VU LEARNING HUB

vulearninghub.xyz

HIGHLIGHTED HANDOUT

CS406

Web Based Database Applications

CS406 by VU Learning HUB

Prepared for VU students • Free study resources

https://vulearninghub.xyz

Handouts: Web-based Database Applications

Table of Contents

TOPIC-1:Course Introduction 8

Module-01 8

Module-02 10

Module-03 12

Module-04 14

Module-05 15

Module-06 18

Module-07 20

Module-08 22

Module-09 23

Module-10 25

Module-11 26

Module-12 28

Module-13 32

Module-14 34

Module-15 37

Module-16 40

Module-17 42

Module-18 44

Module-19 46

Module-20 48

Module-21 50

Module-22 52

Module-23 54

Module-24 57

Module-25 60

Module-26 64

Module-27 67
Module-28 70
Module-29 73
Module-30 76
TOPIC-2: Web application vs. Web site 78
Module-31 78
Module-32 81
Module-33 83
TOPIC-3: Building E-Commerce SMB 86
Module-34 86
Module-35 89
Module-36 92
Module-37 95
Module-38 97
Module-39 100
Module-40 103
Module-41 106
Module-42 109
Module-43 112
Module-44 115
Module-45 118
Module-46 122
Module-47 125
Module-48 128
Module-49 131
Module-50 134
TOPIC-4: How databases work on web? 135
Module-51 135
Module-52 138

Module-53 141
Module-54 143
Module-55 144
Module-56 146
Module-57 149
Module-58 152
TOPIC-5: Database driven website 155
Module-59 155
Module-60 159
TOPIC-6: Understanding database schemas 162
Module-61 162
Module-62 166
Module-63 169
Module-64 172
Module-65 175
Module-66 178
Module-67 181
Module-68 185
Module-69 189
Module-70 193
Module-71 196
Module-72 199
Module-73 202
TOPIC-7: Web database connectivity 205
Module-74 205
Module-75 208
Module-76 211
Module-77 213
Module-78 216

Module-79 219
Module-80 221
Module-81 223
Module-82 225
Module-83 229
Module-84 232
TOPIC-8: Web database operations 235
Module-85 235
Module-86 237
Module-87 240
Module-88 243
TOPIC-9: Rapid Application Development (RAD) 247
Module-89 247
Module-90 250
Module-91 252
Module-92 255
Module-93 258
Module-94 262
Module-95 265
Module-96 268
Module-97 271
Module-98 275
Module-99 277
Module-100 279
Module-101 282
TOPIC-10: SQL on the web 284
Module-102 284
Module-103 287
Module-104 289

Module-105 292
Module-106 295
Module-107 298
Module-108 302
Module-109 304
Module-110 307
Module-111 310
Module-112 313
Module-113 315
Module-114 317
Module-115 320
Module-116 323
Module-117 325
Module-118 328
Module-119 330
Module-120 332
TOPIC-12: JavaScript 334
Module-121 334
Module-122 336
Module-123 338
Module-124 340
Module-125 342
Module-126 344
Module-127 347
Module-128 349
Module-129 352
Module-130 355
Module-131 357
Module-132 359

TOPIC-13: Server side operations 362
Module-133 362
Module-134 365
Module-135 368
Module-136 371
Module-137 374
Module-138 377
TOPIC-14: NoSQL 380
Module-139 380
Module-140 383
Module-141 385
Module-142 387
Module-143 390
Module-144 392
Module-145 394
Module-146 396
Module-147 399
Module-148 402
Module-149 405
Module-150 408
Module-151 411
Module-152 415
TOPIC-15: Search Engine Optimization (SEO) 418
Module-153 418
Module-154 419
Module-155 423
Module-156 425
Module-157 429
Module-158 432

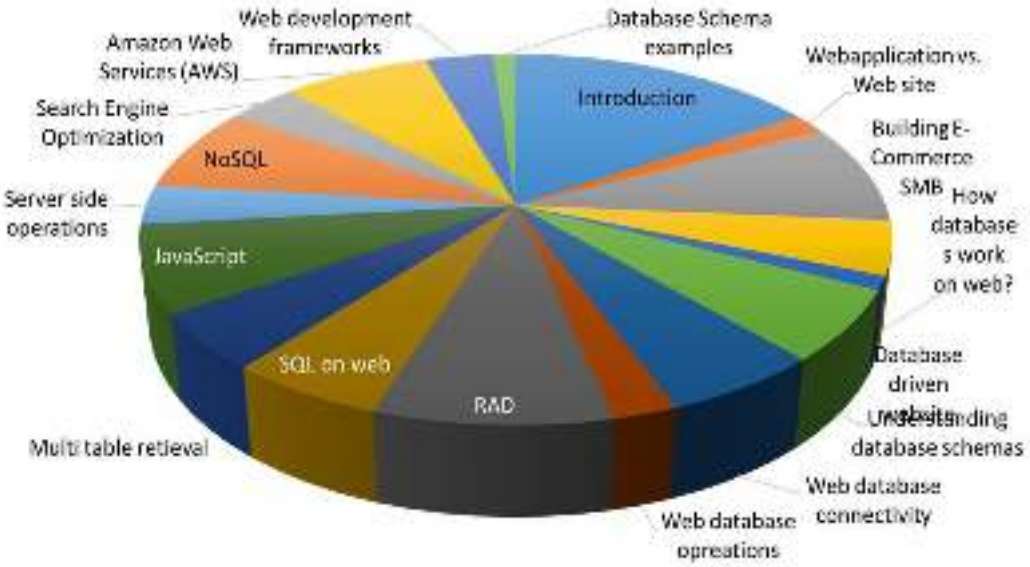
TOPIC-16: Amazon Web Services (AWS) 435
Module-159 435
Module-160 439
Module-161 441
Module-162 443
Module-163 445
Module-164 447
Module-165 448
Module-166 451
Module-167 455
Module-168 458
Module-169 461
Module-170 464
TOPIC-17: Web development frameworks 467
Module-171 467
Module-172 470
TOPIC-18: Database Schema examples 472
Module-173 472
Module-174 475
Module-175 476
Module-176 478
Module-177 479

TOPIC-1: Course Introduction

Module-01

1. Course Introduction
2. Web application vs. Web site
3. Building E-Commerce SMB
4. How databases work on web?
5. Database driven website
6. Understanding database schemas
7. Web database connectivity
8. Web database operations
9. Rapid Application Development (RAD)
10. SQL on web
11. Multi table retrieval
12. JavaScript
13. Server side operations
14. NoSQL
15. Search Engine Optimization
16. Amazon Web Services (AWS)
17. Web development frameworks
18. Database Schema examples

Course Breakdown



Main Text

1. Database-Driven Web Sites (1999), by Jesse Feiler
2. Beginning PHP programming (2009), by Matt Doyle
3. JavaScript Step by Step, 3rd edition (2013), By Steve Suehring
4. A step-by-step guide for launching your SMB on the web (2016) by Andrea Rowland
5. NoSQL For Dummies (2015) by Adam Fowler
6. Amazon Web Services For Dummies (2015) by Bernard Golden
7. SEO.101 Learn the Basics of Search Engine Optimization (2020) By Loren Baker et. al

Web References

A step-by-step guide for launching your SMB on the web
https://uk.godaddy.com/blog/wp-content/uploads/2016/03/Get-your-small-business-online-free-ebook.pdf
Web database connectivity (MIT notes Chap-17)
https://www.cs.uct.ac.za/mit_notes/database/pdfs/chp17.pdf
Database Management Systems
https://www.tutorialspoint.com/dbms/dbms_pdf_version.htm
Database Answers (Schemas)
http://www.databaseanswers.org/data_models/index.htm
Web Application Architecture Guide, publisher Microsoft
http://cis.msjc.edu/CSIS116B/Resources/WebArchitecturePocketGuide.pdf

About the Instructor

Education
PhD Computer Science
University of Stirling, UK
MSc Computer Science
University of Southern California, USA
MSc Computer Engineering
University of Southern California, USA
BS Electrical Engineering (honors)
University of Engineering & Technology, Lahore

Module-02

WEF (World Economic Forum) Report

1. Analysis in partnership between WEF's **New Metrics CoLab**, and data scientists at partner companies: **Burning Glass Technologies, Coursera and LinkedIn**.
2. The report provides insights into emerging employment opportunities across global economy and details regarding the skill sets needed to leverage those opportunities

Key findings of report

1. Demand for both "digital" and "human" factors is driving growth in the professions of the future.
2. **5+2** emerging professional clusters and **96** jobs of tomorrow varying in individual rate of growth and scale of job opportunities offered in the aggregate.
3. Growth in these clusters and jobs is largest among **care** roles and smallest among **green** professions.
4. The highest-growth jobs of tomorrow span all seven profession clusters.
5. The highest-demand skills required in these emerging professional clusters span both technical and cross-functional skills.

Professional Cluster	2020	2022
figues extrapolated from data for 20 economies (LinkedIn)		
	Number of oppertunities (per 10,0000)	
Data and AI	78	123
Engineering and Cloud Computing	60	91
People and Culture	47	58
Product Department	32	44
Sales, Marketing and Content	87	125
figures extrapolated from data for the United States (Burning Glass)		
Care Economy	193	260
Green Economy	9	14
ALL CLUSTERS	506	715

Emerging Jobs

1 Site Reliability Engineer / Cloud Computing /
2 Python Developer / Engineering /
3 Full Stack Engineer / Engineering /
3 Javascript Developer / Engineering /
5 Back End Developer / Engineering /
6 Frontend Engineer / Engineering /
6 Software Developer Dotnet / Engineering /
8 Platform Engineer / Cloud Computing /
9 Development Specialist / Engineering /
10 Cloud Engineer / Cloud Computing /
10 DevOps Engineer / Cloud Computing /
12 Cloud Consultant / Cloud Computing /
13 DevOps Manager / Cloud Computing /
14 Technology Analyst / Engineering /

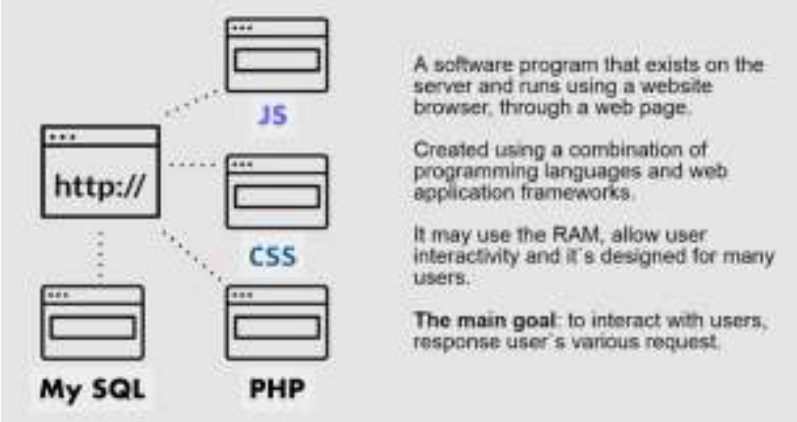
Top 10 skills

1 Development Tools
2 Web Development
3 Data Storage Technologies
4 Software Development Life Cycle (SDLC)
5 Computer Networking
6 Human Computer Interaction
7 Technical Support
8 Digital Literacy
9 Business Management
10 Employee Learning & Development

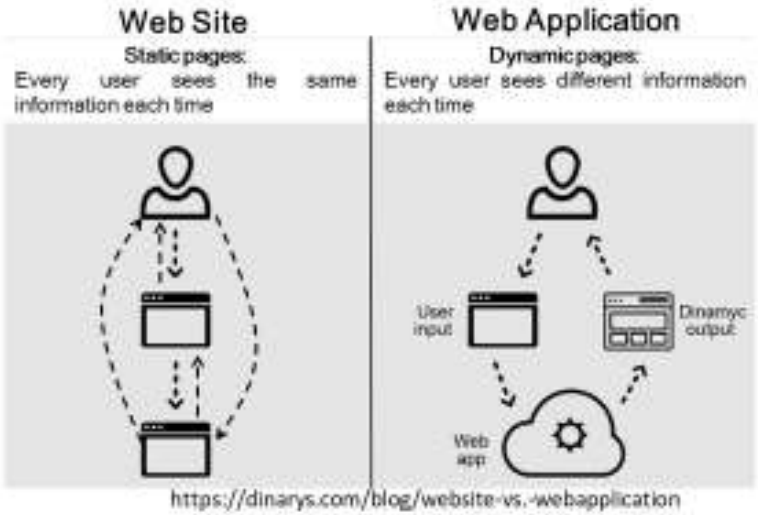
Module-03

What is a website?

What is a web application?



Website Vs. Web application



Web Site	Web Application
1. Marketing focused used to tell customers about business brand.	1. Web tools that perform certain functions and are less focused around brand marketing.
2. Perform data presentation (reading & comprehending).	2. Permit data manipulation (analysis, search).
3. Don't require authentication services.	3. Can be called as 'web portals' or 'online stores' and usually require authentication.
4. Used for communication purposes.	4. Used for automating tasks to achieve certain goals.
5. Less resource intensive and need less processing power.	5. Resource intensive and need high processing power.
6. Less complex and relatively easy to code.	6. Generally complex as well as coding difficulty is high.

Module-04

Internet and Web facts

87% of adults in America use the internet – and they turn to the web more than any other source to find local businesses and services.

if you' re not online, **you don' t exist** to countless potential customers.

Benefits of being online

- Be More Accessible
- Build Your Brand
- Establish Credibility
- Connect With Customers
- Generate Sales Leads

Step-1: Get a Domain Name

- What' s a domain name?
- What' s a domain extension?
- Worksheet: Pick the perfect domain
- Register your domain

Step-2: Plan your website

- Start with the End in Mind
- Worksheet: Planning Your Website
- Prepare to sell online

Step-3: Select your designer

- Site builders & designers
- Chart: Choose Your Best Website Building Option

Module-05

Step-4: Create Remarkable content

Be mobile-ready

Worksheet: Content Creation for Websites

- Product/service page
- About page
- Testimonials page
- Contact us page

Tell your story

Design basics

- Logo
- Color
- Fonts
- Layouts

Image tips

- Show not Tell
- Take them yourself
- Use Quality Stock Images

Step-5: Get Found

SEO (Search Engine Optimization)

How does SEO work?

- Page content
- Keywords
- Meta tags

- Website navigation
- Site map
- Link building
- Image optimization

Fun with keywords

- Relevant
- Specific
- Strategic

How to optimize images

- Align target keyword and image
- Use and optimize image ALT text
- Page text corresponds to image

Tag, you're it

- Title
- Header
- Meta

Do I need a blog?

Reviews, directories & maps

Step-6: Connect with Social Media

Why social media matters?

- Build relationships
- Build your brand
- Drive traffic to your website
- Establish expertise

- Make sales

Chart: Comparing Popular Social Networks

- Twitter
- Facebook
- LinkedIn
- Instagram

Module-06

Step-7: Drive traffic to your website

How to get visitors?

- Create compelling content
- Write professional emails
- Maximize Blogs
- Join Online Communities
- Run Promotions
- Leverage Reviews

Email marketing

Paid options

- Fast and cheap(er).
- Captive audience.
- Easy to measure.

Step-8: Generate leads

- Perform conversions
- Collect email addresses
- Invest in email marketing
- The call-to-action

Email marketing best practices

1. Only for registered users.
2. Stick to a schedule.
3. Only include requested subject matter.
4. Create interest groups.
5. Avoid spam and trash folders.
6. Give them reading options.
7. Put a name to your email.
8. Keep building your contact list.

Step-9: Measure, Improve and Grow

What is next?

- Stay social
- Get mobile
- Hone your focus
- Keep website in shape

Track your website's performance

- Visitors
- Pages
- Referrers
- E-Commerce
- Mobile
- Social Media
- Advertising

Module-07

At various times, "database" can refer to any, several, or all of the following:

1. A database may be a body of data—often a large body of information—usually stored and indexed on a computer or other electronic device.
2. A database may be the software (and sometimes hardware) that is used to store, retrieve, and manipulate data.
3. A database may be a combination of data, software, and custom procedures that address a given problem. CRM consisting of customer records (data); a database management system (software) such as Access, DB2, or Oracle; and custom layouts, reports, and procedures that run operational systems.

In this course, above concepts are expressed with three different terms:

1. Database is used in the first sense: a collection of data.
2. Database management system (DBMS) is the term used for the second sense.
3. Database project used for the final sense the combination of data, software, and the reports, layouts, and procedures that make everything work together for a given purpose.

The Web described

The World Wide Web (WWW), commonly known as the Web, is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs, such as https://www.example.com/), which may be interlinked by hypertext, and are accessible over the Internet. The resources of the WWW are transferred via the Hypertext Transfer Protocol (HTTP) and may be accessed by users by a software application called a web browser and are published by a software application called a web server.

British scientist Tim Berners-Lee invented the World Wide Web in 1989. He wrote the first web browser in 1990 while employed at CERN near Geneva, Switzerland. The browser was released outside CERN in 1991, first to other research institutions starting in January 1991 and then to the general public in August 1991. The World Wide Web has been central to the development of the Information Age and is the primary tool billions of people use to interact on the Internet.

Attributes of the WWW

- The web can manage large amounts of data

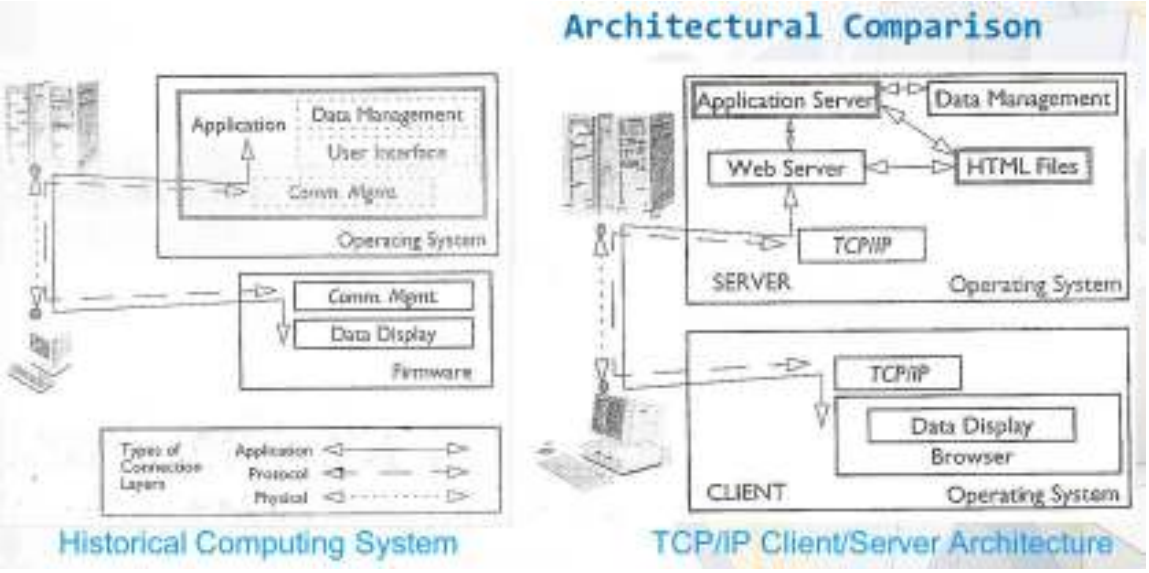
- Web data changes quickly (often unpredictably)
- Web data needs to be selected and displayed in different ways (sometimes unpredictably)
- The web has tools to manipulate its data
- Web pages contain meta-data
- Web pages are often shared across time and space

Module-08

To understand how databases work on the Web, it is helpful to briefly look back at the evolution of contemporary software architecture. This path led from the earliest days of mainframe computers and dumb terminals (originally teletype machines) to the modern architecture that may incorporate mainframes, personal computers, and a variety of networking technologies. This module examines the major steps along this road; it then deals with the contemporary design of systems.

This history is an oversimplification. Furthermore, it has the benefit of hind- sight: at the time, it was not always clear where the road would lead.

1. Mainframes and dumb terminals (from the earliest years of the computer age–the **1950s**)
2. The rise of operating systems and structured programming (starting in the **1960s**)
3. The development of the personal computer (in the late **1970s**)
4. The growth of the Internet and the World Wide Web (early **1990s**)



Module-09

A database schema represents the logical configuration of all or part of a relational database. It can exist both as a visual representation and as a set of formulas known as integrity constraints that govern a database. These formulas are expressed in a data definition language, such as SOL. As part of a data dictionary, a database schema indicates how the entities that make up the database relate to one another, including tables, views, stored procedures, and more.

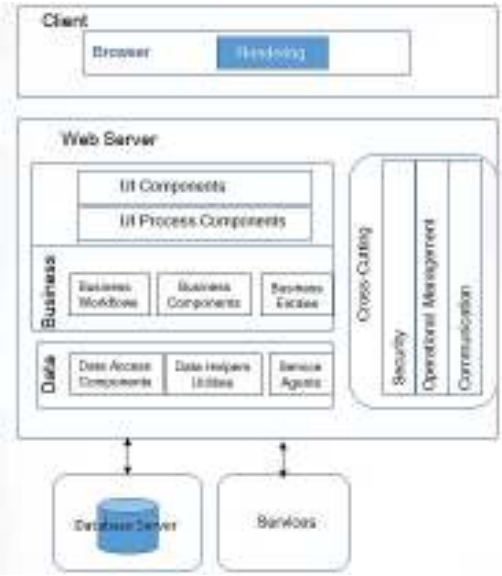


Figure: 3-Tier Architecture

Typically, a database designer creates a database schema to help programmers whose software will interact with the database. The process of creating a database schema is called data modeling. When following the three-schema approach to database design, this step would follow the creation of a conceptual schema. Conceptual schemas focus on an organization's informational needs rather than the structure of a database.

https://www.guru99.com/dbms-schemas.html

Database systems comprise of complex data structures. Thus, to make the system efficient for retrieval of data and reduce the complexity of the users, developers use the method of Data Abstraction.

There are mainly three levels of data abstraction or schemas:

- Internal Level: Actual PHYSICAL storage structure and access paths.
- Conceptual or Logical Level: Structure and constraints for the entire database
- External or View level: Describes various user views

Web Application Frame

- Authentication
- Authorization
- Caching
- Exception Management
- Logging and Instrumentation
- Navigation
- Page Layout (UI)
- Page Rendering
- Request Processing
- Validation
- Deployment Considerations

Module-10

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

Entities are represented by means of their properties called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

i) Simple attribute ii) Composite attribute and iii) Derived attribute iv) single-valued v) multi-valued

Any database which has tables and constraints is not a relational database system (RDBMS). There are certain rules for a database to be perfect RDBMS. These rules are developed by Dr Edgar F Codd (EF Codd) in 1985 to define a perfect RDBMS. For a RDBMS to be a perfect RDBMS, it has to follow his rules. But no RDBMS can obey all his rules. Till now, there is hardly any commercial product that follows all the 13 **Codd's rules**.

Codd's twelve rules are a set of thirteen rules (numbered zero to twelve) proposed by Edgar F. Codd, a pioneer of the relational model for databases, designed to define what is required from a database management system in order for it to be considered relational, i.e., a relational database management system (RDBMS).

Module-11

The Web has been expanding at an incredible speed and even while you are reading this, hundreds and thousands of people are getting 'online' and hooked to the Web. Reactions to this technology are understandably mixed.

The database technology has been around for a long time now, and for many business and government offices, databases systems have already become an essential and integral part of the organization. Now the new technology has given the 'old' a shot in the arm, and the combination of the two creates many exciting opportunities for developing advanced database applications.

As far as database applications are concerned, a key aspect of the WWW technology is that it offers a brand new platform to collect, deliver and disseminate information. Via the Web, a database application can be made available, interactively, to users and organizations anywhere in the world.

Components of a database application

- Browser layer
- Application logic layer
- Database connection layer
- Database layer

2-tier client-server architecture

3-tier client-server architecture

Components of a database application

Web database applications may be created using various approaches. However, there are a number of components that will form essential building blocks for such applications. In other words, a Web database application should comprise the following four layers (i.e. components):

- Browser layer
- Application logic layer
- Database connection layer
- Database layer

Database gateways

A Web database gateway is a bridge between the Web and a DBMS, and its objective is to provide a Web-based application the ability to manipulate data stored in the database. Web database gateways link stateful systems (i.e. databases) with a stateless, connectionless protocol (i.e. HTTP). HTTP is a stateless protocol in the sense that each connection is closed once the server provides a response. Thus, a Web server will not normally keep any record about previous requests.

A Web server will not normally keep any record about previous requests. This results in an important difference between a Web-based client-server application and a traditional client-server application:

a) In a Web-based application, only one transaction can occur on a connection. In other words, the connection is created for a specific request from the client. Once the request has been satisfied, the connection is closed. Thus, every request involving access to the database will have to incur the overhead of making the connection.

b) In a traditional application, multiple transactions can occur on the same connection. The overhead of making the connection will only occur once at the beginning of each database session.

There are a number of different ways to create Web database gateways. Generally, they can be grouped into two categories: client-side solutions and server-side solutions, as illustrated.

Module-12

Server side web database programming

CGI (Common Gateway Interface): A protocol for allowing Web browsers to communicate with Web servers, such as sending data to the servers. Upon receiving the data, the Web server can then pass them to a specified external program (residing on the server host machine) via environment variables.

Advantages: The main advantages of CGI are its simplicity, language independence, Web server independence and its wide acceptance.

Disadvantages: The communication between a client (browser) and the database server must always go through the Web server in the middle, which may cause a bottleneck.

Extended CGI

- Language independence: As with CGI, FastCGI is a protocol and not dependent on any specific language.
- Open standard: Like CGI, FastCGI is positioned as an open standard.

HTTP server APIs and server modules: The server equivalent of browser extensions.
Server vendor modules: Prefabricated applications written in some server APIs.

Advantages of server APIs and modules:

- Server speed
- Resource sharing
- Range of functionality

Important issues:

- Server architecture dependence
- Platform dependence
- Programming language

Proprietary HTTP servers: A server application that handles HTTP requests and provides additional functionality that is not standard or common among available HTTP servers.

Connecting to the databases

Database API libraries: Are at the core of every Web database application and gateway. Regardless how a Web database application is built (whether by manually coding CGI programs

or by using a visual application builder), database API libraries are the foundation of database access.

Database-independent APIs: ODBC Pioneered by Microsoft. It is called ODBC (Open Database Connectivity) and is supported by all of the most popular databases such as Microsoft Access and Oracle. ODBC requires a database-specific driver or client to be loaded on the database client machine.

Benefits of database APIs: Applications created with native database APIs are more efficient than those with database-independent APIs.

Shortcomings of database APIs: The most notable disadvantage of programming in database API is complexity.

GUI application builders: For developers accustomed to point-and-click application programming, these tools help speed the development process. Visual Basic and/or Microsoft Access developers should find such a tool intuitive and easy to use.

Shortcomings of visual tools: Depending on the sophistication of the package used, the resulting programs may be slower to execute than similar programs coded by an experienced programmer.

Security Issues in Web Database Applications

Security risks exist in many areas of a Web database application. This is because the very foundations of the Internet and Web – TCP/IP and HTTP – are very weak with respect to securities. Without special software, all Internet traffic travels in the open and anyone with a little bit skill can intercept data transmission on the Internet. If no measures are taken, there will be many security loopholes that can be explored by malicious users on the Internet.

In general, security issues in Web database applications include the following:

- Data transmission (communication) between the client and the server is not accessible to anyone else except the sender and intended receiver (privacy).
- Data cannot be changed during transmission (integrity).
- The receiver can be sure that the data is from the authenticated sender (authenticity).
- The sender can be sure the receiver is the genuinely intended one (nonfabrication).
- The sender cannot deny he/she sent it (non-repudiation).
- The request from the client should not ask the server to perform illegal or unauthorized actions.

• The data transmitted to the client machine from the server must not be allowed to contain executables that will perform malicious actions.

Security Issues in Web Database Applications

Proxy servers: A proxy server is a system that resides between a Web browser and a Web server.

Firewalls: A firewall is a system designed to prevent unauthorized access to or from a private network (intranet).

Digital signatures: A digital signature consists of two pieces of information: a string of bits that is computed from the data (message) that is being signed along with the private key of the requester for the signature. The signature can be used to verify that the data is from a particular individual or organization.

Digital certificates: It is an attachment to a message used for verifying the sender's authenticity. Such a certificate is obtained from a Certificate Authority (CA), which must be a trust-worthy organization.

Kerberos: It is a server of secured usernames and passwords.

Secure sockets layer (SSL) and secure HTTP (S-HTTP): SSL is an encryption protocol developed by Netscape for transmitting private documents over the Internet.

Java security: If Java is used to write the Web database application, then many security measures can be implemented within Java.

ActiveX security: Java applet programming provides as many features as possible without compromising the security of the client. In contrast, ActiveX's security model places the responsibility for the computer's safety on the user (client).

Performance issues in Web database applications

Web database applications are very complex, more so than stand-alone or traditional client-server applications. They are a hybrid of technology, vendors, programming languages, and development techniques. Many factors work together in a Web database application and any one of them can hamper the application's performance. The following is a list of issues concerning performance:

• **Network consistency:** The availability and speed of network connections can significantly affect performance.

• **Client and server resources:** This is the same consideration as in the traditional client-server applications. Memory and CPU are the scarce resources.

• **Database performance:** It is concerned with the overhead for establishing connections, database tuning, and SQL query optimization.

• **Content delivery:** This is concerned with the content's download time and load time.

• **State maintenance:** It should always minimize the amount of data transferred between client and server and minimize the amount of processing necessary to rebuild the application state.

• **Client-side processing:** If some processing can be carried out on the client-side, it should be done so. Transmitting data to the server for processing that can be done on the client-side will degrade performance.

• **Programming language:** A thorough understanding of the tasks at hand and techniques available can help choose the most suitable language for implementing the application.

Module-13

Overview

Alan Turing and John von Neumann are considered two of the most important developers of modern computers. Both realized that a computer could store data in electronic form and that it could store instructions in electronic form—and that the instructions could act upon stored instructions in precisely the same way in which they could act upon data. In the 1940s, the ability to treat data and computer instructions as (sometimes) interchangeable made modern computers possible. Within two decades, however, this flexibility started to cause problems, and designers (particularly designers of operating systems) starting erecting barriers to the manipulation of computer instructions as if they were data (the famous cry of "no non-reentrant code“ was first heard at this time). Within not too much more time, the modification of program instructions within applications (rather than just within operating systems) started to cause problems.

Programming with databases

The development of database software—in which the data was physically removed from the application software—helped to rebuild the wall that Turing and von Neumann had demolished. Database programming consists of this very rigid distinction between data and the instructions that act upon it. The pendulum of computer science development continues to swing back and forth be-tween this pole and the other pole—that at which data and instructions are indistinguishable. (In some ways, objects—particularly distributed objects—represent this opposite pole: the union of data and instructions.)

Meanwhile, back in the world of Web page design, the distinction between data and computer instructions started out not mattering very much. Recent developments (such as applets and XML) are raising these barriers again as ontology does appear to recapitulate phylogeny in the world of software design. This module covers four major topics in database programming:

- Result Sets
- Timing and Performance
- Transaction Processing
- The Data and Nothing But the Data

Database programming is somewhat different from procedural programming—the step-by-step processing of data as you encounter it. The principles of database programming—particularly result sets and transaction processing make it quite amenable for use with Web sites.

Procedural programming

Procedural Programming may be the first programming paradigm that a new developer will learn. Fundamentally, the procedural code is the one that directly instructs a device on how to finish a task in logical steps. This paradigm uses a linear top-down approach and treats data and procedures as two different entities. Based on the concept of a procedure call, Procedural Programming divides the program into procedures, which are also known as routines or functions, simply containing a series of steps to be carried out.

Simply put, Procedural Programming involves writing down a list of instructions to tell the computer what it should do step-by-step to finish the task at hand.

Programming with databases

Database-driven Web sites often merge data from databases with data that is contained in standard HTML pages. (Your name and address, for instance, may be part of your HTML page rather than being retrieved from the database.) Since it is almost always easier to type information into an HTML page than to enter it into a database (and to plan for its retrieval), you may find that with time more and more of your site's data finds its way from the database environment into the HTML environment. Knowing that this is a danger, you should be able to guard against it. Whereas it is almost always easier to enter data into an HTML page than into a database, it is almost always easier to maintain it when it is in a database. This is a typical short- versus long-term trade-off, but the long-term benefits of database-driven Web sites are so great that you should be ever vigilant.

Module-14

Introducing RAD

Rapid application development is an agile software development approach that focuses more on ongoing software projects and user feedback and less on following a strict plan.

1. Define Requirements: Rather than making you spend months developing specifications with users, RAD begins by defining a loose set of requirements. We say loose because among the key principles of rapid application development is the permission to change requirements at any point in the cycle.

2. Prototype: In this rapid application development phase, the developer's goal is to build something that they can demonstrate to the client. This can be a prototype that satisfies all or only a portion of requirements (as in early stage prototyping).

3. Absorb Feedback: With a recent prototype prepared, RAD developers present their work to the client or end-users. They collect feedback on everything from interface to functionality—it is here where product requirements might come under scrutiny.

4. Finalize Product: During this stage, developers may optimize or even re-engineer their implementation to improve stability, maintainability, and a third word ending in '-ility.' They may also spend this phase connecting the back-end to production data, writing thorough documentation, and doing any other maintenance tasks required before handing the product over with confidence.

RAD Advantages

Speed

In the traditional waterfall approach, developers were unlikely to go on vacation after delivering the product. Clients would invariably request changes ranging from interface to functionality after first delivery. With RAD, projects are more likely to finish on time and to the client's satisfaction upon delivery.

Cost

In rapid application development, developers build the exact systems the client requires, and nothing more. In waterfall, IT risks building and fleshing out complex feature sets that the client may choose to gut from the final product. The time spent building

Developer Satisfaction

Regardless of how proud developers are of their work, if the client isn't satisfied, developers don't receive the accolades they so desperately seek. In RAD, the client is there every step of the way and the developer has the opportunity to present their work frequently. This gives them the confidence that when the final product is delivered, their work receives appreciation.

RAD Disadvantages

Scale

A close-knit team of developers, designers, and product managers can easily incorporate RAD practices because they have direct access to one another. When a project expands beyond a single team or requires inter-team communication, the development cycle invariably slows and muddles the direction of the project. Simply put, it's difficult to keep a large group of people on the same page when your story is constantly changing.

Commitment

In waterfall, the client spent most of their time apart from the development team after completing specifications. This allowed clients to focus on their primary tasks and developers to focus on building. In RAD, the frequent cycle of prototypes requires

developers and clients to commit to frequent meetings that, on the outset, may appear to consume unnecessary cycles.

Interface-Focus

RAD methodology motivates developers to find the perfect solution for the client. The client judges the quality of the solution by what they can interact with—and often, all they interact with is a facade. As a consequence, some developers forego best practices on the back-end to accelerate development of the frontend-focused prototype. When it's time to deliver a working product, they patch up the jerry-rigged server code to avoid a refactor.

What is PHP?

PHP is a programming language for building dynamic, interactive Web sites. PHP programs run on a Web server, and serve Web pages to visitors on request. You can embed PHP code within HTML Web pages, making it very easy for you to create dynamic content quickly.

Introducing PHP

Why use PHP?

Large number of Internet service providers (ISPs) and Web hosting companies that support it. Today hundreds of thousands of developers are using PHP, and it's not surprising that there are so many, considering that several million sites are reported to have PHP installed.

ASP and ASP.NET have a commercial license, which can mean spending additional money on server software, and hosting is often more expensive as a result. Secondly, ASP and ASP.NET are fairly heavily tied to the Windows platform, whereas the other technologies in this list are much more cross - platform.

Java: Java is another general - purpose language that is commonly used for Web application development. Thanks to technologies like JSP (JavaServer Pages) and servlets, Java is a great platform for building large - scale, robust Web applications.

You can easily build and deploy Java - based Web sites on virtually any server platform, including Windows, Linux, and FreeBSD. A steep learning curve, it ' s harder to find a Web hosting company that will support JSP, whereas nearly all hosting companies offer PHP hosting.

Module-15

PHP Basics

Variables are a fundamental part of any programming language. A variable is simply a container that holds a certain value. Variables get their name because that certain value can change throughout the execution of the script.

Naming Variables: A variable consists of two parts: the variable's name and the variable ' s value. Because you ' ll be using variables in your code frequently, it ' s best to give your variables names you can understand and remember.

Creating Variables: Creating a variable in PHP is known as declaring it. Declaring a variable is as simple as using its name in your script: $my_first_variable;

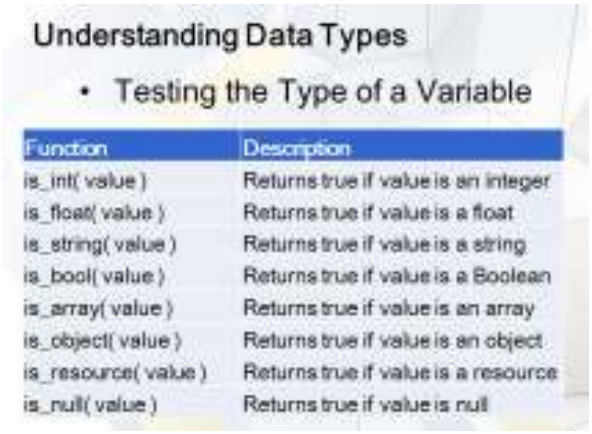
On first seeing a variable ' s name in a script, PHP automatically creates variable at that point.

About Loose Typing: PHP converts a variable's data type automatically, depending on the context in which the variable is used. For example, you can initialize a variable with an integer value; add a float value to it, thereby turning it into a float; then join it onto a string value to produce a longer string. In contrast, many other languages, such as Java, are *strongly – typed*

Testing the Type of a Variable: You can determine the type of a variable at any time by using PHP ' s gettype() function. To use gettype() , pass in the variable whose type you want to test. The function then returns the variable's type as a string.

Changing a Variable's Data Type: Earlier, you learned how to change a variable ' s type by assigning different values to the variable. However, you can use PHP ' s settype() function to change the type of a variable while preserving the variable ' s value as much as possible.

Changing Type by Casting: You can also cause a variable ' s value to be treated as a specific type using a technique known as type casting . This involves placing the name of the desired data type in parentheses before the variable ' s name. Note that the variable itself remains unaffected; this is in contrast to settype() , which changes the variable ' s type.



Operators and Expressions:

So far you ’ ve learned what variables are, and how to set a variable to a particular value, as well as how to retrieve a variable ’ s value and type. However, life would be pretty dull if this was all you could do with variables. This is where operators come into play. Using an operator, you can manipulate the contents of one or more variables to produce a new value. For example, this code uses the addition operator (+) to add the values of $x and $y together to produce a new value:

echo $x + $y;

So an *operator* is a symbol that manipulates one or more values, usually producing a new value in the process. Meanwhile, an *expression* in PHP is anything that evaluates to a value; this can be any combination of values, variables, operators, and functions. In the preceding example, $x + $y is an expression. Here are some more examples of expressions:

$x + $y + $z

$x - $y

$x

5

true

gettype($test_var)

Understanding Operator Precedence

With simple expressions, such as 3 + 4 , it ’ s clear what needs to be done (in this case, “ add 3 and 4 to produce 7 ”). Once you start using more than one operator in an expression, however, things aren ’ t so clear - cut. Consider the following example:

3 + 4 * 5

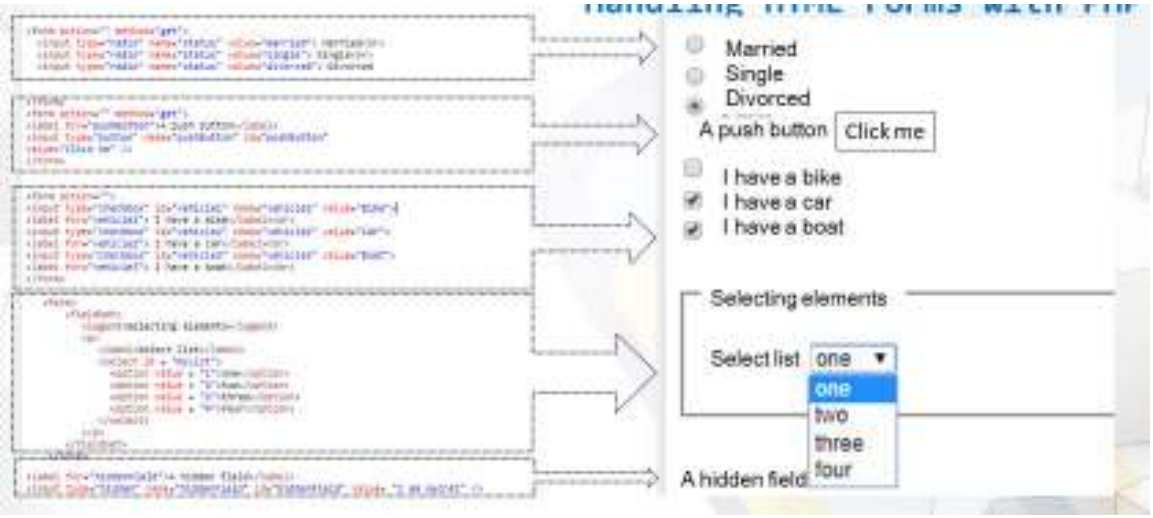
Is PHP supposed to add 3 to 4 to produce 7, then multiply the result by 5 to produce a final figure of 35?

Or should it multiply 4 by 5 first to get 20, then add 3 to make 23?

This is where operator precedence comes into play. All PHP operators are ordered according to precedence. An operator with a higher precedence is executed before an operator with lower precedence. In the case of the example, * has a higher precedence than + , so PHP multiplies 4 by 5 first, then adds 3 to the result to get 23.

Module-16

Handling HTML Forms with PHP



How It Works:

This XHTML Web page contains the most common types of form controls you're likely to come across. First, the form itself is created:

<form action="" method="get">

Notice that the form is created with the get method. This means that the form field names and values will be sent to the server in the URL. You learn more about the get and post methods shortly. Meanwhile, the empty action attribute tells the browser to send the form back to the same page (web_form.html). In a real-world form this attribute would contain the URL of the form handler script.

Next, each of the form controls is created in turn. Most controls are given a name attribute, which is the name of the field that stores the data, and a value attribute, which contains either the fixed field value or, for fields that let the users enter their own value, the default field value. You can think of the field

names and field values as being similar to the keys and values of an associative array.

Most controls are also given an associated label element containing the field label. This text describes the field to the users and prompts them to enter data into the field. Each label is associated with its control using its for attribute, which matches the corresponding id attribute in the control element.

Handling HTML Forms with PHP

Capturing Form Data with PHP: You now know how to create an HTML form, and how data in a form is sent to the server. How do you write a PHP script to handle that data when it arrives at the server?

Handling multi - value fields : The trick is to add square brackets ([]) after the field name in your HTML form. Then, when the PHP engine sees a submitted form field name with square brackets at the end, it creates a nested array of values within the $_GET or $_POST (and $_REQUEST) superglobal array, rather than a single value. You can then pull the individual values out of that nested array.

Generating Web Forms with PHP: As with generating any HTML markup, you can use two common approaches to generate a form within PHP: you can use echo or print statements to write out the markup for the form, or you can separate the PHP code from the form markup using the < ?php and ? > tags. You can also use a mixture of the two techniques within the same script.

Storing PHP Variables in Forms: Earlier in the chapter you were introduced to hidden fields. A hidden field is a special type of input element that can store and send a string value, just like a regular text input control. However, a hidden field is not displayed on the page (although its value can be seen by viewing the page source), and therefore its value cannot be changed by the users when they are filling out the form.

You have already seen how to create a file select field at the start of this chapter. In addition, a form containing a file select field must use the post method, and it must also have an enctype="multipart/form - data" attribute in its <form> tag

Once form data is uploaded to server, the PHP engine recognizes that the form contains an uploaded file or files, and creates a superglobal array called $_FILES containing various pieces of information about the file or files. Each file is described by an element in the $_FILES array keyed on the name of the field that was used to upload the file.

PHP allows you to limit the size of uploaded files in a few ways which is done using php.ini file. Once a file has been successfully uploaded, it is automatically stored in a temporary folder on the server. To use the file, or store it on a more permanent basis, you need to move it out of the temporary folder.

URL redirection is often used within form handling code. Normally when you run a PHP script — whether by typing its URL, following a link, or submitting a form — the script does its thing, displays some sort of response as a Web page, and exits. However, by sending a special HTTP header back to the browser from the PHP script, you can cause the browser to jump to a new URL after the script has run.

Module-17

Introducing Databases and SQL

How much data will the application use?

How often will it need access to the data?

How often will it need to modify the data?

How many users are likely to want access to the data at once?

How much will the data grow over time?

How much do I stand to lose if the data is broken, stolen, or lost?

Embedded Databases: An embedded database engine, as its name implies, sits inside the application that uses it (PHP in this case). Therefore it always runs — and stores its data — on the same machine as the host application.

Client - Server Databases: Client - server databases are, generally speaking, more powerful and flexible than embedded databases. They are usually designed for use over networks, enabling many applications in a network to work simultaneously with the same data.

Simple Databases: Simple database engines are, as the name implies, just about the simplest type of database to work with. Essentially, the simple model is similar to an associative array of data.

Relational Databases: Relational databases offer more power and flexibility than simple databases, and for this reason they tend to be a more popular choice.

In principle, you can use any of these database systems in your PHP applications. You can even hook one application up to several different database engines. To keep these chapters to a reasonable length, however, you ’ ll focus on just one database engine: MySQL.

Compared to the other choices, it offers several advantages:

- It ’ s one of the most popular databases being used on the Web today
- It ’ s freely available as a download to install and run on your own machine
- It ’ s easy to install on a wide range of operating systems (including UNIX, Windows, and Mac
- OS X) It ’ s available as a relatively cheap feature in many Web hosting packages

- It ’ s simple to use and includes some handy administration tools
- It ’ s a fast, powerful system that copes well with large, complex databases, and should stand you in good stead when it comes to larger projects

A quick play with MySQL

Now that you ’ ve set up the MySQL root user, you can start working with databases. In the following sections, you create a new database, add a table to the database, and add data to the table. You also learn how to query databases and tables, update data in tables, and delete data, tables, and databases.

Most of the examples in the following sections show commands, statements, and other SQL keywords being entered using all - uppercase letters. Though SQL keywords are traditionally in uppercase, MySQL also lets you enter keywords in lowercase. So use lowercase if you prefer.

Connecting to MySQL from PHP

Now that you ’ ve set up the MySQL root user, you can start working with databases. In the following sections, you create a new database, add a table to the database, and add data to the table. You also learn how to query databases and tables, update data in tables, and delete data, tables, and databases.

Most of the examples in the following sections show commands, statements, and other SQL keywords being entered using all - uppercase letters. Though SQL keywords are traditionally in uppercase, MySQL also lets you enter keywords in lowercase. So use lowercase if you prefer.

.

Module-18

A brief introduction of JOINs

Why store data in multiple tables?

Basically a single **table** is good when **data** is one-to-one. When you have thousands of rows and multiple columns of **data**, where the **data** is one-to-many, **multiple tables** are better to reduce duplicate **data**

How to retrieve data from multiple tables?

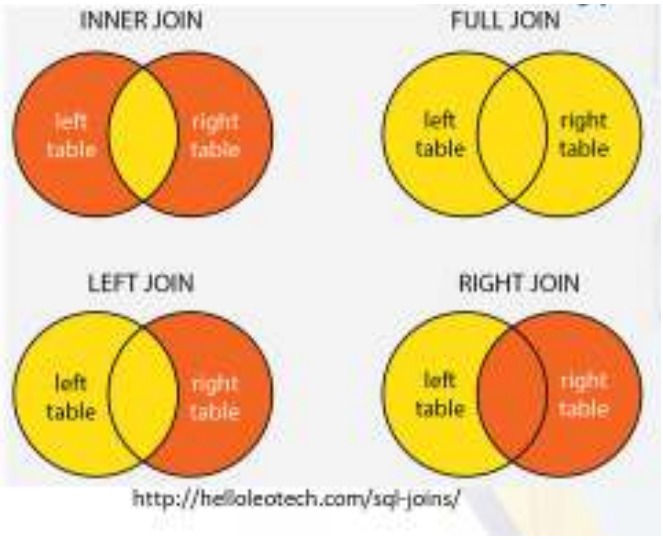
SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables. JOIN Keyword is used in SQL queries for joining two or more tables.

What is a table join?

An SQL **join** clause - corresponding to a **join** operation in relational algebra -combines columns from one or more **tables** in a relational database. It creates a set that can be saved as a **table** or used as it is. A **JOIN** is a means for combining columns from one (self-**join**) or more **tables** by using values common to each.

How many types of joins are there?

There are four basic types of SQL joins: inner, left, right, and full. The easiest and most intuitive way to explain the difference between these **four types** is by using a Venn diagram, which shows all possible logical relations between data sets.



Basic JOIN types

Using Subselects

One of the features that MySQL 4.1 introduces is subselect support, which is a long-awaited capability that allows one SELECT query to be nested inside other. The following is an example that looks up the IDs for event records corresponding to tests ('T') and uses them to select scores for those tests:

```
SELECT * FROM score
WHERE event_id IN (SELECT event_id FROM event WHERE type = 'T');
```

Rewriting Subselects as Joins

For versions of MySQL prior to 4.1, subselects are not available. However, it's o□en possible to rephrase a query that uses a subselect in terms of a join. In fact, even if you have MySQL 4.1 or later, it's not a bad idea to examine queries that you might be inclined to write in terms of subselects; a join is sometimes more efficient than a subselect.

Retrieving from Multiple Tables with UNION

If you want to create a result set by selecting records from multiple tables one a□er the other, you can do that using a UNION statement. UNION is available as of MySQL 4, although prior to that you can use a couple of workarounds (shown later).

Module-19

Overview

Up to now, we have concentrated mainly on connecting to MySQL, either through the command - line tool or through PHP ' s PDO extension, and on creating tables and filling them with data. One of the first SQL statements you came across was a basic SELECT query. There ' s quite a lot more you can do with SELECT , and this module focuses on the different ways you can use queries in PHP scripts to get at the data stored in a MySQL database.

You start off by creating a couple of MySQL tables for a fictional book club database. These tables are used in the examples and scripts throughout this module and the next.

You then take a close look at how to construct SQL SELECT statements so that they access the data you want, arranged in the way you want. You learn how to:

- Limit the number of results returned
- Order and group results
- Query multiple tables at once
- Use various MySQL functions and other features to build more flexible queries

After exploring the theory of SELECT statements, you create a member viewer application that you can use to access the book club tables you will create in the detailed module later.

The BINARY Attribute and Collations: All character data types have a *collation* that is used to determine how characters in the field are compared. By default, a character field ' s collation is case insensitive. This means that, when you sort the column alphabetically (which you learn to do shortly), " a " comes before both " b " and " B ". It also means that queries looking for the text " banana " will match the field values " banana " and " Banana " .
However, by adding the BINARY attribute after the data type definition, you switch the field to a binary collation, which is case sensitive.

The UNIQUE Constraint: You ' ve already seen how you can use the keywords PRIMARY KEY to create an index on a column that uniquely identifies each row in a table. The UNIQUE constraint is similar to PRIMARY KEY in that it creates an index on the column and also ensures that the values in the column must be unique.

The ENUM Data Type: You briefly looked at ENUM columns when learning about data types in the last chapter. An ENUM (enumeration) column is a type of string column where only predefined string values are allowed in the field. For the members table, you created two ENUM fields:
gender ENUM('m', 'f'),

favoriteGenre ENUM(‘crime’, ‘horror’, ‘thriller’, ‘romance’, ‘sciFi’,
‘adventure’, ‘nonFiction’),

Pulling data from multiple tables

Grouping Results: You have seen how to use functions such as count() and sum() to retrieve overall aggregate data from a table, such as how many female members are in the book club. What if you wanted to get more fine-grained information? For example, say you want to find out the number of different page URLs that each member has viewed. You might try this query:

That is no good. All this query has given you is the total number of rows in the table! Instead, you need to *group* the pageUrl count by member ID. To do this, you add a GROUP BY clause. For example:

That is better. By combining an aggregate function, count() , with a column to group by (memberId), you can view statistics on a per - member basis. In this case you can see that members 1 and 6 have each viewed one distinct page, whereas member 3 has visited two different pages.

Now, of course, the member ID on its own isn ’ t very helpful. If you want to know the *names* of the members involved, you have to run another query to look at the data in the members table:

Module-20

JavaScript is more than you might think

JavaScript was originally developed at Netscape sometime in 1995–1996 and was called LiveScript. That was a great name for a new language—and the story could have ended there. However, in an unfortunate decision, marketing people renamed to JavaScript; confusion.

What's in a JavaScript program?: A JavaScript program consists of statements and expressions formed from tokens of various categories, including keywords, literals, separators, operators, and identifiers placed together in an order that is meaningful to a JavaScript interpreter, which is contained in most web browsers. These statements are really not all that complicated to anyone who has programmed in just about any other language. An expression might be: var smallNumber = 4;

What JavaScript can do: JavaScript is largely a complementary language, meaning that it's uncommon for an entire application to be written solely in JavaScript without the aid of other languages like HTML and without presentation in a web browser. Some Adobe products support JavaScript, and Windows 8 begins to change this, but JavaScript's main use is in a browser.

What JavaScript can't do: JavaScript relies on another interface or host program for its functionality. This host program is usually the client's web browser, also known as a *user agent*. Because JavaScript is a client-side language, it can do only what the client allows it to do.

Then came windows: In Windows 8, Microsoft has elevated JavaScript to the same level as other client-side languages, such as Visual Basic and C#, for developing Windows 8 applications.

JavaScript development options: Because JavaScript isn't a compiled language, you don't need any special tools or development environments to write and deploy JavaScript applications. Likewise, you don't need special server software to run the applications. Therefore, your options for creating JavaScript programs are virtually limitless.

Configuring your environment: One useful JavaScript development tool is Visual Studio 2012. A simple web server—the ASP.NET Development Server—comes with the installation of Visual Studio 2012, which makes deploying and testing the applications in this book a little easier. However, you can still test the JavaScript code in this book with other IDEs, such as Eclipse. Likewise, you can test the JavaScript code even if you don't use an IDE at all.

Debugging JavaScript: Debugging JavaScript can be an alarming experience, especially in more complex applications. Some tools, such as Venkman (*http://www.mozilla.org/projects/venkman/*), can assist in JavaScript debugging, but the primary tool for debugging JavaScript is the web browser. Major web browsers include some JavaScript debugging capabilities. Among the programs you should consider using is Firebug, a notable add-on to Firefox. Firebug is available at *http://www.getfirebug.com/*.

A bit of housekeeping: Case sensitivity, white space, comments and semicolon.

JavaScript statements: A collection of tokens of various categories including keywords, literals, separators, operators, and identifiers that are put together to create something that makes sense to the JavaScript interpreter. Here are some examples of basic statements in JavaScript:

```
var x = 4;

var y = x * 4;

alert("Hello");
```

Reserved words in JavaScript: Certain words in JavaScript are *reserved*, which means you can't use them as variables, identifiers, or constant names within your program because doing so will cause the code to have unexpected results, such as errors. About 30 reserved words.

A quick look at functions: JavaScript has several *built-in functions* defined by the language itself. Which built-in functions are available depends on the language version you're using.

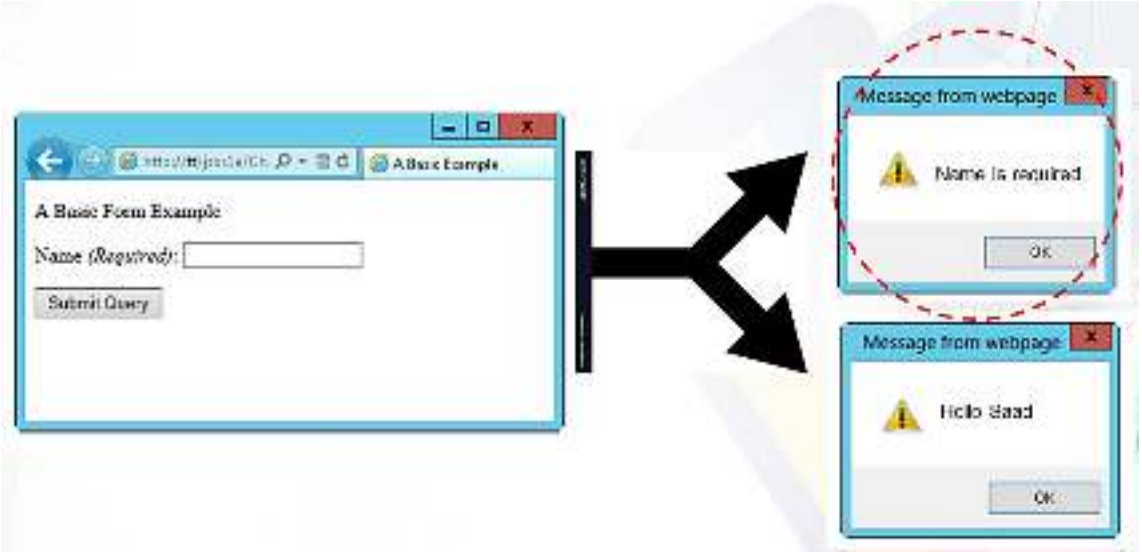
JavaScript's strict mode: ECMA-262 edition 5 introduced a strict variant, commonly referred to as *strict mode*, which adds enhanced error checking and security.

Module-21

JavaScript has been used with web forms for a long time—typically, to quickly verify that a user has filled in form fields correctly before sending that form to the server, a process called client-side validation. Prior to JavaScript, a browser had to send the form and everything in it to the server to make sure that all the required fields were filled in, a process called server-side validation.

Important When using JavaScript, you must perform server-side validation, just in case a user has disabled JavaScript or is purposefully doing something malicious.

Using JavaScript with web forms



The user is required to input a name to proceed to the next page or event. The JavaScript code checks if the text box is empty or not, if it is empty and "Submit Query" button is pressed then an error message box is generated (red dotted circle), else the name entered is displayed in a box and pressing "OK" button proceeds ahead.

Note that the code to check name can be made more powerful, such as checking length of name i.e. number of characters entered or allowing only alphabets and ignoring numbers, special characters etc.

Working with form information

You can access all individual elements of web forms through the DOM. The exact method for accessing each element differs depending on the type of element. For text boxes and select boxes (also known as *drop-downs*), the *value* property holds the text that a visitor types in or selects. This is accessed using the *value* property when using JavaScript or with the *val()* function when using jQuery.

The code that processes the radio buttons is similar to the code you saw that processed the check boxes. The main difference is that radio buttons all share the same name and logical grouping, meaning that they are grouped together and only one can be checked at a time.

Pre-validating form data

JavaScript is frequently used to validate that a given form field is filled in correctly. You saw an example of this behavior earlier in this chapter, when a form asked you to fill in a name. If you didn't put anything in the field, an error alert appeared. JavaScript is good at pre-validating data to make sure that it resembles valid input. However, JavaScript is poor at actually validating the data that makes it to your server.

You should never assume that what gets to the server is valid. I can't count the number of web developers whom I've heard say, "We have a JavaScript validation on the data, so we don't need to check it on the server." This assumption couldn't be further from the truth. People can and do have JavaScript disabled in their browsers; and people also can send POST-formatted and GET-formatted data to the server-side program without having to follow the navigation dictated by the browser interface. No matter how many client-side tricks you employ, they're just that—tricks. Someone will find a way around them.

Module-22

Overview

At first glance, this module's title may seem backward to you: you get data from Web sites; you don't send data to them. But transmitting data to Web servers (and thence to application servers and databases) is important because it is this data that lets users specify just what they want to see—data on or before a certain date, balances greater than a given amount, or information about a specific subject. Those values—the date, the amount, or the subject—must be sent to the database in order for it to do its work. First, you will find a discussion of HTTP — the means by which you request resources on the Web. Not only is HTTP the mechanism by which people get to your Web pages, but it also provides for the transmission of data as part of the request.

Following that, you will find a discussion of forms. HTML forms are the very commonly used feature that lets users enter data in a structured manner and send it to a Web server (and thence possibly on to an application server and database). Forms are not just a formatting device— they, too, are an important way to transmit data to Web sites.

Forms are the easiest way to collect data to be sent to data-bases. They can combine hidden fields with user-entered data that you generate in any of a number of ways, sending everything to be processed when it is complete. You can further extend the power of forms by providing scripts that perform error checking, validation, and automatic submission. The next module deals with some of those issues.

Setting it up

This part of the course may be the most difficult: it deals with issues that you normally need to deal with only when you set up your Web site. Any task that is done infrequently is hard to master. Unfortunately, these are the issues that you need to address at the beginning of your project. Do not be discouraged: once you have set up your database-driven Web site, it is much easier to manage and maintain than a traditional Web site. One way to make life easier for yourself is to hire a consultant to do the setup work for you; you will also find many Internet service providers who will include these services as part of a bundle. Even if you do not do all the work yourself you should cover the module in this part of the course in order to see what is being done for you and to understand how to answer some of the questions that you will be asked. This module is about the details of setting up your database-driven Web site. You may be starting from scratch, or you may be starting from an existing Web site, building on a corporate intranet, or integrating several sites and databases. Database-driven Web sites are easier to maintain than traditional Web sites, but they are usually more complex and require more attention to their design and setup than other sites. This added complexity is outweighed by the

ease of maintenance, but it does mean that you have to pay more attention to your site's design than you might be used to.

Setting up your database-driven Web site involves choosing your tools and vendors as well as designing and implementing the initial site. Once it is up, you need to have procedures in place to manage it so that updates to both databases and Web pages are moved into production appropriately (and with adequate testing). In the public world of the Internet, your site says a lot about you: its design and layout no less than its implementation and maintenance are your face before the world (or at least your coworkers on an intranet). And yet that is not all: having made your site easy to under-stand and use and having promoted it appropriately, you have to think about how to keep people out and how to re-strict your information. This is a different mindset from that involved in the topics covered in this module, and so security is discussed in the next one.

Module-23

Overview

The data landscape has changed. During the past 15 years, the explosion of the World Wide Web, social media, web forms you have to fill in, and greater connectivity to the Internet means that more than ever before a vast array of data is in use.

New and often crucial information is generated hourly, from simple tweets about what people have for dinner to critical medical notes by healthcare providers. As a result, systems designers no longer have the luxury of closeting themselves in a room for a couple of years designing systems to handle new data. Instead, they must quickly create systems that store data and make information readily available for search, consolidation, and analysis. All of this means that a particular kind of systems technology is needed.

The good news is that a huge array of these kinds of systems already exists in the form of NoSQL databases. The not-so-good news is that many people don't understand what NoSQL databases do or why and how to use them. Not to worry, though. That's why I wrote this book. In this chapter, I introduce you to NoSQL and help you understand why you need to consider this technology further now.

Common features

Four core features of NoSQL, shown in the following list, apply to most NoSQL databases. The list compares NoSQL to traditional relational DBMS:

✓Schema agnostic: A database schema is the description of all possible data and data structures in a relational database. With a NoSQL database, a schema isn't required, giving you the freedom to store information without doing up-front schema design.

✓Nonrelational: Relations in a database establish connections between tables of data. For example, a list of transaction details can be connected to a separate list of delivery details. With a NoSQL database, this information is stored as an aggregate — a single record with everything about the transaction, including the delivery address.

✓Commodity hardware: Some databases are designed to operate best (or only) with specialized storage and processing hardware. With a NoSQL database, cheap off-the-shelf servers can be used. Adding more of these cheap servers allows NoSQL databases to scale to handle more data.

✓Highly distributable: Distributed databases can store and process a set of information on more than one device. With a NoSQL database,

NoSQL Database Design and Terminology

New data management challenges have triggered a new database technology — NoSQL. NoSQL thinking and technology mark a shift away from traditional data management technologies. With all the new terms and techniques and the wide variety of options, it's not easy to come up with a succinct description of NoSQL.

NoSQL databases aren't a mere layer on top of existing technologies used to address a slightly different use case. They're different beasts entirely. Each type of NoSQL database is designed to manage different types of data. Understanding the data you want to manage will help you apply the right NoSQL solution.

The popularity of NoSQL databases lies largely in the speed they provide for developers. NoSQL solutions are quicker to build, update, and deploy than their relational forerunners. Their design is tuned to ensure fast response times to particular queries and how data is added to them.

This speed comes with tradeoffs in other areas, including ensuring data consistency — that is, data that has just been added or updated may not be immediately available for all users. Understanding where consistency should and shouldn't be applied is important when deciding to deploy a NoSQL solution.

MongoDB

MongoDB is the poster child for the NoSQL database movement. If asked to name a NoSQL database, most people will say MongoDB, and many people start with MongoDB when looking at NoSQL technology. This popularity is both a blessing and a curse. It's obviously good for MongoDB, Inc. (formerly 10gen). On the flip side, though, people try to use MongoDB for purposes it was not designed for, or try to apply relational database approaches to this fundamentally different database model.

MongoDB is a good NoSQL document database with a range of features that, in the open-source NoSQL world, are hard to beat. Starting your NoSQL career with MongoDB is a good approach to take.

Effective indexing: Retrieving a document using a document ID, or (primary) key, is supported by every NoSQL document database. In many situations, though, you may want a list of all comments on a web page or all recipes for puddings (my personal favorite!). This requires retrieving a list of documents based not on their key but on other information within the document — for example, a page_id JSON property. These indexed fields are commonly

referred to as *secondary indexes.* Adding an index to these fields allows you to use them in queries against the MongoDB database.

In some situations, you may want to search by several of these fields at a time, such as for all pudding recipes that contain chocolate but are gluten free. MongoDB solves this issue by allowing you to create a *compound index,* which is basically an index for all three fields (recipe type, ingredients, is gluten free), perhaps ordered according to the name of the recipe.

Module-24

10 Advantages of NoSQL

1. Less Need for ETL
2. Support for Unstructured Text
3. Ability to Handle Change over Time
4. No Reliance on SQL Magic
5. Ability to Scale Horizontally on Commodity Hardware
6. Breadth of Functionality
7. Support for Multiple Data Structures
8. Vendor Choice
9. No Legacy Code
10. Executing Code Next to the Data

Less Need for ETL

NoSQL databases support storing data "as is." Key-value stores give you the ability to store simple data structures, whereas document NoSQL databases provide you with the ability to handle a range of flat or nested structures.

Most of the data flying between systems does so as a message. Typically, the data takes one of these formats:

✓A binary object to be passed through a set of layers

✓An XML document

✓A JSON document

Being able to handle these formats natively in a range of NoSQL databases lessens the amount of code you have to convert from the source data format

Breadth of Functionality

Most relational databases support the same features but in a slightly different way, so they are all similar. NoSQL databases, in contrast, come in four core types: key-value, columnar, document, and

triple stores. Within these types, you can find a database to suit your particular (and peculiar!) needs. With so much choice, you're bound to find a NoSQL database that will solve your application woes.

10 NoSQL Misconceptions

1. NoSQL Is a Single Type of Database
2. NoSQL Aren't ACID-Compliant
3. NoSQL Databases Lose Data
4. NoSQL Databases Aren't Ready for Mission-Critical Enterprise Applications
5. NoSQL Databases Aren't Secure
6. All NoSQL Databases Open-Source
7. NoSQL Only for Web 2.0 Applications
8. NoSQL Is Just Hype
9. NoSQL Developers Don't Know How to Use an RDBMS
10. Updated RDBMS Technology Will Remove the Need for NoSQL

NoSQL Databases Aren't ACID-Compliant

ACID compliance is the gold standard of data safety. By ensuring that operations are atomic, views of data are consistent, operations don't interfere with each other, and data is durably saved to disk, you protect your data. People often think NoSQL databases do not provide ACID compliance.

Many NoSQL databases provide full ACID support across clusters. MarkLogic Server, OrientDB, Aerospike, and Hypertable are all fully ACID-compliant, providing either *fully serializable* or *read-commit* ACID compliance.

Many other NoSQL databases can provide ACID-like consistency by using sensible settings in client code. This typically involves a *Quorum* or *All* setting for both read and write operations. These databases include Riak, MongoDB, and Microsoft DocumentDB.

10 reasons why developers love NoSQL

1. No Need to Write SQL
2. Don't Have to Spend Months Designing Schema
3. Less Data Transform Code (ETL)

4. Easier to Maintain Code
5. Execute Code Close to the Data for Best Performance
6. Lots of Open-Source Options
7. Easy to Scale
8. Eventual Consistency Data Model
9. Esoteric Language Support
10. JavaScript End-to-End

No Need to Write SQL

Writing Structured Query Language (SQL) — and doing it well — is the bane of many enterprise developers' existence. This pain is because writing very complex queries with multiple joins across related tables isn't easy to do.

Moreover, in light of regular database changes over time, maintaining complex query code is a job in and of itself. Enterprise developers have invented a number of ways to avoid writing SQL. One of the most popular ways is through the use of the Object-Relational Mapping (ORM) library, Hibernate. Hibernate takes a configuration file and one or more objects and abstracts away the nasty SQL so that developers don't have to use it. This comes at a cost in terms of performance, of course, and doesn't solve all query use cases. Sometimes you have to fall back to SQL.

NoSQL databases provide their own query languages, which are tuned to the way the data is managed by the database and to the operations that developers most often perform. This approach provides a simpler query mechanism than nested SQL statements do.

Some NoSQL databases also provide an SQL interface to query NoSQL databases, in case developers can't break the SQL habit!

Module-25

Amazon, started out as an online bookstore, has become *the* leading cloud computing provider. This module solves that mystery by discussing the circumstances that led Amazon into the cloud computing services arena and why Amazon Web Services, far from being an oddly different offering from a retailer, is a logical outgrowth of Amazon's business.

Will also compare Amazon's cloud offering to other competitors in the market and explain how Amazon's approach differs. As part of this comparison, will present some statistics on the size and growth of Amazon's offering, while describing why it's difficult to get a handle on the exact size.

Characteristics of Cloud Computing

- No upfront capital expense
- Pay for what you consume
- Low cost
- Self-service infrastructure
- True elastic capacity
- Improve business agility

Cloud computing

Generally cloud computing refers to the delivery of computing services from a remote location over a network. The following information is drawn directly from NIST. Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly arranged and released with minimal management effort or service provider interaction.

Cloud computing is commonly characterized as providing three types of functionality, referred to *IaaS, PaaS,* and *SaaS,* where *aaS* is shorthand for "as a service" and *service* implies that the functionality isn't local to the user but rather originates elsewhere (a location in a remote place accessed via a network). The letters *I, P,* and *S* in the acronyms refer to different types of functionality, as the following list makes clear: 1) Infrastructure as a Service (Iaas): 2) Platform as a Service (PaaS): 3) Software as a Service (SaaS):

This cloud model is composed of five essential characteristics:

1. On-demand self-service:
2. Broad network access:
3. Resource pooling:
4. Rapid elasticity:
5. Measured service:

Measuring the scale of AWS

Table 1-1	Total AWS Servers	
AWS Region	*Number of Server Racks*	*Number of Servers*
US East	5,030	321,920
US West (Oregon)	41	2,624
US West (California)	630	40,320
EU West (Ireland)	814	52,096
AP Northeast (Japan)	314	20,096
AP Southeast (Singapore)	246	15,744
SA East (Brazil)	25	1,600
Total	**7,100**	**454,400**

Amazon is the pioneer of cloud computing and, because you'd have to have been living in a well not to have heard about "the cloud," being the pioneer in this area is a big deal. The obvious question is this: If AWS is the leader in the market and if Cloud computing is the hottest thing since sliced email, how big are we talking about?

That's an interesting question, because Amazon reveals little about the extent of its business. Rather than break out AWS revenues, the company lumps them into an **Other** category in its financial reports.

A very clever consultant named Huan Liu examined AWS IP addresses and projected the total number of server racks held by AWS, based on an estimate of how many servers reside in a rack. Table breaks down the numbers by region.

That's a lot of servers. (To see the original document outlining Liu's estimates, along with his methodology, go to http://huanliu.wordpress.

com/2012/03/13/amazon-data-center-size).

If you consider that each server can support a number of virtual machines (the number would vary, of course, according to the size of the virtual machines), AWS could support several million running virtual machines.

Introducing AWS APIs

API: application programming interface, represents a way for one program to interact with another via a defined *interface* — in other words, a mechanism by which any other program that communicates with the program can be assured that it will fulfill its role. The idea is that if a calling program provides the right information within the correct syntax, the called program with the API will respond in the requested manner.

The AWS environment acts as an integrated collection of hardware and software services designed to enable the easy, quick, and inexpensive use of computing resources. With respect to AWS, nothing gets done without using the AWS API. The AWS API is the sole way that external users interact with AWS resources, and there's literally no way to use AWS resources without the API being involved. In fact, if you access AWS through the AWS Management Console or the command line tools, you are actually using tools that make calls to the AWS API.

Amazon API Gateway: Amazon API Gateway is an AWS service offering that allows a developer to connect non-AWS applications to AWS back-end resources, such as servers or code. Amazon API Gateway allows an AWS customer to increase the overall utility of Amazon's other cloud services.

An AWS user creates, manages and maintains APIs within Amazon API Gateway, which accepts and processes concurrent API calls. The service manages traffic, authorizes end users and monitors performance.

Module-26

Overview

Ready to start working with Amazon Web Services (AWS) and cloud computing. But how? Well, it turns out that the services part of Amazon Web Services refers to the fact that all interaction with Amazon's cloud computing service is performed with the help of numerous Application Programming Interface (API) calls over the Internet. These calls are accomplished by either SOAP or REST interfaces carrying data in XML or JSON formats.

No need to be afraid. Amazon offers its own, web-based interface to enable users to work with AWS. This interface, the AWS Management Console, hides all the complex details of interacting with the AWS API. You interact with the console, and Amazon's program deals with all the complexity under the hood. Many people never interact with AWS except through the Console — it's that powerful. This module introduces you to the Console, goes through setting up your very own AWS account, and even provides your first taste of cloud computing.

Storage is an increasingly important topic to IT because of the recent staggering increase in the amount of data that businesses use in their day-to-day operations

Storage is the first AWS offering that Amazon offered. Storage therefore holds a significant place in the AWS ecosystem, including some extremely innovative uses of its storage services by AWS customers over the years.

AWS management console

Setting Up Your Amazon Web Services Account

The first thing to do is to create your very own AWS account. In this multistep process, you sign up for the service, provide your billing information, and then confirm your agreement with AWS to create your account. Ready?

1. Your account is now set up as a general AWS account. You can use AWS resources anywhere in the AWS system — the US East or either of the two US West regions, Asia Pacific (Tokyo, Singapore, or Australia), South America (Brazil), and Europe (Ireland). Put another way, your account is scoped over the entirety of AWS, but resources are remotely located within a specific region.

2. You have given AWS a credit card number to pay for the resources you use. In effect, you have an open tab with AWS, so be careful about how much computing resource you consume. For the purposes of this course, you don't have to worry much about costs — your initial sign-up provides a free level of service for a year that should be sufficient for you to perform the steps in this course as well as experiment on your own without any financial pain.

4. Load data in S3: Create an Amazon **S3 bucket** and then **upload** the **data** files to the **bucket**. Launch an Amazon Redshift cluster and create database tables. Use COPY commands to **load** the tables from the **data** files on Amazon **S3**. Troubleshoot **load** errors and modify your COPY commands to correct the errors.

AWS landing page

Setting Up Your Amazon Web Services Account

The first thing to do is to create your very own AWS account. In this multistep process, you sign up for the service, provide your billing information, and then confirm your agreement with AWS to create your account. Ready?

1. Your account is now set up as a general AWS account. You can use AWS resources anywhere in the AWS system — the US East or either of the two US West regions, Asia Pacific (Tokyo, Singapore, or Australia), South America (Brazil), and Europe (Ireland). Put another way, your account is scoped over the entirety of AWS, but resources are located within a specific region.

2. You have given AWS a credit card number to pay for the resources you use. In effect, you have an open tab with AWS, so be careful about how much computing resource you consume. For the purposes of this course, you don't have to worry much about costs — your initial sign-up provides a free level of service for a year that should be sufficient for you to perform the steps in this course as well as experiment on your own without any financial pain.

AWS global physical infrastructure

Setting Up Your Amazon Web Services Account

The first thing to do is to create your very own AWS account. In this multistep process, you sign up for the service, provide your billing information, and then confirm your agreement with AWS to create your account. Ready?

1. Your account is now set up as a general AWS account. You can use AWS resources anywhere in the AWS system — the US East or either of the two US West regions, Asia Pacific (Tokyo, Singapore, or Australia), South America (Brazil), and Europe (Ireland). Put another way, your account is scoped over the entirety of AWS, but resources are located within a specific region.

2. You have given AWS a credit card number to pay for the resources you use. In effect, you have an open tab with AWS, so be careful about how much computing resource you consume. For the purposes of this course, you don't have to worry much about costs — your initial sign-up provides a free level of service for a year that should be sufficient for you to perform the steps in this course as well as experiment on your own without any financial pain.

Setting up AWS storage

The term *Amazon's storage service* (which may be the largest in the industry) is a misnomer: The company offers five different storage services within AWS. The scale of the overall storage service that covers all five specific services is enormous. Module-25 presents information about the number of objects stored in the AWS Simple Storage Service (known as *S3)*; in just over six years, S3 grew to 2 trillion objects. To put the growth into perspective, AWS spent six years reaching 1 trillion objects, and in less than ten months growing from 1 to 2 trillion objects. The AWS storage services are:

Simple Storage Service (S3): Provides highly scalable object storage in the form of unstructured data

Elastic Block Storage (EBS): Provides highly available and reliable data volumes that can be attached to a virtual machine (VM), detached, and then reattached to another VM

Glacier: A data archiving solution; provides low-cost, highly robust archival data storage and retrieval

DynamoDB: Key-value storage; provides highly scalable, high performance storage based on tables indexed by data values referred to as *keys*

Elastic File System (EFS): Provides scalable, elastic, concurrent with some restrictions, and encrypted file storage for use with both AWS cloud services and on-premises resources.

Module-27

Overview

There are many users of AWS who struggle to describe why they adopted it. Still others are interested in AWS, but aren't sure about exactly what it is. And others who know what it is, and why they adopted it, but get tongue tied when asked to justify their decision by higher management. To solve all those problems in one fell swoop, here is a list of the ten best reasons to use AWS.

Those in the know will tell you that you have to use the right tool for the job. For the new generation of webscale applications like Pinterest, AWS is the right tool. Overlooked in that truism is the undeniable fact that using a tool effectively requires having the right skills. With respect to AWS, the rigaht skills involve aligning your application design with AWS's operational characteristics. It's critical to get the application design right — so here are ten design principles to help you get your alignment straight.

10 Reasons for using AWS

1. Provides IT Agility
2. Provides Business Agility
3. Offers a Rich Services Ecosystem
4. Simplifies IT Operations
5. Spans the Globe
6. Is the Leading Cloud-Computing Service Provider
7. Enables Innovation
8. Is Cost Effective
9. Aligns Your Organization with the Future of Technology
10. AWS Is Good for Your Career

AWS Provides IT Agility: IT has a reputation as the "Department of No." Though it's true that some IT organizations seem to revel in a *Dilbert*-like obstinacy, where innumerable and inexplicable roadblocks are placed in the way of anyone seeking access to the wizardry of "infrastructure," others are frustrated by the sheer complexity of coordinating many different resources, each with its own interface and configuration rules, all of which must be successfully stitched together to provide access to computing resources. Most of these multidepartment, manual, time-consuming efforts are the result of the years-long build-up of established processes executed in serial fashion, resulting in IT provisioning cycles that commonly require weeks to months to deliver computing resources. The result of all this: It's slower than molasses and widely despised.

AWS Is Cost Effective: Commenters who analyze Silicon Valley trends note that the cost of starting an Internet business is now less than 10 percent of what it cost a mere decade ago. Much of that cost reduction is due to AWS: its on-demand low pricing and easy termination with no penalties make it possible to use and pay for exactly as much computing capacity as you need, when you need it.

AWS Is Good for Your Career: Great careers are built on being the right person in the right place at the right time. Being the right person is all about you — your capacity for hard work, productive work relationships, and intelligence, for example. These characteristics will help you be successful no matter which field or role you work in.

10 Design principles for cloud

1. Everything Fails All the Time
2. Redundancy Protects Against Failure
3. Geographic Distribution Protection
4. Monitoring Prevents Problems
5. Utilization Review Prevents Waste
6. Application Management Automation
7. Security Design Prevents Breaches & Data Loss
8. Encryption Ensures Privacy
9. Tier-Based Design and Efficiency
10. Good Application Architecture & Debt

Everything Fails All the Time: The truism "Everything fails all the time" is adapted from Werner Vogels, the chief technology officer of Amazon. IT departments have traditionally attempted to render both infrastructure and applications impervious to failure: A hardware resource or an application component that "fell down on the job" increased the urgency of the search for

perfection in order to banish failure. Unfortunately, that search was never successful — the failure of resources and applications has been part of the IT world from the beginning.

Monitoring Prevents Problems: Redundancy is good, and it's important to avoid a situation in which your application, once neat and tidy with redundant resources, becomes non-redundant through the failure of a redundant resource. The question then is, how to *know* when the formerly neat-and-tidy redundant application is no longer so because of failure?

Tier-Based Design Increases Efficiency: A tiered design makes it possible to improve security by partitioning security groups. It may not be as obvious that using a tier-based application design, particularly one that uses redundant, *scalable* tiers (tiers that can grow and shrink by the addition or subtraction of instances to the tier) can also improve the efficiency of your application.

Module-28

Search engine optimization (SEO) very much revolves around Google today. However, the practice we now know as SEO actually pre-dates the world's most popular search engine co-founded by Larry Page and Sergey Brin.

Although it could be argued that SEO and all things search engine marketing began with the launch of the first website published in 1991, or perhaps when the first web search engine launched, the story of SEO "officially" begins a bit later, around 1997.

Before Search Engine Optimization became the official name, other terms were used as well. For example:

- Search engine placement
- Search engine positioning
- Search engine ranking
- Search engine registration
- Search engine submission
- Website promotion

The Google revolution

In 2000, Yahoo pulled off the worst strategic move in the history of search and partnered with Google and let Google power their organic results instead of Inktomi. Beforehand Google was a little-known search engine. Hardly known! The end result: every Yahoo search result said "Powered by Google" and they ended up introducing their largest competitor to the world and Google became a household name.

What is SEO?: The process of optimizing a website – as well as all the content on that website – so it will appear in prominent positions in the organic results of search engines. SEO requires an understanding of how search engines work, what people search for, and why and how people search. Successful SEO makes a site appealing to users and search engines. It is a combination of technical and marketing.

Glossary

Bounce Rate: The percentage of website visitors who leave without visiting another page on that website. Bounce rates range widely depending on industry and niche. Although bounce

rate can indicate potential content or website issues, it is not a direct ranking factor, according to Google.

Click Bait: Content that is designed to entice people to click, typically by overpromising or being intentionally misleading in headlines, so publishers can earn advertising revenue.

Googlebot: The web crawling system Google uses to find and add new websites and webpages to its index.

The Science behind Intent

In 2006, a study conducted by the University of Hong Kong found that at a primary level, search intent can be segmented into two search goals. That a user is specifically looking to the find information relating to the keyword(s) they have used, or that they are looking for more general information about a topic.

Do-Know-Go: A concept that search queries can be segmented into three categories.

DO (Transactional): When a user performs a "do" query, they are looking to achieve a specific action, such as purchasing a specific product or booking a service. These are important to e-commerce websites for example, where

a user may be looking for a specific brand or item.

KNOW (Information): A "know" query is an informational query, where the user is wanting to learn about a particular subject. Know queries are closely linked to micro-moments.

GO (Navigation): Queries are typically brand or known entity queries, where a user is looking to go to a specific website or location. If a user is specifically searching for Adidas, serving them Puma as a result wouldn't meet their needs.

Periodic table of SEO

Since it first debuted in 2011, Search Engine Land's Periodic Table of SEO has been downloaded nearly 100,000 times by professionals from 74 different countries, and has been referenced and linked to thousands of times by marketing websites, blogs and pieces of industry content marketing.

While much of the foundation of search engine optimization has either stayed the same or has become further entrenched, much has also changed as the web has become more mobile, instantly accessible and aligned to new Internet-connected devices. 2019 marks the 150th anniversary of Dmitri Mendeleev's original Periodic Table of Chemical Elements.

Previous versions of this table focused on elements dubbed “success factors,” but this year’s table contains elements that are either foundational, increasingly toxic to an SEO strategy or represent verticals that are still emerging. With this relaunched infographic, we are offering both long-time SEOs and those new to the industry an overview of what’s important when you’re looking to achieve success in SEO. It isn’t all about rankings, but it is about achieving positive results from greater visibility in search engines.

What is content?

“High quality, useful information that conveys a story presented in a contextually relevant manner with the goal of soliciting an emotion or engagement. Delivered live or asynchronously, content can be expressed using a variety of formats including text, images, video, audio, and/or presentations.”

Here are some other key benefits of content, in relation to SEO:

Quality Content Generates High CTR - Google considers your CTR as an important factor to rank your website - the more you get users to click on your links, the greater are your chances of getting better rankings on search engines.

Quality Content Helps You Generate Backlinks - One of the best SEO strategies is to gain high-quality backlinks from high-authority websites. For Google, high-quality backlinks indicate credibility and trust. The more quality backlinks you have, the higher you are likely to rank on Google.

Content Allows You to Incorporate Keywords - Quality content is the only way to make sure that you can strategically use your keywords. This will help you compete with other brands from your industry.

Quality Content Provides a Great User Experience - SEO involves various strategies such as generating backlinks, writing quality blog posts and using good keywords. It also involves creating a website that has a good structure that users can navigate easily, optimizing your robots.txt files, and writing good meta tags.

Module-29

Web frameworks have transformed the world of programming and become vitally important in every development process. Even the smallest unit of an application is comprised of coding, and a web framework simply automates it. You might try browsing different sites, books and articles about it, but find only general and ambiguous information – nothing but endless definitions and difficult terms that make your head spin. Well, it's time to handle this issue and get a clear understanding of web frameworks.

What Are Web Frameworks?

A web framework is a software tool that provides a way to build and run web applications. As a result, you don't need to write code on your own and waste time looking for possible miscalculations and bugs.

In the early days of web development, all applications were hand-coded, and only the developer of a certain app could change or deploy it. Web frameworks introduced a simple way out of this trap. Since 1995, all the hassle connected with changing an application's structure has been put in order because of the appearance of a general performance. And that's when web-specific languages appeared. Their variety is now working well for both static and dynamic web pages. You can choose one framework that covers all your needs or merge several, depending on your task.

Types of Web Frameworks

There are two main functions of frameworks: to work on the server side (backend), or on the client-side (frontend), corresponding to their type. This division is not complicated and looks like this:

Server-side frameworks. The rules and architecture of these frameworks allows you to create simple pages, landings and forms of different types. However, in order to

build a web application with a well-developed interface, you should have a wider functionality. These frameworks can also form the output data and improve security in case of web attacks. All of these can definitely simplify the development process. Server-side frameworks work mostly on particular but important details without which an application can't work properly.

Here are top backend frameworks and the languages they work in:

Client-side frameworks. Unlike the server side, client-side frameworks have nothing to do with business logic. Their work takes place inside the browser. Thus, one can improve and implement new user interfaces. Numerous animated features can be created with frontend

frameworks as well as SPA (single-page applications). Each of the client-side frameworks differs in function and use. For comparison purposes, here they are:

Cross-functional frameworks. Meteor is known as a full-stack web framework. It means that it covers almost all the needs of both the client and server sides, which makes Meteor extremely popular and widely used. You don't have to waste your time merging two frameworks using ; you can simply choose this one and accelerate your development process.

Top 5 Client-side web frameworks

Angular can be considered the best framework for web applications, first released in 2009, and completely rewritten in 2016. A comparative significant advantage is its flexibility and rich set of functions. Angular is suitable for bulky apps due to Typescript support. It requires less development effort and offers high performance due to two-way binding and dependency injection.

Ember is one of the most entrusted and mature JavaScript web dev frameworks. Released in 2011, it has been rapidly growing and gaining more and more influence. It's core features are strict organization, advanced version management, and support of both modern standards and older technologies at the same time. Ember allows making properties out of functions which is useful when it comes to working with bindings.

Flutter created to give developers a fast development framework, and to users, a great engaging and fast experience. Flutter is a code-compatible implementation that is rendered using standards-based web technologies: HTML, CSS and JavaScript.

React is not exactly a web app framework, but a JavaScript library. React has gained its fame due to the revolutionary component-based architecture that other frameworks began to use much later.

Vue.js is one of the newer frameworks for web development which is growing in popularity very quickly. Its greatest advantage is, if you already have a product, you use Vue.js on its part and everything will function just fine. No lags, no troubles.

Comparison of web frameworks

Angular can be considered the best framework for web applications, and it definitely

takes the lead among the products of the Google company for developers. AngularJS,

it's predecessor, has bee first released in 2009, and completely rewritten in 2016. A comparotove significant advantage of this platform is its flexibility and rich set of functions. Angular is suitable for bulky apps due to Typescript support. It requires less development effort

and offers high performance thanks to features such as two-way binding and dependency injection respectively.

Ember is one of the most entrusted and mature JavaScript web dev frameworks. Released in 2011, it has been rapidly growing and gaining more and more influence in

the world of professional web development.

Ember's core features are its strict organization, advanced version management system, and support of both the most modern standards and older technologies at the same time. Ember allows you to make properties out of functions which is pretty

useful when it comes to working with bindings.

Flutter has been created to give developers a fast **development framework**, and to users, a great engaging and fast experience. **Flutter** for **web** is a code-compatible implementation of **Flutter** that is rendered using standards-based **web** technologies: HTML, CSS and JavaScript.

Angular can be considered the best framework for web applications, and it definitely

takes the lead among the products of the Google company for developers. AngularJS,

it's predecessor, has bee first released in 2009, and completely rewritten in 2016. A comparotove significant advantage of this platform is its flexibility and rich set of functions. Angular is suitable for bulky apps due to Typescript support. It requires less development effort and offers high performance thanks to features such as two-way binding and dependency injection respectively.

Ember is one of the most entrusted and mature JavaScript web dev frameworks. Released in 2011, it has been rapidly growing and gaining more and more influence in

the world of professional web development.

Ember's core features are its strict organization, advanced version management system, and support of both the most modern standards and older technologies at the same time. Ember allows you to make properties out of functions which is pretty

useful when it comes to working with bindings.

Flutter has been created to give developers a fast **development framework**, and to users, a great engaging and fast experience. **Flutter** for **web** is a code-compatible implementation of **Flutter** that is rendered using standards-based **web** technologies: HTML, CSS and JavaScript.

Module-30

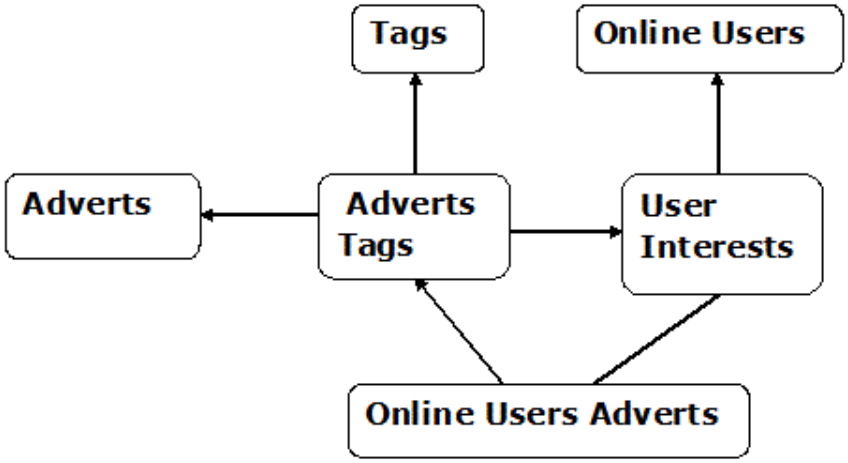
Online advertising

Early Years: First online advertising was initiated when Hot-Wired signed up fourteen advertisers for its online debut (October 27, 1994). After this initiation we saw the emergence and public acceptance of the Web as an interactive medium in the following years.

Growth: The year 1994 saw the first online advertisement that was quickly followed by a period of research on advertiser and publisher ad formats and technology. In late 1990s, billions of money were invested in online advertising.

Current Scenario: Banner ads today, as they were more than a decade ago, are no more effective online advertising medium. Online advertising has been constantly rising since 2004. With the number of hours an internet user spends browsing websites, advertisers have realized the significance and advantage of manipulating user tendency to scour the web.

From SEO marketing, blogs and social media to stylish ads, interactive tools and branding technologies, advertisers are now using a wide array of platforms to increase business visibility



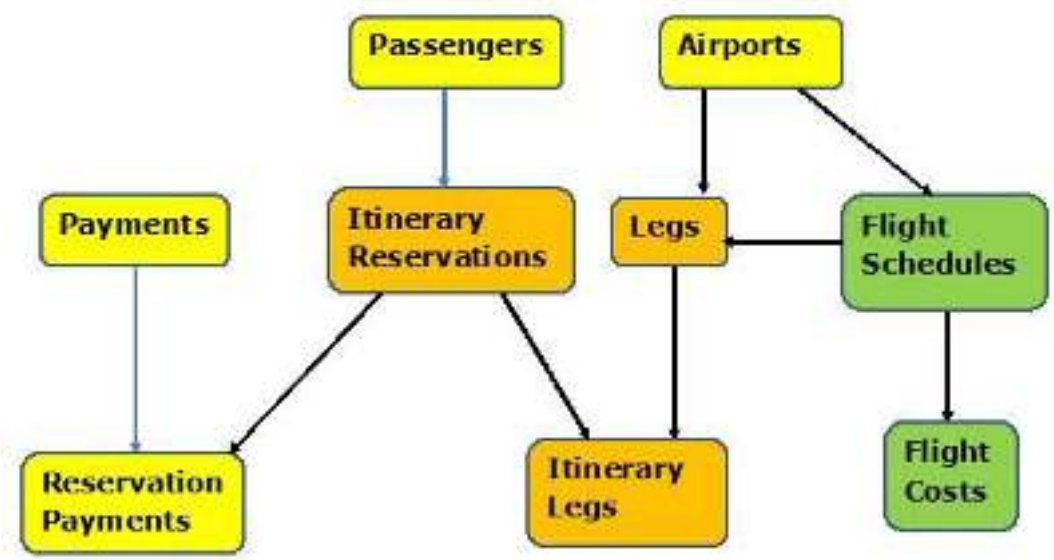
http://www.databaseanswers.org/data_models/advertising_online/index.htm

Airline reservation

Airline reservation systems incorporate airline schedules, fare tariffs, passenger reservations and ticket records. An airline's direct distribution works within their own reservation system, as well as pushing out information to the GDS. The second type of direct distribution channel are consumers who use the internet or mobile applications to make their own reservations. Travel agencies and other indirect distribution channels access the same GDS as those accessed by the airline reservation systems.

Reservation systems may host "ticket-less" airlines and "hybrid" airlines that use e-ticketing in addition to ticket-less to accommodate code-shares and interlines.

In addition to these "standardized" GDS, some airlines have proprietary versions which they use to run their flight operations. A few examples are Delta's OSS and Deltamatic systems and EDS SHARES.



http://www.databaseanswers.org/data_models/airline_reservations/index.htm

TOPIC-2: Web application vs. Web site

Module-31

Choosing between a web application and a website, you may wonder what the exact difference is. At one point, it may seem that there's no difference at all. The definitions are controversial, and sometimes they overlap. Both websites and web applications run in browsers, both require access to the internet, both have a front end and a back end written in the same programming languages. What is more, they both possess such attributes as interactivity, integration, and authentication.

Still, we believe that the 'web application vs. website' difference not only exists but also is vital to understand clearly when you are looking for an online solution for your business. Web application development differs significantly from the development of a website. So let's dot the I's and find out what distinguishes these kinds of web software and which option is better for you.

Point 1. Interactivity

The first point to start 'web application vs. website' differentiation with is interactivity. A website provides visual and text content that the user can see and read, but not affect in any way. In the case of a web application, the user can not only read the page content but also manipulate the data on this page. The interaction takes the form of a dialog: the user clicks a button or submits a form and gets a response from the page. This response may take a form of a document download, online chat, electronic payment and more.

An illustrative example of a web application interactivity is an online banking application that performs transactions based on a customer's input. Similar functionality can be found in an online store that allows visitors to search through the catalog and buy items instantly. Social networks are another impressive example. They connect users via chats and blog platforms, generate feed content based on users' preferences and allow for almost unlimited content sharing, not to speak of their built-in mini-applications for user entertainment.

The problem is that today one can rarely encounter a website without a hint of interactivity. Modern websites usually contain small web application elements. For example, a restaurant's website may contain a Google Maps widget showing a route to this restaurant. However, in the case of websites, the balance between the informational content and interactivity is shifted towards the former. A typical website contains far fewer interactive elements than informational content, and the user usually spends most of the time on a website reading,

viewing, or listening. The situation is the opposite with web applications, as their core functionality is based on interaction.

Point 2. Integration

Integration means bringing together different components to build a more comprehensive system. Both websites and web applications can be integrated with other software (CRM, ERP, etc.). Still, integration is more typical for web applications, because their complex functionality often requires interaction with extra systems.

Take integration of a business web application (say, an e-shop) with a CRM (Customer Relationship Management) system. A CRM stores all customer data in one place, providing easy access to them for the employees. The integration will allow automatic collection of web application user data and storing it in the CRM. This way, your team will get access to a full set of data about customers, their inquiries, communication, and feedback. This enables exploring customer behavior and buying habits, as well as settling their claims faster. Moreover, any change in customer data will be reflected in the CRM instantly. Always staying up to date with your customer preferences, you will reduce churn rates and increase sales.

A website also can be integrated with a CRM. This allows providing users with more personalized content. However, for a website, it's rather a rarely implemented feature than a part of the core functionality.

Point 3. Authentication

Authentication is the procedure that involves entering a user's login and password to get access to the system. It is a must for the web software that requires any personal information. User accounts must be secured to prevent unauthorized access and leakage of sensitive data.

Web applications mostly require authentication, as they offer a much broader scope of options than websites. Consider an example of social networks. When you register, you create an account and get a unique identification number. The system warns you if your login and password are weak. If you leave them unchanged, hackers may reach your account and steal your information, as well as irritate other users with junk emails under your name.

Authentication is not obligatory for informational websites. The user may be offered to register to get access to additional options unavailable to unregistered website visitors. For example, you may look through news and featured articles on a news website without bothering to register. However, if you want to leave a comment you will have to log in. This way, users confirm their identity allowing the system to block spammers. As you can see, both websites

and web applications may require authentication. However, for web applications, it is obligatory due to security reasons.

Web applications with special names

Web portals: A portal is an environment for integrating diverse applications and various content. It enables to configure content and offers personalized user experience, which means that the user gets only the content tailored to their needs. Consider a bank web portal It may provide links to account information, bill payments, and deposits. Each of them is a web application, but they are accessed from the central hub – the web portal.

Online stores: An online store (or an e-shop) is an application used for selling goods or services over the internet. The process goes the following way: a customer chooses a product and clicks a button to order it; then, the system processes the order. One of the features of an online store is the users' ability to make online payments. To pay online, the user should indicate their credit card number, and, in some cases, passport details, email, or telephone number. To make the transaction secure, the user has to be authorized.

Who you need: website developers or web application developers: Deciding on which specialists to hire, you should consider your business needs first. If you need a website, not a web application, a small web studio may be the best choice. Such a company can provide you with a unique and good-looking website, where you can display the information about your company. Still, later on, you may decide to add web applications to your website, and this may cause the need for more qualified assistance.

If you need a web application, not a website, turn to web application developers. These specialists usually have extensive development skills and are able to implement a broader range of functions. So, if you need intense interactivity, integration with other corporative systems and top-notch security level, opt for the companies offering web application development.

Module-32

What is Web Analytics?

Web analytics is the measurement, collection, analysis and reporting of web data for purposes of understanding and optimizing web usage. However, Web analytics is not just a process for measuring web traffic but can be used as a tool for business and market research, and to assess and improve the effectiveness of a website.

What is Web Analytics used for?

Know your visitors. ...
Track where traffic is coming from. ...
Use an IP lookup database. ...
Use a personalized URL. ...
Gauge the success of other marketing activities and see your match rate rise. ...
Know exactly what your customers are looking for. ...
See where you are losing customers.

How Web Analytics Work?

Most **analytics tools** 'tag' their **web** pages by inserting a snippet of JavaScript in the **web** page's code. Using this tag, the **analytics tool** counts each time the page gets a visitor or a click on a link, places some text (cookie) on visitors machine to identification , knows where the visitor came from etc.

Google Analytics: It is one of the most popular digital analytics software. It is Google's free web analytics service that allows you to analyze in-depth detail about the visitors on your website. It provides valuable insights that can help you to shape the success strategy of your business.

Google Analytics Certification Individual Qualification certification is well worth your time. You'll gain in-depth insight into Google Analytics, which will help you better understand your website data. Plus, your certification enables you to become a qualified web analyst for your company

CrazyEgg: Takes the guesswork out of trying to figure out whether or not certain elements of your website are being used. How? It shows you visually with various types of overlay maps, so that you can see exactly what's being looked at and clicked on when people visit your site.

Clicky: Clicky is a web analytics tool similar to Google Analytics. Show in a glance, traffic data for multiple websites at the same time on the dashboard. This is helpful for someone who runs online businesses. It's less intimidating, cleaner, and a lot more user friendly.

GTmetrix: A free tool that you can use to check the speed and performance of your website. Just insert your URL -you'll get a quick analysis and report of how long it took to load, along with a grade and what parts of your site are slowing you down.

Module-33

Organization is a fundament skill in the oftentimes chaotic world of business. There are organizational tools for business and just about every other aspect of life. Task management software helps us get through the week and to keep on top of our seemingly never-ending to-do's. Most importantly, it assists in capturing concepts and strategies for new exciting projects.

What Are Organizational Tools?

An organizational tool as an app or software created to optimize your daily task performance. There are several different types of tools you could encounter, including project management software, note-taking programs, journals, and day planners.

Reasons to Use Organizational Tools

Organization management tools help in sorting out one's daily responsibilities and being more efficient with time: calendars, to-do lists, mind maps, document storage clouds.

One can never be too organized, and there is always room for improvement.

If you're someone who wants to make your business better, take time to look at what you can do to improve time management and make your business grow faster, all the while keeping your expenses under control. We've put together this short list of must-have productivity apps for the new year.

What is meant by productivity?

A measure of the efficiency of a person, machine, factory, system, etc., in converting inputs into useful outputs. **Productivity** is computed by dividing average output per period by the total costs incurred or resources (capital, energy, material, personnel) consumed in that period.

What is the effect of technology on productivity?

Technology can help to organize every task that you need to complete and improve **productivity** along the way. **Technology** will minimize your employees taking unnecessary steps or getting overwhelmed with all the tasks they have to complete

How does technology increases productivity in workplace?

Here are some ways today's **technologies** help **increase** the **productivity** of your business: Connect dispersed workforces. ... **Productive** employees need instant access to business information, data, and their team; mobile-friendly **technologies** are a sure-fire way to provide that. Enable collaboration tools.

1. Track and limit how much time you're spending on tasks: Some research suggests only around 17 percent of people are able to accurately estimate the passage of time. A tool like Rescue Time can help by letting you know exactly how much time you spend on daily tasks, including social media, email, word processing, and apps.

2. Take regular breaks: It sounds counterintuitive, but taking scheduled breaks can actually help improve concentration. Some research has shown that taking short breaks during long tasks helps you to maintain a constant level of performance; while working breakless leads to steady decline in performance.

3. Set self-imposed deadlines: While we usually think of a stress as a bad thing, a manageable level of self-imposed stress can actually be helpful in terms of giving us focus and helping us meet our goals. For open-ended tasks or projects, try giving yourself a deadline, and then stick to it. You may be surprised to discover just how focused and productive you can be when you're watching the clock.

4. Follow the "two-minute rule.": Entrepreneur Steve Olenski recommends implementing the "two-minute rule" to make the most of small windows of time that you have at work. The idea is this: If you see a task or action that you know can be done in two minutes or less, do it immediately as it takes less time than having to get back to it later.

5. Just say no to meetings: According to Atlassian, the average office worker spends over 31 hours each month in unproductive meetings. Before booking your next meeting, ask yourself whether you can accomplish the same goals or tasks via email, phone, or Web-based meeting (which may be slightly more productive).

6. Hold standing meetings: If you absolutely must have a meeting, there's some evidence that standing meetings can result in increased group arousal, decreased territoriality, and improved group performance.

Basecamp: Manage multiple projects help organize all of that stuff and keep things in order. You may still need physical folders for all your projects, but primarily for milestone and goal tracking purposes, not so much the record keeping and communication between team members. There is a free trial period for Basecamp, but after the trial is over there is no free continuation plan.

Evernote: Primarily treats like second workplace. You can store everything you want to learn later (just not right now) and organize all your notes from public speaking presentations, to books that you plan to publish, to courses, contact information, everything. Whenever you get a

new idea for a project you can pull out Evernote app and add a little note to a project folder, and when you go home and work on your desktop. You can sign up for a free account, but there are also Plus and Premium paid accounts. Those plans will give more storage and access to more features.

1Password: It generates and stores passwords for various things you sign up for online so that you only have to remember one master password. It comes with a small fee, but it's totally worth not having to write your passwords down onto paper or just forget your passwords all of the time.

Dropbox: Affordable cloud storage—primarily because it's easy to use and incredibly affordable. You get 2GB of storage for free, although people pay per month for a Pro account to handle more storage. When you download the application to your computer, your computer is essentially Dropbox enabled. From there, right click on anything and get the share link to paste into emails or chat messages.

TOPIC-3: Building E-Commerce SMB

Module-34

PLAY TO WIN

Putting your business on the web means putting your business in the game. And playing to win means taking a strategic approach to building an online presence, including:

- Getting a domain name that represents who you are and what you do at a glance
- Launching a website dedicated to telling your story, your way
- Creating valuable content
- Making sure the right people see that content (think social media and SEO)
- Using your website, email and directories to generate leads
- Measuring, then tweaking for improvement

ONLINE REAL ESTATE

Want an easy way to picture how your domain name and website work together? Real estate. Your website is your house, sitting on a piece of property with a street address. That property is known as your hosting account — the space on the Internet where you place the files that make up your website. People find those files (which they see as your website) by going to your online street address. This is your domain name.

What is a domain name?

A domain name is your website name. A domain name is the address where Internet users can access your website. A domain name is used for finding and identifying computers on the Internet. Computers use IP addresses, which are a series of number. However, it is difficult for humans to remember strings of numbers. Because of this, domain names were developed and used to identify entities on the Internet rather than using IP addresses.

A domain name can be any combination of letters and numbers, and it can be used in combination of the various domain name extensions, such as .com, .net and more. The domain name must be registered before you can use it. Every domain name is unique. No two websites can have the same domain name.

1. Keep it short. Would you remember it if you saw it on the side of a bus?
2. Make it easy to type. Avoid hyphens and unusual spellings.
3. Include keywords. Try to use words that people might enter when searching for your type of business.

4. Target your area. Use your city or state in your domain name to appeal to local customers.
5. Pick the right extension. One of those industry- or geo-specific domain endings might be a better fit for your business than a more generic .com.

Here is a simple set of steps to develop your dream domain name which describes who you are and what you do and where you do it. Once you have created a domain name, you need to check it's availability. For this we use a domain registrar which has been around for about 20 years. If the domain is available, proceed ahead and buy it before it is gone, if it is not available try for alternate domains. If the alternate domains are also not available, then you may like to contact the domain owner you are interested in, provided the domain owner is interested in selling the domain.

To find out who the owner is there are many web sites from where you can get help, we again try a site that has been around for a long time. From here you may get the contact info of the domain owner or you may get a link that leads you to the domain owner. So now you are able to contact the domain owner.

Alternately you may like to make an offer to the domain owner, there are several sites i.e. domain brokers that do that, you may like to check they have a joining fee and charge 10-20% commission on purchase.

Best of luck.

A **domain name registrar is** a company that manages the reservation of Internet **domain names**. A **domain name registrar** must be accredited by a generic top-level **domain** (gTLD) registry or a country code top-level **domain** (ccTLD) registry.
Sometimes, it's smart to register more than one domain name to represent your business. Here's why:

TO PIVOT FOR GROWTH
If you have a domain name with a traditional extension, like yourbusinessname.com, you can register additional domains that represent specific products or services you offer, or markets you serve, as your business grows.

TO PROTECT YOUR BRAND If you own a domain, your competition doesn't. This is one easy way to help protect your brand online.

TO IDENTIFY WITH A LOCAL MARKET

If you serve customers in a specific geographic area, it's important to make that local connection clear in your website address.

TO IDENTIFY WITH A SPECIFIC INDUSTRY

There's a good chance people will search for products and services by industry — like "wedding photographer in savannah" — so including your industry in your domain can make you stand out in search results.

PRO TIP: If you register more than one domain, attach your website to your primary domain and point any secondary domains to that address. It's really easy.

Module-35

Finding the perfect domain name might be getting easier thanks to hundreds of new generic top-level domains (gTLDs), but if you've ever set up a website for personal or commercial use, you're probably familiar with the frustrating feeling that comes with finding out the domain name you want isn't available.

There's no need to despair, though—your perfect name might belong to someone else, but that's not necessarily a reason to give up on the dream of owning it. Domain auctions are a great way to score (or sell, if that's your intention) previously-owned domain names in the global marketplace.

Market Places: If you're in the market to buy or sell a domain name, sites such as GreatDomains, Bido.com, and Afternic are ready and waiting to help make it happen. These sites (and hundreds like them) operate in a fashion similar to popular online auction sites such as eBay. Because thousands of domain names are purchased and dropped (or deregistered), on a daily basis, the pool of names available for resale swells and contracts almost constantly.

How It Works: Depending on the site they choose, sellers will generally be charged a listing fee (and possibly a sales commission). They can list their domain names and choose from various pricing options, including setting a reserve (ie, a minimum acceptable amount for sale, invisible to bidders), minimum offer (the absolute minimum price a buyer must submit in order to bid), and more. On the other side of the equation, some sites, such as NameJet, offer buyers interested in making high-value purchases a bidder verification service in order to review their eligibility and allow them to secure high-dollar bidding privileges.

Protect Yourself: Buying or selling, it's important for all parties involved to protect their assets. If you're buying, make sure you have the latest records on the availability of your name, and choose a reputable auction site that offers some measure of buyer protection (eBay, for example, extends its buyer protection plan to those who purchase domains on their site). Buyers **and** sellers alike may want to take advantage of an third-party plan such as Escrow.com's Domain and Website Escrow Service to protect both parties while payment clears and the domain is transferred.

You Have Options

Discovering your domain name is unavailable doesn't have to be the end of the road for your website. With some careful searching, judicious bidding, and a little luck, you might be able to snag your first-choice domain name from a second-hand source.

Buying Domains at Auction

1 AfterNic.com 2 Sedo.com 3 SnapNames.com 4 NameJet.com 5 Flippa.com

Listing fees start at $9. They charge a 15% sale commission, capped at $2,000.

Browse the website, and see what you find.

A fully developed site.

These will be hard to purchase at a sensible price.

No website at all.

The website may be dead, but it may also mean the owner just never bothered to put a website up on the domain.

Site with little content but an indication of sale price.

If a site lists the sale price, you can negotiate with the owner.

Don't bother asking about the price first thing–they will be turned off from the sale.

A parked website.

This means the owner is just making money off of advertising.

Check the Alexa rank to see how much traffic it already gets.

You may be able negotiate a reasonable price.

Site diverts to another site full of ads.

Check the Alexa rank.

Contact the owner to negotiate a sale.

Expired domains: aged domains the previous owner has let expire.

- FreshDrop.com is a good way to find expired options.
  - •Inventory constantly changes. See if you can figure out why the domain was dropped.
- Owner felt it wasn't worth renewing
  - It's not likely worth anything to you, if it wasn't to them.
- Registrant no longer with the company
  - Could offer value to you.
- Email address is no longer valid and not receiving emails about the domain name expiry
  - Could offer value to you.

- Forgot about owning the domain name.
    - Could offer value to you.
- Legal reason
    - May not be worth investing in.
- Banned by Google
    - No value to you, because Google won’t rank it.

Module-36

Benefits of selling online

If you're on the fence about whether or not to start selling online, jump off and start selling! Kidding, but here are some major perks to consider:

Wide customer reach

Expand your sales beyond friends, family, and your local community. Taking your business online will allow you to grow your business by reaching customers across the globe.

Measured results

Having your products and services online will allow you to quickly see how well your business is performing due to advanced tracking and analytics.

Low startup costs

While owning a brick-and-mortar store might be your end goal, it's very costly to have a storefront these days. Putting your goods and services online is an inexpensive way to start making sales and getting your product in the hands of eager customers.

Easy, inventory management

With most shopping cart software on the web today, you'll be able to easily track what sells and when, so you'll always know your inventory — no need to manually count items anymore!

START WITH THE END IN MIND

Understand the **end** result. **Begin with the End in Mind means to begin** each day, task, or project with a clear vision of your desired direction and destination, and then continue by flexing your proactive muscles to make things happen.

With that domain name in hand, you're ready to start *thinking* about your website. That's right — thinking about your website. Like most everything else in business and life, a bit of planning goes a long way toward ensuring that your site will do all the things you want it to do for your business.

Do you want your website to inform? To inspire? To generate sales leads? To actually sell products or services?

The nature of most business websites is either informational or sales-driven. Is it enough for your website to showcase your products and services, or do you want visitors to be able to buy them directly from the site? If so, you'll want an ecommerce website (more on that later). When

you figure out what you want your site to do for your business, you can begin building the type of site that will achieve those objectives.

If you plan to sell products or services on your website, you'll need an ecommerce website, also known as an online store. This type of website requires a few more specialized parts than a standard website. Here's what you'll need to plan for:

A SHOPPING CART

This tool will let you display product images and descriptions. Look for a cart option that includes important features like shipping options and inventory tracking.

PAYMENT PROCESSING

While you can use a third-party service like PayPal to collect payments, establishing your own merchant account to accept credit card payments will give your business more credibility. Merchant accounts let you accept payments from major credit, debit and gift cards on your site — so your customers don't have to leave your online store to pay for their goods.

SSL

The first thing most savvy customers look for when they make a purchase online is an SSL (Secure Sockets Layer) certificate. SSL certificates are digital certificates that encrypt the information your customers send when they purchase products or complete forms on your website. Visual indicators of an SSL can include a padlock icon in the browser, https:// before the website address, and a green address bar.

Secure your ecommerce website with an SSL certificate to make sure customers feel safe making purchases on your site.

What is an ecommerce shopping cart?

Have you ever purchased a pair of shoes or a new song or movie online? If so, you have participated in ecommerce. You have probably heard the term "ecommerce" before, and know it means commercial transactions that are conducted electronically on the Internet — but where it might get fuzzy is how this ties to your business and your website and what you need to do to sell online.

That's where the shopping cart comes in.

The term “shopping cart” might bring to mind the basket where you store items at the supermarket until you’re ready to check out with the cashier. That’s exactly how an ecommerce shopping cart works as well. It’s a virtual shopping basket where your web visitors can select and store items for eventual purchase.

Module-37

So you think you want to sell your products online? Making the leap from selling your goods and services locally to putting them online for the world to see can be confusing and scary. With myriad shopping cart software providers and online store builders to choose from, knowing where to get started is sometimes the hardest part.

Think you're ready? If you're dreaming of making sales and shipping products now, your next big decision is deciding what tool or service you will use for your shopping cart and how it will integrate with your website. From third-party marketplaces to plugins on your existing website, there are a lot, and we mean a LOT, of options out there.

Don't worry. We're here to help you navigate this tricky landscape so you can get your products on the web and earning money for you right away. Sound like a good deal?

Top features to seek in a shopping cart

One easy way to help determine if a tool is the right fit is to check out the features and pick what's most important to you. Here's a list of some of the most valuable ecommerce features:

Simple setup and ecommerce design. If you are building your online store yourself, it's important that whatever solution you select is user friendly. A good way to test this is to try a free trial or read customer reviews and testimonials.

Professional, customizable themes. The way your site looks is incredibly important in converting sales. If your site does not look professional and trustworthy, customers will not buy from you. Many online store site builders will showcase their available templates and designs on their website for you to check out before signing up.

Built-in payment processing. This is the bread and butter of online selling. Make sure you thoroughly understand the payment process — both how your customers will be billed and how you will receive payment. Many marketplace websites have additional fees for using their service

Easy shipping and mailing. Once you start selling products online, you have the pesky task of actually getting those products to your customers. Many ecommerce solutions remove the pain of shipping — from handling the mailings for you to providing easy-to-print shipping labels.

Customer support. Make sure whatever shopping cart solution you decide on offers the level of support you need.

1. Marketplace ecommerce solutions: Ready to get your ecommerce feet wet, but not sure you need a full-fledged website? Or maybe you are concerned customers won't find your website, so you want some customer acquisition assistance. Whatever the case, using a marketplace service can help you get your products online fast, while reaching a broad consumer audience. Ex **eBay, Etsy, Amazon and Alibaba** provide an easy entry into selling online. Making a page and listing your products on a marketplace gives you a place to send customers online and enables you to tap into a vast user-base of thousands of potential customers.

Tip: Set up both a dedicated online store (see below) AND a page on a popular marketplace to make the most of your ecommerce opportunity.

2. Hosted, Software-as-a-Service (SaaS) solutions: Do you need a website in addition to a shopping cart solution? Do you want something easy and cost effective? If so, hosted, software-as-a-service ecommerce tools are probably for you. Ecommerce solutions such as **Shopify, Magento Small Business** are great for the small retailer who wants a professional, quick-to-implement shopping cart.

Hosted, SaaS shopping carts typically require little technical knowledge and are inexpensive to launch. With thousands of templates and designs to choose from, you can easily build your website on these platforms without having to hire a designer or web professional.

3. Online payment services, apps and plugins: If you already have a website, chances are the thought of creating a new one doesn't sound too exciting. If your website isn't built to support an online store, there are various other solutions you can explore to maintain your existing site but also add an ecommerce component.

Ecommerce apps and plugins are an easy way to allow your website visitors to view and purchase your products and services without ever leaving your site.

As an integrated solution to an existing website and hosting provider, these shopping cart apps make it easy for you to go from website to online store in a snap. Examples of these easy-to-integrate solutions include **PayPal, Stripe and WooCommerce**.

Module-38

OK, I know that you might not be well-versed in the art of website building — and I don't expect you to be. The good news is that you've got options. Self-starter? You can do it yourself with a template-based site builder or content management
system like WordPress. Too busy for that jazz? Hire a professional.

But whatever path you take, you'll want to consider these primary factors: cost, customization, complexity, time, and ongoing maintenance. A good method for making your final decision is to prioritize your most important considerations, then weigh the options against your needs.

Q1. When considering which local business to use, which of these statements about 'local business websites' applies most to you?

Key Findings: 36% of respondents say that a clear & smart website gives a local business more credibility

•only 5% of respondents said that an bad or ugly website would put them off using a local business

Gender breakdown: Men are more design influenced and give more credibility to a 'clear & smart' website. Female respondents said they more likely to contact a local business if they have a website

Age group breakdown:

Older consumers (55+) give more credibility to a business which has a 'clear & smart' website. Consumers aged 18-34 are twice as likely than those aged 55+ to contact a local business if they have a website. Younger consumers more influenced by website aesthetics

Analysis:

Of those surveyed, 68% said that having a website – ideally one that is well designed – is a key factor in the opinion they hold about a business and directly influences their decision to use a local business. Only 27% were not bothered about whether a local business has a website or not, and wouldn't judge them if they didn't. Additionally, if a local business has a bad or ugly website then it won't put of most customers from contacting you. Perhaps people expect that larger, national & multi-national brands to have more impressive websites, but for a local business this takes on less importance.

SITE BUILDER: Site builders like, Wix and Squarespace are great if you're a DIY-type who wants an affordable, attractive, basic website in a short amount of time. Simply choose a pre-designed

template and then replace the text and images to meet your needs. Drag-and-drop. Easy to create and update. Plus, most popular site builder plans including website hosting.

WORDPRESS: Like the idea of building and updating your own website without learning HTML, but want more flexibility than a site builder? If you've got a little skill and some extra time, a CMS such as WordPress might be for you. You can choose from myriad free or paid WordPress themes (designs for the overall style of your website). A plethora of plugins also can boost your site's functionality. Some WordPress offerings even bundle hosting, security and support into one plan to make it easier to set up and maintain a WordPress website.

cPANEL: cPanel is a Linux-based web hosting control panel. If a person purchases the hosting from large companies or even from smaller local companies, it is more likely that the hosting will come with cPanel.

PROFESSIONAL DESIGNER: Hiring a professional designer is a great option if you have an idea for your website, but don't want to build it yourself. A pro can collaborate with you to turn your vision into a fully functional, customized website that meets your online goals. Unless you've got a designer buddy who owes you a big favor, this is the most expensive route to get online. But it can be a great investment when you consider the time and effort it will require for you to build a custom website.

Advantages

- It is very easy to host a website using cPanel.
- It is multilingual and available in many languages.
- It is responsive and adapts to any screen size, it can be used efficiently in mobile and Tablet too.
- cPanel has in–built File Manager, which helps in managing your files without the hassles of an FTP.
- cPanel has integrated webmail software, which helps in sending and receiving emails through online webmail client.
- You can easily create a backup of your website in a few steps. This will help you to restore your backup, if your website encounters any error.
- You can easily manage your website databases, as cPanel has integrated phpMyAdmin, which helps to directly manage databases.

Disadvantages

cPanel does not have any significant drawbacks, however here are a couple of its notable disadvantages:

- Need to know HTML to edit the templates as per your requirement.
- cPanel is meant for small and medium websites only. Creating a large website using cPanel is not recommended.

Module-39

This is the first part of the series of three modeless with the same topic i.e. creating remarkable content. In the part-I we will talk upto color psychology. In part-II we will talk about upto homepage and in part-III will do a worksheet for your website.

TELL YOUR STORY

So, what's your story? Every business has one. And a compelling story drives an effective small business website. How did you get started? What do you tell customers about your business, or how do you explain to friends how you make a living? What problem can you solve for your customers, as only you can? Think about your mission, your passion, and what makes you stand apart from the competition.

It could be as simple as this:
"I got hold of cotton pest incidence data and cotton yield data for about a decade. After I adjusted the frequency, generated a line plot I was surprised to see a negative correlation between yield and pesticide use i.e. when pesticide use increased the yield went down and vice versa i.e. when I am sick and I take medication I get sicker..." And that, my friends, could be the anchor text for your home page.

Logo, Fonts, Colors, Layout and more, by putting a little thought into these basic design elements, you'll make big strides in telling your story, building your brand, and framing your products and services in the best possible light on the web.

FIVE CORE PAGES

You're going to tell that story throughout a handful of pages on the your website. I think five core pages — **Home, About Us, Products/Services, Testimonials,** and **Contact Us** — makes for a solid small business website, but you really can do it in as few as three pages.

Developing the content for these pages requires telling your story with words that count (text) and images (photos, videos, your logo, etc.), presented in an appealing way (that's the design factor).

It's important to make this space — your website — easy for visitors to travel through, or navigate. A click here, a link there, and they'll get a clear idea of who you are and what you can offer to them.

GET INSPIRED

Not sure how to start telling your business's unique story on your website? Well, you don't have to do it alone. Here are a few places you might find inspiration:
– talking to friends
– customer referrals
– existing brochures and other marketing materials
– company newsletters or cards
– other websites
Take a look at what other successful businesses in your industry do on their websites, and note what you like and what you don't. There's nothing wrong with getting a little online inspiration, just don't copy someone else's content.

DESIGN BASICS

Even if you hire a pro to build your site for you, you'll need to make (or at least approve) decisions about the look of the site — its design. Few design fundamentals

1. LOGO: Think about how you want to incorporate your company logo into your website's design. Maybe you want to echo the colors of your logo on your site, or make the logo "pop" against a contrasting background.

2. COLORS: For brand harmony, it's important to choose the right color palette for your website. Do you own a creative company? Perhaps vibrant colors like hot pink and tangerine speak to your brand. If you're in the professional services industry, more subtle hues such as charcoal and blue might be more appropriate.

THINK ABOUT THE FEELINGS COLORS EVOKE FOR YOU AND MATCH THEM WITH HOW YOU WANT YOUR CUSTOMERS TO FEEL WHEN THEY VISIT YOUR WEBSITE.

3. FONTS: The way type on your website looks impacts readers subtly, but because text is everywhere on your site, it adds up to a substantial impression. Think about the fonts that might best represent your particular business — from bold, linear styles to more delicate, feminine fonts.

4. LAYOUTS AND MORE: Consider the amount of "whitespace" (space between elements) in your design. A lot of whitespace can denote clarity or simplicity, while having very little of it can make your site look active or intense. Other elements, like background colors, gradients, and the "texture" of your overall design can contribute to your online impression.

Colors affect people's psychology and their performance. The choice of color in marketing has a significant effect on business.

Colors affect a person's mood, choices and activities. People's perception of colors differs from culture to culture. However, some colors' have universal meanings based on biological reasons while other colors have semantics rooted in the history, values etc. i.e. cultural. Some universal meanings of colors are related to the responses of optic nerves to the colors. Differences in color meanings between the eastern and western cultures. For example, while black color in Western cultures indicates power, control and death, it indicates wealth, health and prosperity in Eastern cultures.

The world seems to be getting a little smaller each day thanks to online communities and social networking. In turn, this "world-wide community" has created an international readership for a variety of websites.

Designers must weigh carefully the messages they send to that potentially broad user-base.

One aspect of design that can have far reaching and sometimes unintentional effects on readers is color. Colors have a variety of associations within North American culture alone, and can mean something radically different to Japanese or Middle Eastern readers, where color meanings are frequently much more specific and defined.

It is important to understand how color associations vary from culture to culture, and within different possible audiences, when planning a website.

Understanding color can be a tricky challenge and many color meanings can almost seem contradictory — particularly in the West, where color meanings are extremely broad. When working with color, remember to think about context and how color is used with other elements such as text and photos.

Module-40

: Can you spot something in this logo? The FedEx logo, designed in 1994 by Linden Leader & Landor Associates, at first appears simple and straightforward. However, if you look at the white space between the "E" and "x" you can see a right-facing arrow. This "hidden" arrow was intended to be a subliminal symbol for speed and precision.

Leader and his team at Landor Associates, the consulting firm that was tasked with reinventing **FedEx's** brand identity, developed over 400 versions of the **logo**, before noticing that putting a capital "E" and a lowercase "X" together created the suggestion of an **arrow**.

Baskin-Robbins: In 2005, as part of its 60th anniversary celebration, Baskin-Robbins launched a new brand identity. The new logo was intended to "capture the fun and energy of Baskin-Robbins." In the old logo, the number "31" appeared within a simple arc, suggestive of a scoop of ice cream, and next to the name. In the new logo, you can see that the "31" still exists. It is now formed by the pink portion of the ice cream store's two initials: "B" and "R.“

Data2BI: Data->Information->knowledge->intelligence. This quantum leap shown by the arrows emanating from the database in the mind. The first arrow at left looks like an eye closed peacefully after data converted to intelligence.

WEBSITE IMAGE TIPS

How much time would you spend on site where you’re expected you to read everything to understand what’s going on? Not much, I wager. By using a few thoughtfully chosen pictures and/or short videos, you can quickly convey the information and impression you’re going for.

1. SHOW INSTEAD OF TELLING If it would take a thousand words to explain something, showing it is better. This is especially true of product shots and video demonstrations. Use them to create an impact.

2. TAKE THEM YOURSELF If you need to snap a few shots of a product or make a video, you already have a tool you can use — your smartphone. Just adjust the lighting, be aware of the background you’re using, and snap away. When you’re done, you can edit them using an online editor, or an application like Adobe Photoshop.

3. USE QUALITY STOCK IMAGES With stock images, you can save yourself time and get high-quality images at a fraction of the cost of a professional photographer. To learn more, check out The Smart Guide to Using Stock Images.

IF YOU'RE SELLING PRODUCTS ONLINE, IT MIGHT BE WORTH THE INVESTMENT TO HAVE PROFESSIONAL PHOTOS TAKEN OF EACH PRODUCT

BE MOBILE-READY

If you recall, in Module-5 I gave latest world digital statistics. As per the statistics, the internet users were 4 billion i.e. 53% penetration, but unique mobile phone users were 5.135 billions i.e. 68% penetration. All mobile phone users may not be smartphone users, but this data gives a fair idea about the mobile phone market size.

Here's a staggering mobile fact from our friends at Google: 80 percent of smartphone users do shopping research on their phones. They tap and swipe their way through search results, websites and reviews to figure out what they're going to buy, and from which seller. What then? A full 50 percent of those mobile-savvy searchers visit a store within a day.

More than just a site you can see on a phone, a mobile website is optimized for use on a smartphone and other mobile devices. It's *designed* for a positive user experience on mobile. Whether you design your own website or hire a designer to do it for you, make sure your small business website is mobile-responsive.

If your website isn't easy to tap and swipe, you're out of luck.

Remind me: How many chances do you get to make a good impression? Oh, that's right. One. Just one.

If you don't have the best homepage possible, that first impression you lose that first impression forever.

Will the visitor come back? Maybe. But you're playing with fire.

There aren't any new statistics on web design aesthetics and first impressions, but an older study demonstrated that 94 percent of people's first impressions of a business were related to web design. That's pretty illustrative.

If you have a beautiful, functional, easily navigable homepage, you're more likely to retain visitors and convince them to come back for more.

Without one, you're practically shooing visitors away. And that's bad for business.

Creating the best homepage for your business can pay off big time, especially if most of your visitors land on the homepage first (rather than a paid ad landing page or a blog post found via organic search). Someone tells a friend to look up Company XYZ — that's you — so the friend types "Company XYZ" in Google's search box.

Your homepage pops up first.

Module-41

Creating great content for your audience is only half the battle when visitors come to your website. The information they seek must also be well organized and easy to find within the structure of the site. By organizing your content into categories you will not only provide a clear path for your visitors, but it will also allow you to give stronger definition to your products or services, plus the search engines will be able to keep your latest updates as fresh as possible within their index.

When it it comes to the site structure aspects of effective technical SEO methods, there is one tried and true type of site architecture that has consistently proven itself in reaping the greatest benefits. The **Silo Structure** organizes website pages into main categories, each with their own sub-categories (or sub-topics), which are then supported additionally by articles written within the Blog section of the website. The benefits this type of site structure provides take into consideration your 2 most important visitors: potential customers and search engines with whom you want to achieve higher ranks.

HOME PAGE

On your home page, you want to include the top things visitors need to know in order to decide to do business with you. Who are you? What do you do/sell? Why should I trust you? How do I contact you?

Describe your business in one short sentence (tagline).
Describe your business and what's great about it in just a few sentences

Call-to-Action. What is the one, most important thing you want your visitors to do before they leave your site? Be clear, concise and tell the what you want them to do. *Examples: Call for a quote • Schedule appointment online • Watch our demo • Sign up for a class • Donate now • Email us for a free quote*

Primary Contact Information. Include one primary way for customers to contact your business on the home page; typically, this is your phone number. Your "Contact Us" page can include all of the various ways to contact your business.

Products/Services Section. Include a short bulleted list or a few photos of your products and services on the home page, and then add a link to the full product/services page to view all. List general categories of products or services. *Example: Specializing in Residential Glass Services: Dual Pane Glass Replacement, Custom Showers and Mirrors, Glass Shelves and Tabletops, and more.*

Sign Up Form. What information do you need to collect? (i.e. email, name, phone, etc.). **Tip: The less information you require, the more sign ups you'll get.** How will you motivate visitors to sign up? Very briefly describe why your visitors should sign up and include this with your sign up form. *Example: "Sign up for exclusive deals and VIP access to special events."*

Credibility. Include one customer quote or review on the home page and link to the testimonial page for visitors to read them all. Web **credibility** is about making your **website** in such a way that it comes across as trustworthy and knowledgeable. A **credible website** can reap huge benefits on to your **website** and your business. Just 52.8% of web users believe online information to be **credible** (source: UCLA)

How to Establish Credibility in Your Writing

- Provide Evidence. Whatever claims you make in your writing, you need to be able to back them up. ...
- Show, Don't Tell. Don't just tell your readers that you have the best product or that they should side with your opinion of an argument. ...
- Use Active Voice.
- Share Personal Stories.
- Use Solid Writing Skills.

1. Blog: A blog lets you share your ideas and working practices, giving a level of transparency to your business activities. This makes your business look honest which will boost your online credibility. People also see that you regularly update your content and that you aren't about to run off with their money. Then there's the fact that blogs can boost search rankings, which in turn boosts credibility.

2. Social Media: To be a credible online business nowadays you need to be active on social media. If you share real content with your followers you'll also be able to boost your credibility further: pictures of your work environment, your completed work, staff members, happy customers.

3. Customer reviews/testimonials: Review websites (Google, Yell, Yelp, Trip Advisor, etc.) can be a great way to boost your credibility. The views of other people on your business will do far more for your credibility than anything else: people are always going to trust other people more than they trust any business. Some would say that testimonials have a poor impact because they can easily be made up and put onto the website by the business. But the fact of the matter is that one of the most-visited pages on any website is the testimonial page.

4. Be an expert: Use your online presence to portray yourself as an expert. You can link this with blogs by picking topics you know a great deal about and writing about them, aiming to provide some useful information for your audience/potential customers.

5. Be easy to reach: There is nothing that shouts "dodgy website" more than the absence of contact details. Don't be that business. A contact form isn't good enough. And having just an email is definitely not a good way to establish credibility. Adding a phone number to that email is a good start, especially if it's a landline.

6. Have a good website: Doesn't just mean having a beautiful design, it's about the language that you use, the kind of things you share about your business in your content, the availability of contact details, and how user-friendly it is. Your website is how people assess your business, and it usually happens instantly.

Module-42

Congratulations! You've already accomplished the first rule of thumb for getting your website found — creating relevant content. The type of content that will resonate with your target customer. Keep at it. (Oh, didn't I mention that content creation is an ongoing process? Keep it fresh, that's the mantra.) Now it's time to take the extra steps needed to boost your business's visibility online.

Enter search engine optimization, fondly known as SEO.

SEARCH ENGINE OPTIMIZATION (SEO): THE PROCESS OF REFINING A WEBSITE TO GET HIGHER SEARCH ENGINE RANKINGS AND ORGANIC VISITORS TO YOUR SITE, WITHOUT PAYING FOR SEARCH ENGINE PLACEMENT.

Imagine you're a chef who's spent a week perfecting a new recipe. If you don't get it on the menu, it's going to collect dust back in the kitchen because most of your customers aren't going to know about it. It's the same way with your website — you've cooked up all this amazing content, and you want search engines like Google and Bing to serve it up to potential customers in their search results pages. Bake it and they will come? Um, no.

A strong **organic search** ranking will work wonders at attracting visitors to your website. Unlike paid listings — advertisements that display in sponsored areas — organic search results are "free" and based on, among other things, the site's content and how closely it matches the keywords being searched.

Details of SEO discussed in subsequent modules, here give a brief intro.

While each search engine has its own set of ranking criteria, they all look at these same basic but key elements:

RELEVANT PAGE CONTENT(KING): A good rule of thumb is to include between 300- 700 words per page, focusing page content on its target keyword's subject matter.

KEYWORDS: Do some research to discover what **select words and phrases** people use to search for your type of business. Sprinkle these keywords throughout the content on your website

META TAGS: These are HTML tags that contain info to help search engines know what your site's about. They help describe your website in search engine results.

WEBSITE NAVIGATION: This covers all the links on your site and how visitors navigate from page to page. Like visitors, search engines rely on good navigation to get around your site. **SURE ALL THE URLS IN YOUR NAVIGATION ARE VALID.**

SITEMAP

A sitemap is essentially a map or directory of all the pages in your website. It guides search engines throughout your site with the names and locations of pages.

LINK BUILDING

Search engines use **internal links** and **backlinks** to rank your site. Internal link links a keyword or sentence on one page of your website to another page on your site. More important for ranking are backlinks; links from other websites that point to your site.

With an estimated 90% of online shoppers saying online reviews sway their buying decisions, cant deny significance of your business's presence on review sites.

GOOD CHANCE YOU'VE GONE TO THE LOCAL RESTAURANT WITH 3 STARS RATHER THAN 2.5 STARS — SO WHY WOULDN'T YOUR POTENTIAL CUSTOMERS ACT THE SAME WAY?

Online reviews and directory sites (like Google My Busi-ness, Foursquare and Yelp) are a potential option for new business. If people like what they see, they'll click the web-site link in the review or directory listing, and you've got a new visitor on your site. Plus, search engines typically pull information from business listing sites and display under your website address for local search results. You want your website correct address (and other information) to be listed on these sites, few options here:

1. Individually go to each site and update your business's pertinent information.
2. Subscribe to a service that enables you to enter and update that per-tinent info in one place, before pushing it out to countless review and directory sites.

Online maps are a go-to source for local business information. They're a fantastic tool for helping potential customers find your business.

Serious about getting your website found? Find, claim, and verify your business information on key online review sites, directories and maps.

DO I NEED A BLOG?

If you want to increase traffic to your website, boost search engine rankings, and establish yourself as a subject-matter expert — yes. A blog also is a great way to interact with your customers and capture feedback about your products and services.

Again, this is all about adding valuable content to the mix. A blog gives you an amazing content marketing vehicle — a means to talk about your company's values, provide a look behind the scenes, and offer useful solutions without sounding like you're "selling."

Your blog's content makes excellent fodder for your social media sites (more on that soon), and it's a great way to build relationships with industry influencers and customers by linking to relevant content on their sites. Post consistently for the best results, at least once every two weeks — more often is better.

1. A blog gives you content to share on social media: Having multiple social media accounts such as Facebook, LinkedIn, and Twitter has become a crucial part of every digital marketing strategy. Not only does a blog serve the purpose of providing original and relevant content for all of your social platforms, but it also drives traffic back to your website; remember backlinks

2. A blog humanizes your brand: Your customers are used to know about your products and services in a professional, persuasive manner. A blog gives your website viewers the opportunity to get to know you on a deeper, more personal level. This is a space where you can have a little more fun with your brand, share your opinions, and provide a "behind the scenes" look into your company. Still not convinced? According to Quick Sprout, 60% of consumers feel positive about a corporate brand after reading their blog, and 61% have made a purchase based solely on a blog post.

3. A blog gives you industry credibility: A blog positions you as an expert in your field. Having a consistent and well-written blog tells all of your customers, competitors, and employees that your company is a credible source that keeps up with industry innovations and developments.

4. A blog increases your SEO performance: Most people think of SEO technically such as enhancing metadata and URLs. However, if you stay up-to-date with the latest news in SEO then you know that the future of SEO lies in having a lot of rich, relevant content on your site. What better way to constantly be adding fresh content to your website than with a blog? Because Google praises websites with both robust and consistent content, a blog section kills two SEO birds with one stone.

Module-43

Ever wonder how to pick the perfect keyword topic for your blog or website, or find one that will have the most value? Most SEO strategists and content creators focus on, at best, two of the eight dimensions they should analyze when creating a keyword strategy for a website. Much like other SEO strategies, keyword research has evolved to a level that goes well beyond "how many people searched for a topic." It now includes many more variables that help define how much value a topic will bring to a website.

Here are the **eight dimensions of a perfect keyword**, which when combined and consistently executed upon, earn search rankings, drive traffic and goal completions, and build brands.

1. Search volume: This is the variable that has been around the longest. It's what most search optimizers use to determine the value of a keyword or key phrase. They go to a keyword tool (we love SEMrush) that gives them a basic representation of how many people are searching for a topic, and then they target those keywords which have the highest search volume. This might be a great place to start for some initial analysis and direction, but when defining the value of a keyword or topic, this dimension only tells a small part of a much larger picture.

2. Conversion value: When determining the value of a keyword, you also need to look at its potential *conversion value* — one of many measurable points that define progress towards a sale. A few measurable conversion points include:

- Email signup
- Product sale
- Ad click
- Social share

3. Brand value: One of the challenges when business start to scale is keeping the original brand values intact, and making sure that each team member is representing the brand in a consistent way. **The content on your blog or website is the digital salesperson in the organization, and should follow the same brand values and guidelines.** A few questions to determine if the topic is "on-brand" include:

- Does the keyword represent the brand in a positive way?
- Would you talk about the product or brand using this keyword?
- Does it align with the brand positioning and tone of voice?

4. Persona value: In traditional advertising (print, radio, TV) uses simple demographic data to define and target audiences. In the online world, marketers dive deeper into the psychographics of the user to define what we call *personas* to help sculpt targeted content

experiences that speak directly to the interests of the audience. At a minimum, you should ask yourself if the keyword or topic aligns with your target audience in the following ways:

- Is it what they like to read? Are they interested in the topic?
- Is it written in words they use, and not in your "marketing lingo?"
- Would they find the information inclusive enough, or would they need to go to another website to learn more?
- Would they be likely to share this information in their social circles?

5. Trending value: When we analyze a keyword we like to determine if it's an "evergreen topic" or a "timely topic." Evergreen topics have long lasting value, so when looking at the trend, it should be pretty consistent throughout the year. Traditionally, these tend to be slightly more competitive, but have a much longer lifespan for being relevant. Timely topics are seasonal (like holiday recipes) or more focused on current events or news coverage. Timely topics have a much shorter lifespan, but can drive large bursts of traffic if done well. You can use Google Trends to gain insights into what type of topic it might be. Once in Google Trends, try to answer the following basic questions about the topic:

- Is this topic currently trending? Does it show consistent trend growth?
- Does it have a seasonality? would require little updating six months from now?

Having a good mix of evergreen content topics and timely content topics is important for long-term growth, while still staying relevant with your personas.

6. Competitive value

It's important to understand and answer what it will take to drive traffic for that keyword:

- What are others doing to compete for this keyword?
- What type of content are they creating?
- Are they bigger, faster or stronger than you, or do you have a chance to compete on the target keyword?

When thinking about the importance of competitive value, think about ***jungle-lion-joggers.***

Types of competitive SEO audits to perform when determining keyword value that help understand how to compete for keyword that you're trying to rank for:

Experience-based competitive audits analyze variables that would affect the experience someone has on your website compared to that of your competitors.

Search-based competitive audits uncover the competitive landscape within the search results for a defined set of high-value topics.

7. Funnel value

What are your goals? Are you trying to inform and drive familiarity of your business, drive a single purchase, build loyalty, or simply become part of the consideration set? Funnel value defines the type of asset that would work best to accomplish your goal.

8. Social value

Since Google is starting to include social signals in its ranking algorithm, therefore, questions to answer when trying to determine if a topic has social value include:

- Do people like sharing content like what you're creating?
- Do they like talking about the topic on social media?
- Has other content about this topic been shared often on social media?

Now we wouldn't expect someone to go from using one or two of these targets to using all eight overnight; it takes time to evolve your keyword research process. What we would suggest is that when you're searching for your next keyword topic for a blog post or product, try to include some of these other variables in your analysis to strengthen the keyword value.

Module-44

Establishing a business presence on the social networks where your customers and prospects spend their time is sort of like setting up a booth at the world's largest virtual trade show (although sometimes it feels like a circus). You can bet your competitors are there, competing for the attention of all those potential customers with flashy banners and giveaways. There's a crowd gathered around the booth playing a funny video and the vendor live-streaming a demo.

Why is social media so important for businesses today?, and more importantly, how can you get in on that action?

SOCIAL MEDIA SITES AND BLOGS REACH 8 OUT OF 10 OF ALL U.S. INTERNET USERS.

The true power of social media is influence. Social provides an avenue for companies to not only engage with customers, but also influence them with the right content that helps them make a decision. Many companies are not leveraging the power of social media to the best of their capabilities.

BUILD RELATIONSHIPS

By developing a following on popular social media sites, you can connect with your customers and prospects and share content with large number of people. Social media makes it easy to target specific customer questions or issues, while making the same information available to all existing and potential customers.

BUILD YOUR BRAND

Social media gives you the opportunity to share valuable information about your business and showcase your personality.

DRIVE TRAFFIC TO YOUR WEBSITE

Including a link back to your website in interesting social posts can generate serious traffic.

ESTABLISH EXPERTISE

Share your knowledge and experience to build credibility as an industry thought leader.

MAKE SALES

Ultimately, these activities — connections, brand building, driving traffic, and establishing expertise— all lead to the same place: sales.

Establishing and maintaining a strong social media presence for your business definitely requires a commitment of your time, but the benefits make that investment worthwhile. By getting social, you will continue to build credibility, raise brand awareness, and, especially, engage with current and potential customers.

Locate your customers

Just like IRL (in real life) you can't be everywhere — in your apartment, at your son's cricket match, at the grocery store, oon the treadmill— at the same time. Same for social media. You've got to figure out where the people who are most interested in your type of products or services are spending their time, and go there.

Different social networks appeal to different demographics. Some platforms specialize in visual content — perfect for product-centric businesses. Some networks allow you to share long-form content, live-stream video, stay on top of industry trends, and also "spy" on your competitors.

Do some research. Where is your target audience spending time? How about your competitors? What benefits can each network offer your business? Analyze and prioritize.

Take all that data, look at it in the context of the marketing goals you've established for your business, and develop a social media strategy that works for you.

1. Fill in your profile: If people are interested in the type of content you're sharing and discussing, they'll likely be interested in finding out where they can get more.

2. Promote your blog content: It's been found that brands who create 15 blog posts per month (and share that content through social media) average 1,200 new leads per month – proving it's not impossible to get your content seen by ideal customers.

3. Make your content easy to share: Over 41% of people measure the social influence of a blog by the number of shares it gets. If you're making this information available to your site visitors, it could build trust and lead to higher conversion rates.

4. Post when your audience is active: What good is posting on social media if your target audience isn't online to see it? The best time to post on social media is when your target audience is most active – their "peak time."

5. Focus on sharing visual content: It's easy to miss content that might be interesting or otherwise helpful while scrolling endlessly through a social media feed and. Stop potential site visitors from doing the same by sharing visual content that stands out.

6. Engage with your audience (consistently): Social media is unique in that you can engage directly with your target audience. You can answer questions and obtain feedback in real-time, and improve the experience people have with your brand.

7. Optimize your calls-to-action: Calls-to-action tell a user exactly what you want them to do. Add phrases like: "Click Here" "Read More" or "Visit Our Site"

8. Test paid social advertising: Social advertising is an effective way to reach new people who haven't yet heard of your brand or website. If you have the marketing budget, social advertising is worth experimenting with. That's because you don't need to break the bank.

Module-45

BEST PRACTICES FOR USING SOCIAL MEDIA FOR BUSINESS

Once you're all set up on your chosen networks, and listening to get a pulse on your customers and competitors, it's time to start posting and sharing content. The fun part. To get the most out of social media for your business, it helps to follow a few basic guidelines:

MAKE FRIENDS: Social media is about people connecting with people. So, even if you're representing a business, be sure to let your human side shine through in your posts. Return favors (such as a sharing and liking posts and pages).

ADD VALUE: Provide useful information in your posts, such as links to related articles and videos.

SHOW SOME RESPECT: Treat your followers with dignity and respect, and you're likely to get the same in return.

PRACTICE RESTRAINT: One of the quickest ways to lose followers is to overdo it. Posting quality content once or twice a day is enough to engage with your audience.

GET IT RIGHT: Be sure to check your facts, spelling and grammar.

ENCOURAGE OPEN DIALOGUE: Craft posts that encourage followers to respond. Ask questions.

BE RESPONSIVE: Acknowledge followers' feedback and engage them in conversation.

GET VISUAL: Post photos, videos, and other visual elements to boost engagement.

LEVERAGE PROMOTIONS: Come up with special offers for social media followers.

DRIVE TRAFFIC TO YOUR WEBSITE

Include your website address in posts for followers who want to learn more.

YOUR WEBSITE AND SOCIAL PROFILES

You want your business's website and social media profiles to be connected "naturally". Texting each other as soon as you find time. Sharing friends and trading gossip. You get the point. After you set up your social media profiles, post links to them everywhere — on your website, storefront, marketing materials, even your email signature.

Buttons on website. Make it easy for website visitors to become followers by adding social media icons or links to your website.

Links in social posts. Include your website's URL in your social media posts. Also, add widgets or add-ons to your site so live feeds of your social media posts show up there.

Share, Follow, Like. When people "like" your business's social page, they become members of your business's online community. Take every opportunity to "share," "follow," and "like" others' pages so they'll return the favor!

TIME-SAVING TOOLS

As a small business owner one thing's for sure — you're busy. And social media has a well-deserved reputation as a bit of a time-suck. These tools can help you manage your business's social media presence in less time, with fewer headaches.

Hootsuite. This is a popular tool for scheduling posts in advance and using keywords and hashtags to listen in on conversational streams.

Buffer. It's super-easy to batch schedule updates and share pages on all your social networks.

Sprout Social. This powerful platform features publishing, engagement and analytics functions that make is much easier to manage and engage on all your social networks.

Likeable Local. This robust tool does the legwork for you, automating the process of providing content on social media and making sure it reaches the right audience for your business.

Feedly. This RSS reader will help you keep track of new posts that go out from your competition.

TweetJukebox. Enter as many tweets as you like, then set a calendar for when you want them to go live.

Tip: Use the scheduling feature available for Facebook business pages.

5 ways to engage with followers

Still not sure the best way to engage with social media for small business? We've got you covered. Check out these five tips:

Promote your content. If you've written an engaging blog post or produced an interesting report or white paper, you can promote this content on your social channels.

Share others' content. That's right. It doesn't have to be all your own original content that you share. Now you might not want to share your competitor's content, but national associations, trade groups and the like often produce great content.

Dole out mentions. If you're sharing someone else's content or perhaps you've just met with them, you can engage through a mention (on Twitter) or tag (on Facebook).

Offer replies and favorites. When an individual or organization mentions you on a social media channel, you can keep the conversation going with a reply. You might also simply "favorite" a tweet.

Leverage private messaging. Just because you're on social media doesn't mean everything has to be public. You can take advantage of the private messaging features within the platforms to engage

How to Measure success?

So, how do you know if your work has paid off? What does success look like? Are you getting a return on your investment? Consider these key metrics.

Followers, fans and views. On their own, these numbers don't mean much. But they certainly give you a sense of your reach.

Engagement. Are people responding to and commenting on your content?

Shares. What content is motivating your followers to share it?

Clicks. How many people are following the links in your posts and ending up on your website?

Action. And once they're there, are they signing up for a newsletter, submitting a contact form or buying anything?

Now that you've decided what you want to measure, how do you accomplish that feat? Here are a few tools that can help:

Platform analytics. As an account administrator, you have access to tons of data from within Facebook, Twitter, LinkedIn and Pinterest.

Google Analytics. Get a handle on where people go once they land on your site and how much time they spend there.

bit.ly. When you create a unique URL with this URL shortener, you have the ability to easily track clicks. (Plus, a bit.ly link is great for Twitter's limited character count.)

Schedulers. Applications like Hootsuite and Buffer not only schedule your posts but track key data as well.

Module-46

We've talked about how your social profiles can drive visitors to your business website (it's huge), but we're not quite finished on this topic. Driving traffic to your website is an ongoing process. Like maintaining your hair color. And raising your children.

Like any worthwhile endeavor, attracting visitors to your business's online home takes time and patience. You might even need to use your ATM card. Let's look at a handful of proven strategies

CREATE COMPELLING, SEARCH-WORTHY CONTENT

Yes, that's again. Whether it's a shareable infographic, entertaining how-to video, or blog post about why spending time in your online store is more fun than watching a popular movie, make content that people can't find anywhere else. This is the kind of stuff people will want to link to from their own websites and social profiles (remember *backlinks?*). Make sure they find that amazing content by optimizing it for search engines.

SEND BUSINESS-CLASS EMAIL

Remember that domain you registered? The one connected to your website? You can use it to send branded email to your customers and subscribers. Every time you send an email from that business-class account, you're marketing your website.

To really drive traffic back to your website, include strong calls-to-action (CTAs) in your emails. Come up with short, catchy phrases that inspire readers to take a specific action on your website. And that's just one of many ways you can use email marketing to drum up business on your site — we're going to dive deeper in the next modules.

MAXIMIZE BLOGS

Share snippets from your blog on your social channels, with links back to the entire post on your website. Drive even more visitors to your site by hosting guest posts from industry influencers and satisfied customers — and ask them to share the link to the post on your website with their social followers. And don't forget that all that fresh content you add to your blog is like candy for search engines.

JOIN ONLINE COMMUNITIES

Like local meetups, there are online communities for just about every interest and industry you can imagine. Check out forums, blogs and chats related to your market. Leave meaningful comments with links back to relevant content on your website.

SET UP SHOP ON A MARKETPLACE

Maybe you're already got an Etsy site or a Fiverr profile — these marketplace sites are handy for promoting your products or services before you set up a website of your own. After you're up and running, you can use your marketplace pages to drive traffic to your new website. Feature your hand-knitted mittens on your Etsy site, for example, with a link to your online store for the matching scarf. Offer your headline writing services for $50/hour on Elance, and link to a full menu of your copywriting services on your website. You get the idea.

RUN PROMOTIONS

Everybody loves a deal. Offer online-only promotions, redeemable through your website. Get the word out by sharing the promotion on your blog and social media profiles.

LEVERAGE LOCAL REVIEWS

Make sure your website address is listed correctly on all those online review sites that 88 percent of your potential customers are visiting for info. If they like what they see — including your feedback — they're likely to click the link.

With its massive ROI and ability to keep your business connected with subscribers — people who've expressed an interest in receiving your emails — you're missing out if you don't embrace email marketing. Think of every newsletter and campaign you create as a chance to throw a party for your customers:

Send exclusive invites. Offer subscriber-only savings that recipients can redeem on your site.

Give party favors. Provide useful tips and tricks that link back to blog posts on your site.

Play spin the bottle. Include a shout-out to a customer who's doing something cool with your product or service (with a link to their website).

Crown a party queen. Introduce exciting new products and services with links to your website for more info.

Have cake. Include links to entertaining videos and other stuff that's just plain fun.

Call to action

You want your website visitors to *do* something — in this case, to give you their contact information. The call-to-action is a means to that end. It's a concise, convincing and (hopefully) clever offer that requests information from the visitor in exchange for something they might want. You can even link a call-to-action directly to a contact form.

Let's say you've got an online education or distance learning website; say PAKtuitions.com. This call-to-action encourages visitors to give you their contact information: "Looking for the perfect tutor? Sign up here and we'll notify you daily about new tutors!" In two short sentences — well-placed on your website — you're giving visitors an easy way to opt-in for a daily email from you. They get something of value, and you get something even more valuable — one step closer to a sale.

PAID OPTIONS

Sometimes you've gotta give a little to get more **"Money attracts Money".** Enter online ads, "sponsored" posts and other paid options designed to put your business info front-and-center and to drive traffic to your website. A few benefits:

Fast and cheap(er). Online ads cost a fraction of traditional print ads, and you can execute them quickly.

Captive audience. Your ads show up in front of targeted consumers glued to their favorite social networks and other websites.

Easy to measure. You'll get plenty of handy metrics for measuring the success of your online ad campaigns.

THE TOP THREE PAID ADVERTISING SPOTS ON A SEARCH RESULTS PAGE GET A WHOPPING 46% OF THE CLICKS OF THE PAGE.

Module-47

Driving traffic to your website is key, but converting your website's visitors into leads is how you'll stay in business. That means figuring out who's visiting your site and contacting them directly so you can work to convert them into customers.

At the end of the day, you're in sales. So don't be timid about asking website visitors to share a little bit about themselves. A little bit. A name and email address. That's all you need to start converting, and most people are used to giving out that info online. Ask for a phone number if you've got a callback strategy in place — but don't be shocked if some visitors decline giving their phone numbers.

INCLUDE YOUR BUSINESS PHONE NUMBER OR EMAIL ADDRESS ON EVERY PAGE OF YOUR WEBSITE.

Traffic Doesn't Matter Without Conversions

Lots of people brag about their website traffic. One blogger might say, *"I get 2 million unique visitors every month!"*

That's great, but I have a follow-up question: "How many of those 2 million visitors buy your product or service?" that put food on the table or cash in your kids' university education funds.

There's nothing wrong with creating a website and failing to monetize it. If you just want to write articles or post pictures or host videos, that's your choice. However, if you want to make money, you need to know how to convert website visitors into customers.

There are two main types of conversions:

Prospect to lead: In this conversion, you're turning a website visitor into a potential lead. Visitor A lands on a page of your website and sees that you have a mailing list. He or she signs up. That's a conversion.

Lead to customer: The more important conversion occurs when a lead buys something from you. He or she might follow you on social, read your blog, check out your marketing emails, and finally hand over the cash for whatever you're selling.

Visitor monetization: YouTube Partner Program (YPP), from 10,000 lifetime views to **1,000 subscribers** and **4,000** hours of watch time within the past 12 months.

Collect email addresses

306.4 billion emails sent daily in 2020, so email isn't going anywhere anytime soon. It's the primary way for businesses to connect with customers. But how do you get those valuable email addresses? Sure, you can place some sort of sign-up form on the counter if you have a brick-and-mortar store. You can collect emails at trade shows and other events. But your website is a much more powerful tool for soliciting email contact info from people who are interested in what you have to offer. Here's how you can collect email addresses and other contact info on your website:

Include a contact form. Visitors will fill out the form on your site because they have a question, need information, or want you to contact them. It's where they go to connect directly with you.

Add a sign-up form for visitors to receive information about products and services, updates, discounts, etc.

Include an email opt-in box on your checkout page (for ecommerce sites).

Give a special offer for joining the email list.

Pretty straightforward, right? Most popular website builders include a contact form in the template.

EMAIL MARKETING — WORTH THE INVESTMENT

Now that you've got all those email addresses, what are you going to do with them? How about reaching out to those customers and prospects though a branded, professional-looking email campaign that's brimming with value for its recipients? Great idea!

Email marketing offers a targeted approach to connect with current and potential customers. After all, they've opted to receive your email correspondence. It can build brand awareness and loyalty. Even better? The cost. Harvard Business Review called email marketing "the most cost-effective advertising method available today." When you invest in a legitimate email marketing program you get all kinds of goodness:

– It's easy to add a sign-up widget to your website.
– You'll store email addresses in a user-friendly database.
– You can export email addresses for use in social media campaigns.
– You can create attractive email campaigns using templates.
– You can track open rates/who clicks on what links.

– You can create contact interest groups.

BUSINESSES THAT USE EMAIL TO NURTURE LEADS GENERATE 50% MORE SALES-READY LEADS AND AT 33% LOWER COST.

EMAIL MARKETING BEST PRACTICES

The inbox is a crowded place. You want to send business emails that pass spam filters, stand out, and entice recipients to open and click. With that in mind:

1 Only send email marketing campaigns to people who have signed up for or requested them. Look for email marketing programs with opt-in buttons or sign-up forms to help build permission-based contact lists.

2 Stick to a schedule. Test different days and times to see which combination has the best open rate (the number of list subscribers who opened the email message, a percentage of the total number of emails sent). Be consistent.

3 Only include requested subject matter. Deliver information your contacts signed up for, and you'll build credibility and trust. Send them a newsletter about party planning when they signed up for plumbing tips, and you'll probably lose a subscriber.

4 Create interest groups. Placing your contacts in different groups based on their interests or preferences to market more effectively.

5 Avoid spam and trash folders. To prevent triggering spam filters, avoid using ALL CAPS or multiple exclamation marks in the subject line or body of your email.

6 Put a name to your email. Use your personal name or the name of your business as the From or Sender name.

7 Give 'em reading options. Send both HTML and plain text versions of your newsletter so all your contacts can read it on their computer or mobile device.

8 Keep building your contact list. Add calls-to-action and sign-up forms to your website and social media profiles, and collect contact information in person at events or conferences.

Module-48

Remember those goals you set for your website? Sure, you'll know as you get closer to meeting them. But you won't really know how well your website is working for your business — and how to adjust your online marketing strategy — without help from web analytics. As a small business owner, you need to know how many people are visiting your site, how they found it, and if your online marketing efforts are working. Web analytics products can give you all of those answers, and more.

WEB ANALYTICS: THE MEASUREMENT, COLLECTION, ANALYSIS AND REPORTING OF INTERNET DATA FOR THE PURPOSES OF UNDERSTANDING AND OPTIMIZING WEB USAGE.

There are a number of products that can analyze your site to tell you what's working and what's not. Google Analytics is a good place to start

VISITORS: Learn about your website visitors, such as whether they are new or returning, how they found you, their location, browser used etc.

UNIQUE VISITOR TRAFFIC: How many individual visitors are coming to your site in a predefined timeframe. An upward trend indicates you're providing content that is valuable to your target audience and shows your marketing campaigns are successful.

PAGES: Gather information on specific pages so you can analyze the effectiveness or popularity of each page on your site.

REFERRERS Find out which websites, URLs, search engines and keywords lead visitors to your site.

ECOMMERCE Get specifics about revenue-boosting and lead-generating visitor activities, such as purchasing products and signing up for newsletters.

LANDING PAGE CONVERSIONS: A standalone web page, created specifically for a marketing or advertising campaign. It's where a visitor "lands" after they click on a link in an email, or Ads. It's important to keep eyes on all types of conversions in your marketing funnel (visitor to lead, lead to customer, and visitor to customer) to ensure that you're avoiding any roadblocks or bottlenecks that can keep them from converting.

WHAT'S NEXT?

You've got a website and social profiles in place, plus the know-how to drive traffic to your site, engage with followers, and generate leads — so you're in a great position to grow your business. What now?

1. STAY SOCIAL: Tune your social media strategy as you gain some experience engaging with followers. What types of content get most positive response? When engagement rates are highest? Is it time to try some paid options? Which social networks seem to be working best for your business? Make smart use of your time, focus on deeper engagement on the sites that matter most to your audience.

2. GET MOBILE: If it isn't easy for your customers to navigate through your website on their phones and tablets, fix it. According to Kissmetrics, 73% of mobile Internet users say that they've encountered a website that was too slow to load, and 51% have tried to navigate through mobile sites that crashed, froze, or spit back error messages. As more and more consumers turn to their mobile devices to browse and buy, it's important to keep your site off that undesired mobile performance list.

3. HONE YOUR FOCUS: Channel your time and energy into online efforts that prove most effective for your business — email marketing, a particular social media platform, a call-to-action on your home page, whatever works best for you. After you get some idea about the online game, take a step back and evaluate your progress.

4. KEEP YOUR WEBSITE IN SHAPE: Updating and maintaining your website's content and functionality is key to staying current and delivering what your visitors want and need. Plus, continually updating your site with relevant content will work wonders at improving your search engine visibility. Keep it fresh with consistent blog posts, up-to-date photos and videos, and design tweaks that reflect evolving trends.

CHECKLIST: 10-POINT WEBSITE REVIEW

Wondering if you have an effective small business website? Your site should be more than a **glorified business card**. It should entice visitors to learn more about your products and services, establish that you're a credible business, and turn your visitors into customers. Even a website with just the bare essentials should help your business grow.

Answer the questionnaire to see if you've covered the basics:

All done? Great! Now tally up the number of Y's you circled to get your diagnosis.

0 - 4 Sounds like you're just getting started. You need help to take the next step.

5 - 7 You’re on track. Pick one area to focus on this week. Make it happen.
8 -10 You’re a guru. It might be time to take the next step. Email marketing?

Module-49

Since the dawn of the internet, links have been the way to get from one place to another online. Think about it, you either start by searching for something and then clicking a link, or by typing a link directly into your browser's address bar. There's no other way to get around

Link shorteners were originally created to address stubborn email systems that wrapped an email after 80 characters and broke any long URLs that might have been in the message. Once Twitter (and other social media) took off and introduced the 140-character limit, that shortened link became even more important.

It wasn't long before link shorteners quickly became more than mere link shorteners. They began to allow publishers to track the links they posted with analytics. They keep URLs that are loaded with **UTM tracking tags** from looking ugly by hiding the length and characters in the UTM tracking system.

A URL shortening service is a third-party website that converts that long URL to a short, case-sensitive alphanumeric code. Simply put, this means that a URL shortening service takes ridiculously long URLs (web addresses) and makes them short. Technically speaking, a URL is created manually or automatically by your web application, so when people connect to your website the server will know which page to direct the user. As a key part of your web presence, there are several good reasons for using URL shortening.

Link Masking

Proper link masking might be where you take a URL from a strong piece of content that you are looking to share, and simplify it to portray a key point in your social message. Other good reasons for masking a link might include:

- Hiding, "beautifying," or branding an ugly affiliate link.
- Shortening a lengthy domain such as: "**professionalbusinessinformation.com**" to something simpler like "**probizinfo.link**"
- Tracking – How many people click on this one specific link. This is a digital marketing best practice when sharing links online.

Obviously shortening a URL allows you to mask the original web address. This allows for spammers and hackers to hide malicious links from us. Thankfully, with security protection features from Chrome and other browsers we no longer have to worry about malicious link masking.

Link Shortening

It's much simpler to share a short and memorable URL than a lengthy one, especially when those lengthy URLs contain random numbers in them.

- Which URL are you going to shout to a room full of people while giving a talk? The short one, or the long one?
- Which URL would you link to in your presentation?
- Which URL would you tell people at a networking event w/o business cards?
- Which URL do you want to put into a forum when sharing some new ideas?

Link Tracking: Link tracking has got to be the number one reason to shorten a link. We, as digital marketers and social media managers, need to know the fruits of our labors. And we probably have to do monthly reports on our efforts too. If we can't prove our clicks – and hopefully conversions – then we have very little job security..

Link Retargeting: Have you heard of retargeting? It's why you get hit with Amazon ads right after viewing a product on its website. Retargeting is essentially the act of adding a little piece of code, to your site, so that when people visit, you can serve ads to them later.

Link Rotating: Ever wanted to run an A/B test but weren't sure how to do it? With advanced linking features you can actually split traffic 50/50 – or 75/25, or whatever – from within the link itself and thus perform a simple landing page test. Now this isn't necessary for advertising, as you can just run two ads, and there are other tools, like Optimizely, that let you handle this on the page instead of within the link, but there are some opportunities for testing split traffic for your links.

Link Swapping or Changing: You create a shortened link and share it across 12 different networks, then one day down the road, the page your link points to is taken down. Good luck finding and changing all those old links. With Rebrandly, you own your links. If you want to change out the destination URL, you can. It takes just a couple of seconds and those 12 links you shared all those months ago will automatically update. Crisis averted.

When posting a link across multiple sites on the web. As discussed earlier, it's a lot easier to track and change one link in one place, rather than hunt down all versions of it posted across your marketing channels and make the necessary changes. And that's assuming you can even make those changes.

You should use short links in all social profile bio's – for the same reason.
You should not use a URL shortener:

When linking from one page of your own site to another. Especially if you are using anchor text – which you should almost always do. There's no need for additional tracking here, as you should be able to see your results clearly, say if you are using Google Analytics.

When sending an email or chat message to friends. There's no need to shorten your link. You can embed the link in anchor text or just paste it right in there. Your friend trusts you so there is no need to worry about them not clicking due to a funky looking or lengthy URL.

Module-50

1. My Links
2. Analytics
3. Tools
4. Admin

TOPIC-4: How databases work on web?

Module-51

What's important in a hosting provider?

"Great hosting boils down to the 3 S's: speed, support and security," Scalability is also critical. "You need the ability to rapidly scale your website as your target audience grows and the **resiliency** to handle sudden bursts of high traffic,".

Hosting services are available in a wide range of monthly hosting prices ranging from a few hundred rupees to several thousand. If you're a small business getting started, you can probably do quite well with a cloud, virtual private server, or managed service.

https://www.cnet.com/how-to/how-to-choose-a-web-hosting-provider/

Decide how much hand-holding you'll need. Basic customer service provides access to email, ticket and phone support. Turnaround time on requests, however, will vary. Some service providers even offer 24-hour phone support. The limiting factor to non-managed service is that while a vendor may answer questions about basic configuration, it won't be your systems manager.

If you want to delegate the management of your site completely, then you want to consider managed service. Providers of managed service will make sure your system is configured properly for your load, keep an eye on security issues, patch your software as needed and manage backups among other tasks.

Estimate the amount of traffic you expect (and be honest). Hosting providers generally charge based on storage and bandwidth usage. Bandwidth is a measure of how many bytes you serve over a given period. If you expect only a few visitors to your site, bandwidth will be low. But if you're suddenly featured at the top of Google or your product goes viral or your advertisement campaign takes-off you can expect bandwidth requirements to jump.

As long as you're honest with yourself, there's not much of a risk. For example, if you plan to only serve a few pages to a few local customers, you'll never cross any limits. But if you know that you're really building a site that will stress low-end shared servers, be sure to pick a dedicated or cloud-based server.

Understand server types. Cheapest hosting is available on **1)** shared servers, where one box may run hundreds of websites. The performance of your site depends on the load that all the other sites are putting on the server. Shared hosting also limits your access to the server's capabilities, generally limiting you to uploading files via FTP or SFTP, preventing shell access,

restricting what programs you can run on the server and limiting the amount of database access your site can perform.

The next tier up is **2)** VPS (for virtual private server), which is a full instance of a virtual machine (a simulated computer) running on a box. Usually, hosting providers run many VPS instances on one box, but performance is almost always better than base-level shared services. If you use a VPS, you should be familiar with basic server maintenance and management.

3) Cloud servers may be a better choice. They usually run on the giant public clouds, like AWS or Microsoft Azure. Service providers can build whatever configuration suits the needs of their customers. The big benefit of cloud servers is that you can scale seamlessly. If you need to be able to handle that big traffic surge, just pay your provider more money. Nothing needs to be moved or rebuilt.

If you don't want to share, consider a **4)** dedicated server, a physical box that's rented to you. It's the same as having a server sitting behind your desk, except it's located in a service provider's data center. System management skills needed.

Be wary of unlimited offers. Some hosting providers offer so-called unlimited storage and bandwidth for little additional monthly charges. This deal often isn't what it seems to be. If you pay a little a month for hosting, there will likely be something in your terms of service allowing your hosting provider to either throttle your performance or shut you down after a certain usage level.

Choose a portable content management system to avoid lock-in. Most hosts are pretty good, but times change. Management changes, acquisitions and technology shifts can alter your web hosting plans. Make sure your site isn't locked to any one host -- and that you have a backup practice in place.

Make sure you use an open source content management system. Many people use WordPress on top of PHP, which will run on just about anything. Do regular updates and site backups, so you always have access to your site's data, media and structure. This approach means all you need to do is load your backup on another provider's service and point your domain name to that provider.

Own your domain name. All fledgling businesses own their domains. Make sure you own the domain. That way you can change providers if needed, and own any earned SEO benefits

E-commerce requires certain level of security for transmissions, such as Secure Sockets Layer (SSL), secure HTTP (S-HTTP), and Secure Electronic Transfer (SET): they must be implemented within the user's browser as well as within the Web server software at the ISP. Often, an ISP

that specializes in e-commerce will also offer additional services such as: the ability to accept credit cards for transactions. The ISP (or a business partner) allows you to use its merchant credit card accounts to accept payment for your goods and services.

You can hire designers to create a Web site for you; you may hire the same people or different ones to maintain it over time. Some ISPs offer Web design as part of their add-on services. ISPs and DSPs may also offer database development services to enable you to take advantage of their Internet database offerings.

ISPs may price off—site backups at an additional fee, and some ISPs provide highly redundant hardware (often in widely separated locations) to help minimize the danger of power outages. ISPs providing services to the largest companies charge rates far higher, but they are basically offering an uninterruptible Internet connection.

Cybercasting requires large bandwidth. There are companies that specialize in this technology. If your organization needs such services, you can use one of those companies—-or use an ISP that provides such services. This is often a cost-effective alternative to providing high-speed bandwidth to your site for relatively infrequent events.

Module-52

At various times, "database" can refer to any, several, or all of the following:

4. A database may be a body of data—often a large body of information—usually stored and indexed on a computer or other electronic device.
5. A database may be the software (and sometimes hardware) that is used to store, retrieve, and manipulate data.
6. A database may be a combination of data, software, and custom procedures that address a given problem. CRM consisting of customer records (data); a database management system (software) such as Access, DB2, or Oracle; and custom layouts, reports, and procedures that run operational systems.

In this course, above concepts are expressed with three different terms:

4. Database is used in the first sense: a collection of data.
5. Database management system (DBMS) is the term used for the second sense.
6. Database project used for the final sense the combination of data, software, and the reports, layouts, and procedures that make everything work together for a given purpose.

Further distinction among these three concepts:

1. Databases are owned or created by users.
2. DBMS created by software developers—Microsoft, Oracle, IBM etc.
3. Database projects are created by architects, end-users, or others familiar with both the data in the database and the capabilities of the DBMS.
4. **Databases handle large amount of structured data:** Individual values are not databases—your name is a value, not a database, and its relationship to your address is not a database. A collection of names, however, can be a database. It makes sense to think of a database of the children in a class (or a school) or the aisle in which you have parked your car at the shopping mall is scarcely a database, but the collection of license plate tags and aisle numbers that a valet parking clerk maintains does sound like a database.

5. **Databases can change quickly (often unpredictably):** It is possible to present the data from the database in a non-database form, such as a printed list. In the case of the children in a class, a class list could be—and normally is—produced once a semester. The relatively few enrollments and dropouts can be marked in pencil on the printed list. In the case of a database of parked cars, however, the situation is likely to be impossible to handle with a printed list. Each time a car is parked or removed, the list needs to be updated. The line of patrons waiting to park or retrieve their cars would quickly grow long if a new list had to be prepared each time.

6. **Database data needs to be selected and displayed in different ways (sometimes unpredictable):** Whether or not data changes, databases allow it to be selected and displayed in various ways. Whereas a printed list of the students in a class will always be presented in one way alphabetically by last name, in rows and columns corresponding to classroom seats, by birth date, or by address—the fact that databases can present data on an ad-hoc basis means that the way in which data is stored has little to do with the way in which it is retrieved.

Databases have tools to manipulate their data: DBMS are designed to perform the manipulations needed in storing, retrieving, updating, selecting, and displaying data. They are usually highly optimized for these purposes. Data stored in standard computer files ("flat files") must be manipulated by custom-written software that manipulates the data directly. This is usually less efficient and much more expensive.

Databases contain meta data: DBMS store meta-data in addition to data. Meta-data (data about data) describes the data. Aspects of meta-data that are stored include a name for each data field ("birth date," "customer name," etc). The name of the data field is distinct from the values in the database ("Ismail," "Siddiqi," "Hadi," etc.).

Perform data validation and integrity: DBMS contain routines that can be invoked to edit data as it is entered. Some typical types of validation and integrity features:

- Values must be unique (for example, ID numbers).
- Values must correspond to other databases (if you enter an invoice, the account number must be valid).
- Values can be generated automatically (such as serial numbers).

Data often shared across time-space: If you consider the facts that data can change, and that the data can be summarized and displayed in various ways, it is obvious that the database can appear—and be—different at various times. In fact, one way of thinking of a database is not as

a static body of data but as a static structure for the storage, retrieval, and manipulation of a body of dynamic data.

In general, the more similar the data instances are, the more efficient a database can be. If you know that the database will be used to store names and addresses of students rather than of clients, you can often construct a more efficient database.

The following terms are used to describe the data in a database:

A **record (row)** is a given data instance—0ne student, client, inventory record, appointment, message, etc. Sometimes used synonymously are instance, observation, row, and case. Each record within a database has the same structure as every other record.

A **field (column)** is a single piece of data within each record the date of an appointment, the time of an appointment, the location of an appointment, etc. Sometimes used synonymously are data point, column, and variable. All fields are present in each record of a database, but they need not have data within them.

A **key** is a field that is used to retrieve data. Often keys are unique identification numbers, for example); it is sometimes necessary to construct such a field when the real-world data may not be unique (people names).

Databases can be normalized: There are many theories and techniques that have been developed to make information storage and retrieval more efficient. Some specific methods have been developed and are general termed normalization. Your database does not have to be normalized, many are not. However, normalized databases function more efficiently than others for certain applications.

Module-53

The World Wide Web (WWW), commonly known as the Web, is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs, such as https://www.example.com/), which may be interlinked by hypertext, and are accessible over the Internet. The resources of the WWW are transferred via the Hypertext Transfer Protocol (HTTP) and may be accessed by users by a software application called a web browser and are published by a software application called a web server.

British scientist Tim Berners-Lee invented the World Wide Web in 1989. He wrote the first web browser in 1990 while employed at CERN near Geneva, Switzerland. The browser was released outside CERN in 1991, first to other research institutions starting in January 1991 and then to the general public in August 1991. The World Wide Web has been central to the development of the Information Age and is the primary tool billions of people use to interact on the Internet.

1. The Web is distributed. One of the driving factors in the proliferation of the Web is the freedom from a centralized authority. Since Web is the product of many individuals, the lack of central control presents many challenges for reasoning with the information it presents. First, different communities will use different vocabularies, resulting in problems of synonymy (when two different words have the same meaning) and polysemy (when the same word is used with different meanings). Second, the lack of editorial review or quality control means that each Web page's reliability must be questioned. An intelligent Web agent simply cannot assume that all of the information it gathers is correct and consistent.

There have been quite a number of well-known “Web hoaxes" in which information was published on the Web with the intent to amuse or mislead. Furthermore, since there can be no global enforcement of integrity constraints on the Web, information from different sources may be in conflict. Some of these conflicts may be due to philosophical disagreement; different political groups, religious groups, or nationalities may have fundamental differences in opinion that will never be resolved. Any attempt to prevent such inconsistencies must favor one opinion, but the correctness of the opinion is very much in the eye of the beholder.

2. The Web is dynamic. The Web changes at an incredible pace, much faster than a user or even a softbot can keep up with. While new pages are being added, the content of existing pages is changing. Some pages are fairly static, others change on a regular basis, and still others change at unpredictable intervals. These changes may vary in significance: although the addition of punctuation, correction of spelling errors, or reordering of a paragraph does not affect the semantic content of document, other changes may completely alter meaning or even remove large amounts of data. We must assume that Web data can and often will be outdated.

The rapid pace of information change on the Internet poses an additional challenge to any attempt to create standard vocabularies and provide formal semantics. As understanding of a given domain changes, both the vocabulary may change and the semantics may be refined. it is important that such changes do not adversely alter the meaning of existing content.

3. The Web is massive. Recent estimates place the number of indexed Web pages at over 5.7 billion. Even if each page contained only a single piece of agent-gatherable knowledge, the cumulative database would be large enough to bring most reasoning systems to their knees. To scale to the size of the ever-growing Web, we may use incomplete reasoning algorithms.

4. Web is open. Open Web is about the ability to openly do three kinds things:

1. **publish** content and applications on the web in open standards
2. **code and implement** the web standards that content/apps depend on
3. **access and use** content / code / web-apps / implementations

1. This ability to openly publish depends on a few things:

open formats for freely publishing what you write, photograph, video and otherwise create, author, or code (e.g. HTML, CSS, Javascript, JPEG, PNG, Ogg, WebM etc.). Domain name registrars and web hosting services that, like phone companies, don't judge your content.

2. Code and implement

- openly documented, in this day and age, on the web itself.
- freely accessible. there should be no charge to view a web standards specification.
- unencumbered by patents, either patent-free, or available under unconditional royalty free licenses.

3. Access to content and applications

This open access depends on the open ability to browse and use any web page or application (i.e. URL) on your:

- web browsing device
- internet service

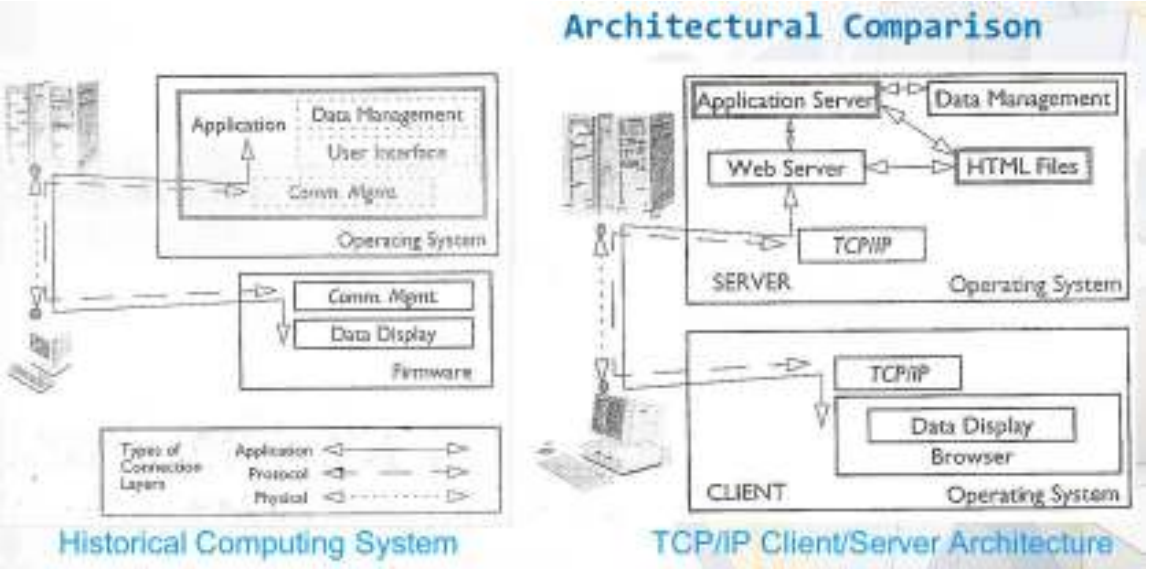
https://tantek.com/2010/281/b1/what-is-the-open-web

Module-54

To understand how databases work on the Web, it is helpful to briefly look back at the evolution of contemporary software architecture. This path led from the earliest days of mainframe computers and dumb terminals (originally teletype machines) to the modern architecture that may incorporate mainframes, personal computers, and a variety of networking technologies. This module examines the major steps along this road; it then deals with the contemporary design of systems.

This history is an oversimplification. Furthermore, it has the benefit of hind- sight: at the time, it was not always clear where the road would lead.

5. Mainframes and dumb terminals (from the earliest years of the computer age–the **1950s**)
6. The rise of operating systems and structured programming (starting in the **1960s**)
7. The development of the personal computer (in the late **1970s**)
8. The growth of the Internet and the World Wide Web (early **1990s**)



Module-55

When you chat to somebody on the Net or send them an e-mail, do you ever stop to think how many different computers you are using in the process? There's the computer on your own desk (or in your palm), of course, and another one at the other end where the other person is present, ready to communicate with you. But in between your two machines, making communication between them possible, there are probably about a dozen other computers bridging the gap. Collectively, all the world's linked-up computers are called **the Internet** i.e. network of networks. How do they talk to one another? Let's take a closer look!

Global communication is easy now thanks to an intricately linked worldwide computer network that we call the Internet. In less than 20 years, the Internet has expanded to link up around 210 different nations. Even some of the world's poorest developing nations are now connected.

Lots of people use the word "Internet" to mean going online. Actually, the "Internet" is nothing more than the basic computer network. Think of it like the telephone network or the network of highways that criss-cross the world. Telephones and highways form networks, just like the Internet. The things you say on the telephone and the traffic that travels down roads run on "top" of the basic network. In much the same way, things like the World Wide Web (the information pages we can browse online), instant messaging chat programs, MP3 music downloading, and file sharing are all things that run on top of the basic computer network that we call the Internet.

The Internet is a collection of standalone computers (and computer networks in companies, schools, and colleges) all loosely linked together, mostly using the telecommunication network. The connections between the computers are a mixture of old-fashioned copper cables, fiber-optic cables (which send messages in pulses of light), wireless radio connections (which transmit information by radio. waves), and satellite links.

Chart: Internet use around the world: This chart compares the estimated percentage of households with Internet access for different world regions and economic groupings. Although there have been dramatic improvements in all regions, there are still great disparities between the "richer" nations and the "poorer" ones.

The world average, shown by the black-outlined orange center bar, is still only 46.4 out of 100 (less than half). Not surprisingly, richer nations are to the left of the average and poorer ones to the right. Source: Redrawn from Chart 1.5 of the Executive Summary of Measuring the Information Society 2015, International Telecommunication Union (ITU).

What does the Internet do?

The Internet has one very simple job: to move digitized content (known as **data**) from one place to another. That's it! The machines that make up the Internet treat all the content they handle in exactly the same way. In this respect, the Internet works a bit like the postal service. Letters are simply passed from one place to another, no matter who they are from or what messages they contain. The job of the mail service is to move letters from place to place, not to worry about why people are writing letters in the first place; the same applies to the Internet.

Just like the mail service, the Internet's simplicity means it can handle many different kinds of content helping people to do many different jobs. It's not specialized to handle emails, Web pages, chat messages, videos or anything else: all content is handled equally and passed on in exactly the same way. Because the Internet is so simply designed, people can easily use it to run new "applications"—new things that run on top of the basic computer network. That's why, when two European inventors developed Skype, a way of making audio calls over the Net, they just had to write a program that could turn speech into digitized data and back to speech again. No-one had to rebuild the entire Internet to make Skype possible.

Module-56

Circuit switching

Much of the Internet runs on the telecommunications network—but there's a big difference between how a telephone call works and how the Internet carries data. If you ring a friend, your telephone opens a direct connection (or circuit) between your home and theirs. If you had a big map of the worldwide telephone system (and it would be a *really* big map!), you could theoretically mark a direct line, running along hundreds of miles of cable, all the way from your phone to the phone in your friend's house. For as long as you're on the phone, that circuit stays permanently open between your two phones. This way of linking phones together is called **circuit switching**. In the old days, when you made a call, someone sitting at a "switchboard" (literally, a board made of wood with wires and sockets all over it) pulled wires in and out to make temporary circuits that connected one home to another. Now the circuit switching is done automatically by an electronic telephone exchange.

If you think about it, circuit switching is a really inefficient way to use a network. All the time you're connected to your friend's house, no-one else can get through to either of you by phone. (Imagine being on your computer, typing an email for an hour or more—and no-one being able to email you while you were doing so.) Suppose you talk very slowly on the phone, leave long gaps of silence, or go off to make a cup of coffee. Even though you're not actually sending information down the line, the circuit is still connected—and still blocking other people from using it.

Packet switching

The Internet could, theoretically, work by circuit switching—and some parts of it still do. If you have a traditional "dialup" connection to the Net (where your computer dials a telephone number to reach ISP in what's effectively an ordinary phone call), you're using circuit switching to go online. You'll know how maddeningly inefficient this can be. No-one can phone you while you're online; you'll be billed for every second you stay on the Net; and your Net connection will work relatively slowly.

Most data moves over the Internet in a completely different way called **packet switching**. Suppose you send an email to someone in China. Instead of opening up a long and complex circuit between your home and China and sending your email down it all in one go, the email is broken up into tiny pieces called **packets**. Each one is tagged with its ultimate destination and allowed to travel separately. In theory, all the packets could travel by totally different routes. When they reach their ultimate destination, they are reassembled to make an email again.

Packet switching is much more efficient than circuit switching. You don't have to have a permanent connection between the two places that are communicating, for a start, so you're not blocking an entire chunk of the network each time you send a message. Many people can use the network at the same time and since the packets can flow by many different routes, depending on which ones are quietest or busiest, the whole network is used more evenly—which makes for quicker, cheaper and more efficient communication all round.

There are hundreds of millions of computers on the Net, but they don't all do exactly the same thing. Some of them are like electronic filing cabinets that simply store information and pass it on when requested. These high performance machines are called **servers**. Machines that hold ordinary documents are called file servers; ones that hold people's mail are called mail servers; and the ones that hold Web pages are Web servers. There are tens of millions of servers on the Internet.

A computer that gets information from a server is called a **client**. When your computer connects over the Internet to a mail server through your ISP so you can read your messages, your computer is the client and the ISP computer is the server. There are far more clients on the Internet than servers—probably billion by now!

When two computers on the Internet swap information back and forth on a more-or-less equal basis, they are known as peers. If you use an instant messaging program to chat to a friend, and you start swapping party photos back and forth, you're taking part in what's called **peer-to-peer** (**P2P**) communication. In P2P, the machines involved sometimes act as clients and sometimes as servers.

Internet is also made up of intermediate computers called **routers**, whose job is really just to make connections between different systems. If you have several computers at home or school, you probably have a single router that connects them all to the Internet. The router is like the mailbox on the end of your street: it's your single point of entry to the worldwide network.

Internet as a postal service was just an analogy, as of 2019, there are about 294 billion emails sent and received each day. If everything is sent by packet-sharing, and no-one really controls it, how does that vast mass of data ever reach its destination without getting lost?

The answer is called TCP/IP, which stands for Transmission Control Protocol/Internet Protocol. It's the Internet's fundamental "control system" and it's really two systems in one. A "protocol" is simply a standard way of doing things—a tried and trusted method that everybody follows to ensure things get done properly.

Internet Protocol (IP) is simply the Internet's addressing system. All machines on the Internet—yours, mine, and everyone else's—are identified by an Internet Protocol (IP) address such as

such as 12.34.56.78 If all the machines have numeric addresses, every machine knows exactly how (and where) to contact every other machine. When it comes to websites, we usually refer to them by easy-to-remember names (like www.data2bi.biz) rather than their actual IP addresses—and there's a relatively simple system called **DNS (Domain Name System)** that enables a computer to look up the IP address for any given website.

The other part **Transmission Control Protocol (TCP)**, sorts out how packets of data move back and forth between one computer (in other words, one IP address) and another. It's TCP that figures out how to get the data from the source to the destination, arranging for it to be broken into packets, transmitted, resent if they get lost, and reassembled into the correct order at the other end.

Module-57

For most people on the go, Internet access involves using a mobile network and for those at home/office Internet access involves connecting to a hub/router which is connected to a coaxial cable, phone line or optical fiber.

You request a Web page by typing in an address— such as http:/ /www.data2BI.biz/index.html. Your ISP (internet service provider) connects to the ISP that serves that domain. Conceptually that ISP is a domain. Within a domain, there may be many different computers. The lnternet routing tables are used to map these names to specific Internet addresses.

What your browser actually receives is a string of text characters, the text on the Web page as well as the HTML formatting information and more.

The architecture described here has two different types of connections. The connection between the user and the lSP, often over a telephone line or optical cable etc. using modems – is a session.

The request for a Web page is a transaction: a request from the user sent to appropriate domain, and a response provided in the form of a Web page.

A site is a registered location on the Internet (or on an intranet). A domain name server resolves the domain name into an IP address, and in-between sits an ISP.

There are several ways or methods of connecting to the Internet. There are two access methods **direct and Indirect** and these can be either **fixed or mobile**.

Indirect Access

This is most common method used in home and office networks.

The device e.g. computer connects to a network using **Ethernet or WiFi** and the network connects to the Internet using ADSL, cable or fibre.

Direct Access

This is most common method used when traveling.

The device e.g. smart phone connects directly to the Internet using 3G/4G mobile networks or public Wi-Fi.

Fixed Internet- Home/Office

Fixed access is usually much faster and reliable than mobile, and is used for connecting homes/offices. The main Access mechanisms are i) ADSL over traditional Phone Lines (most common) ii) Cable (limited to cable TV areas) iii) Fiber broadband

Pros	Cons
Very Fast and reliable	Requires a fixed connection
Good for streaming video	Not usable when at a remote location
Cheap when compared to Mobile	Most common for businesses and home
Can easily share the connection	

Domain Name Servers (DNS) are the Internet's equivalent of a phone book. They maintain a directory of domain names and translate them to Internet Protocol (IP) addresses.

This is necessary because, although domain names are easy for people to remember, computers or machines, access websites based on IP addresses. lnternet routing tables are used to map these names to specific Internet addresses, but none of that is the responsibility either of the user or of the content provider.

Information from all the domain name servers across the Internet are gathered together and housed at the Central Registry. Host companies and Internet Service Providers interact with the Central Registry on a regular schedule to get updated DNS information.

When you type in a web address, e.g., www.data2BI.biz, your Internet Service Provider views the DNS associated with the domain name, translates it into a machine friendly IP address (for example 96.44.146.234 is the IP for data2BI.biz) and can direct your Internet connection to the correct website.

ln order to handle large number of users, some lnternet sites use mirrors duplicate computers with duplicate data; they route users www1, www2, www3, or whatever their mirror computers are called so that the load is spread evenly.

Contents:

What your browser actually receives is a string of text characters- the text on the Web page as well as the HTML formatting information. There may be non-textual elements on the page-—images, sound, or even executable code (such as Java applets)—but they are wrapped in the stream of text characters.

Interpreting:

In many cases, Web pages are simply text files: on request, the file is read by the lSP's Web software, and it is transmitted character by character to the receiving ISP and then to the user’s

browser. This need not be the case, however. Remember that what happens within the domain to which you connect is its business: all that is required is for the ISP to return that string of text characters (with possibly some embedded images, sound, or applets) to the user.

Generating:

So a computer client program i.e. a web browser can generate a Web page on request rather than reading it from a file. Of course, this means that the computer needs the browser to format the Web page. But for dynamically created Web pages often combine template or format information from a file with dynamic information from a database and that is performed at the server end.

The architecture described here has two types of connections. The connection between the user and the ISP—over a telephone line or using smartphones is a **session**. It is a relationship between two individual devices, and it is usually billed to a single account. The session is not fixed, neither the type of activity.

The request for a Web page as described is a **transaction**: a request from the user is sent to the appropriate domain, and a response is provided in the form of a Web page consisting of text and possibly embedded nontextual information. The transaction is not designed to be lengthy, and it consists only of the request and the response. (If no response is received, the user's browser presents an error message.) There is normally no relationship between any transaction and any other.

Using a shared resource such as a database is not like a session: you often need a password to access a database, and you may initiate many queries. In fact, the normal implementation of database access on the Web is transactional, not session based. Each request contains all of the information needed to let the database accomplish its work. In order to do this, many hidden fields of data are sent back and forth both in the requests and in the responses.

State-less systems do not keep track of information about users between requests. The system i.e. the web server do not keep track of the state of any individual user, and each transaction or request is completely self-contained.

Module-58

Experienced computer users almost all would offer the same advice to newcomers: do not let yourself be locked into a particular vendor's products. It is your data, your Web site, and your development cost. You have every reason to expect that those investments are yours and that you can reorganize your database and Web assets just as you would the chairs in your office.

When you are evaluating a new product or service, you will often be shown how easy it is to convert your existing data or Web site to the new product. Keep your eyes open: ask how easy it will be to convert your data or Web site from that new product to yet another new product (possibly from another vendor). Nothing is forever-— particularly in the world of computers.

The people who have had the most success in developing database-driven Web sites have often changed many, if not all of the components of their sites several times before settling on a particular combination that works for them.

Your first step is to find an internet service provider (ISP). Your ISP together with a telephone company (sometimes a cable company or wireless company) provides you with your physical connection to the internet.

It is very common to select a second company to host your Web site and to provide your database services. Such a hosting company is a database service provider (DSP).

The heart of your database-driven Web site is your database software that comes in three flavors.

Web Server, Application Server and Database Server

In some cases, however, may need more: scripts and commands that interact with other applications on your 'Web server or other computers. CGi (common gateway interface) scripts are a standard means of implementing such "glue" routines.

Often you are not starting from scratch. This can pose yet another constraint in the design and implementation of your database Web application and in the tools that you use resulting in transitions.

https://geekflare.com/mysql-hosting-platform/

The web+database hosting combo is a preferred option for new or low traffic websites since the combo frees the system administrator from all the hassles of managing diverse services.

But when data management becomes critical in high volume applications or websites, it could make sense to decouple both services and keep a dedicated hosting just for the database.

DBaaS (database as a service) is also a preferred choice if you are setting up the data layer of an application before knowing how you are going to access that data.

The management tools you get with your databases is another important decision factor. DB hosting providers usually offer a management front-end that is usually friendly and easy to use. But it is equally important that you can connect to the database via API calls or remote tools that give you the freedom to access and manage your data at your will.

Database software comes in many varieties; the major categories are:

- Software designed originally for desktop applications. This includes products such as Microsoft Access, File-Maker Pro etc. Over the years, these products have been expanded and strengthened to be able to support network and multiuser configurations.
- In the middle, object-oriented databases store data models created by object-oriented programming language. Programmers can create, modify, and store object datasets within these databases, such as Visual FoxPro, dBASE Plus etc.
- At the high end, enterprise database products such as DB2, Oracle, Informix, SQL Server, Sybase and Teradata have been the workhorses of databases for years. These are equipped with interfaces to application servers, and some have application server functionality built into their suite of services.
- Beyond the high end, new concepts such as data warehousing are being developed that tie together hetrogenous databases (often using a variety of products) into very large repositories of data.
- Firstly - "Server" can refer to a physical thing (a computer), or a logical thing (a piece of software). Web, application and database server software can all run on the same physical server machine, or be running across multiple physical machines.
- The **Web server** deals with HTTP(S) requests, and passes these requests on to "handlers". They have built-in handlers for file requests - HTML pages, images, CSS, JavaScript etc. You can add additional handlers for requests that they cannot manage - e.g. dynamic pages delivered by the application server. Web servers implement the HTTP specification, and know how to manage request and response headers.
- The **application server** handles requests which create dynamic pages. So instead of serving an HTML page that is stored on the hard drive, they dynamically generate the

HTML and send to the end user. Common languages/frameworks for this are Java/JSP, .Net (aspx), PHP, Ruby (on Rails or not), Python etc. Most of the time, this application server software is running on the same physical server machine as the web server.

- The **database server** software is where the application stores its structured data. Typically, this means custom software which allows the application server to ask questions like "how many items does user x have in their basket?", using a programming language. Examples are MySQL, SQL Server, Oracle (all "relational databases"), and MongoDB, Redis and CouchDB ("NoSQL" non-relational database solutions).

https://stackoverflow.com/questions/13042840/difference-between-web-server-application-server-and-database-server

There are transitions possible at every step of the way. You may have an environment that is based largely on CGI scripts written in Perl; converting them to a more sophisticated application server may not be feasible.

On the other hand, living with old designs can be very expensive. A Telnet-based database access system may well have worked for years or decades with dumb terminals being used to access mainframe-based systems. The people who design, develop, support, and use such systems often have a great deal invested in the systems. The prospect of using new technology and of throwing out hardware and software to which they have become accustomed is not always welcome.

The fact that database-driven Web sites are often much cheaper to design, develop, implement, support, and use can be easily obscured when people perceive that their expertise is being questioned.

The fact that this is a “people” issue (rather than a technical issue) in no way diminishes its importance. Developing a new system that no one will use is not productive for anyone. When you are replacing or augmenting an old system, remember how much people may be identified with it; they may take statements of fact as personal attacks and may question what to you may be the most obvious statements.

TOPIC-5: Database driven website

Module-59

This module covers the basics of choosing and working with your Internet service provider and database service provider. These may be commercial services, in-house services within your organization, or services that you must provide for yourself and for others. The combination of services that you need may be provided by one organization or by several; one computer or several may be used. This module and the next one deals with the basics of how to connect a computer with server software to the Internet (or your intranet); it proceeds through various other services that may be provided and that you may or may not need.

Internet Services allows us to access huge amount of multi-media data such as text, graphics, sound and software over the internet.

There are various services available that provide exchange of data with individuals or groups. The following gives a brief introduction to these services:

1. **Electronic Mail:** Used to send electronic message over the internet.

2. **File Transfer Protocol (FTP):** Enable the users to transfer files.

3. Telnet: Used to log on to a remote computer that is attached to internet.

4. Internet Telephony (VoIP): Allows the internet users to talk across internet to any PC equipped to receive the call.

5. World Wide Web (WWW): It offers a way to access documents spread over the several servers over the internet. These documents may contain texts, graphics, audio, video, hyperlinks. The hyperlinks allow the users to navigate between the documents.

6. Instant Messaging: Offers real time chat between individuals and group of people. Eg. Yahoo messenger, MSN messenger.

You start by figuring out what you have now. If you have Internet access, you probably have personal email. If you have your own Web site, you probably have the ability to upload Web pages to your site and you may have FTP access. (If you do not, you either have FTP access provided under a proprietary name or the functional equivalent—unless you are unable to change your own Web pages.)

If you have your own domain name, such as data2BI.biz you have access to domain mail POP (post office protocol) and email addressed to you@data2BI.biz is properly delivered. If you do not yet have a database server, you need to obtain one in order to produce your database driven Web site. Your options are simple:

- Your ISP may offer database servers.
- You can find another ISP (or database service provider) to provide that service.
- As part of the previous step, it may be cost effective to reshuffle your internet services, possibly moving your Web site and domain name to the DSP (hosting service)

Whatever you do, remember that you need to construct a proper networked environment whether you purchase it as part of an all-in—one package from a vendor or construct it yourself, that is what you are looking toward.
https://www.highspeedinternet.com/resources/choosing-an-internet-service-provider

1. Find internet service providers in your area.

Not every ISP is available in every area. Coverage areas differ from provider to provider, so your choices will be limited to the providers that offer service in your area. **Prices, speeds, special offers, and package lineups also vary by location.** What you see advertised online is not necessarily what you can get. Always check the availability of a package in your area before you decide it's the one for you.

2. Compare plans, pricing, speeds, and more.

All internet providers have their own specialties, and it helps to pick one that coincides with what you need the most. Here are the biggest issues to consider:

i) Plans and pricing ii) Speeds iii) Installation and equipment costs iv) Customer satisfaction ratings v) Data caps and overage fees

Some providers deliver ultrafast speeds, while others have more straightforward plans that are easier on the wallet. Many providers impose data caps that limit how much internet you can use per month—though some of them offer unlimited data.

3. Find how much internet speed you need.

You'll need an internet plan with adequate download and upload speeds to accomplish all of your everyday Wi-Fi tasks with ease. You want internet that's fast—but you don't need it to be *too* fast necessarily, or else you'll end up paying too much for bandwidth you won't use.

If you do not have your own domain name, your Web site is addressed part of your hosting or ISP site, something like www.thishost.com/users/yoursite. If you move your site to another vendor, your address changes to something like www.anotherhost.com/customers/~yoursite. Your customers cant find you now you.

When you have your own domain name, the change from one ISP to another is done by the internet addressing agency (www.internic.net in the United States). Your site-www.mydomain.com, for example is listed in internal Internet tables where just the IP address of the host gets changed, and your website visitors who type in www.mydomain.com simply arrive at the new IP address.

When you buy a domain name, there are typically at least two contact people i.e. a technical and a billing contact. Often, an lSP will set up your domain name for you and unless instructed, they will identify themselves as both contacts. If you are not setting up your domain name yourself, make certain that you (or someone at your organization) is set up as one of the contacts.

Either of the two contacts can authorize a change in the domain's address. If your ISP is the only party who can change that address, they may not act particularly speedily to change your domain's address when you are transferring it to another ISP.

If your domain has already been established and you are not one of the contacts, ask your ISP to make the change now. You can check by looking at www.internic.net to see who the contacts are.

Some of the most common types of connections are described here and there can be combinations, which include:

- A dial-up connection with a static IP address.
- A permanent (non-switched) connection via frame relay, integrated services digital network (ISDN), or leased lines.

- Colocation in which your computer and its server software are physically located on the premises of an internet service provider.

- Database hosting in which your database runs on your ISP's (or DSP's) computer on its premises.

Dial-up connection with static IP

A static IP address is an IP Address associated with your account that never changes and can be assigned to a specific device. Every time that you connect to your ISP network the static IP address routes traffic to the computer or device that can be assigned an IP (such as a router or firewall). This allows you to host a variety of applications that can be accessed remotely. This is useful to i) Host a web, mail or FTP server ii) Access a corporate network remotely iii) Host a webcam for video streaming or iv) Use video conference applications

Module-60

https://www.hostingadvice.com/how-to/how-to-choose-a-web-host/

5 Expert Tips: How to Choose a Web Host & Top 6 Services

For developers, bloggers, small business owners, and others, learning how to choose a web host is like searching for Mr. Right. It's a familiar question: Which hosting provider should I trust with my heart (er, websites)?

If done right, you can spend a lifetime of happiness with a reliable and high-performing host who is always available through phone, chat, or email to answer your burning late-night questions. However, rushing into a hosting relationship without doing your research could lead to feeling trapped, misled, or extorted. Choosing the wrong host often ends with headaches and a messy, expensive divorce — and you again alone, holding on to all the files you used to share.

Shared hosting

In shared hosting, several customers and websites share the same server. On one hand, shared hosting is— simple and uncomplicated. Most first-time hosting customers should turn to a shared package when entering the web hosting world, then decide when it's time to upgrade to a VPS or dedicated plan to meet your increasing needs.

VPS hosting

VPS, is a middle ground between the shared hosting and the commitment of a dedicated server. The server is divided into virtual machines, which act as independent dedicated servers. VPS customers still share a server, but they each have much larger portions and greater control than those with a shared hosting plan.

Dedicated hosting

High-performing sites need dedicated hosting, which entails using an entire server to power your website or applications. As the name implies, dedicated servers are ready to wait on you hand and foot and meet your every configuration need. Customers have complete control over the doting architecture, meaning they can customize security systems, operating systems, load balancers, and more.

Cloud servers may be a better choice. They usually run on the giant public clouds, like Amazon Web Services or Microsoft Azure. Service providers can build whatever configuration suits the needs of their customers.

Blogging site: Some features, i) modern design with better content presentation ii) typography that pleases readers' eyes iii) highly navigable menus iv) Email subscription system v) multiple comment options and more.

Online store: The top hosts take care of the added security requirements associated with protecting customer and payment information while also providing access to shopping cart software, and integrations with services such as PayPal.

Online Portfolio or Résumé: For those job applicants without any technological know-how — or, let's be honest, tech-savvy folks who just don't feel like going through the motions of developing a site — website builders are the fastest way to produce a professional online presence that showcases your work.

Personal Site: Personal websites need to convey information in a visually appealing way. Hosting customers don't need to spend a lot to create a stunning site, hosts attract beginners and hobbyists by making web hosting affordable and easy to use.

Business Site: Even if you don't plan on using your website to sell products, your business is counting on the online presence to increase brand recognition. Entrepreneurs can expect their business website to grow 10 to 20% each month if all goes well, so you'll want to find a hosting provider that can handle a booming business.

4. Understand How Web Hosting Pricing Works

Free hosting

Proceed with caution. The convenience and savings are attractive, but the added features, support, and security you can gain by signing up with a reputable hosting provider are well worth the slight cost

Cheap hosting

Because hosting companies can pack thousands of hosting customers onto a shared server, providers can afford to include dozens of value-added services with hosting plans. By signing up for an affordable hosting plan, you can experiment with luxury options, such as content delivery networks, automatic backups, website builders, and eCommerce tools, to explore different avenues of online success.

5. Think Ahead: Choose a Host With Which You Can Grow

Don't be afraid to daydream about the bright, busy futures of your websites. Some of the more budget-driven web hosts concentrate solely on shared hosting, meaning you'll have to part

ways and take your chances out in the hosting dating pool when you’re ready to move forward with VPS or dedicated services.

TOPIC-6: Understanding database schemas

Module-61

https://www.lucidchart.com/pages/database-diagram/database-schema

A database schema represents the logical configuration of all or part of a relational database. It can exist both as a visual representation and as a set of formulas known as integrity constraints that govern a database. These formulas are expressed in a data definition language, such as SOL. As part of a data dictionary, a database schema indicates how the entities that make up the database relate to one another, including tables, views, stored procedures, and more.

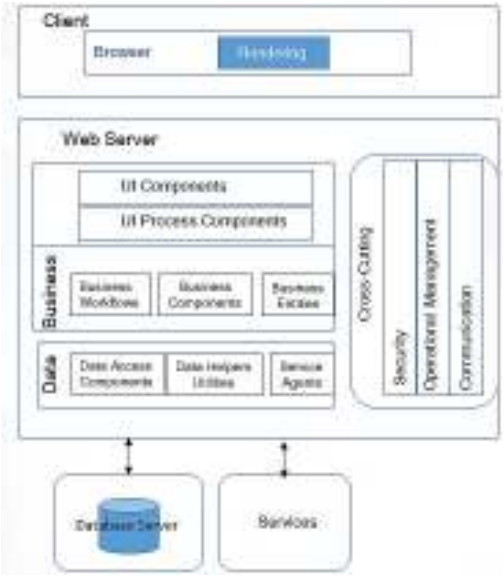


Figure: 3-Tier Architecture

Typically, a database designer creates a database schema to help programmers whose software will interact with the database. The process of creating a database schema is called data modeling. When following the three-schema approach to database design, this step would follow the creation of a conceptual schema. Conceptual schemas focus on an organization's informational needs rather than the structure of a database.

https://www.guru99.com/dbms-schemas.html

Database systems comprise of complex data structures. Thus, to make the system efficient for retrieval of data and reduce the complexity of the users, developers use the method of Data Abstraction.

There are mainly three levels of data abstraction or schemas:

- Internal Level: Actual PHYSICAL storage structure and access paths.
- Conceptual or Logical Level: Structure and constraints for the entire database
- External or View level: Describes various user views

- A physical database schema lays out how data is stored physically on a storage system in terms of files and indices.

- A logical database schema conveys the logical constraints that apply to the stored data. It may define integrity constraints, views, and tables.

- Each external view is defined using an external schema, which consists of definitions of various types of external record of that specific view.

At the most basic level, a database schema indicates which tables or relations make up the database, as well as the fields included on each table. Thus, the terms schema diagram and entity-relationship diagram are often interchangeable.

https://www.tutorialspoint.com/dbms/dbms_data_schemas.htm

Facts about Physical schema:

- The internal schema is the lowest level of data abstraction
- It helps you to keeps information about the actual representation of the entire database. Like the actual storage of the data on the disk in the form of records
- The internal view tells us what data is stored in the database and how
- It never deals with the physical devices. Instead, internal schema views a physical device as a collection of physical pages

Facts about Logical schema:

- Defines all database entities, their attributes, and their relationships
- Security and integrity information
- In the conceptual level, the data available to a user must be contained in or derivable from the physical level

Facts about external schema:

- An external level is only related to the data which is viewed by specific end users.
- This level includes some external schemas.
- External schema level is nearest to the user

- The external schema describes the segment of the database which is needed for a certain user group and hides the remaining database details from the specific user group

https://www.tutorialspoint.com/dbms/dbms_data_schemas.htm
https://www.lucidchart.com/pages/database-diagram/database-schema

Database schema Vs. Database Instance

These terms, though related, do not mean the same thing. A database schema is a sketch of a planned database. It doesn't actually have any data in it.

A database instance, on the other hand, is a snapshot of a database as it existed at a particular time. Thus, database instances can change over time, whereas a database schema is usually static, since it's difficult to change the structure of a database once it is operational.

Database schemas and database instances can affect one another through a database management system (DBMS). The DBMS makes sure that every database instance complies with the constraints imposed by the database designers in the database schema.

A modern DBMS has the following characteristics:

Real-world entity: Uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.

Relation-based tables: DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the relationships among the tables.

Isolation of data and application: A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.

Query Language: DBMS is equipped with a query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and as different filtering options as required to retrieve a set of data.

Multiple views: DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production

department. This feature enables the users to have a concentrated view of the database according to their requirements.

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows:

Administrators: Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain isolation and force security. Administrators also look after DBMS resources like system license, required tools, and other software and hardware related maintenance.

Designers: Designers are the group of people who actually work on the designing part of the database. They keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints, and views.

End Users: End users are those who actually reap the benefits of having a DBMS. End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.

Advantages Database Schema

- You can manage data independent of the physical storage
- Faster Migration to new graphical environments
- DBMS Architecture allows you to make changes on the presentation level without affecting the other two layers
- As each tier is separate, it is possible to use different sets of developers
- It is more secure as the client doesn't have direct access to the database business logic
- In case of the failure of the one-tier no data loss as you are always secure by accessing the other tier

Disadvantages Database Schema

- Complete DB Schema is a complex structure which is difficult to understand for every one
- Difficult to set up and maintain
- The physical separation of the tiers can affect the performance of the Database

Module-62

https://www.guru99.com/data-modelling-conceptual-logical.html

Data modeling (data modeling) is the process of creating a data model for the data to be stored in a Database. This data model is a conceptual representation of Data objects, the associations between different data objects and the rules. Data modeling helps in the visual representation of data and enforces business rules, regulatory compliances, and government policies on the data.

Data model emphasizes on what data is needed and how it should be organized instead of what operations need to be performed on the data. Data Model is like architect's building plan which helps to build a conceptual model and set the relationship between data items.

https://stackoverflow.com/questions/25093452/difference-between-data-model-and-database-schema-in-dbms

A schema is a blueprint of the database which specifies what fields will be present and what would be their types. For example an employee table will have an *employee_ID* column represented by a string of 10 digits and an *employee_Name* column with a string of 45 characters.

Data model is a high level design which decides what can be present in the schema. It provides a database user with a conceptual framework in which we specify the database requirements of the database user and the structure of the database to fulfill these requirements.

A data model can, for example, be a relational model where the data will be organized in tables whereas the schema for this model would be the set of attributes and their corresponding domains.

Types of Data Models

There are mainly three different types of data models:

1. **Conceptual:** This Data Model defines **WHAT** the system contains. This model is typically created by Business stakeholders and Data Architects. The purpose is to organize, scope and define business concepts and rules.

2. **Logical:** Defines **HOW** the system should be implemented regardless of the DBMS. This model is typically created by Data Architects and Business Analysts. The purpose is to developed technical map of rules and data structures.

3. **Physical**: This Data Model describes **HOW** the system will be implemented using a specific DBMS system. This model is typically created by DBA and developers. The purpose is actual implementation of the database.

1. Conceptual Model

Establishes the entities, their attributes, and their relationships. In this Data modeling level, there is hardly any detail available of the actual Database structure. The 3 basic tenants of Data Model are

- Entity: A real-world thing
- Attribute: Characteristics or properties of an entity
- Relationship: Dependency or association between two entities

Characteristics of a conceptual data model

Designed and developed for a business audience offers organization-wide coverage of the business concepts. Developed independently of hardware specifications like data storage capacity, location or software specifications like DBMS vendor and technology. The focus is to represent data as a user will see it in the "real world."

2. Logical Data Model

Adds further information to the conceptual model elements. It defines the structure of the data elements and set the relationships between them. No primary or secondary key is defined. At this Data modeling level, you need to verify and adjust the connector details that were set earlier for relationships.

Characteristics of a Logical data model

- Designed and developed independently from the DBMS.
- Data attributes will have data-types with exact precisions and length.
- Normalization processes to the model is applied typically till 3NF.

3. Physical Data Model

A Physical Data Model describes the database specific implementation of the data model. It offers an abstraction of the database and helps generate schema. This is because of the richness of meta-data offered by a Physical Data Model.

This type of Data model also helps to visualize database structure. It helps to model database columns keys, constraints, indexes, triggers, and other RDBMS features.

Characteristics of a physical data model:

- Describes data need for a single project or application though it may be integrated with other physical data models based on project scope.
- Data Model contains relationships between tables that which addresses cardinality and nullability of the relationships.
- Developed for a specific version of a DBMS, location, data storage or technology to be used in the project.
- Columns should have exact data-types, lengths assigned and default values.
- Primary and Foreign keys, views, indexes, access profiles, and authorizations, etc. are defined.

Advantages and Disadvantages of Data Model:

Advantages of Data model:

- The main goal of a designing data model is to make certain that data objects offered by the functional team are represented accurately.
- The data model should be detailed enough to be used for building the physical database.
- The information in the data model can be used for defining the relationship between tables, primary and foreign keys, and stored procedures.
- Data Model helps business to communicate the within and across organizations.
- Data model helps to documents data mappings in ETL process
- Help to recognize correct sources of data to populate the model

Advantages and Disadvantages of Data Model:

Disadvantages of Data model:

- To develop Data model one should know physical data stored characteristics.
- This is a navigational system produces complex application development, management. Thus, it requires a knowledge of the biographical truth.
- Even smaller change made in structure require modification in the entire application.
- There is no set data manipulation language in DBMS.

Module-63

https://www.guru99.com/data-modelling-conceptual-logical.html
https://www.guru99.com/dbms-data-independence.html

What is Data Independence of DBMS?

Data Independence is defined as a property of DBMS that helps you to change the Database schema at one level of a database system without requiring to change the schema at the next higher level. Data independence helps you to keep data separated from all programs that make use of it.

You can use this stored data for computing and presentation. In many systems, data independence is an essential function for components of the system.

Data Independence: If a database system is not multi-layered, then it becomes difficult to make any changes in the database system. Database systems are designed in multi-layers as we learnt earlier. Metadata itself follows a layered architecture, so that when we change data at one layer, it does not affect the data at another level. This data is independent but mapped to each other.

The two types of Data Models are

- Entity Relationship (E-R) Model
- UML (Unified Modeling Language)

https://www.tutorialspoint.com/dbms/dbms_pdf_version.htm

Data Independence

A database system normally contains a lot of data in addition to users' data. For example, it stores data about data, known as metadata, to locate and retrieve data easily. It is rather difficult to modify or update a set of metadata once it is stored in the database. But as a DBMS expands, it needs to change over time to satisfy the requirements of the users. If the entire data is dependent, it would become a tedious and highly complex job.

Logical Data Independence

Logical data is data about database, that is, it stores information about how data is managed inside. For example, a table (relation) stored in the database and all its constraints applied on that relation. Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk. If we do some changes on table format, it should not change the data residing on the disk.

Physical Data Independence

All the schemas are logical, and the actual data is stored in binary on the disk. Physical data independence is the power to change the physical data without impacting the schema or logical data. For example, in case we want to change or upgrade the storage system itself —suppose we want to replace hard-disks with SSD (solid state drive) — it should not have any impact on the logical data or schemas.

Examples of changes under Physical Data Independence

Due to Physical independence, any of the below change will not affect the conceptual layer.

- Using a new storage device like Hard Drive or Magnetic Tapes
- Modifying the file organization technique in the Database
- Switching to different data structures.
- Changing the access method.
- Modifying indexes.
- Changes to compression techniques or hashing algorithms.
- Change of Location of Database from say C drive to D Drive

Examples of changes under Logical Data Independence

Due to Logical independence, any of the below change will not affect the external layer.

- Add/Modify/Delete a new attribute, entity or relationship is possible without a rewrite of existing application programs
- Merging two records into one
- Breaking an existing record into two or more records

https://pediaa.com/what-is-the-difference-between-uml-and-erd/

UML is a standard modeling language that helps to get a pictorial understanding of the software. There are various diagrams in UML such as class, object, use case, activity and many more. Whereas, ERD helps to design a database. It also represents the entities and how these entities connect with each other. Overall, UML is a modeling language whereas ERD is a diagram.

What is UML

UML stands for Unified Modeling Language. It is a language that helps to visualize, construct and documents software systems. It is not like other programming languages such as C, Java, and PHP because UML does not contain any programming statements. Moreover, it provides a graphical representation of the entire software. It also helps to model Object Oriented Programming concepts.

What is ERD

Most software systems have a database. ERD stands for **Entity Relationship Diagram**. It helps to design the database. Moreover, this diagram is based on the entity-relationship model, which is a model that represents the relationships among data.

https://pediaa.com/what-is-the-difference-between-uml-and-erd/

Module-64

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

Entities are represented by means of their properties called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

i) Simple attribute ii) Composite attribute and iii) Derived attribute iv) single-valued v) multi-valued

Types of Attributes

Simple attribute: Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.

Composite attribute: Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.

Derived attribute: Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For

example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.

Single-value attribute: Single-value attributes contain single value. For example: Social_Security_Number.

Multi-value attribute: Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

These attribute types can come together in a way like i) simple single-valued attributes ii) simple multi-valued attributes iii) composite single-valued attributes iv) composite multi-valued attributes

Entity-Set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set. For example, the roll_number of a student makes him/her identifiable among students.

Super Key: A set of attributes (one or more) that collectively identifies an entity in an entity set.

Candidate Key: A minimal super key is called a candidate key. An entity set may have more than one candidate key.

Primary Key: A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

Relationship

The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.

Relationship Set: A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

Degree of Relationship: The number of participating entities in a relationship defines the degree of the relationship.

Binary = degree 2

Ternary = degree 3

n-ary = degree n

Mapping Cardinalities

https://support.airtable.com/hc/en-us/articles/218734758-A-beginner-s-guide-to-many-to-many-relationships#onetomany

Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

One-to-one: One entity from entity set A can be associated with at most one entity of entity set B and vice versa.

- People-Passports (Each person has only one passport from a particular country and each passport is intended for only one person.)
- Country-Flag (Each country has only one flag and each flag belongs to only one country.)
- Spousal Relationships (Each person has only one spouse.)

One-to-many: One entity from entity set A can be associated with more than one entities of entity set B, however an entity from entity set B can be associated with at most one entity.

- People-Addresses (Each person can live at one address, but each address can house one or more people.)
- Owners-Pets (Each pet has one owner, but each owner can have one or more pets.)
- Farmer-Equipment (Each piece of farming equipment is owned by one farmer, but each farmer can own many pieces of equipment.)

Mapping Cardinalities

Many-to-one: More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.

- There are many states that are in a given region, but no states are in two regions.
- A university has many students, but no person can be student of many universities.

Many-to-many: One entity from A can be associated with more than one entity from B and vice versa.

- Ingredients-Recipes (Each food item can be used in multiple recipes and each recipe requires multiple ingredients.)
- Doctors-Patients (Each doctor sees many patients and each patient sees many doctors.)
- Employees-Tasks (Each employee works on many tasks at a time while each task is being worked on by one or more employees.)

Module-65

An Entity Relationship (ER) diagram is a visual representation of your database. It highlights the entities of your system and the relationship between those entities. An ER diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.

ER diagrams are related to data structure diagrams (DSDs), which focus on the relationships of elements within entities instead of relationships between entities themselves. ER diagrams are also often used in conjunction with data flow diagrams (DFDs), which map out the flow of information for processes or systems.

Entity

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

Attributes

Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity

If the attributes are **composite**, they are further divided in a tree like structure.

Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.

Multivalued attributes are depicted by double ellipse.

Derived attributes are depicted by dashed ellipse.

Relationship

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship are connected to it by a line.

Binary Relationship and Cardinality: A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

- **One-to-one:** When only one instance of an entity is associated with the relationship. Observe that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.

- **One-to-many:** When more than one instance of an entity is associated with a relationship. Observe that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship.

- **Many-to-one:** When more than one instance of entity is associated with the relationship. Observe that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship.

- **Many-to-many: M**ore than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship.

https://www.ques10.com/p/9460/explain-total-participation-and-partial-particip-1/

Participation Constraints

The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in. There are two types of participation constraints: Total and Partial

- **Total Participation**

  Total Participation is when each entity in the entity set occurs in at least one relationship in that relationship set.

  For example, consider the relationship borrower between customers and loans. A double line from loan to borrower, indicates that each loan must have at least one associated customer.

- **Partial Participation**

  Partial Participation is when each entity in the entity set may not occur in at least one relationship in that relationship set. For example, a company policy states that employee (manager) must manage a department, However every employee may not manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is partial

i.e. some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all.

Note: Partial Participation is represented by single line connecting entities in relationship.

https://www.tutorialspoint.com/dbms/dbms_pdf_version.htm

The ER Model has the power of expressing database entities in a conceptual hierarchical manner. As the hierarchy goes up, it generalizes the view of entities, and as we go deep in the hierarchy, it gives us the detail or specialization of every entity included.

Generalization

Going up in this structure is called **generalization**, where entities are clubbed together to represent a more generalized view. For example, pigeon, house sparrow, crow, and dove can all be generalized as Birds.

Specialization

Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics. Take a group 'Person' for example. A person has name, date of birth, gender, etc. These properties are common in all persons, human beings. But in a company, persons can be identified as employee, employer, customer, or vendor, based on what role they play in the company.

Module-66

Any database which has tables and constraints is not a relational database system (RDBMS). There are certain rules for a database to be perfect RDBMS. These rules are developed by Dr Edgar F Codd (EF Codd) in 1985 to define a perfect RDBMS. For a RDBMS to be a perfect RDBMS, it has to follow his rules. But no RDBMS can obey all his rules. Till now, there is hardly any commercial product that follows all the 13 **Codd's rules**.

Codd's twelve rules are a set of thirteen rules (numbered zero to twelve) proposed by Edgar F. Codd, a pioneer of the relational model for databases, designed to define what is required from a database management system in order for it to be considered relational, i.e., a relational database management system (RDBMS). https://www.tutorialcup.com/dbms/codds-rule.htm

Codd's Rule 0

This is the foundational Rule. This rule states that any database system should have characteristics as relational, as a database and as a management system to be RDBMS.

That means a database should be a relational by having the relation/mapping among the tables in the database. They have to be related to one another by means of constraints/relation. There should not be any independent tables hanging in the database.

RDBMS is management system – that means it should be able to manage the data, relation, retrieval, update, delete, permission on the objects. It should be able handle all these administrative tasks without affecting the objectives of database. It should be performing all these tasks by using query languages.

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following: data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule-5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

Rule-6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

Module-67

In this module we will briefly cover the Codd's rules using some simple examples.

Rule-1: Information Rule

The order of rows and columns in the table should not affect the meaning of the table. Each cell should have single data. There should not be any group/range of values separated by comma, space or hyphen (Normalized data). This should be the only way to store the data in a database. This rule is satisfied by all the databases.

For example:

Order of storing personal details about 'Ahsan' and 'Ismail' in EMPLOYEE table should not have any difference. There should be flexibility of storing them in any order in a row. Similarly, storing Employee name first and then his/her email address should be same as storing email address and then his/her name. It does not make any difference on the meaning of table.

Rule-2: Guaranteed Access Rule

This rule refers to the primary key. It states that any data/column/attribute in the table should be able to logically accessed by using the table in which it is stored, the primary key column of the table and the column which we want to access. When combination of these 3 is used, it should give the correct result. Any column/cell value should not be directly accessed without specifying the table and primary key.

For example:

Email address of Ahsan Abdullah EMPLOYEE + EmpID(Ahsan) + EmpEmail is the right way of getting any cell value.

Email address of Ahsan ahsan1010@yahoo.com should be able to access like this.

Rule-3: Systematic Treatment of NULL

This rule states about handling the NULLs in the database. As database consists of various data-types, each cell will have different data-types. If any of the cell value is unknown, or not applicable or missing, it cannot be represent as zero or empty. It will be always represented as NULL. This NULL should be acting irrespective of the data-type used for the cell i.e. EmpEmail type text vs. EmpSal type integer. When used in logical or arithmetical operation, it should result in value correctly.

For example:

Increasing salary by 100 for each employee will result in new salary of 100 for Ismail

100 + 0 = 100 but for Aazar **100 + NULL = NULL** i.e. arithmetic operations using NULL value should not result in any zero or numeric value. DBMS should be strong enough to handle these NULLs according to the situation and the data-types.

Rule-4: Active Online Catalog-This rule illustrates data dictionary. Metadata should be maintained for all the data in the database. These metadata should also be stored as tables, rows and columns and having access privileges. In short, metadata stored in the data dictionary should also obey all the characteristics of a database. Also, it should have correct up to date data. We should be able to access metadata by using same query language that we use to access the database.

SELECT * FROM ALL_TAB;

ALL_TAB is the table which has the table definitions that the user owns and has access. It is queried using the same SQL query that we use in the database.

Rule-5: Comprehensive Data Sub language Rule (Example)

SQL is a structured query language which support creating tables / views/ constraints/indexes, accessing the records of tables/views (SELECT), manipulating the records by insert/delete/update, provides security by giving different level of access rights (GRANT and REVOKE) and integrity and consistency by using constraints.

Any database without any query language is not a RDBMS. Database can be accessed by using query language directly or using them in the application.

Rule-6: View Updating Rule

Views are the virtual tables created by using queries to show the partial table. That is views are subset of table, it is only partial table with few rows and columns. This rule states that views are also be able to get updated as we do with its table.

For example:

Consider EMPLOYEE table, in which we have details of the employees who are in a certain salary bracket, say 'not paid'. Here EMPLOYEE is the whole table and EMAPLOYEE_NULLSAL is the view with unpaid Employees. According to this rule, we should be able to update the records in EMAPLOYEE_NULLSAL. But in real DBMS, we cannot give this privilege on views, view created is to give subset of data to the user in the form of table. When lengthy queries have to be written to get some details from the database, view shortens the length of the query and gives more meaningful and shorter query. In such case, updating the view is not feasible. Although updating the view will update the table used for creating it, it is not recommended by most of the database. Hence this rule is not used in most of the database.

Codd's Rule 7: High-level insert, update, and delete

This rule states that every query language used by the database should support INSERT, DELETE and UPDATE on the records. It should also support set operations like UNION, UNION ALL, MINUS, INTERSECT and INTERSECT ALL. All these operation should not be restricted to single table or row at a time. It should be able to handle multiple tables and rows in its operation.

For example: Suppose employees got 5% hike in a year. Then their salary has to be updated to reflect the new salary. Since this is the annual hike given to the employees, this increment is applicable for all the employees. Hence, the query should not be written for updating the salary one by one for thousands of employees. A single query should be strong enough to update the entire employee's salary at a time.

Codd's Rule 8: Physical Data Independence

If there is any change in the physical storage of the data, it should not affect the data at the logical or external view.

For example: If the data stored in one disk is transferred to another disk, then the user viewing the data should not feel the difference or delay in access time. The user should be able to access the data as he was accessing before. Similarly, if the file name for the table is changed in the memory, it should not affect the table or the user viewing the table. This is known as physical independence and database should support this feature.

Codd's Rule 9: Logical Data Independence

This is similar to physical data independence. Here if there are any changes to the logical view, then it should not be reflected in the user view.

For example: If we split the EMPLOYEE table according to his department into multiple employee tables, the user viewing the employee table should not feel that these records are coming from different tables. These split tables should be able to get joined and show the result. In our example we can use UNION and display the results to the user. But in ideal scenario, this is difficult to achieve since all the logical and user view will be tied so strongly that they will be almost same.

Codd's Rule 10: Integrity Independence

Database should be able apply integrity rules by using its query languages. It should not be dependent on any external factor or application to maintain the integrity. The keys and constraints in the database should be strong enough to handle the integrity. A good RDBMS

should be independent of the frontend application. It should at least support primary key and foreign key integrity constraints.

For example: Suppose we want to insert an employee for department 50 using an application. But department 50 does not exists in the system. In such case, the application should not perform the task of fetching if department 50 exists, if not insert the department and then inserting the employee. It should all handled by the database.

Codd's Rule 11: Distribution Independence

The database can be located at the user server or at any other network. The end user should not be able to know about the database servers. He should be able to get the records as if he is pulling the records locally. Even if the database is located in different servers, the accessibility time should be comparatively less.

Codd's Rule 12: Non-Subversion Rule

When a query is fired in the database, it will be converted into low level language so that it can be understood by the underlying systems to retrieve the data. In such case, when accessing or manipulating the records at low level language, there should not be any loopholes that alter the integrity of the database. In other words, even though the query written does not change the integrity of the tables, the converted low level language should be same as the query written. It should not be converted into some other low level language which changes the data integrity in the database or performs some unwanted actions in the database.

For example:

Update Student's address query should always be converted into low level language which updates the address record in the student file in the memory. It should not be updating any other record in the file nor inserting some malicious record into the file/memory.

Module-68

Overview

The core of a Web application is its server-side logic. The Web application layer itself can be comprised of many distinct layers. The typical example is a three-layered architecture comprised of presentation, business, and data layers. Figure 1 illustrates a common Web application architecture with common components grouped by different areas of concern.

Design Considerations

When designing a Web application, the goals of a software architect are to minimize the complexity by separating tasks into different areas of concern while designing a secure and high performance application.
When designing Web application, consider following guidelines:

• **Partition your application logically.** Use layering to partition your application logically into presentation, business, and data access layers. This helps you to create maintainable code and allows you to monitor and optimize the performance of each layer separately. A clear logical separation also offers more choices for scaling your application.

• **Use abstraction to implement loose coupling between layers**. This can be accomplished by defining interface components, such as a façade with well-known inputs and outputs that translates requests into a format understood by components within the layer. In addition, you can also use **Interface** types or abstract base classes to define a shared abstraction that must be implemented by interface components.

• **Understand how components will communicate with each other**. This requires an understanding of the deployment scenarios your application must support. You must determine if communication across physical boundaries or process boundaries should be supported, or if all components will run within the same process.

• **Reduce round trips.** When designing a Web application, consider using techniques such as caching and output buffering to reduce round trips between the browser and the Web server, and between the Web server and downstream servers.

• **Consider using caching.** A well-designed caching strategy is probably the single most important performance-related design consideration. ASP.NET caching features include output caching, partial page caching, and the cache API. Design your application to take advantage of these features.

• **Consider using logging and instrumentation.** You should audit and log activities across the layers and tiers of your application. These logs can be used to detect suspicious activity, which frequently provides early indications of an attack on the system.

• **Avoid blocking during long-running tasks.** If you have long-running or blocking operations, consider using an asynchronous approach to allow the Web server to process other incoming requests.

• **Consider authenticating users across trust boundaries.** You should design your application to authenticate users whenever they cross a trust boundary; for example, when accessing a remote business layer from your presentation layer.

• **Do not pass sensitive data in plain text across network.** Whenever you need to pass sensitive data such as a password or authentication cookie across the network, consider encrypting and signing the data or using SSL.

• **Design to run your Web application using a least-privilege account.** If an attacker manages to take control of a process, the process identity should have restricted access to the file system and other system resources in order to limit the possible damage.

Web Application Frame: There are several common issues to consider as your develop your design. These issues can be categorized into specific areas of the design.

Authentication

• Lack of authentication across trust boundaries.
• Storing passwords in a database as plain text.
• Designing custom authentication mechanism instead of using built-in capabilities.

Authorization

• Lack of authorization across trust boundaries.
• Incorrect role granularity.
• Using impersonation and delegation when not required.

Caching

• Caching volatile data.
• Not considering caching page output.
• Caching sensitive data.
• Failing to cache data in a ready-to-use format.

Exception Management

- Revealing sensitive information to the end user.
- Not logging sufficient details about the exception.
- Using exceptions for application logic.

Logging and Instrumentation

- Failing to implement adequate instrumentation in all layers.
- Failing to log system-critical and business-critical events.
- Not supporting runtime configuration of logging and instrumentation.

Navigation

- Mixing navigation logic with user interface components.
- Hard-coding relationships between views.
- Not verifying if the user is authorized to navigate to a view.

Page Layout (UI)

- Using table-based layout for complex layouts.
- Designing complex and overloaded pages.

Page Rendering

- Using excessive postbacks that impact user experience.
- Using excessive page sizes that reduce performance.

Request Processing

- Mixing processing and rendering logic.
- Choosing an inappropriate pattern.

Service Interface Layer

- Breaking the service interface.
- Implementing business rules in a service interface.
- Failing to consider interoperability requirements.

Validation

- Relying on client side validation.
- Lack of validation across trust boundaries.
- Not reusing the validation logic.

Authentication

Designing an effective authentication strategy is important for the security and reliability of your application. Improper or weak authorization can leave your application vulnerable to spoofing attacks, dictionary attacks, session hijacking, and other types of attack.

When designing an authentication strategy, consider following guidelines:

- Identify trust boundaries within Web application layers. This will help you to determine where to authenticate.
- Use a platform-supported authentication mechanism such as Windows Authentication when possible.
- If you are using Forms authentication, use the platform features when possible.
- Enforce strong account management practices such as account lockouts and expirations.
- Enforce strong passwords policies. This includes specifying password length and complexity, and password expiration policies.

Authorization

Authorization determines the tasks that an authenticated identity can perform and identifies the resources that can be accessed. Designing an effective authorization strategy is important for the security and reliability of your application. Improper or weak authorization leads to information disclosure, data tampering, and elevation of privileges. Defense in depth is the key security principle to apply to your application's authorization strategy.

When designing an authorization strategy, consider following guidelines:

- Identify trust boundaries within the Web application layers and authorize users across trust boundaries.
- Use URL authorization for page and directory access control.
- Consider the granularity of your authorization settings. Too fine granularity increases management overheads and too coarse granularity reduces flexibility.
- Access downstream resources using a trusted identity based on the trusted sub-system model.
- Use impersonation and delegation to take advantage of the user-specific auditing and granular access controls of the platform, but consider the effect on performance and scalability.

Module-69

In part-I we covered the design guidelines for framer topics 1 and 2, in this module we will cover the remaining framer topics 3-11.

- Authentication
- Authorization
- Caching
- Exception Management
- Logging and Instrumentation
- Navigation
- Page Layout (UI)
- Page Rendering
- Request Processing
- Validation
- Deployment Considerations

3. Caching: Caching improves the performance and responsiveness of your application. However, incorrect caching choices and poor caching design can degrade performance and responsiveness. You should use caching to optimize reference data lookups, avoid network round trips, and avoid unnecessary and duplicate processing. To implement caching, you must decide when to load the cache data. Try to load the cache asynchronously or by using a batch process to avoid client delays. When designing caching, consider following guidelines:

- Avoid caching volatile data.
- Use output caching to cache pages that are relatively static.
- Consider partial page caching through user controls for static data in your pages.
- Pool expensive shared resources e.g. network connections, instead of caching
- Cache data in a ready-to-use format.

4. Exception Management: Designing an effective exception management strategy is important for the security and reliability of your application. Correct exception handling in your Web pages prevents sensitive exception details from being revealed to the user, improves application robustness, and helps to avoid leaving your application in an inconsistent state in the event of an error. Following guidelines:

• Do not use exceptions to control logic flow, and design your code to avoid exceptions where possible.
• Do not catch exceptions unless you can handle them or you need to add information to the exception.
• Design a global error handler to catch unhandled exceptions.
• Display user-friendly messages to end users whenever an error or exception occurs.

5. Logging and Instrumentation: Effective strategy is important for the security and reliability of your application. You should audit and log activity across the tiers of your application. These logs can be used to detect suspicious activity, which frequently provides early indications of an attack on the system, and help to address the repudiation threat where users deny their actions. Log files may be required in legal proceedings to prove the wrongdoing of individuals. Generally, auditing is considered most authoritative if the audits are generated at the precise time of resource access and by the same routines that access the resource. Following guidelines:

• Consider auditing for user management events and unusual activities.
• Consider auditing for business critical operations.
• Create secure log file management policies, e.g. restricting the access to log files, allowing only write access to users, etc.
• Do not store sensitive information in the log or audit files.

6. Navigation

Separate navigation from the processing logic. It should allow users to navigate easily through your screens or pages. Designing a consistent navigation structure for your application will help to minimize user confusion as well as reducing the apparent complexity of the application. Following guidelines:

• Use well-known design patterns, such as Model-View-Presenter (MVP), to decouple UI processing from output rendering.
• Consider encapsulating navigation in a Master Page for consistency across pages.
• Design a site-map to help users and allow search engines to crawl the site if desired.
• Consider using visual elements such as embedded links, navigation menus, and breadcrumb text in the UI to help users understand where they are etc.

7. Page Layout (UI): Page layout to be separable from the specific UI components and UI processing. When choosing a layout strategy, consider whether designers or developers will be

building the layout. If designers will be building the layout, choose a layout approach that does not require coding or the use of development-focused tools. Following guidelines:

- Use Cascading Style Sheets (CSS) for layout whenever possible.
- Use table-based layout for a grid layout, table based layout can be slow, does not have full cross browser support, and there may be issues with complex layout.
- Common layout for pages where possible to maximize accessibility & ease of use.
- Use Master Pages in ASP.NET applications to provide a common look and feel for all of the pages.
- Avoid designing and developing large pages that accomplish multiple tasks, particularly where usually only a few tasks are executed with each request.

8. Page Rendering: Must ensure that you render the pages efficiently and maximize interface usability. Following guidelines:

- Consider data binding options. For example, you can bind custom objects or datasets to controls. However, be aware that binding only applies to rendered data in ASP.NET.
- Consider using AJAX for an improved user experience and better responsiveness.
- Consider using data paging techniques for large amounts of data to minimize scalability issues.
- Consider designing to support localization in user interface components.
- Abstract the user process components from data rendering and acquisition functions.

9. Request Processing: Ensure separation of concerns by implementing the request processing logic separately from the user interface. Following guidelines/considerations:

- Centralizing the common pre-processing and post-processing steps of web page requests to promote logic reuse across pages. E.g. consider creating a base class derived from the **Page** class to contain your common pre- and post-processing logic.
- If you are designing views for handling large amounts of data, consider giving access to the model from the view using the Supervising Controller pattern, which is a form of the MVP pattern.
- If your application does not have a dependency on view state and you have a limited number of control events, consider using MVC pattern.
- Consider using the Intercepting Filter pattern to implement the processing steps as pluggable filters when appropriate.

10. Validation: Important for the security and reliability of your application. Improper or weak authorization can leave your application vulnerable to cross-site scripting attacks, SQL injection attacks, buffer overflows, and other types of input attack. Following guidelines:

• Identify trust boundaries within Web application layers, and validate all data crossing these boundaries.
• Assume that all client-controlled data is malicious and needs to be validated.
• Design your validation strategy to constrain, reject, and sanitize malicious input.
• Design to validate input for length, range, format, and type.
• Use client side validation for user experience, and server side validation for security.

Module-70

In part-I we covered the design guidelines 1 and 2, in part-II we covered design guidelines 3-10 and in this module we will cover the remaining and last guidelines on deployment considerations.

- Authentication
- Authorization
- Caching
- Exception Management
- Logging and Instrumentation
- Navigation
- Page Layout (UI)
- Page Rendering
- Request Processing
- Validation
- **Deployment Considerations**
-

11. Deployment Considerations

When deploying a Web application, you should take into account how layer and component location will affect the performance, scalability and security of the application. You may also need to consider design trade-offs. Use either a distributed or a non-distributed deployment approach, depending on the business requirements and infrastructure constraints.

Consider the following guidelines for deployment:

• Consider using non-distributed deployment to maximize performance.

• Consider using distributed deployment to achieve better scalability and to allow each layer to be secured separately.

Non-Distributed Deployment

In a non-distributed deployment scenario, all the logically separate layers of the Web application are physically located on the same Web server, except for the database. You must consider how the application will handle multiple concurrent users, and how to secure the layers that reside on the same server. Figure shows this scenario.

Consider the following guidelines:

• Consider using non-distributed deployment if your Web application is performance sensitive, because the local calls to other layers provide performance gains.
• Consider designing a component-based interface for your business layer.
• If your business logic runs in the same process, avoid authentication at the business layer.
• Consider using a trusted identity (through the trusted subsystem model) to access the database. This improves the performance and scalability of your application.
• Consider encrypting and digitally signing sensitive data passed between the Web server and database server.

Distributed Deployment

In a distributed deployment, the presentation and business layers of the Web application reside on separate physical tiers, and communicate remotely. You will typically locate your business and data access layers on the same sever. Figure shows this scenario.

Consider the following guidelines when choosing a distributed deployment:

• Do not physically separate your business logic components unless this is necessary.
• If your security concerns prohibit you from deploying your business logic on your front-end web server, consider distributed deployment.
• Consider using a message-based interface for your business layer.
• Consider using the TCP protocol with binary encoding to communicate with the business layer for best performance.
• Consider protecting sensitive data passed between different physical tiers.

Load Balancing

When you deploy your Web application on multiple servers, you can use load balancing to distribute requests so that they are handled by different Web servers. This helps to maximize response times, resource utilization, and throughput. Figure shows this scenario.

Consider the following guidelines when designing your Web application to use load balancing:

• Avoid server affinity when designing scalable Web applications. Server affinity occurs when all requests from a particular client must be handled by the same server. It usually occurs when you use locally updatable caches, or in-process or local session state stores.
• Consider designing stateless components for your Web application; for example, a Web front end that has no in-process state and no stateful business components.
• Consider using Windows Network Load Balancing (NLB) as a software solution to implement redirection of requests to the servers in an application farm.

What does a server farm do?

A **server farm** or **server** cluster is a collection of computer servers – usually maintained by an organization to supply **server** functionality far beyond the capability of a single machine. **Server farms** often consist of thousands of computers which require a large amount of power to run and to keep cool.

Web Farm Considerations

A Web farm allows you to scale out you application, which can also minimize the impact of hardware failures. When you add more servers, you can use either a load balancing or a clustering approach.

Consider the following guidelines:

• Consider using clustering to minimize the impact of hardware failures.

• Consider partitioning your database across multiple database servers if your application has high input/output requirements.

• Consider configuring the web farm to route all requests from the same user to the same server to provide affinity where this is required.

• Do not use in-process session management in a web farm when requests from the same user cannot be guaranteed to be routed to the same server. Use an out-of-process state server service or a database server for this scenario.

Module-71

Overview

In this series of three modules, we will describe the key guidelines for the design of the data layer of an application. The guidelines are organized by category. They cover the common issues encountered, and commonly made mistakes, when designing the data layer.

Data Layer Components

• **Data access logic components**. Data access components abstract the logic necessary to access your underlying data stores. Doing so centralizes the data access functionality, which makes the application easier to configure and maintain.

• **Data Helpers / Utilities**. Helper functions and utilities assist in data manipulation, data transformation, and data access within the layer. They consist of specialized libraries and/or custom routines especially designed to maximize data access performance and reduce the development requirements of the logic components and the service agent parts of the layer.

• **Service agents**. When a business component must use functionality exposed by an external service, you may need to create code that manages the semantics of communicating with that service. Service agents isolate your application from the idiosyncrasies of calling diverse services, and can provide additional services such as basic mapping between the format of the data exposed by the service and the format your application requires.

Approach

A correct approach to designing the data layer will reduce development time and assist in maintenance of the data layer after the application is deployed. This section briefly outlines an effective design approach for the data layer. Perform the following key activities in each of these areas when designing your data layer:

1. **Create an overall design for your data access layer:**
a. Identify your data source requirements
b. Determine your data access approach
c. Choose how to map data structures to the data source
d. Determine how to connect to the data source
e. Determine strategies for handling data source errors.

2. **Design your data access components**:
a. Enumerate the data sources that you will access

b. Decide on the method of access for each data source

c. Determine whether helper components are required or desirable to simplify data access component development and maintenance

d. Determine relevant design patterns. For example, consider using the Table Data Gateway, Query Object, Repository, and other patterns.

Approach

3. **Design your data helper components:**

a. Identify functionality that could be moved out of the data access components and centralized for reuse

b. Research available helper component libraries

c. Consider custom helper components for common problems such as connection strings, data source authentication, monitoring, and exception processing

d. Consider implementing routines for data access monitoring and testing in your helper components

e. Consider the setup and implementation of logging for your helper components.

4. **Design your service agents:**

a. Use the appropriate tool to add a service reference. This will generate a proxy and the data classes that represent the data contract from the service

b. Determine how the service will be used in your application. For most applications, you should use an abstraction layer between the business layer and the data access layer, which will provide a consistent interface regardless of the data source. For smaller applications, the business layer, or even the presentation layer, may access the service agent directly.

Design Guidelines

• **Choose the data access technology.** The choice of an appropriate data access technology will depend on the type of data you are dealing with, and how you want to manipulate the data within the application. Certain technologies are better suited for specific scenarios.

• **Use abstraction to implement a loosely coupled interface to the data access layer**. Define interface components, such as a gateway with well-known inputs and outputs, which translate requests into a format understood by components within the layer. In addition, you can use interface types or abstract base classes to define a shared abstraction that must be implemented by interface components.

• **Consider consolidating data structures.** For table-based entities in your data access layer, consider using Data Transfer Objects (DTOs) to help organize the data into unified structures. In

addition, DTOs encourage coarse-grained operations while providing a structure that is designed to move data across different boundary layers.

• **Decide how you will manage connections.** As a rule, the data access layer should create and manage all connections to all data sources required by the application. Choose an appropriate method for storing and protecting connection information that conforms to application and security requirements.

• **Determine how you will handle data exceptions.** The data access layer should catch and (at least initially) handle all exceptions associated with CRUD operations. Data exceptions and timeout errors, should be handled in this layer and passed to other layers only if the failures affect application responsiveness or functionality.

BLOB

A BLOB is a Binary Large Object. When data is stored and retrieved as a single stream of data, it can be considered to be a BLOB. BLOBs may have structure within them, but that structure is not apparent to the database that stores it or the data layer that reads and writes it. Databases can store the BLOB data or can store pointers to them within the database. The BLOB data is usually stored in a file system if not stored directly in the database. BLOBs are typically used to store image data, but can also be used to store binary representations of objects.

When designing for BLOBs, consider the following guidelines:

• Store images in a database only when it is not practical to store them on the disk.

• Use BLOBs to simplify synchronization of large binary objects between servers.

• Consider whether you need to search the BLOB data. If so, create and populate other searchable database fields instead of parsing the BLOB data.

• When retrieving the BLOB, cast it to the appropriate type for manipulation within your business or presentation layer.

• Do not consider storing BLOB in the database when using buffered transmission.

Module-72

Batching

Batching database commands can improve the performance of your data layer. Each request to the database execution environment incurs an overhead. Batching can reduce the total overhead by increasing throughput and decreasing latency. Batching similar queries is better because the database caches and can reuse a query execution plan for a similar query.

Connections

Connections to data sources are a fundamental part of the data layer. All data source connections should be managed by the data layer. Creating and managing connections uses valuable resources in both the data layer and the data source. To maximize performance, follow guidelines for creating, managing, and closing connections

Data Format

Data formats and types are important to properly interpret the raw bytes stored in the database and transferred by the data layer. Choosing the appropriate data format provides interoperability with other applications, and facilitates serialized communications across different processes and physical machines. Data format and serialization are also important to allow the storage and retrieval of application state by the business layer.

Exception Management

Design a centralized exception management strategy so that exceptions are caught and thrown consistently in your data layer. If possible, centralize exception-handling logic in your database helper components. Pay particular attention to exceptions that propagate through trust boundaries and to other layers or tiers. Design for unhandled exceptions so they do not result in application reliability issues or exposure of sensitive application information.

Object Relational Mapping Considerations

When designing an Object Oriented (OO) application, consider the impedance mismatch between the OO model and the relational model that makes it difficult to translate between them. For example, encapsulation in OO designs, where fields are hidden, contradicts the public nature of properties in a database. Other examples of impedance mismatch include differences in the data types, structural differences, transactional differences, and differences in how data is manipulated. The two common approaches to handling the mismatch are data access design patterns such as Repository, and Object/Relational Mapping (O/RM) tools. A common model associated with OO design is the Domain Model, which is based on modeling entities after objects within a domain. As a result, the term domain represents an object-oriented design in the following guidelines.

Queries

Queries are the primary data manipulation operations for the data layer. They are the mechanism that translates requests from the application into create, retrieve, update and delete (CRUD) actions on the database. As queries are so essential, they should be optimized to maximize database performance and throughput.

Stored Procedures

In the past, stored procedures represented a performance improvement over dynamic SQL statements. However, with modern database engines, performance is no longer a major factor. When considering the use of stored procedures, the primary factors are abstraction, maintainability, and your environment. This section contains guidelines to help you design your application when using stored procedures. For guidance on choosing between using stored procedures and dynamic SQL statements, see the section that follows.

When it comes to security and performance, the primary guidelines are to use typed parameters and avoid dynamic SQL within the stored procedure. Parameters are one of the factors that influence the use of cached query plans instead of rebuilding the query plan from scratch. When parameter types and the number of parameters change, new query execution plans are generated, which can reduce performance.

Stored Procedures vs. Dynamic SQL

The choice between stored procedures and dynamic SQL focuses primarily on the use of SQL statements dynamically generated in code instead of SQL implemented within a stored procedure in the database. When choosing between stored procedures and dynamic SQL, you must consider the abstraction requirements, maintainability, and environment constraints.

Transactions

A transaction is an exchange of sequential information and associated actions that are treated as an atomic unit in order to satisfy a request and ensure database integrity. A transaction is only considered complete if all information and actions are complete, and the associated database changes made permanent. Transactions support undo (rollback) database actions following an error, which helps to preserve the integrity of data in the database.

Validation

Designing an effective input and data validation strategy is critical to the security of your application. Determine the validation rules for data received from other layers, from third party

components, as well as from the database or data store. Understand your trust boundaries so that you can validate any data that crosses these boundaries.

Module-73

XML

XML is useful for interoperability and for maintaining data structure outside the database. For performance reasons, be careful when using XML for very large amounts of data. If you must handle large amounts of data, use attribute-based schemas instead of element-based schemas. Use schemas to validate the XML structure and content.

When designing for the use of XML, consider the following guidelines:

- Use XML readers and writers to access XML-formatted data.
- Use an XML schema to define formats and to provide validation for data stored and transmitted as XML.
- Use custom validators for complex data parameters within your XML schema.
- Store XML in typed columns in the database, if available, for maximum performance.
- For read-heavy applications that use XML in SQL Server, consider XML indexes.

Manageability Considerations

Manageability is an important factor in your application. A manageable application is easier for administrators and operators to install, configure, and monitor. It also makes it easier to detect, validate, resolve, and verify errors at runtime. You should always strive to maximize manageability when designing your application.

When designing for manageability, consider the following guidelines:

- Use common interface types or a shared abstraction (Dependency Inversion) to provide an interface to the data access layer.

- Consider the use of custom entities, or decide if other data representations will better meet your requirements. Coding custom entities can increase development costs; however, they also provide improved performance through binary serialization and a smaller data footprint.

- Implement business entities by deriving them from a base class that provides basic functionality and encapsulates common tasks. However, be careful not to overload the base class with unrelated operations, which would reduce the cohesiveness of entities derived from the base class, and cause maintainability and performance issues.

Performance Considerations

Performance is a function of both your data layer design and your database design. Consider both together when tuning your system for maximum data throughput.

When designing for performance, consider the following guidelines:

• Use connection pooling and tune performance based on results obtained by running simulated load scenarios.

• Consider tuning isolation levels for data queries. If you are building an application with high throughput requirements, special data operations may be performed at lower isolation levels than the rest of the transaction. Combining isolation levels can have a negative impact on data consistency, so you must carefully analyze this option on a case-by-case basis.
• Consider batching commands to reduce round-trips to the database server.
• Use optimistic concurrency with non-volatile data to mitigate the cost of locking data in the database. This avoids the overhead of locking database rows, including the connection that must be kept open during a lock.
• If using a DataReader, use ordinal lookups to for faster performance.

Security Considerations

The data layer should protect the database against attacks that try to steal or corrupt data. It should allow only as much access to the various parts of the data source as is required. It should also protect the mechanisms used to gain access to the data source. When designing for security, consider the following guidelines:

• When using Microsoft SQL Server, consider using Windows authentication with a trusted sub-system.
• Encrypt connection strings in configuration files instead of using a system or user Data Source Name (DSN).
• When storing passwords, use a salted hash instead of an encrypted version of the password.
• Require that callers send identity information to the data layer for auditing purposes.
• If you are using SQL statements, consider the parameterized approach instead of string concatenation to protect against SQL injection attacks.

Deployment Considerations

When deploying a data access layer, the goal of a software architect is to consider the performance and security issues in the production environment. When deploying the data access layer, consider the following guidelines:

• Locate the data access layer on the same tier as the business layer to improve application performance.
• If you need to support a remote data access layer, consider using the TCP protocol to improve performance.

• You should not locate the data access layer on the same server as the database.

Technology Considerations

The following guidelines will help you to choose an appropriate implementation technology and techniques depending on the type of application you are designing and the requirements of that application:

• If you require basic support for queries and parameters, consider using ADO.NET objects directly.
• If you require support for more complex data-access scenarios, or need to simplify your data access code, consider using the Enterprise Library Data Access Application Block.
• If you are building a data-driven Web application with pages based on the data model of the underlying database, consider using ASP.NET Dynamic Data.
• If you want to manipulate XML-formatted data, consider using the classes in the System.Xml namespace and its subsidiary namespaces.
• If you are using ASP.NET to create user interfaces, consider using a DataReader to access data to maximize rendering performance. DataReaders are ideal for read-only, forward-only operations in which each row is processed quickly.
• If you are accessing Microsoft SQL Server, consider using classes in the ADO.NET SqlClient namespace to maximize performance.
• If you are accessing Microsoft SQL Server 2008, consider using a FILESTREAM for greater flexibility in the storage and access of BLOB data.
• If you are designing an object oriented business layer based on the Domain Model pattern, consider using the ADO.NET Entity Framework.

TOPIC-7: Web database connectivity

Module-74

The Web has been expanding at an incredible speed and even while you are reading this, hundreds and thousands of people are getting 'online' and hooked to the Web. Reactions to this technology are understandably mixed.

The database technology has been around for a long time now, and for many business and government offices, databases systems have already become an essential and integral part of the organization. Now the new technology has given the 'old' a shot in the arm, and the combination of the two creates many exciting opportunities for developing advanced database applications.

As far as database applications are concerned, a key aspect of the WWW technology is that it offers a brand new platform to collect, deliver and disseminate information. Via the Web, a database application can be made available, interactively, to users and organizations anywhere in the world.

a. **Internet:** It is a worldwide collection of interconnected computer networks, which belong to various organizations. It is not synonymous to the WWW.

b. **The WWW or Web:** Comprises software (e.g. Web servers and browsers) and data (e.g. Web sites). It simply represents a (huge) set of information resources and services that live on the Internet.

c. **Intranet:** A Web site or group of sites which belongs to an organization and can only be accessed by members of that organization.

d. **Extranet:** An intranet which allows partial access by authorized users from outside the organization via the Internet.

e. **HTTP (HyperText Transfer Protocol):** The standard protocol for transferring Web pages through the Internet. HTTP defines how clients (i.e. users) and servers (i.e. providers) should communicate.

f. **HTML (HyperText Markup Language):** A simple yet powerful language that is commonly used to format documents which are to be published on the Web.

g. **URL (Uniform Resource Locator):** A string of alphanumeric characters that represents the location of a resource (e.g. a Web page) on the Internet and how that resource should be accessed.

h. There are two types of Web pages: static and dynamic.

i. 1. **Static:** An HTML document stored in a file is a typical example of a static Web page. Its contents do not change unless the file itself is changed.

j. 2. **Dynamic:** For a dynamic Web page, its contents are generated each time it is accessed. As a result, a dynamic Web page can respond to user input from the browser by, for example, returning data requested by the completion of a form or returning the result of a database query. A dynamic page can also be customized by and for each user.

k. From the above, it can be seen that dynamic Web pages are much more powerful and versatile than static Web pages, and will be a focus for developing Web database applications. When the documents to be published are dynamic, such as those resulting from queries to databases, the appropriate hypertext needs to be generated by the servers. To achieve this, we must write scripts that perform conversions from different data formats into HTML 'on-the-fly'. These scripts also need to recognize and understand the queries performed by clients through HTML forms and the results generated by the DBMS.

l. In short, a Web database application normally interacts with an existing database, using the Web as a means of connection and having a Web browser or client program on the front end. Typically such applications use HTML forms for collecting user input (from the client); CGI (Common Gateway Interface, to be discussed later module) to check and transfer the data from the server; and a script or program which is or calls a database client to submit or retrieve data from the database. The diagram gives a graphical illustration of such a scenario.

m. **Platform independence:** Web clients are platform-independent and do not require modification to be run on different operating systems. Traditional database clients require extensive porting efforts to support multiple platforms. This is one of the most compelling reasons for building a Web-based client-server database application.

n. **Interpreted applications:** Web applications are written in interpreted languages (e.g. HTML and Java). This has an adverse effect on performance. In many applications, however, this is a price worth paying to gain the advantage of being platform independent. Time-critical applications not a good choice to implement on the Web.

o. **No need for installation:** Another benefit of Web database applications is that the need for installing special software is eliminated on the clients' side. It is pretty safe to assume that the clients have already had a Web browser installed, which is the only piece of software needed for the clients to run the applications.

p. **Simple client:** As a client needs just a browser to run a Web-based database application, the potential complications are minimized.

q. **Common interface across applications:** Again, because there is no need for specialized software, users can use a browser for different applications.

r. **Limited GUI (Graphical User Interface):** Big limitation of Web-based database applications. Highly customized application interfaces and highly interactive clients may not translate well as Web applications. This is because of the HTML limitations.

s. **Integrate with other applications:** Because of the benefit of being platform independent, different applications that adhere to the HTML standard can be integrated without many difficulties.

t. **Non-persistent connection to database:** Persistent database connections are highly efficient data channels between a client and the DBMS, and therefore, are ideal for database applications. However, Web-based applications do not have this benefit.

u. Apart from the above differences, there are some other important concerns for Web-based applications:

- **Reliability of the Internet:** At the moment, there are reliability problems with the Internet. It may break down; data may be lost on the net; large amounts of data traffic may slow down or even overwhelm the network system.

- **Security:** Security on the Internet is of great concern for any organization which has developed Web-based database applications. For example, the database may be broken into, or confidential data may be intercepted during transmission by unauthorized parties or even criminals.

v. At the present, a lot of research and development work is being carried out to address these concerns. There is no doubt that the potential problems can be overcome and over time, the Internet will be more reliable and more secure for connecting the world.

Module-75

Components of a database application

- Browser layer
- Application logic layer
- Database connection layer
- Database layer

2-tier client-server architecture

3-tier client-server architecture

Components of a database application

Web database applications may be created using various approaches. However, there are a number of components that will form essential building blocks for such applications. In other words, a Web database application should comprise the following four layers (i.e. components):

- Browser layer
- Application logic layer
- Database connection layer
- Database layer

Browser layer

The browser is the client of a Web database application, and it has two major functions. First, it handles the layout and display of HTML documents. Second, it executes the client-side extension functionality such as Java, JavaScript etc.

All browsers implement the HTML standard. Browsers are also responsible for providing forms for the collection of user input, packaging the input, and sending it to the appropriate server for processing. For example, input can include registration for site access, guest books and requests for information. HTML, Java, JavaScript may be used to implement forms.

Application logic layer

The application logic layer is the part of a Web database application with which a developer will spend the most time. It is responsible for:

• Collecting data for a query (e.g. a SQL statement).

• Preparing and sending the query to the database via the database connection layer.

• Retrieving the results from the connection layer.

• Formatting the data for display.

Most of the application's business rules and functionality will reside in this layer. Depending on the implementation methods used for the database application, the application logic layer may have different security responsibilities.

Database connection layer: This is the component which actually links a database to the Web server. Because manual Web database programming can be a daunting task, many current Web database building tools offer database connectivity solutions, and they are used to simplify the connection process.

The database connection layer provides a link between the application logic layer and the DBMS. Connection solutions come in many forms, such as DBMS net protocols, API (Application Programming Interface [see note below]) or class libraries, and programs that are themselves database clients. Some of these solutions resulted in tools being specifically designed for developing Web database applications.

The connection layer within a Web database application must accomplish a number of goals. It has to provide access to the underlying database, and also needs to be easy to use, efficient, flexible, robust, reliable and secure.

Database layer: This is the place where the underlying database resides within the Web database application. As we have already learned, the database is responsible for storing, retrieving and updating data based on user requirements, and the DBMS can provide efficiency and security measures.

In many cases, when developing a Web database application, the underlying database has already been in existence. A major task, therefore, is to link the database to the Web (the connection layer) and to develop the application logic layer.

1-tier client-server architecture

The simplest of Database Architecture are **1-tier** where the Client, Server, and Database all reside on the same machine. Anytime you install a DB in your system and access it to practice SQL queries it is 1-tier architecture. But such architecture is rarely used in production.

2-tier client-server architecture

Traditional client-server applications typically have a 2-tier architecture as illustrated in the figure. The client (tier 1) is primarily responsible for the presentation of data to the user, and the server (tier 2) is responsible for supplying data services to the client. The client will handle user interfaces and main application logic, and the server will mainly provide access services to the underlying database.

If such a 2-tier architecture is used to implement a Web database application, tier 1 will contain the browser layer, the application logic layer and the connection layer. Tier 2 accommodates the DBMS. This will inevitably result in a fat client.

3-tier client-server architecture

In order to satisfy requirements of increasingly complex database applications, a 3-tier architecture replaces the 2-tier one. The three tiers of which each can potentially run on a different platform.

The first tier is the client, which contains user interfaces. The middle tier accommodates the application server, which provides application logic and data processing functions. The third tier contains the actual DBMS, which may run on a separate server called a database server.

The 3-tier architecture is more suitable for implementing a Web database application.

The browser layer can reside in tier 1, together with a small part of the application logic layer. The middle tier implements the majority of the application logic as well as the connection layer. Tier 3 is for the DBMS.

Referring to the figure, for example, it can be seen that the Web Client is in the first tier. The Web Server and Gateway are in the middle tier and they form the application server. The DBMS and possibly other data sources are in the third tier.

Module-76

Database gateways

A Web database gateway is a bridge between the Web and a DBMS, and its objective is to provide a Web-based application the ability to manipulate data stored in the database. Web database gateways link stateful systems (i.e. databases) with a stateless, connectionless protocol (i.e. HTTP). HTTP is a stateless protocol in the sense that each connection is closed once the server provides a response. Thus, a Web server will not normally keep any record about previous requests.

A Web server will not normally keep any record about previous requests. This results in an important difference between a Web-based client-server application and a traditional client-server application:

a) In a Web-based application, only one transaction can occur on a connection. In other words, the connection is created for a specific request from the client. Once the request has been satisfied, the connection is closed. Thus, every request involving access to the database will have to incur the overhead of making the connection.

b) In a traditional application, multiple transactions can occur on the same connection. The overhead of making the connection will only occur once at the beginning of each database session.

There are a number of different ways to create Web database gateways. Generally, they can be grouped into two categories: client-side solutions and server-side solutions, as illustrated.

Client-side solutions

The client-side solutions include two types of approaches for connections:

- Browser extensions and external applications. Browser extensions are add-ons to the core Web browser that enhance and augment the browser's original functionality.
- External applications are helper applications or viewers. They are typically existing database clients that reside on the client machine and are launched by the Web browser in a particular Web application. Using external applications is a quick and easy way to bring legacy database applications online, but the resulting system is neither open nor portable. Legacy database clients do not take advantages of the platform independence and language independence available through many Web solutions. Legacy clients are resistant to change, meaning that any modification to the client program must be propagated via costly manual installations throughout the user base.

Server-side solutions

Server-side solutions are more widely adopted than the client-side solutions. A main reason for this is that the Web database architecture requires the client to be as thin as possible. The Web server should not only host all the documents, but should also be responsible for dealing with all the requests from the client.

3. Responsibilities of Web Server

a. Listening for HTTP requests.

b. Checking the validity of request.

c. Finding the requested resource.

d. Requesting authentication, if necessary.

e. Delivering requested resource.

f. Spawning programs if required.

g. Passing variables to programs.

h. Delivering output of programs to requester.

i. Displaying error message, if necessary.

4. Responsibilities of Client (browser)

j. Rendering HTML documents.

k. Allowing user navigate HTML links.

l. Displaying image.

m. Sending HTML form data to a URL.

n. Interpreting Java applets.

o. Executing plug-ins.

p. Executing external helper applications.

q. Interpreting JavaScript and other scripting language programs.

r. Executing ActiveX controls for IE.

vulearninghub.xyz | by VU Learning HUB

Module-77

Major tasks of client-side Web database application programming include the creation of browser extensions and the incorporation of external applications. These types of gateways take advantage of the resources of the client machine, to aid server-side database access. Remember, however, it is advantageous to have a thin client. Thus, the scope of such programming on the client-side should be limited. A very large part of the database application should be on the server side.

JavaScript

JavaScript is a scripting language that allows programmers to create and customize applications on the Internet and intranets. On the client side, it can be used to perform simple data manipulation such as mathematical calculations and form validation. JavaScript code is normally sent as a part of an HTML document and is executed by the browser upon receipt (the browser must have the script language interpreter). Note that JavaScript has little to do with Java language. JavaScript provides developers with a simple way to access certain properties and methods of Java applets on the same page, without having to understand or modify the Java source code of the applet.

Connection to databases

As a database gateway, JavaScript on the client side does not offer much without the aid of a complementary approach such as Java, plug-ins and CGI (Common Gateway Interface, to be discussed later).

Performance

JavaScript can improve the performance of a Web database application if it is used for client-side state management. It can eliminate the need to transfer state data repeatedly between the browser and the Web server. Instead of sending an HTTP request each time it updates an application state, it sends the state only once as the final action. However, it may result in the application becoming less robust if state management is completely on the client side. If the client accidentally or deliberately exits, the session state is lost.

ActiveX

ActiveX is a way to extend Microsoft IE's (Internet Explorer) capabilities. An ActiveX control is a component on the browser that adds functionality which cannot be obtained in HTML, such as access to a file on the client side, other applications, complex user interfaces, and additional hardware devices. ActiveX is similar to Microsoft OLE (Object Linking and Embedding), and ActiveX controls can be developed by any organization and individual. At the present, more

than one thousand ActiveX controls, including controls for database access, are available for developers to incorporate into Web applications.

Connection to databases

A number of commercial ActiveX controls offer database connectivity. Because ActiveX has abilities similar to OLE, it supports most or all the functionality available to any Windows program.

Performance

Like JavaScript, ActiveX can aid in minimizing network traffic. In many cases, this technique results in improved performance. ActiveX can also offer rich GUIs. The more flexible interface, executed entirely on the client side, makes operations more efficient for users.

Plug-ins

Plug-ins are Dynamic Link Libraries (DLL) that give browsers additional functionality. Plug-ins can be installed to run seamlessly inside the browser window, transparent to the user. They have full access to the client's resources, because they are simply programs that run in an intimate association with the Web browser.

There are a number of important issues concerning plug-ins:

• Plug-ins incur installation requirements. Because they are native code, not packaged with the browser itself, plug-ins must be installed on the client machine.

• Plug-ins are platform dependent. Whenever a change is made, it must be made on all supported platforms.

Connection to databases

Plug-ins can operate like any stand-alone applications on the client side. They can be used to create direct socket connections to databases via the DBMS net protocols (such as SQL *Net for Oracle). Plug-ins can also use JDBC, ODBC, OLE and any other methods to connect to databases.

Performance

Plug-ins are loaded on demand. When a user starts up a browser, the installed plug-ins are registered with the browser along with their supported MIME types, but the plug-ins themselves are not loaded. When a plug-in for a particular MIME type is requested, the code is then loaded into memory. Because plugins use native code, their executions are fast.

External applications

External helper applications can be new or legacy database clients, or a terminal emulator. If there are existing traditional client-server database applications which reside on the same machine as the browser, then they can be launched by the browser and execute as usual. This approach may be an appropriate interim solution for migrating from an existing client-server application to a purely Web-based one. It is straightforward to configure the browser to launch existing applications. It just involves the registration of a new MIME type and the associated application name. For organizations that cannot yet afford the time and funds needed to transfer existing database applications to the Web, launching legacy applications from the browser provides a first step that requires little work.

Maintenance issues

Using the external applications approach, the existing database applications need not be changed. However, it means that all the maintenance burdens associated with traditional client-server applications will remain. Any change to the external application will require a very costly reinstallation on all client machines. Because this is not a pure Web-based solution, many advantages offered by Web-based applications cannot be realized.

Performance

Traditional client-server database applications usually offer good performance. They do not incur the overhead of requiring repeated connections to the database. External database clients can make one connection to the remote database and use that connection for as many transactions as necessary for the session, closing it only when finished.

Module-78

CGI is the de facto standard for interfacing Web clients and servers with external applications, and is arguably the most commonly adopted approach for interfacing Web applications to data sources (such as databases). This module will basically cover CGI.

With CGI, a process is spawned on the server each time a request is made for a CGI program. There is no method for keeping a spawned process alive between successive requests, even if they are made by the same user. Furthermore, CGI does not inherently support distributed processing, nor does it provide any mechanism for sharing commonly used data or functionality among active and future CGI requests. Any data that exists in one instance of a CGI program cannot be accessed by another instance of the same program.

a.) CGI is a protocol for allowing Web browsers to communicate with Web servers, such as sending data to the servers. Upon receiving the data, the Web server can then pass them to a specified external program (residing on the server host machine). The external program is called a CGI program or CGI script. Because CGI is a protocol, not a library of functions written specifically for any particular Web server, CGI programs/scripts are language independent. As long as the program/script conforms to the specification of the CGI protocol, it can be written in any language such as C, C++ or Java. In short, CGI is the protocol governing communications among browsers, servers and CGI programs.

b.) In general, a Web server is only able to send documents and to tell a browser what kinds of documents it is sending. By using CGI, the server can also launch external programs (i.e. CGI programs). When the server recognizes that a URL points to a file, it returns the contents of that file. When the URL points to a CGI program, the server will execute it and then send back the output of the program's execution to the browser as if it were a file.

c.) The CGI approach enables access to databases from the browser. The Web client can invoke a CGI program/script via a browser, and then the program performs the required action and accesses the database via the gateway. The outcome of accessing the database is then returned to the client via the Web server. Invoking and executing

CGI programs from a Web browser is mostly transparent to the user.

The following steps need to be taken in order for a CGI program to execute successfully:

- The user (Web client) calls the CGI program by clicking on a link or by pressing a button. The program can also be invoked when the browser loads an HTML document (hence being able to create a dynamic Web page).

- The browser contacts the Web server, asking for permission to run the CGI program.

• The server checks the configuration and access files to ensure that the program exists and the client has access authorization to the program.

• The server prepares the environment variables and launches the program.

• The program executes and reads the environment variables and STDIN.

• The program sends the appropriate MIME headers to STDOUT, followed by the remainder of the output, and terminates.

• The server sends the data in STDOUT (i.e. the output from the program's execution) to the browser and closes the connection.

• The browser displays the information received from the server.

As mentioned earlier, when preparing data for the browser to display, the CGI program has to include a header as the first line of output. It specifies how the browser should display the output. This header may be one of the following types:

Primarily, there are four methods available for passing information from the browser to a CGI program. In this way, clients' input (representing users' specific requirements) can be transmitted to the program for actions.

1. Passing parameters on the command line.

2. Passing environment variables to CGI programs.

3. Passing data to CGI programs via STDIN.

4. Using extra path information.

Detailed discussions on these methods are beyond the scope of this course.

Advantages

The main advantages of CGI are its simplicity, language independence, Web server independence and its wide acceptance.

Disadvantages

1. Communication between a client (browser) and the database server must always go through the Web server in the middle, which may cause a bottleneck if there is a large number of users accessing the Web server simultaneously. For every request submitted by a Web client or every response delivered by the database server, the Web server has to convert data from or to an HTML document.

2. Lack of efficiency and transaction support in a CGI program. For every query submitted through CGI, the database server has to perform the same logon and logout procedure, even for subsequent queries submitted by the same user. The CGI program could handle queries in batch mode, but then support for online database transactions that contain multiple interactive queries would be difficult.

3. The server has to generate a new process or thread for each CGI program. For a popular site (like Yahoo), there can easily be hundreds or even thousands of processes competing for memory, disk and processor time. This situation can incur significant overhead.

4. Extra measures have to be taken to ensure server security. CGI itself does not provide any security measures, and therefore developers of CGI programs must be security conscious.

In order to overcome these problems of classical CGI, an improved version of CGI, called FastCGI, has been developed with the following features:

• **Language independence:** As with CGI, FastCGI is a protocol and not dependent on any specific language.

• **Open standard:** Like CGI, FastCGI is positioned as an open standard. It can be implemented by anyone. The specifications, documentation and source code (in different languages) can be obtained at the Web site

https://soramimi.jp/fastcgi/fastcgispec.html.

• **Independence from the Web server architecture:** A FastCGI application need not be modified when an existing Web server architecture changes. As long as the new architecture supports the FastCGI protocol, the application will continue to work.

• **Distributed computing:** FastCGI allows the Web application to be run on a different machine from the Web server. In this way, the hardware can be tuned optimally for the software.

• **Allocating processes:** FastCGI applications do not require the Web server to start a new process for each application instance. Instead, a certain number of processes are allotted to the FastCGI application. The number of processes dedicated for an application is user-definable. These processes can be initiated when the Web server is started or on demand.

Module-79

CGI is the de facto standard for interfacing Web clients and servers with external applications, and is arguably the most commonly adopted approach for interfacing Web applications to data sources (such as databases). This module will basically cover CGI.

With CGI, a process is spawned on the server each time a request is made for a CGI program. There is no method for keeping a spawned process alive between successive requests, even if they are made by the same user. Furthermore, CGI does not inherently support distributed processing, nor does it provide any mechanism for sharing commonly used data or functionality among active and future CGI requests. Any data that exists in one instance of a CGI program cannot be accessed by another instance of the same program.

HTTP server APIs and server modules

HTTP server (Web server) APIs and modules are the server equivalent of browser extensions. The central theme of Web database sites created with HTTP server APIs or modules is that the database access programs coexist with the server. They share the address space and run-time process of the server. This approach is in direct contrast to the architecture of CGI, in which CGI programs run as separate processes and in separate memory spaces from the HTTP server.

Instead of creating a separate process for each CGI program, the API offers a way to create an interface between the server and the external programs using dynamic linking or shared objects. Programs are loaded as part of the server, giving them full access to all the I/O functions of the server. In addition, only one copy of the program is loaded and shared among multiple requests to the server.

Server vendor modules

Server modules are just prefabricated applications written in some server APIs. Developers can often purchase commercial modules to aid or replace the development of an application feature. Sometimes, the functionality required in a Web database application can be found as an existing server module. Vendors of Web servers usually provide proprietary server modules to support their products. There are a very large number of server modules that are commercially available, and the number is still rising. For example, Oracle provides the Oracle PL/SQL module, which contains procedures to drive database-backed Web sites. The Oracle module supports both NSAPI and ISAPI.

- **Server speed:** API programs run as dynamically loaded libraries or modules. A server API program is usually loaded the first time the resource is requested, and therefore, only the first user who requests that program will incur the overhead of loading the dynamic libraries. Alternatively, the server can force this first instantiation so that no user will incur the loading

overhead. This technique is called preloading. Either way, the API approach is more efficient than CGI.

• **Resource sharing:** Unlike a CGI program, a server API program shares address space with other instances of itself and with the HTTP server. This means that any common data required by the different threads and instances need exist only in one place. This common storage area can be accessed by concurrent and separate instances of the server API program. The same principle applies to common functions and code. The same set of functions and code are loaded just once and can be shared by multiple server API programs. The above techniques save space and improve performance.

• **Range of functionality:** A CGI program has access to a Web transaction only at certain limited points. It has no control over the HTTP authentication scheme. It has no contact with the inner workings of the HTTP server, because a CGI program is considered external to the server. In contrast, server API programs are closely linked to the server; they exist in conjunction with or as part of the server. They can customize the authentication method as well as transmission encryption methods. Server API programs can also customize the way access logging is performed, providing more detailed transaction logs than are available by default.

Proprietary HTTP servers

A proprietary HTTP server is defined as a server application that handles HTTP requests and provides additional functionality that is not standard or common among available HTTP servers. The functionality includes access to a particular database or data source, and translation from a legacy application environment to the Web.

Examples of proprietary servers include IBM Domino, Oracle Application Express Listener and Hyper-G. These products were created for specific needs. For Domino, the need is tight integration with legacy Lotus Notes applications, allowing them to be served over the Web. Oracle Application Express Listener was designed to provide highly efficient and integrated access to back-end Oracle databases. For Hyper-G, the need is to have easily maintainable Web sites with automatic link update capabilities.

The main objectives of creating proprietary servers are to meet specialized and customized needs, and to optimize performance. However, the benefits of proprietary servers must be carefully weighed against their exclusive ties to a Web database product (which may bring many shortcomings). It requires a thorough understanding of the business requirements in order to determine whether or not a proprietary Web server is appropriate in a project.

Module-80

In previous modules, we have studied various approaches that enable browsers (Web clients) to communicate with Web servers, and in turn allow Web clients to have access to databases. For example, CGI, FastCGI or API programs can be invoked by the Web client to access the underlying database. In this module and in the next module, we are going to discuss how database connections can actually be made via those CGI/FastCGI/API programs. We will learn what specific techniques, tools and languages are available for making the connections.

Database API libraries are at the core of every Web database application and gateway. Regardless how a Web database application is built (whether by manually coding CGI programs or by using a visual application builder), database API libraries are the foundation of database access.

Database API libraries are collections of functions or object classes that provide source code access to databases. They offer a method of connecting to the database engine (under a username and password if user authentication is supported by the DBMS), sending queries across the connection, and retrieving the results and/or error messages in a desired format.

Traditional client-server database applications have already employed database connectivity libraries supplied by vendors and third-party software companies (i.e., third party class libraries). Because of this fact of wider user base, database APIs have the advantage over other gateway solutions for Web database connectivity.

The Web database applications that require developers to use database API libraries are mainly CGI, FastCGI or server API programs. Web database application building tools, including template-driven database access packages and visual GUI builders, use database APIs as well as the supporting gateways (such as CGI and server API), but all these interactivities are hidden from the developers.

Native database APIs

Native database APIs are platform-dependent as well as programming language dependent. However, most popular databases (such as Oracle) support multiple platforms in the first place, and therefore, the porting between different platforms should not require excessive effort.

In general, programs that use native database APIs are faster than those using other methods, because the libraries provide direct and low-level access. Other database access methods tend to be slower, because they add another layer of programming to provide the developer a different, easier, or more customized programming interface. These additional layers slow the overall transaction down.

Native database API programming is not inherently dependent on a Web server. For example, a CGI program using native API calls to Oracle that works with the Netscape server should also work with other types of servers. However, if the CGI program also incorporates Web server-specific functions or modules, it will be dependent on that Web server.

Benefits of database APIs

Database APIs (native or independent) arguably offer the most flexible way in which Web database applications are created. Applications created with native database APIs are more efficient than those with database-independent APIs. This database connectivity solution is the fastest way to access database functionality and has been tested rigorously in the database software industry. It is worth noting that database APIs have been used successfully for years even before the invention of the Web.

Shortcomings of database APIs

The most notable disadvantage of programming in database API is complexity. For rapid application development and prototyping, it is better to use a high-level tool, such as template-driven database access software or visual application builders.

Another disadvantage is with ODBC. Because ODBC standardizes access to databases from multiple vendors, applications using ODBC do not have access to native SQL database calls that are not supported by the ODBC standard.

In some cases, this can be inconvenient and may even affect application performance.

vulearninghub.xyz | by VU Learning HUB

Module-81

Template-driven database connectivity packages are offered by database vendors and third-party developers to simplify Web database application programming. Such a package usually consists of the following components:

- Template consisting of HTML and non-standard tags or directives
- Template parser
- Database client daemons

Template-driven packages are very product dependent. Different DBMSs require database access templates in different formats. An application developed for one product will be strongly tied to it. Migrating from one product to another is very difficult and requires a rewrite of all the database access, flow control and output-formatting commands.

An example of a template-driven package is PHP.

Benefits of template-driven packages

a.) The most important benefit is speed of development. Assuming an available package has been installed and configured properly, it takes as little time as a few hours to create a Web site that displays information directly from the database.

Shortcomings of template-driven packages

b.) The structures of templates are normally predetermined by vendors or third party developers. As a result, they only offer a limited range of flexibility and customizability. Package vendors provide what they feel is important functionality, but, as with most off-the-shelf tools, such software packages may not let you create applications requiring complex operations.

c.) Although templates offer a rapid path to prototyping and developing simple Web database applications, the ease of development is obtained for the cost of speed and efficiency. Because the templates must be processed on demand and require heavy string manipulation (templates are of a large text type or string type; they must be parsed by the parser), using them is slow compared with using direct access such as native database APIs.

d.) The actual performance of an application should be tested and evaluated before the usefulness of such a package is ruled out. The overhead of parsing templates may be negligible if using high-performance machines. **e.)** Other factors, such as development time or development expertise, may be more important than a higher operational speed.

Visual Web database building tools offer an interesting development environment for creating Web database applications. For developers accustomed to point-and-click application programming, these tools help speed the development process. For instance, Visual Basic and/or Microsoft Access developers.

a.) The approach: The architectures of visual building tools vary. In general, they include a user-friendly GUI (Graphical User Interface), allowing developers to build a Web database application with a series of mouse clicks and some textual input. These tools also offer application management so that a developer no longer needs to juggle multiple HTML documents and CGI, NSAPI or ISAPI programs manually.

b.) At the end of a building session, the tool package can generate applications using various techniques. Some applications are coded using ODBC; some use native database APIs for the databases they support; and others may use database net protocols.

c.) Some of these tools create their own API, which can be used by other developers. Some generate code that works but can still be modified and customized by developers using various traditional IDEs, compilers and debuggers.

d.) May generate a CGI program or a Web server API program (such as NSAPI and ISAPI). Some sophisticated tools even offer all the options. Unlike native database APIs or template-driven database connectivity packages, GUI tools tend to be as open as possible. Many offer development support for the popular databases.

Benefits of visual tools

a.) Visual development tools can be of great assistance to developers who are familiar and comfortable with visual application development techniques. They offer rapid application development and prototyping, and an organized way to manage the application components. **b.)** Visual tools also shield the developer from low-level details of Web database application development. As a result, a developer can create a useful Web application without the need to know what is happening in the code levels.

Shortcomings of visual tools

c.) Depending on the sophistication of the package used, the resulting programs may be slower to execute than similar programs coded by an experienced programmer. Visual application building tools, particularly Object-oriented ones, tend to generate fat programs with a lot of unnecessary sub-classing. **d.)** Another potential drawback is cost. A good visual tool may be too expensive for a small one-off development budget.

Module-82

a.) State is an abstract concept of being, which can be explained by a set of rules, facts or truisms. A state in a database application includes a set of variables and/or other means to record who the user/client is, what tasks he/she has been doing, at what position he/she is at a particular instance in time, and many other useful pieces of information about a database session.

b.) Persistence is the capability of remembering a state and tracking state changes across different applications or different periods of time within an instance of an application or multiple instances.

The requirement of state maintenance in Web database applications results in the increased complexity. As mentioned before in the Context section,

c.) HTTP is connectionless, which means that once an HTTP request is sent and a response is received, the connection to the server is closed. If a connection were to be kept open between client and server, the server could at any time query the client for state information and vice versa. The server would be able to know the identity of the user throughout the session once the user logged in. However, the reality is that there is no constant connection throughout the session. Thus, the server

d.) cannot have memory of the user's identity even after user login. In this situation, programmers must find a way to make session state persist.

The URL approach

It works as follows:

• A registration or login page is delivered to the user.

• The user types in a username and password, and then submits the page.

• The username and password pair are sent to a server-side CGI program, which extracts the values from the QUERY_STRING environment variable.

• The values are checked by the server to determine whether or not the user is authenticated.

• If he/she is authenticated, the authenticated state is reflected in a randomly generated session ID (SID), which is stored in a database along with other necessary data to describe the state of the user session.

• The SID can then be stored in all URLs within HTML documents returned by the server to the client, therefore tracking the identity of the user throughout the session.

Benefits of the URL approach

a.) The URL approach is easy to use and maintain state. To retrieve a state, the receiving CGI program need only collect the data from environment variables in the GET method and act on it as necessary. To pass on, set or change the state, the program simply creates new URLs with the appropriate data.

Shortcomings of the URL approach

b.) If the state information has to be kept in the URL, the URL becomes very long and can be very messy. Also, such a URL displays part of the application code and low-level details. This causes security concerns, and may be used by hackers.

c.) If an application manages state on the client side using the URL method, the state will be lost when the user quits the browser session unless the user bookmarks the URL. A bookmark saves the URL in the browser for future retrieval.

d.) If state is maintained solely in the URL without any server-side state data management, bookmarking is sufficient to recreate the state in a new browser session. However, having the user perform this maintenance task is obviously undesirable.

URL Query_String: This is another popular method of maintaining state. A registered user in a site has a hidden form appended to each page visited within the site. This form contains the username and the name of the current page. When the user moves from one page to another, the hidden form moves as well and is appended to the end of the succeeding HTML page.

Benefits of the hidden fields approach

Like the URL approach, it is easy to use to maintain state. In addition, because the fields are hidden, the user has a seamless experience and sees a clean URL.

Another advantage of using this approach is that, unlike using URLs, there is no limit on the size of data that can be stored.

Shortcomings of the hidden fields approach

As with the URL approach, users can fake states by editing their own version of the HTML hidden fields. They can bring up the document source in an editor, change the data stored, and then submit the tampered form to the server. This raises serious security concerns.

Data is also lost between sessions. If the entire session state is stored in hidden fields, that state will not be accessible after the user exits the browser unless the user specifically saves the

HTML document to disk or with a bookmark. Again, it is undesirable to involve users in this kind of maintenance task.

HTTP cookies: A cookie is a small text file containing: 1) Name of the cookie 2) Domains for which the cookie is valid 3) Expiration time in GMT 4) Application-specific data such as user information

Cookies are sent by the server to the browser, and saved on client's disk. Whenever needed, server can request a desired cookie from the client. The client browser will check cookie presence, if present browser send cookie data to the server.

Benefits of cookies:

1. Cookies can be completely transparent. As long as a user does not choose the browser option to be alerted before accepting cookies, his/her browser will handle incoming cookies and place them on the client disk without user intervention.

2. Cookies are stored in a separate file, location handled by the browser and difficult for user to find. Also, cookies difficult to tamper with. This increases security.

3. As cookies stored on the client disk, their data is accessible even in a new browser session. It does not require the user to do anything.

4. If a programmer chooses to set an expiration date or time for a cookie, the browser will invalidate the cookie at the appropriate time.

Shortcomings of cookies

1. Stored data usually limited to 4 kb. For large state data, consider other techniques.

2. Physically stored on the client disk, they cannot move with the user. This effect is important for applications whose users often change machines.

3. Although difficult to tamper with, still possible for someone to break into them. Remember a cookie is just a text file. If a user can find it can edit it.

Important considerations

Managing state on the client

An application can maintain all of its state on the client-side with any of the methods discussed in the prior modules.

• Benefits of the client-side maintenance

On attraction of maintaining state on the client is simplicity. It is easier to keep all the data in one place, and by doing it on the client, it eliminates the need for server database programming and maintenance.

If an application uses client-side extensions to maintain state, it can also provide a faster response to the user because the need to network access is eliminated.

• Shortcomings of the client-side maintenance

If all the state data is on the client-side, there is a danger that users can somehow forge state information by editing URLs, hidden fields, and cookies. This leads to security risks in server programs.

With the exception of the cookie approach to maintaining state, there is no guarantee that the necessary data will be saved when the client exits unexpectedly. Thus, the robustness of the application is compromised.

Managing state on the server

This approach for maintaining state actually involves using both the client and the server. Usually a small piece of information, either a user ID or a session key is stored on the client-side. The server program uses this ID or key to look up the state data in a database.

• Benefits of the server-side maintenance

Maintaining state on the server is more reliable and robust than the client-side maintenance. As long as the client can provide an ID or a key, the user's session state can be restored, even between different browsing sessions.

Server-side maintenance can result in thin clients. The less dependent a Web database application is on the client, the less code needs to exist on or be transmitted to the client.

Server-side maintenance also leads to better network efficiency, because only small amounts of data need to be transmitted between the client and the server.

• Shortcomings of the server-side maintenance

The main reason an application would not be developed using server-side state maintenance is its complexity, because it requires the developer to write extensive code. However, the benefits of implementing server-side state management outweigh the additional work required.

Module-83

Security risks exist in many areas of a Web database application. This is because the very foundations of the Internet and Web – TCP/IP and HTTP – are very weak with respect to security. Without special software, all Internet traffic data transmission traveling in the open can be intercepted by anyone with a little bit skill. If no measures are taken, there will be many security loopholes that can be exploited by malicious users on the Internet.

In this module we will cover the first four topics, and in the next module-64 will cover topics 5 and 6. In general, security issues in Web database applications include the following:

- Data transmission (communication) between the client and the server is not accessible to anyone else except the sender and intended receiver (privacy).
- Data cannot be changed during transmission (integrity).
- The receiver can be sure that the data is from the authenticated sender (authenticity).
- The sender can be sure the receiver is the genuinely intended one (non-fabrication).
- The sender cannot deny he/she sent it (non-repudiation).
- The request from the client should not ask the server to perform illegal or unauthorized actions.
- The data transmitted to the client machine from the server must not be allowed to contain executables that will perform malicious actions.

At the present, there are a number of measures that can be taken to address some of the above issues. These measures are not perfect in the sense that they cannot cover every eventuality, but they should help get rid of some of the loopholes. It must be stressed that security is the most important but least understood aspect of Web database programming. More work still needs to be done to enhance security.

Proxy servers: A proxy server is a system that resides between a Web browser and a Web server. It intercepts all requests to the Web server to determine if it can fulfill the requests itself. If not, it forwards the requests to the Web server. Due to the fact that the proxy server is between browsers and the Web server, it can be utilized to be a defense for the Web server.

Firewalls

Because a Web server is open for access by anyone on the Internet, it is normally advised that the server should not be connected to the **intranet** (i.e., an organisation's internal network). This way, no one can have access to the **intranet** via the Web server.

However, if a Web application has to use a database on the **intranet**, then the firewall approach can be used to prevent unauthorized access.

A firewall is a system designed to prevent unauthorized access to or from a private network (**intranet**). It can be implemented in either hardware, software, or both. All data entering or leaving the **intranet** (connected to the Internet) must pass through the firewall. They are checked by the firewall system and anything that does not meet the specified security criteria is blocked.

A proxy server can act as a firewall, because it intercepts all data in and out, and can also hide the address of the server and **intranet**.

Digital signatures: (a.) A digital signature consists of two pieces of information: a string of bits that is computed from the data (message) that is being signed along with the private key of the requester for the signature. The signature can be used to verify that the data is from a particular individual or organization. The digital signature technique is very useful for verifying authenticity and maintaining integrity. **(b)** It has the following properties:

• Authenticity is verifiable using a computation based on a corresponding public key.

• If the private key is kept secret, the signature cannot be forged.

• It is unique for the data signed. The computation will not produce the same result for two different messages.

• The signed data cannot be changed, otherwise the signature will no longer verify the data as being authentic.

Digital certificates: (c.) A digital certificate is an attachment to a message used for verifying the sender's authenticity. Such a certificate is obtained from a Certificate Authority (CA), which must be a trust-worthy organization. When a user wants to send a message, he/she can apply for a digital certificate from the CA. The CA issues an encrypted certificate containing the applicant's public key and other identification information. The CA makes its own key publicly available.

(d.) When the message is received, the recipient uses the CA's public key to decode the digital certificate attached to the message, verifies it as issued by the CA, and then obtains the sender's public key and identification information held within the certificate. With this information, the recipient can send an encrypted reply.

Secure sockets layer (SSL) and secure HTTP (S-HTTP)

(a.) SSL is an encryption protocol developed by Netscape for transmitting private documents over the Internet. It works by using a private key to encrypt data that is to be transferred over the SSL connection. Netscape, Firefox, Chrome and Microsoft IE support SSL.

(b.) Another protocol for transmitting data securely over the Internet is called Secure HTTP, a modified version of the standard HTTP protocol. Whereas SSL creates a secure connection between a client and a server, over which any amount of data can be sent securely, S-HTTP is designed to transmit individual messages securely.

In general, the SSL and S-HTTP protocols allow a browser and a server to establish a secure link to transmit information. However, the authenticity of the client (the browser) and the server must be verified. Thus, a key component in the establishment of secure Web sessions using SSL or S-HTTP protocols is the digital certificate. Without authentic and trustworthy certificates, the protocols offer no security at all.

Module-84

In this module we will first cover topics 5 and 6 listed in last module-83, and then address the performance issues and briefly mention 9 of those issues.

Web database applications are very complex, more so than stand-alone or traditional client-server applications. They are a hybrid of technology, vendors, programming languages, and development techniques.

Many factors work together in a Web database application and any one of them can obstruct the application's performance. It is crucial to understand the potential bottlenecks in a Web database application as well as to know effective, well-tested solutions to address the problems.

Java security

If Java is used to write the Web database application, then many security measures can be implemented within Java. Three Java components can be utilized for security purposes:

(.a) The class loader: It not only loads each required class and checks it is in the correct format, but also ensures that the application/applet does not violate system security by allocating a namespace. This technique can effectively define security levels for each class and ensure that a class with a lower security clearance can never be in place of a class with a higher clearance.

(b.) The bytecode verifier: Before the Java Virtual Machine (JVM) will allow an application/applet to execute, its code must be verified to ensure: compiled code is correctly formatted; internal stacks will not overflow or underflow; no illegal data conversions will occur; bytecode instructions are correctly typed; and all class member accesses are valid.

(c.) The security manager: An application-specific security manager can be defined within a browser, and any applets downloaded by this browser are subject to its (security manager's) security policies. This can prevent a client from being attacked by dangerous methods.

a. ActiveX is a deprecated software framework created by Microsoft that adapts its earlier Component Object Model (COM) and Object Linking and Embedding (OLE) technologies for content downloaded from a network, particularly from the World Wide Web. Microsoft introduced ActiveX in 1996. In principle, ActiveX is not dependent on Microsoft Windows operating systems, but in practice, most ActiveX controls only run on Windows. Most also require the client to be running on an x86-based computer because ActiveX controls contain compiled code

b. For Java, security for the client machine is one of the most important design factors. Java applet programming provides as many features as possible without compromising the security

of the client. In contrast, ActiveX's security model places the responsibility for the computer's safety on the user (client).

c. Before a browser downloads an ActiveX control that has not been digitally signed or has been certified by an unknown CA, it displays a dialog box warning the user that this action may not be safe. It is up to the user to decide whether to abort the downloading, or continue and accept a potential damaging consequence.

The following is a list of issues concerning performance:

• **Network consistency:** The availability and speed of network connections can significantly affect performance.

• **Client and server resources:** This is the same consideration as in the traditional client-server applications. Memory and CPU are the scarce resources.

• **Database performance:** It is concerned with the overhead for establishing connections, database tuning, and SQL query optimization.

• **Content delivery:** This is concerned with the content's download time and load time. The size of any content should be minimized to reduce download time; and appropriate format should be chosen for a certain document (mainly images and graphics) so that load time can be minimized.

• **State maintenance:** It should always minimize the amount of data transferred between client and server and minimize the amount of processing necessary to rebuild the application state.

• **Client-side processing:** If some processing can be carried out on the client-side, it should be done so. Transmitting data to the server for processing that can be done on the client-side will degrade performance.

• **Programming language:** A thorough understanding of the tasks at hand and techniques available can help choose the most suitable language for implementing the application.

https://www.apmdigest.com/15-top-factors-that-impact-application-performance

1. Network consistency: Network used by application impacts performance tremendously, especially for mobile & cloud. Inconsistent bandwidth, high jitter, increased latency and packet loss all degrade application performance. Although you can't control mobile or most cloud networks, you can build and test apps with these network conditions in mind. This gives organizations best chance to optimize app performance before users feel the network's impact.

2. Peak Usage (8): Poor understanding of how the application will be used (i.e. how many people will simultaneously use it and for what kind of transactions), and the corresponding application architecture and its scaling assumptions that go into its design and deployment. This lack of understanding of real user transactions and performance manifests itself as bottlenecks in performance during the most critical peak usage period.

3. Application Design (9): Biggest impacts on application performance caused by outsourcing/subcontracting application development outside company's quality control domain. Quality and performance needs to be built into the application and cannot be an afterthought or something that "we'll fix later". The subpar app performance that is accepted in the development phase is bound to show itself in the production stage. Modern APM solutions capture this poor performance, but can't provide the cure. The only way to prevent poor app performance is to expose your app development to the rigorous quality controls and processes early on in the application lifecycle — and actually fix them early in the cycle.

TOPIC-8: Web database operations

Module-85

Result Sets

As noted previously, relational databases manipulate tables:

Data is stored in tables, and the results of database operations are other tables (which, when they contain a single row and column, are single values). All of this is based on the mathematics of set theory—unions, intersections, and logical Operations on elements of sets that have certain properties and attributes. The resultant tables can be called result sets.

Procedural Programming: This type of computer processing environment differs from the procedurally based world that preceded it. The computer code that is generated looks quite different. Typical procedure—based code is structured like this:

```
Get a data record.
Check to see if condition X is met; if so do A, if not do B.
If the results of A are of type T, do C, else do D;
     if the results of B are of type T, do C, else do E.
Get the next record.
```

Database Programming: Database programming tends to move all of the condition checking into the database query itself. Thus, you wind up with several sections of code of this nature:

```
Get all data records in which condition X is met and
where the results of A would be T.
Do A and C for all those records.
Get all data records in which condition X is not met and-
where the-results of B would be T.
Do B and C for all these records.
Get all data records in which condition X is met and
where the results of A would not be T.
Do A and D for all those records.
```

This matters to database-driven Web sites because you must be able to structure your Web pages_ in this way. In fact, because of the nature of HTML, you really have no choice in the matter—you cannot write traditional procedural code on them. The typical Web page as constructed by an application server is generated as shown in Figure.

You send an HTTP request to the Web server (usually by typing in a URL or clicking on a link). The Web server, together with the application server (Which may be a plug-in or part of the Web server) then constructs the page—all of this is shown inside the dotted box of Figure 10-1.

You may provide HTML excerpts or stubs to be used as headers and footers and as templates to surround data returned from the database. However, the database queries are specified as part of the original HTTP request to the Web server there is no further interaction with the user until the page is returned. Many pages encapsulate only one query; those that do incorporate more than one (as the figure shows) need to have both specified in the original HTTP request.

Once you have started to generate an HTML page for a user, you cannot stop in the middle for database accesses and queries. (You can make a subsequent query dependent on the results of a first query on the page, but that is not the same as interacting with the user.)

Thus, you need all the information for your query (or queries) when you start to generate the page. This includes passwords, options (number of records to retrieve), and all other parameters for the query. (See “Sending Data to Web Sites with HTTP and Forms" on page 255 and "Cookies" on page 374 for how to implement this.)

Not only do you need the information, but you need to be able to find it in the database. Having people type in their passport ID numbers will not help you to retrieve any information about them if the passport ID numbers are not in the database.

As you start to design your Web pages, keep this in mind and make certain that you know what you will need to retrieve the data you need—and that is in the database. (In large enterprises, it can be a very time-consuming process to change database designs.)

Some application server products do allow you to interrupt this processing to perform intermediate actions and tests on data as it is retrieved and as HTML is generated. In such cases, the complexity of the project quickly mushrooms.

vulearninghub.xyz | by VU Learning HUB

Module-86

Looking back at Figure 10-l, another point should become clear: the process of generating the database driven Web page is almost always substantially longer than the process of returning a prefabricated Web page (all that is required in that case is to transmit the text of the HTML on the page). Some old-fashioned HTML coders will immediately thumb their noses and say, “See? This database stuff is just an inefficient waste of time.”

What matters in general is that the total time is usually less, but it is spent in different ways. The feel of a database-driven Web site will be very different from that of a Web site without a database, and that in turn will feel different from a database driven application that doesn't use the Web (or even a graphical user interface). Beware of jumping to conclusions based on timing and performance of small pieces of the application.

Automating web page production

The fact that database-driven Web pages take longer to generate than simple HTML pages means no more or less than that: the total cost (and complexity) of a database-driven Web site is almost always less than that of a traditional site. However, those costs increase and decrease in different places:

- The cost of coding HTML pages by hand is substantially reduced with database-driven Web sites.
- The computer resources needed to deliver database-driven Web pages are greater than those for traditional HTML pages. You either need more powerful computers or you need to accept slower responses. (Note: the final HTML page that is generated is typically not much bigger for database-driven Web pages than for manually written Web pages. As a result, it is only the computer resources—not the telecommunications link—that need upgrading.)

You are simply automating the production of Web pages, and it should not be surprising that you need greater computer resources and fewer human resources to do so (that is the history of all automation efforts).

Interface Adaptations: In addition to more computer power for database-driven Web sites, consider the user's site browsing experience. Even with very fast computers, a lot of work is needed to generate each page; the Web server (and application server) needs access to the database in order to fulfill users’ requests: all of this takes time.

When a page presents a search interface to browse the database, producing that page is usually quite simple: database access is needed only for the results. Users who have formulated their

query and asked for a database search are not encounter delay. For other purposes, the possible delay needs to be addressed with an adaptation, such as:

- **Frames.** While using HTML frames, the generated page is actually produced from several separate pages. You can use frames that do not rely on database access to return a banner or navigation frame quickly while the database access is in progress. Thus, the user sees part of the usable page (not just a background) very quickly.

- **Redirection.** Use the redirect command to automatically send user to a new page. This is commonly used when a Web site has moved—the user types in www.olddomain.com, and redirect statement automatically sends to www.newdomain.com after a brief pause. Use the first page for welcome and identification; while the user reads it, the redirect command is executed (and with it the database access). Then the full database-driven page is automatically displayed.

- **Set Home Pages.** On intranets, set default home pages to your site's home page. When people start their browsers, they will automatically be connected to your site and the database accesses will occur as part of the start-up process.

Transaction processing is the collection of a number of operations into what appears to be a single process: that single process either succeeds or fails. When you are working with transactions, you do not worry about leaving an inventory record in one condition while an account balance record is in another: if they are part of a transaction, the charge to a customer account occurs in the same process as the shipment from inventory. Transaction processing can be implemented at a variety of levels:

- Many databases support transactions themselves, allowing you to specify which operations will be part of the transaction.

- Some application servers and application development tools let you create transactions at that level.

- You can create your own transactions. While this is not particularly difficult to do, it involves creating temporary flags to mark transactions as pending or complete.

It is likely that you will have a choice of how to implement transaction processing. What you should remember, though, is that transaction processing is relatively expensive. It is one of those cases in which operations are performed twice—or even three times-—during the process of carrying out the transaction and leaving proper traces behind so that the transaction can be undone if necessary.

The two critical parts of a transaction are its commitment (on successful preparation of the transaction) and its rollback (in the case of failure). Failure can be anything from corruption of a disk or database to a user cancellation of the process. Because transaction processing is relatively expensive, make certain that you use it correctly. If you have a set of related operations that should be carried out but that do not need to be rolled back as a single operation in the case of failure, you do not need transaction processing. The critical thing to remember with transaction processing is that the rollback must be necessary (in the case of problems). Grouping operations together logically into a transaction is a waste of expensive computer time.

Note also that transaction processing can degrade database performance for other processing that occurs while the transaction is being handled. At the critical moment of the transaction, a single-thread operation must occur: the database must write to its log and verify that that write statement has been executed correctly.

Transactions always involve database updates; there is no data retrieval process that cannot be repeated, and so retrieval can never require a rollback. (You may want to lock the database or parts of it during a retrieval to prevent updates while you are retrieving data, but that is not the same as a transaction.)

Database-driven Web sites often merge data from databases with data that is contained in standard HTML pages. (Your name and address, for instance, may be part of your HTML page rather than being retrieved from the database.) Since it is almost always easier to type information into an HTML page than to enter it into a database (and to plan for its retrieval), you may find that with time more and more of your site's data finds its way from the database environment into the HTML environment. Knowing that this is a danger, you should be able to guard against it.

Whereas it is almost always easier to enter data into an HTML page than into a database, it is almost always easier to maintain it when it is in a database. This is a typical short-versus long-term trade-off, but the long-term benefits of database- driven Web sites are so great that you should be ever vigilant.

Summary

Database programming is somewhat different from procedural programming—the step—by—step processing of data as you encounter it. The principles of database programming—particularly result sets and transaction processing—make it quite amenable for use with Web sites.

Module-87

When you type an address into a browser or click a link (technically an HREF attribute of an anchor) on a Web the command is executed, a message is sent; that message has two parts—the header and an optional body. The HTTP specification states three parts to the message header as follows:

1. An action that describes what the Web server is supposed to do about the URL.

2. The URL itself.

3. The version of HTTP to which this message conforms.

The message may contain a message body. All HTTP requests must receive a response (even if it is an error stating that the address is wrong, the server is bug, etc.). If no response is received, the browser that sent the request gives the user an error message indicating that response was received. The Web page that you see in your browser is technically the response to the request that you have specified by typing in a URL.

GET: it requests Web server get the resource (usually a Web page) described in the URL i.e. allows the Web server to generate an HTML message that is not a static page. GET is the action that is assumed when you type in a URL or create a link (with HREF).

POST: it is designed to specifically transfer data to the resource named in the URL. That data can be specified using MIME encoding, i.e. you can transfer pictures, sounds, etc. The POST method is used with forms.

1. It starts with the **scheme** followed by a colon and two slashes (http: / / for the Web).

2. The **host name** is specified either as four sets of digits separated by periods (such as 192.168.1.3) or as a domain name (such as www.yourdomain.com). If the host is omitted, it is assumed to be the computer on which the request is being made.

3. Optionally, a colon and a **port number** can be provided. Each scheme has a default port (for HTTP it is 80) that it automatically connects to.

4. A **slash** follows the specification of the host and port. For the local computer a blank host name and if the default port is used, the URL looks like this at this point: http: ///. For a URL without a port, it looks like this: http: //www.yourdomain.com/, and for a URL with a port, it looks like http: //www.yourdomain.com: 80/.

5. The **path component** is the name of the resource to which the request is directed. Often, it is a Web page with a name that ends in .htm or .html. Thus, the path might be weeklydata.html. If the file is located within a directory on the host computer, that directory

and others are specified and separated by slashes as in project/webfiles/weeklydata.html. In other cases, the path specifies an application; to which the request is directed; the application is responsible for replying to the request.

6. If the request contains **data** to be sent to the application server (whether that is Microsoft Internet Information Server, FileMaker Web Companion etc.) as part of the header, that data (called the search part or query) follows the path and is preceded by a question mark. The data format is described the following section.

The simplest HTTP requests just specify the location of a resource that will provide a response—which is usually a Web page. In order to tell the resource what you want it to do, you must pass additional information as part of the request.

There are two ways to pass information as part of a request:

1. You can pass the information in the URL as a searchpart.

2. You can pass the information in the message body.

Searchpart and queries

You can include a searchpart in the URL that is part of the request. The searchpart contains text data, each element of which is identified by a descriptive name.

If more than one element is provided, the elements are separated by ampersands (&). The name of each descriptor is specified by the resource that you are addressing. The data sent in a searchpart must be text data, adhering to Internet standards. The numbers and letters of the alphabet are the most basic text elements.

When you use the GET action (to retrieve a page from a Web location), data is always sent in the searchpart.

1. Because it is part of the URL, the data that is being sent may be visible in a browser's location display. (You can verify this by looking at the URL that is generated when you click the Search button in a search engine.) This can pose security issues.

2. Only text data can be sent in a searchpart.

3. Large amounts of data can be unwieldy in a search-part. Although the searchpart and the URL have been designed to be read by machines, some browsers have difficulty handling long requests, even though they should not.

The POST action is designed to send relatively large amounts of data to a Web resource; the response is often a Web page indicating that the data has been processed. Accordingly, the POST action relies on a message body to pass data. None of the three disadvantages of searchparts exists in a message body:

1. The message body is never shown in a browser's location display.

2. The message body is used for various purposes; browsers and other Web software are used to dealing with multiple-part, multiple-type messages (such as messages that contain text, sound, video, and custom data types—often all in one message).

3. Message bodies are typically long; unexpected problems with browsers usually do not occur with long messages.

vulearninghub.xyz | by VU Learning HUB

Module-88

A form is a container for controls, such as data entry elements with which users interact. Form attributes enable interaction with programs on remote computer:

- Forms have a **method**. The method instructs the browser how to transmit the data from the form to the action program. GET and POST are commonly used methods.

- Forms have an **action** associated with them. The action is the name of a program (and the path to get to it); the program is executed when the user submits the form.

- Like most HTML elements, forms may also be named. Naming your forms makes the code more readable and can allow you to manipulate them using controls located outside the form on the same Web page

- A form's character encoding determines what characters it will accept and how they will be transmitted.

- The content type of a form specifies how data submitted with the POST method is transmitted. (discussed in last module.)

Form Tags: The starting tag for a form element must contain the form's method and action. Here are typical starting tags:

<FORM METHOD:"POST" ACTION=”NewRec.idc">

<FORM METHOD=“POST” ACTION:“FMPr0“>

If sending a form from a Web page on a different computer, must include the address—as in ACTION=“http://192.168.l.l/NewReC.idC“. Everything after the starting tag until the ending tag

(</FORM>) is part of the form.

Controls are the elements of a form with which a user interacts. Most of them are coded in HTML using the INPUT element. The types of controls and their attributes are described in this module. Controls are familiar to users of any graphical interface.

Buttons may be general or the submit and reset buttons that have special meanings within a form. Other buttons may be used to launch scripts.

Checkboxes allow yes/no choices.

Radio buttons can be grouped (by giving several of them the same name). Only one of a group of radio buttons can be on at one time.

Menus provide single and multiple selections from a designated list of options.

Text controls allow entry onto a single line.

Text areas allow entry into a larger multiline area.

Special text fields for passwords are defined; typing in them does not display the characters that you type.

Image controls display an image.

Hidden controls have no visible representation, but they can contain data that is submitted with the form.

Object and file select controls are also part of the HTML specification.

Control attributes let you manipulate your form's controls. The attributes that matter most to you are these:

Type. You must specify a type (such as radio, text, text area, or hidden) for the control so that the user's browser knows how to display it.

Name. Each control has a name. It is used to identify the control within the form and the control's data within the message that is transmitted to the action program. When you are collecting data within a form that will be sent to a CDML action, the control names are normally the names either of database fields or of requests (e.g. -db , -lay).

Value. Each control has a value. You specify an initial value (often blank as in VALUE=""), but normally users can enter new values. (Buttons, which cause actions, have no data values in this sense; the value of a button is the text that is displayed on it.) The data transmission when the user submits the form includes pairs of NAME=VALUE. Thus, your form may combine control names and values to transmit data such as

first name=John —db=students.fp3

Other attributes let you specify the size (width) of the control, the maximum number of characters for a text or password control, whether or not a radio button or checkbox is checked, etc.

Note that controls flow down the page just as all other data elements do. To position them from side to side or in more complex manners, place them within a table. Note also that controls can be commingled with other HTML elements within a form.

This basic code snippet—<INPUT TYPE=x NAME=y VALUE=z>

Hidden controls are used frequently in forms that update databases. A hidden control has the same three attributes that all other controls do—its type (”hidden"), a name, and a value. Just as with other controls, a hidden control is transmitted as part of the data when a form is submitted. Its transmission is exactly the same as that of other controls-

controlname=controlvalue

This is the way in which data can be transmitted to the database without having been entered by the user. You can create hidden fields and set their values to the values of fields in the current or other databases (using the FMP—Field element); you can also set the values of hidden fields to dates, times, or other items that you construct or calculate when the Web page is displayed.

Hidden controls are a specified type of control: you cannot hide radio buttons, text fields, or menus. A hidden control has no visual representation, and so only its name and value matter. There is a distinction between a hidden control and a control of another type which is placed so that it is not visible.

Any of three things can be done with a form:

1. It can be submitted—the content of its controls are sent using the specified method to the action program.

2. It can be reset—all entered values are replaced with the values specified in the VALUE= attribute of each control.

3. It can be ignored—-the user can move on to another page or another activity.

The SUBMIT input type (<INPUT TYPE=SUBMIT>) is used for a button that submits the form. The value of this control is used as the name for the button.

Likewise, a RESET input type is used for the button that will reset the form's values. Its value is used for the button name.

As soon as a form gets larger than a few fields, design issues come into play. Most important point to note is that there is no substitute for usability testing: not everyone follows directions, and not everyone behaves the way you do.

Size of forms: In general, forms should be as small as possible. The complexity of a large form is daunting, whether it be on the computer or on paper.

Scrolling & screen size: When possible, a form should be seen with as little scrolling as possible. This is hard when designing forms from PCs to smart phones, try to keep as much information visible as possible. Often, you can do this by using a table within your form so various data entry fields are arrayed across as well as down the page.

Logical size: Try to keep your form to the smallest part of your Web page. In general, a form that consists of an entire Web page is too big. Forms can contain text, graphics, and other HTML elements, but if they are not to be submitted as data, it is best to keep them out of the form. Lay out your page—with its graphics, links, etc. and place the form within a small section of it.

Managing large forms: Sometimes, you need to have a lot of data on a form. Split the form into several separate screens. On the first screen, place the first batch of data fields. Using the GET method, send those fields and their data to the next screen. Place them in hidden fields on that screen, and let users enter the visible fields.

In this way, you can have a reasonable number of entry fields on each screen (perhaps 10), and accumulate the entries from prior screens into hidden fields on each current screen. The final submission will send all data from both hidden and visible fields for processing.

TOPIC-9: Rapid Application Development (RAD)

Module-89

What Is Rapid Application Development (RAD)?

Rapid application development is an agile software development approach that focuses more on ongoing software projects and user feedback and less on following a strict plan. As such, it emphasizes rapid prototyping over costly planning. Though often mistaken for a specific model, rapid application development (RAD) is the idea that we benefit by treating our software projects like clay, rather than steel, which is how traditional development practices treat them.

How did RAD come about?

In the 1980s, Barry Boehm, James Martin and others recognized this obvious point: software was not a raw mineral resource. They saw software for what it was: infinitely malleable. Boehm and Martin took advantage of software's inherent pliability when designing their development models: the Spiral Model and the James Martin RAD model, respectively. Since then, RAD has evolved to take on other forms and acted as a precursor to agile.

RAD Methodology

Though exact practices and tools vary between specific rapid application development methods, their underlying phases remain the same:

1. Define Requirements

Rather than making you spend months developing specifications with users, RAD begins by defining a loose set of requirements. We say loose because among the key principles of rapid application development is the permission to change requirements at any point in the cycle. Basically, developers gather the product's "gist." The client provides their vision for the product and comes to an agreement with developers on the requirements that satisfy that vision.

2. Prototype

In this rapid application development phase, the developer's goal is to build something that they can demonstrate to the client. This can be a prototype that satisfies all or only a portion of requirements (as in early stage prototyping).

This prototype may cut corners to reach a working state, and that's acceptable. Most RAD approaches have a finalization stage where developers pay down technical debt amassed by early prototypes.

3. Absorb Feedback

With a recent prototype prepared, RAD developers present their work to the client or end-users. They collect feedback on everything from interface to functionality—it is here where product requirements might come under scrutiny. Clients may change their minds or discover that something that seemed right on paper makes no sense in practice. Clients are only human, after all. With feedback in hand, developers return to some form of step 2: they continue to prototype. If feedback is strictly positive, and the client is satisfied with the prototype, developers can move to step 4.

4. Finalize Product

During this stage, developers may optimize or even re-engineer their implementation to improve stability, maintainability, and a third word ending in '-ility.' They may also spend this phase connecting the back-end to production data, writing thorough documentation, and doing any other maintenance tasks required before handing the product over with confidence.

Both Boehm's Spiral Model and James Martin's RAD Model make use of these four steps to help development teams reduce risk and build excellent products. However, RAD has its drawbacks as well.

With the pros and cons of RAD laid out, we can determine which types of projects benefit most from RAD, and which do not. If you need to build an internal business tool or even a customer-facing portal, like an app or website, RAD techniques will help your team deliver a better experience to your end-user.

However, if you are tasked with building mission-critical software (flight controls, implant firmware, etc.), a RAD approach is not only inappropriate, it may also be irresponsible. A pilot with a failing control module or a heart attack survivor with a malfunctioning pacemaker cannot provide prototype feedback from beyond the grave.

RAD Advanatges

Advantage	Description
Speed	In the traditional waterfall approach, developers were unlikely to go on vacation after delivering the product. Clients would invariably request changes ranging from interface to functionality after first delivery. With RAD, projects are more likely to finish on time and to the client's satisfaction upon delivery.

Cost	In RAD, developers build the exact systems the client requires, and nothing more. In waterfall, IT risks building and fleshing out complex feature sets that the client may choose to gut from the final product. The time spent building zombie features can never be recovered, and that means the budget spent on them is lost. RAD reduces this risk and therefore reduces the cost.
Developer Satisfaction	In the traditional waterfall approach, developers work in silos devoid of feedback and positive affirmation for a product well-made. And when they finally get the opportunity to present their work to the client, the client may not roll out the red carpet for them. Regardless of how proud developers are of their work, if the client isn't satisfied, developers don't receive the accolades they so desperately seek. In RAD, the client is there every step of the way and the developer has the opportunity to present their work frequently. This gives them the confidence that when the final product is delivered, their work receives appreciation.

Module-90

RAD vs. Agile

Those who research development methodology compare one framework to another. Most commonly, RAD is directly contrasted with Agile. Unfortunately, this comparison is challenging to draw. RAD is a forbear of agile, but agile encompasses far more than a development model. Agile is more of a philosophy than a methodology.

In an attempt to show this, we have contrasted the core principles of each concept:

As you can see, agile took several steps beyond the scope of RAD. While agile dictates the ideal working environment (just shy of how many rubber ducks to keep on your desk), RAD focuses on how to build software products for your clients and end-users. Let's take a closer look at what RAD entails.

Rapid Application Development Tools

As you may now understand, rapid application development is more of a software development methodology rather than a specific language, tool, or interface. However, tools can help facilitate rapid design, development, and feedback solicitation.

Design and Prototyping Tools

The products in this category help teams craft interactive designs at impressive speeds. And some tools on this list, like Webflow, allow designers to export the completed design as a functional cross-browser prototype.

User Testing and Feedback Tools

As noted many times thus far, RAD methodology requires frequent feedback from clients and end-users. And in modern workflows, developers who work offsite prefer to solicit feedback remotely rather than book travel and accommodations each and every time they require input from clients.

Development Tools

If your team has strict technology requirements or a limited skill set, it's simpler to stick with what they know. Often you cannot justify the cost of migrating technologies. But if you're willing to consider a new approach to development, the tools in this category will accelerate your production cycle.

Low-code tools, for example, bundle development elements (IDE, APIs, languages, framework, UI components, connectors, etc.) into a single coherent suite of tools for building applications

visually, integrating them with the back-end, and then managing the app lifecycle. No-code tools, by contrast, offer self-service application assembly for business users who are not developers.

Module-91

PHP is a programming language for building dynamic, interactive Web sites. As a general rule, PHP programs run on a Web server, and serve Web pages to visitors on request. One of the key features of PHP is that you can embed PHP code within HTML Web pages, making it very easy for you to create dynamic content quickly.

PHP stands for PHP: Hypertext Preprocessor, which gives you a good idea of its core purpose: to process information and produce hypertext (HTML) as a result. (Developers love recursive acronyms, and PHP: Hypertext Preprocessor is a good example of one.)

PHP is a *server - side scripting language*, which means that PHP scripts, or programs, usually run on a Web server. (A good example of a *client - side* scripting language is JavaScript, which commonly runs within a Web browser.) Furthermore, PHP is an *interpreted* language — a PHP script is processed by the PHP engine each time it is run.

The interesting stuff happens when a PHP script runs. Because PHP is so flexible, a PHP script can carry out any number of interesting tasks, such as:

- Reading and processing the contents of a Web form sent by the visitor
- Reading, writing, and creating files on the Web server
- Working with data in a database stored on the Web server
- Grabbing and processing data from other Web sites and feeds
- Generating dynamic graphics, such as charts and manipulated photos

Common examples of PHP scripts include:

- Web forums that allow visitors to post messages and discuss topics
- Search engines that let people search the contents of a Web site or database
- Straw poll scripts that enable visitors to vote in polls and surveys
- Content management systems and blogs, which enable Webmasters to create sites easily with minimal technical knowledge
- Webmail applications, allowing people to send and receive email using their Web browser
- Online stores, allowing shoppers to purchase products and services over the Internet

Common examples of PHP scripts include:

- Web forums that allow visitors to post messages and discuss topics
- Search engines that let people search the contents of a Web site or database
- Straw poll scripts that enable visitors to vote in polls and surveys
- Content management systems and blogs, which enable Webmasters to create sites easily with minimal technical knowledge

- Webmail applications, allowing people to send and receive email using their Web browser
- Online stores, allowing shoppers to purchase products and services over the Internet

Why Use PHP?
One of the best things about PHP is the large number of Internet service providers (ISPs) and Web hosting companies that support it. Today hundreds of thousands of developers are using PHP, and it is not surprising that there are so many, considering that several million sites are reported to have PHP installed.

Another great feature of PHP is that it is *cross - platform* — you can run PHP programs on Windows, Linux, FreeBSD, Mac OS X, and Solaris, among others. What ' s more, the PHP engine can integrate with all common Web servers, including Apache, Internet Information Server (IIS), Zeus, and lighttpd. This means that you can develop and test your PHP Web site on one setup, then deploy it on a different type of system without having to change much of your code.

ASP (Active Server Pages): Launched in 1997, and was one of the first Web application technologies to integrate closely with the Web server, resulting in fast performance. ASP scripts are usually written in VBScript, a language derived from BASIC. This contrasts with PHP ' s more C - like syntax.

ASP.NET: It is actually a framework of libraries that you can use to build Web sites, and you have a choice of languages to use, including C#, VB.NET (Visual Basic), and J# (Java). Because ASP.NET gives you a large library of code for doing things like creating HTML forms and accessing database tables, you can get a Web application up and running very quickly. PHP, although it has a very rich standard library of functions, does not give you a structured framework to the extent that ASP.NET does.

Perl: Perl among first languages for creating dynamic Web pages, through CGI scripting and, later, integrating tightly into Web servers with technologies like the Apache mod_perl module and ActivePerl for IIS. Though Perl is a powerful scripting language, it is harder to learn than PHP. It is also more of a general - purpose language than PHP, although Perl ' s CPAN library includes some excellent modules for Web development.

Java: Java good for building large - scale, robust Web applications using JSP (JavaServer Pages) and servlets. Using Apache Tomcat, you can easily build and deploy Java - based Web sites on virtually any server platform. Compared to PHP Java has quite a steep learning curve, and you have to write lot of code to get even a simple Web site going (though JSP helps a lot in this regard). PHP is a simpler language to learn, and it is quicker to get a basic Web site up and

running. It is harder to find a Web hosting company that will support JSP, whereas nearly all hosting companies offer PHP hosting.

Module-92

Perl: Perl among first languages for creating dynamic Web pages, through CGI scripting and, later, integrating tightly into Web servers with technologies like the Apache mod_perl module and ActivePerl for IIS. Though Perl is a powerful scripting language, it is harder to learn than PHP. It is also more of a general - purpose language than PHP, although Perl ' s CPAN library includes some excellent modules for Web development.

Java: Java good for building large - scale, robust Web applications using JSP (JavaServer Pages) and servlets. Using Apache Tomcat, you can easily build and deploy Java - based Web sites on virtually any server platform. Compared to PHP Java has quite a steep learning curve, and you have to write lot of code to get even a simple Web site going (though JSP helps a lot in this regard). PHP is a simpler language to learn, and it is quicker to get a basic Web site up and running. It is harder to find a Web hosting company that will support JSP, whereas nearly all hosting companies offer PHP hosting.

Python: Many popular sites such as Google and YouTube are built using Python, and Python Web hosting is starting to become much more common (though it is nowhere near as common as PHP hosting). You can even build and host your Python apps on Google ' s server with the Google App Engine. Overall, Python is a very nice language, but PHP is currently a lot more popular, and has a lot more built - in functionality to help with building Web sites.

Ruby: Very popular due to the excellent Ruby on Rails application framework, which uses the Model - View - Controller (MVC) pattern, along with Ruby ' s extensive object - oriented programming features, to make it easy to build a complete Web application very quickly. As with Python, Ruby is fast becoming a popular choice among Web developers, but for now, PHP is much more popular.

ColdFusion: Around since 1995. It is easy to learn, it lets you build Web applications very quickly, and it is really easy to create database - driven sites. Allows you to build complex Flash - based Web applications. ColdFusion ' s main disadvantages compared to PHP include the fact that it is not as popular (so it is harder to find hosting and developers), it is not as flexible as PHP for certain tasks, and the server software to run your apps can be expensive. (PHP and Apache are, of course, free and open source.)

PHP occupies something of a middle. It is not a general - purpose language like Python or Ruby (although it can be used as one). This makes PHP highly suited to its main job: building Web sites. PHP doesn't have a complete Web application framework like ASP.NET or Ruby on Rails,

meaning that you ’ are left to build your Web sites “ from the ground up ” (or use add - on extensions, libraries, and frameworks).

However, this middle ground partly explains the popularity of PHP. The fact that you don ’ t need to learn a framework or import tons of libraries to do basic Web tasks makes the language easy to learn and use. On the other hand, if you need the extra functionality of libraries and frameworks, they ’ re there for you.

Another reason for PHP ’ s popularity is the excellent — and thorough — online documentation available through www.php.net and its mirror sites.

In the past, PHP has been criticized for the way it handled a number of things — for example, one of its main stumbling blocks was the way in which it implemented object support. However, since version 5, PHP has taken stock of the downfalls of its predecessors and, where necessary, has completely rewritten the way in which it implements its functionality. Now more than ever, PHP is a serious contender for large - scale enterprise developments as well as having a large, consolidated base of small - to medium - sized applications.

The Evolution of PHP

Although PHP only started gaining popularity with Web developers around 1998, it was created by Rasmus Lerdorf way back in 1994. PHP started out as a set of simple tools coded in the C language to replace the Perl scripts that Rasmus was using on his personal home page (hence the original meaning of the “ PHP ” acronym). He released PHP to the general public in 1995, and called it PHP version 2.

In 1997, two more developers, Zeev Suraski and Andi Gutmans, rewrote most of PHP and, along with Rasmus, released PHP version 3.0 in June 1998. By the end of that year, PHP had already amassed tens of thousands of developers, and was being used on hundreds of thousands of Web sites.

For the next version of PHP, Zeev and Andi set about rewriting the PHP core yet again, calling it the “ Zend Engine ” (basing the name “ Zend ” on their two names). The new version, PHP 4, was launched in May 2000. This version further improved on PHP 3, and included session handling features, output buffering, a richer core language, and support for a wider variety of Web server platforms.

Although PHP 4 was a marked improvement over version 3, it still suffered from a relatively poor object - oriented programming (OOP) implementation. PHP 5, released in July 2004,

addressed this issue, with private and protected class members; final, private, protected, and static methods; abstract classes; interfaces; and a standardized constructor/destructor syntax.

Module-93

So far you ’ ve looked at what PHP is, and what you can use it for. You ’ ve also written and tested a simple PHP script to give you a feel for how the language works. Now, in these next few modules, you ’ ll build a solid foundation of knowledge that you can use to create more complex applications and Web sites in PHP.

A *variable* is simply a container that holds a certain value. Variables get their name because that certain value can change throughout the execution of the script. For example, consider the following simple PHP script: echo 2 + 2;

This code outputs the number 4 when it is run. This is all well and good; however, if you wanted to print the value of, say, 5 + 6 instead, you ’ d have to write another PHP script, as follows: echo 5 + 6;

This is where variables come into play. By using variables instead of numbers in your script, you make the script much more useful and flexible: echo $x + $y;

A variable consists of two parts: the variable ’ s name and the variable ’ s value. Because you ’ ll be using variables in your code frequently, it is best to give your variables names you can understand and remember. Like other programming languages, PHP has certain rules you must follow when naming your variables:

Variable names are case - sensitive ($Variable and $variable are two distinct variables), so it is worth sticking to one variable naming method — for example, always using lowercase — to avoid mistakes. It is also worth pointing out that variable names longer than 30 characters are somewhat impractical. Here are some examples of PHP variable names:
$my_first_variable
$anotherVariable
$x
$_123

Creating a variable in PHP is known as *declaring* it. Declaring a variable is as simple as using its name in your script: $my_first_variable;

When PHP first sees a variable ’ s name in a script, it automatically creates the variable at that point.

Many programming languages prevent you from using a variable without first explicitly declaring (creating) it. But PHP lets you use variables at any point just by naming them. This is not always the blessing you might think; if you happen to use a nonexistent variable name by mistake, no error message is generated, and you may end up with a hard - to - find bug. In most cases, though, it works just fine and is a helpful feature.

When you declare a variable in PHP, it is good practice to assign a value to it at the same time. This is known as *initializing* a variable. By doing this, anyone reading your code knows exactly what value the variable holds at the time it is created. (If you don ' t initialize a variable in PHP, it is given the default value of null .) Here ' s an example of declaring and initializing a variable:

$my_first_variable = 3;

Looking back at the addition example earlier, the following script creates two variables, initializes them with the values 5 and 6 , then outputs their sum (11):

```
$x = 5;
$y = 6;
echo $x + $y;
```

All data stored in PHP variables fall into one of eight basic categories, known as *data types* . A variable ' s data type determines what operations can be carried out on the variable ' s data, as well as the amount of memory needed to hold the data. PHP supports four scalar data types. *Scalar* data means data that contains only a single value. Here ' s a list of them, including examples:

As well as the four scalar types, PHP supports two compound types. *Compound* data is data that can contain more than one value. The following table describes PHP ' s compound types:

Finally, PHP supports two *special* data types, so called because they don ' t contain scalar or compound data as such, but have a specific meaning:

PHP is known as a *loosely - typed* language. This means that it is not particularly fussy about the type of data stored in a variable. It converts a variable ' s data type automatically, depending on the context in which the variable is used. For example, you can initialize a variable with an integer value; add a float value to it, thereby turning it into a float; then join it onto a string value to produce a longer string. In contrast, many other languages, such as Java, are *strongly - typed* ; once you set the type of a variable in Java, it must always contain data of that type.

PHP ’ s loose typing is both good and bad. On the plus side, it makes variables very flexible; the same variable can easily be used in different situations. It also means that you don ’ t need to worry about specifying the type of a variable when you declare it. However, PHP won ’ t tell you if you accidentally pass around data of the wrong type. For example, PHP will happily let you pass a floating - point value to a piece of code that expects to be working on an integer value. You probably won ’ t see an error message, but you may discover that the output of your script isn ’ t quite what you expected! These types of errors can be hard to track down. (Fortunately, there is a way to test the type of a variable, as you see in a moment.)

You can determine the type of a variable at any time by using PHP ’ s gettype() function. To use gettype() , pass in the variable whose type you want to test. The function then returns the variable ’ s type as a string.

To pass a variable to a function, place the variable between parentheses after the function name — for example, gettype($x) *. If you need to pass more than one variable, separate them by commas.*

The following example shows gettype() in action. A variable is declared, and its type is tested with gettype() . Then, four different types of data are assigned to the variable, and the variable ’ s type is retested with gettype() each time:

The $test_var variable initially has a type of null , because it has been created but not initialized (assigned a value). After setting $test_var ’ s value to 15 , its type changes to integer . Setting $test_var to 8.23 changes its type to double (which in PHP means the same as float , because all PHP floating - point numbers are double - precision). Finally, setting $test_var to “ Hello, world! ” alters its type to string .

In PHP, a floating - point value is simply a value with a decimal point. So if 15.0 was used instead of 15 in the preceding example, $test_var would become a double, rather than an integer.

You can also test a variable for a specific data type using PHP ’ s type testing functions:

Earlier, you learned how to change a variable ’ s type by assigning different values to the variable. However, you can use PHP ’ s settype() function to change the type of a variable while preserving the variable ’ s value as much as possible. To use settype() , pass in the name of the variable you want to alter, followed by the type to change the variable to (in quotation marks).

Here ’ s some example code that converts a variable to various different types using settype() :

To start with, the $test_var variable contains 8.23 , a floating - point value. Next, $test_var is converted to a string, which means that the number 8.23 is now stored using the characters 8 , . (period), 2 , and 3 . After converting $test_var to an integer type, it contains the value 8 ; in other words, the fractional part of the number has been lost permanently. You can see this in the next two lines, which convert $test_var back to a float and display its contents. Even though $test_var is a floating - point variable again, it now contains the whole number 8 . Finally, after converting $test_var to a Boolean, it contains the value true (which PHP displays as 1). This is because PHP converts a non - zero number to the Boolean value true .

vulearninghub.xyz | by VU Learning HUB

Module-94

You can also cause a variable ' s value to be treated as a specific type using a technique known as *type casting* . This involves placing the name of the desired data type in parentheses before the variable ' s name. Note that the variable itself remains unaffected; this is in contrast to settype(), which changes the variable ' s type. In the following example, a variable ' s value is cast to various different types at the time that the value is displayed: […]

Note that $test_var ' s type isn ' t changed at any point; it remains a floating - point variable, containing the value 8.23, at all times. All that changes is the type of the data that ' s passed to the echo statement. Here's the full list of casts that you can use in PHP: […]

You can also cast a value to an integer, floating - point, or string value using three PHP functions […]

Why would you want to change a variable's type with settype(), or change a value's type with casting? Most of the time, PHP's loose typing handles type conversion automatically, depending on the context in which variables and values used. However, forcing a variable to be of a certain type is useful for security reasons; if you're expecting to pass a user- entered integer value to a database, it is a good idea to cast the value to an integer, just to make sure the user really did enter an integer. Likewise, if passing data to another program, that expects data to be in string format, you can cast the value to a string before you pass it.

So far you've learned what variables are, and how to set a variable to a particular value, as well as how to retrieve a variable ' s value and type. However, life would be pretty dull if this was all you could do with variables. This is where operators come into play. Using an operator, you can manipulate the contents of one or more variables to produce a new value. For example, this code uses the addition operator (+) to add the values of $x and $y together to produce a new value:

```
echo $x + $y;
```

So an *operator* is a symbol that manipulates one or more values, usually producing a new value in the process. Meanwhile, an *expression* in PHP is anything that evaluates to a value; this can be any combination of values, variables, operators, and functions. In the preceding example, $x + $y is an expression. Here are some more examples of expressions:

```
$x + $y + $z
$x - $y
```

$x
5
true
gettype($test_var)

The values and variables that are used with an operator are known as *operands* .

Operators in PHP can be grouped into ten types, as follows:
[…]

In the next few modules we will explore the most frequently used PHP operators.

In PHP, the arithmetic operators (plus, minus, and so on) work much as you would expect, enabling you to write expressions as though they were simple equations. For example, $c = $a + $b adds $a and $b and assigns the result to $c . Here ’ s a full list of PHP ’ s arithmetic operators:
[…]

You ’ ve already seen how the basic assignment operator (=) can be used to assign a value to a variable:
$test_var = 8.23;
It is also worth noting that the preceding expression evaluates to the value of the assignment: 8.23. This is because the assignment operator, like most operators in PHP, produces a value as well as carrying out the assignment operation. This means that you can write code such as:
$another_var = $test_var = 8.23; which means: “ Assign the value 8.23 to $test_var , then assign the result of that expression (8.23) to $another_var . ” So both $test_var and $another_var now contain the value 8.23 .

The equals sign (=) can be combined with other operators to give you a *combined assignment operator* that makes it easier to write certain expressions. The combined assignment operators (such as +=, – =, and so on) simply give you a shorthand method for performing typical arithmetic operations, so that you don ’ t have to write out the variable name multiple times. For example, you can write: […]

This also works for other kinds of operators. For example, the concatenation operator (described later in this chapter) can be combined with the equals sign (as .=), causing the value on the right side to be appended to the existing value on the left, like this: […]

PHP ’ s bitwise operators let you work on the individual bits within integer variables. Consider the integer value 1234. For a 16 - bit integer, this value is stored as two bytes: 4 (the most significant byte) and 210 (the least significant). 4 * 256 + 210 = 1234.
Here ’ s how those two bytes look as a string of bits:

00000100 11010010

A bit with a value of 1 is said to be *set* , whereas a bit with a value of 0 is *unset* (or not set).

PHP ’ s bitwise operators let you manipulate these bits directly, as shown in the following table.[…]

Each example includes both decimal values and their binary equivalents, so you can see how the bits are altered:

You can see that ~ (Not) inverts all the bits in the number. Notice that there are 32 bits in each value, because PHP uses 32 - bit integers. (The other examples show only the last 8 bits of each value, for brevity.) The resulting bit values (11111111111111111111111111110001 *) represent – 15, because PHP uses the* two’s complement *system to represent negative numbers*

vulearninghub.xyz | by VU Learning HUB

Module-95

As you might imagine from the name, *comparison operators* let you compare one operand with the other in various ways. If the comparison test is successful, the expression evaluates to true ; otherwise, it evaluates to false . You often use comparison operators with decision and looping statements such as if and while. Here ’ s a list of the comparison operators in PHP: […]

The following examples show comparison operators in action:

[…]

As you can see, comparison operators are commonly used to compare two numbers (or strings converted to numbers). The = = operator is also frequently used to check that two strings are the same.

Oftentimes it is useful to add or subtract the value 1 (one) over and over. This situation occurs so frequently — for example, when creating loops — that special operators are used to perform this task: the increment and decrement operators. They are written as two plus signs or two minus signs, respectively, preceding or following a variable name, like so:

[…]

The location of the operators makes a difference. Placing the operator before the variable name causes the variable ’ s value to be incremented or decremented before the value is returned; placing the operator after the variable name returns the current value of the variable first, then adds or subtracts one from the variable. For example:

[…]

Interestingly, you can use the increment operator with characters as well. For example, you can “ add ” one to the character B and the returned value is C. However, you cannot subtract from (decrement) character values.

PHP ’ s logical operators work on Boolean values. Before looking at how logical operators work, it is worth taking a bit of time to explore Boolean values more thoroughly.

As you ’ ve already seen, a Boolean value is either true or false . PHP automatically evaluates expressions as either true or false when needed, although as you ’ ve already seen, you can use

settype() or casting to explicitly convert a value to a Boolean value if necessary. For example, the following expressions all evaluate to true :
1
1 == 1
3 > 2
“hello” != “goodbye”

The following expressions all evaluate to false :
3 < 2
gettype(3) == “array”
“hello” == “goodbye”

In addition, PHP considers the following values to be false :
[…]
All other values are considered true in a Boolean context.

PHP ’ s logical operators work on Boolean values. Before looking at how logical operators work, it is worth taking a bit of time to explore Boolean values more thoroughly.

As you ’ ve already seen, a Boolean value is either true or false . PHP automatically evaluates expressions as either true or false when needed, although as you ’ ve already seen, you can use settype() or casting to explicitly convert a value to a Boolean value if necessary. For example, the following expressions all evaluate to true :
1
1 == 1
3 > 2
“hello” != “goodbye”

The following expressions all evaluate to false :
3 < 2
gettype(3) == “array”
“hello” == “goodbye”

In addition, PHP considers the following values to be false :
[…]
All other values are considered true in a Boolean context.

Here are some simple examples of logical operators in action:

vulearninghub.xyz | by VU Learning HUB

[…]

The main use of logical operators and Boolean logic is when making decisions and creating loops. You ’ re probably wondering why the and and or operators can also be written as & & and || . The reason is that and and or have a different precedence to & & and || . Operator precedence is explained in a moment.

Module-96

There ’ s really only one string operator, and that ’ s the *concatenation operator* , . (dot). This operator simply takes two string values, and joins the right - hand string onto the left - hand one to make a longer string.

For example:
echo “Shaken, “ . “not stirred”; // Displays “ Shaken, not stirred ”

You can also concatenate more than two strings at once. Furthermore, the values you concatenate don’t have to be strings; thanks to PHP ’ s automatic type conversion, non - string values, such as integers and floats, are converted to strings at the time they ’ re concatenated:

```
$tempF = 451;
// Displays “ Books catch fire at 232.777777778 degrees C. ”
echo “Books catch fire at “ . ( (5/9) * ($tempF - 32) ) . “ degrees C.”;
```

In fact, there is one other string operator — the combined assignment operator .= — which was mentioned earlier in the chapter. It is useful when you want to join a new string onto the end of an existing string variable. For example, the following two lines of code both do the same thing — they change the string variable $x by adding the string variable $y to the end of it:

```
$x = $x . $y;
$x .= $y;
```

For 3 + 4 , it is clear what needs to be done (in this case, “ add 3 and 4 to produce 7 ”). But with more than one operator in an expression, however, things aren ’ t so clear - cut. Consider the following example: 3 + 4 * 5 i.e. 23 or 35, what should PHP do? This is where operator precedence comes into play.

All PHP operators are ordered according to precedence. An operator with a higher precedence is executed before an operator with lower precedence. In this example, * has a higher precedence than + , so PHP multiplies 4 by 5 first, then adds 3 to the result to get 23. Here ’ s a partial list of operators in order of precedence (highest first):

Order of execution of operators in an expression can be effected by using parentheses. Placing parentheses around an operator and its operands forces operator take highest precedence. So, for example, the following expression evaluates to 35: (3 + 4) * 5

PHP has two logical “and ” operators (&& , and) and two logical “or ” operators (|| , or). You can see in the previous table that && and || have a higher precedence than and and or . In fact, and and or are below even the assignment operators. This means that you have to be careful when using and and or . For example:

$x = false || true; // $x is true
$x = false or true; // $x is false

In the first line, false || true evaluates to true , so $x ends up with the value true , as you ’ d expect.

However, in the second line, $x = false is evaluated first, because = has a higher precedence than or . By the time false or true is evaluated, $x has already been set to false .

Because of the low precedence of the and and or operators, it is generally a good idea to stick with && and || unless you specifically need that low precedence.

You can also define value - containers called *constants* in PHP. The values of constants, as their name implies, can never be changed. Constants can be defined only once in a PHP program.

Constants differ from variables in that their names do not start with the dollar sign, but other than that they can be named in the same way variables are. However, it is good practice to use all - uppercase names for constants. In addition, because constants don ’ t start with a dollar sign, you should avoid naming your constants using any of PHP ’ s reserved words, such as statements or function names. For example, don ’ t create a constant called ECHO or SETTYPE . If you do name any constants this way, PHP will get
very confused!

Constants may only contain scalar values such as Boolean, integer, float, and string (not values such as arrays and objects), can be used from anywhere in your PHP program without regard to variable scope, and are case - sensitive.

To define a constant, use the define() function, and include inside the parentheses the name you’ve chosen for the constant, followed by the value for the constant, as shown here: […]

Constants are useful for any situation where you want to make sure a value does not change throughout the running of your script. Common uses for constants include configuration files and storing text to display to the user.

Module-97

One of the most common ways to receive input from the user of a Web application is via an HTML form. You ' ve probably filled in many HTML forms yourself. Common examples include contact forms that let you email a site owner; order forms that let you order products from an online store; and Web - based email systems that let you send and receive email messages using your Web browser.

- action tells the Web browser where to send the form data when the user fills out and submits the form. This should either be an absolute URL or a relative URL. The script at the specified URL should be capable of accepting and processing the form data; more on this in a moment.

- method tells the browser how to send the form data. You can use two methods: get is useful for sending small amounts of data and makes it easy for the user to resubmit the form, and post can send much larger amounts of form data.

When you submit your form, and see all of the form data in your browser's address bar, it means your form used the get method, which sends the form data in the URL. You can see that the form data is preceded by a ? character, and that the data for each form field is sent as a name/value pair:
http://localhost/web_form.html?textField=Hello&passwordField=secret& ...

The get method is limited in the amount of data it can send, because a URL can only contain a small number of characters (1,024 characters is a safe upper limit). If you need to send larger amounts of data from a form, use the post method instead:

<form action="myscript.php" method="post">

The post method sends the data within the HTTP headers of the request that's sent to the server, rather than embedding the data in the URL. This allows a lot more data to be sent. If the users try to refresh the page after sending a form via the post method, their browser usually pops up a dialog box asking them if they want to resend their form data.

In this example, you create a Web form that contains a variety of form fields. Not only will you learn how to create the various types of form fields, but you can see how the fields look and work in your Web browser.

Save the following file as web_form.html in your document root folder, then open it in your browser to see the form:

The created form fields include:

A text input field —- This allows the user to enter a single line of text. You can optionally prefill the field with an initial value using the value attribute (if you don't want to do this, specify an empty string for the value attribute, or leave the attribute out altogether):
[…]

A password field — This works like a text input field, except that the entered text is not displayed. This is, of course, intended for entering sensitive information such as passwords. Again, you can prefill the field using the value attribute, though it's not a good idea to do this because the password can then be revealed by viewing the page source in the Web browser:
[…]

The created form fields include:

A checkbox field — This is a simple toggle; it can be either on or off. The value attribute should contain the value that will be sent to the server when the checkbox is selected (if the checkbox isn't selected, nothing is sent):

You can preselect a checkbox by adding the attribute checked="checked" *to the* input *tag —- for example:* <input type="checkbox" checked="checked" ... />.
By creating multiple checkbox fields with the same name *attribute, you can allow the user to select multiple values for the same field. (You learn how to deal with multiple field values in PHP later in this chapter.)*

Two radio button fields — Radio buttons tend to be placed into groups of at least two buttons. All buttons in a group have the same name attribute. Only one button can be selected per group. As with checkboxes, use the value attribute to store the value that is sent to the server if the button is selected. Note that the value attribute is mandatory for checkboxes and radio buttons, and optional for other field types:

A submit button — Clicking this type of button sends the filled-in form to the server-side script for processing. The value attribute stores the text label that is displayed inside the button (this value is also sent to the server when the button is clicked):

[…]

A reset button — This type of button resets all form fields back to their initial values (often empty). The value attribute contains the button label text:

[…]
A file select field — This allows the users to choose a file on their hard drive for uploading to the server (see “Creating File Upload Forms” in the later modules). The value attribute is usually ignored by the browser:

[…]

A hidden field — This type of field is not displayed on the page; it simply stores the text value specified in the value attribute. Hidden fields are great for passing additional information from the form to the server, as you see later in the chapter:

[…]

An image field — This works like a submit button, but allows you to use your own button graphic instead of the standard gray button. You specify the URL of the button graphic using the src attribute, and the graphic’s width and height (in pixels) with the width and height attributes. As with the submit button, the value attribute contains the value that is sent to the server when the button is clicked:
[…]

A push button — This type of button doesn’t do anything by default when it’s clicked, but you can make such buttons trigger various events in the browser using JavaScript. The value attribute specifies the text label to display in the button:
[…]

A pull-down menu — This allows a user to pick a single item from a predefined list of options. The size attribute’s value of 1 tells the browser that you want the list to be in a pull-down menu format. Within the select element, you create an option element for each of your options. Place the option label between the <option> ... </option> tags. Each option element can have an optional value attribute, which is the value sent to the server if that option is selected. If you don’t include a value attribute, the text between the <option> ... </option> tags is sent instead:

[…]

vulearninghub.xyz | by VU Learning HUB

A list box — This works just like a pull-down menu, except that it displays several options at once. To turn a pull-down menu into a list box, change the size attribute from 1 to the number of options to display at once:

[…]

A multi-select list box — This works like a list box, but it also allows the user to select multiple items at once by holding down Ctrl (on Windows and Linux browsers) or Command (on Mac browsers). To turn a normal list box into a multi-select box, add the attribute multiple (with a value of “multiple“) to the select element. If the user selects more than one option, all the selected values are sent to the server (you learn how to handle multiple field values later in the chapter):
[…]

A text area field — This is similar to a text input field, but it allows the user to enter multiple lines of text. Unlike most other controls, you specify an initial value (if any) by placing the text between the <textarea> ... </textarea> tags, rather than in a value attribute. A textarea element must include attributes for the height of the control in rows (rows) and the width of the control in columns (cols):
[…]

Module-98

You now know how to create an HTML form, and how data in a form is sent to the server. How do you write a PHP script to handle that data when it arrives at the server?
First of all, the form ’ s action attribute needs to contain the URL of the PHP script that will handle the form. For example:

< form action=”form_handler.php” method=”post” >

Next, of course, you need to create the form_handler.php script. When users send their forms, their data is sent to the server and the form_handler.php script is run. The script then needs to read the form data and act on it. All this will be covered in the module.

To read the data from a form, you use a few superglobal variables. A superglobal is a built - in PHP variable that is available in any scope: at the top level of your script, within a function, or within a class method. For example, $GLOBALS superglobal array, which contains a list of all global variables used in your applications. Here, you learn about three new superglobal arrays:
[…]
Each of these three superglobal arrays contains the field names from the sent form as array keys, with the field values themselves as array values. For example, say you created a form using the get method, and that form contained the following control:

< input type=”text ” name=”emailAddress” value=”” / >

You could then access the value that the user entered into that form field using either the $_GET or the $_REQUEST superglobal:

$email = $_GET[“emailAddress”];
$email = $_REQUEST[“emailAddress”];

In this example, you create a simple user registration form, then write a form handler script that reads the field values sent from the form and displays them in the page.

First, create the registration form. Save the following HTML code as registration.html in your document root folder:

One of the most common ways to receive input from the user of a Web application is via an HTML form. You ’ ve probably filled in many HTML forms yourself. Common examples include contact forms that let you email a site owner; order forms that let you order products from an

online store; and Web - based email systems that let you send and receive email messages using your Web browser.

One of the most common ways to receive input from the user of a Web application is via an HTML form. You ’ ve probably filled in many HTML forms yourself. Common examples include contact forms that let you email a site owner; order forms that let you order products from an online store; and Web - based email systems that let you send and receive email messages using your Web browser.

Module-99

You learned in the prior modules that you can create form fields that send multiple values, rather than a single value. For example, in this module we will discuss form fields that are capable of sending multiple values to the server:

The first form field is a multi - select list box, allowing the user to pick one or more (or no) options. The second two form fields are checkboxes with the same name (newsletter) but different values (widgetTimes and funWithWidgets). If the user checks both checkboxes then both values, widgetTimes and funWithWidgets , are sent to the server under the newsletter field name.

So how can you handle multi - value fields in your PHP scripts? The trick is to add square brackets ([]) after the field name in your HTML form. Then, when the PHP engine sees a submitted form field name with square brackets at the end, it creates a nested array of values within the $_GET or $_POST (and $_REQUEST) superglobal array, rather than a single value. You can then pull the individual values out of that nested array. So you might create a multi - select list control as follows:

< select name=”favoriteWidgets[]” id=”favoriteWidgets” size=”3”
multiple=”multiple” ... < /select >
You ’ d then retrieve the array containing the submitted field values as follows:

$favoriteWidgetValuesArray = $_GET[“favoriteWidgets”]; // If using get method
$favoriteWidgetValuesArray = $_POST[“favoriteWidgets”]; // If using post method)

As before, fill out the form, and try selecting a couple of the “favorite widget” options and both “newsletter” checkboxes. Now submit the form. Notice how the PHP script handles the multi-value fields. You can see a sample form here and the resulting script output in next slide.

The Web form, registration_multi.html, is largely similar to the previous registration.html page. However, this form contains a multi-select list box (favoriteWidgets) and two checkboxes with the same name (newsletter). Because these controls are capable of sending multiple values, two empty square brackets ([]) are appended to the field names:

The square brackets tell the PHP engine to expect multiple values for these fields, and to create corresponding nested arrays within the relevant superglobal arrays ($_POST and $_REQUEST in this case).

The form handler, process_registration_multi.php, displays the user's submitted form data in the page. Because most fields contain just one value, it's simply a case of displaying the relevant $_POST values using the echo() statement.

For the multi-value fields, however, the script needs to be a bit smarter. First it creates two empty string variables to hold the list of field values to display:

```
$favoriteWidgets = "";
$newsletters = "";
```

Next, for the favoriteWidgets field, the script checks to see if the corresponding $_POST array element ($_POST["favoriteWidgets"]) exists. (Remember that, for certain unselected form controls such as multi-select lists and checkboxes, PHP doesn't create a corresponding $_POST/$_GET/$_REQUEST array element.) If the $_POST["favoriteWidgets"] array element does exist, the script loops through each of the array elements in the nested array, concatenating their values onto the end of the $favoriteWidgets string, along with a comma and space to separate the values:

If any field values were sent for these fields, the resulting strings now have a stray comma and space on the end, so the script uses a regular expression to remove these two characters, tidying up the strings:

```
$favoriteWidgets = preg_replace( "/, $/", "", $favoriteWidgets );
$newsletters = preg_replace( "/, $/", "", $newsletters );
```

Now it's simply a case of outputting these two strings in the Web page, along with the other single-value fields:

Module-100

The following all-in-one PHP script does the following things:

- It displays a registration form for the user to fill out. Certain fields are required to be filled in; these are labeled with asterisks in the form. The remaining fields are optional
- When the form is sent, the script checks that the required fields have been filled in
- If all required fields are filled, the script displays a thank-you message
- If one or more required fields are missing, the script redisplays the form with an error message, and highlights the fields that still need to be filled in. The script remembers which fields the user already filled in, and prefills those fields in the new form

Now browse the script's URL in your Web browser. You'll see a blank registration form. Try submitting an empty form by clicking Send Details. You should see an error message, with the missing required fields highlighted. If you fill in some values and resubmit, the script keeps checking to see if you've filled in the required fields. If not, it redisplays the form, including any data you've already entered, and highlights the missing fields, as shown in Figure

The script kicks off with the standard XHTML page header. It includes an additional CSS class for the red error boxes:
[…]

Next, the script checks to see if the form has been submitted. It does this by looking for the existence of the submitButton field. If present, it means that the Send Details button has been clicked and the form received, and the script calls a processForm() function to handle the form data. However, if the form hasn't been displayed, it calls displayForm() to display the blank form, passing in an empty array (more on this in a moment):
[…]

Next the script defines some helper functions. validateField() is used within the form to display a red error box around a form field label if the required field hasn't been filled in. It's passed a field name, and a list of all the required fields that weren't filled in. If the field name is within the list, it displays the markup for the error box:
[…]

setValue() is used to prefill the text input fields and text area field in the form. It expects to be passed a field name. It then looks up the field name in the $_POST superglobal array and, if found, it outputs the field ' s value:
[…]

setChecked() is used to preselect checkboxes and radio buttons by inserting a checked attribute into the element tag. Similarly, setSelected() is used to preselect an option in a select list via the selected attribute. Both functions look for the supplied field name in the $_POST array and, if the field is found and its value matches the supplied field value, the control is preselected:
[…]

Next comes the form handling function, processForm() . This sets up an array of required field names, and also initializes an array to hold the required fields that weren ' t filled in:
[…]

Now the function loops through the required field names and looks for each field name in the $_POST
array. If the field name doesn ' t exist, or if it does exist but its value is empty, the field name is added to
the $missingFields array:

If missing fields were found, the function calls the displayForm() function to redisplay the form, passing in the array of missing field names so that displayForm() can highlight the appropriate fields. Otherwise, displayThanks() is called to thank the user:
[…]

The displayForm() function itself displays the HTML form to the user. It expects an array of any missing required field names. If this array is empty, the form is presumably being displayed for the first time, so displayForm() shows a welcome message. However, if there are elements in the array, the form is being redisplayed because there were errors, so the function shows an appropriate error message:
[…]

Next, the form itself is displayed. The form uses the post method, and its action attribute points back to the script ' s URL:
[…]

Then each form control is created using HTML markup. Notice how the validateField() , setValue() , setChecked() , and setSelected() functions are called throughout the markup in order to insert appropriate attributes into the elements.

With the password fields, it is unwise to redisplay a user ’ s password in the page because the password can easily be read by viewing the HTML source. Therefore, the two password fields are always redisplayed as blank. The script checks to see if the form is being redisplayed due to missing required field values; if so, the password field labels are highlighted with the red error boxes to remind the users to reenter their password:
[…]

vulearninghub.xyz | by VU Learning HUB

Module-101

You learned in the prior modules that you can create form fields that send multiple values, rather than a single value. For example, in this module we will discuss form fields that are capable of sending multiple values to the server:

You can use hidden fields to create a series of forms that guide the user through the data entry process step by step. Within each form, you can store the current step — so that the script knows what stage the user has reached — as well as the data already entered by the user in other steps.

- Here's an example that splits the previous registration.php form into three steps:
- First name/last name
- Gender/favorite widget
- Newsletter preference/comments

Save the following script as registration_multistep.php in your document root folder and run the script in your Web browser. Try filling in some field values and using the Back and Next buttons to jump between the three steps. Notice how the field values are preserved when you return to a previously completed step. Figure 9-8 shows the first step of the form, and Figure 9-9 shows the second step.

To keep things simple, this script doesn't validate any form fields in the way that registration.php *does. However, you could easily use the same techniques used in* registration.php *to validate eachstep of the form as it is submitted.*

For each step of the signup process, the script displays a form with a hidden field, step, to track the current step. For example:

<input type="hidden" name="step" value="1" />

The script starts by testing for the presence of this field in the submitted form data. If found, and its value is valid (between 1 and 3), the script uses PHP's call_user_func() function to call the appropriate processing function — processStep1(), processStep2(), or processStep3(). If the step field wasn't submitted (or its value was invalid), the script assumes the user has just started the signup process and displays the form for the first step:

```
if ( isset( $_POST["step"] ) and $_POST["step"] >= 1 and $_POST["step"] <= 3 ) {
call_user_func( "processStep" . (int)$_POST["step"] );
```

```
} else {
displayStep1();
}
```

The next three functions — setValue(), setChecked(), and setSelected() — are identical to their counterparts in registration.php.
Next come the three functions to process the forms submitted from each of the three steps.
processStep1() simply displays step 2:

```
function processStep1() {
displayStep2();
}
```

processStep2() checks to see if the user clicked the Back button. If he did, step 1 is redisplayed; otherwise it is assumed the user clicked the Next button, so step 3 is displayed:

```
function processStep2() {
if ( isset( $_POST[“submitButton”] ) and $_POST[“submitButton”] ==
“ < Back” ) {
displayStep1();
} else {
displayStep3();
}
}
```

The remaining four functions — displayStep1() , displayStep2() , displayStep3() , and displayThanks() — display forms for each of the three steps in the signup process, as well as the final thank - you page. Notice that each of the step functions includes all of the form fields for the entire signup process; the fields for the current step are displayed as normal, and the fields for the other two steps are displayed as hidden fields. For example, displayStep2() outputs hidden fields to store the values for firstName , lastName , newsletter , and comments , while displaying the fields for the current step (gender and favoriteWidget):
[…]

By including (and populating) all the fields — whether visible or hidden — in each of the three steps, the script ensures that the entire signup data is sent back to the server each time a form is submitted, thereby allowing the data to be carried across the three steps.

Steps 2 and 3 also include Back and Next buttons, whereas step 1 just includes a Next button. Finally, displayThanks() simply displays the thank - you message to the user.

TOPIC-10: SQL on the web

Module-102

As pointed out earlier, there are four primary ways of using databases to drive Web sites. Each of these has different database needs.

In this case, you often have a relatively large amount of data. The database may have been created for purposes other than Web publication, and it may have complex relationships among its data elements. Your job is often to facilitate access to such an existing database (or to parts thereof). These projects have some typical database characteristics:

- These are often large databases.
- Often the database predates the Web project. The mission of the Web site is to allow access to an already existing database.
- Queries are likely to be at least partly generated by users, instead of providing buttons on your Web site that automatically retrieve all records related to zebras, you are likely to provide a form into which users can type zebra, snake, ice cream, or whatever they want to search on.
- Since queries are not predictable, you may need to manage special cases including massive successful results (”l4,235,632 records found”), repeated failures due to user error (”no entries for zebra”), and counter intuitive results (”zebra see under circulation”). in such cases, the messenger--the Web site—is often blamed.
- Furthermore, you need to be able to pass information about unsuccessful queries on to the user if you cannot handle them automatically. If the user has constructed a bizarre query, there is more likelihood of the user’s being able to handle the resulting mess than there is of your HTML page being able to do so.

Here, you are typically concerned with a relatively small amount of data that is relatively simple—interoffice messages, bulletin boards, etc.

Among the database characteristics involved with using databases to share data are the following:

- As part of such a project, you often design and implement the database.

- The database is likely to be quite simple: each entry is likely to be one item (a message, a bulletin board posting, etc.), and the other fields in the database are supporting information (date entered, recipient, subject, etc.)

- Queries are likely to be predictable (by date, subject, and a free-form scan of text within messages) and can be implemented on your Web site with buttons rather than data entry fields.

This may entail aspects of both of the previous cases: You may interact with a large customer or inventory database as well as with a separate database containing sales information which components can be put together with one another, advertising copy and images, etc.

In addition to those issues, database used for e-commerce web sites often need to address these characteristics:

- They need to implement security with regard to all aspects of their operations.
- They need to manage transactions—multiple database accesses that must be treated as a single unit (such as a sale).
- Their operations need to be recoverable in the case of hardware or software failures after a customer believes that a transaction has been completed.

In these situations, the fact that you are using a database is usually irrelevant (and often invisible) to the user. As a result you have several special points to consider:

- Performance must be as fast as possible. (Since then no clear benefit to the user for using database, it is inappropriate to ask the user to suffer from more sluggish performance than would be the case if a database were not used.)

- You generally design and create our own database; you can implement it in whatever way is easiest for you (and most convenient for driving the Web site).

- The database should be hidden; this means that no strange error messages (”unfermented primary data base key found") should appear before the user’s eyes.
- In most cases, your database—driven Web site needs a reliable, fast database manager; your demands will probably not be particularly great. Except in the case in which you publish a database and let users do their own searches, you will not wind up using complex outer joins, recursive table expressions, or sensitive cursors. (Even in these cases, you will likely not be involved-you will let users type in their complex outer joins, recursive table expressions, or sensitive cursors and let them have it out directly with the database.)
-
- What you need to worry about are the basics of databases. If you think of databases as little more than automated file storage mechanisms, that is usually sufficient. (For some

vulearninghub.xyz | by VU Learning HUB

reason, the word "database" scares many otherwise reasonable people. They have decided at some point in their lives that databases are too complicated for them to understand, and they tune out whenever the subject comes up. This is an irrational reaction.)

- **You will also need to worry about the issues involved in physically placing your databases on your Web site and issues involved**

vulearninghub.xyz | by VU Learning HUB

Module-103

There is a specific (if changing) conceptual model underlying today's databases—the relational model. Databases differ from some other contemporary software in that they are based on such a model—and it is a model that is studied extensively in universities and other institutions that are not necessarily engaged in the development of commercial products. (There is absolutely nothing wrong with the development of commercial products; however, there is a distinction between a conceptual foundation based on mathematics and computer science and one based on marketing strategies.)

You can sum up the relational model in two simple points:

1. All data in a relational database is stored in tables—two-dimensional structures with rows and columns (like spreadsheets).

2. Everything that you do to a table results in another table (which—if it has a single column and row may contain one data value).

From these two simple points, a number of important consequences ensue (including many books, a wide variety of products and careers, and a large number of dissertations).
There are three parts to the relational model, two of which matter to you:

1. Objects-the database structures in which data is stored.
2. Integrity—rules governing the content of the database. (This is the part that you can safely ignore.)
3. Operators—the commands that manipulate the data in the database.

These parts of the relational model are discussed briefly in the next modules.

Data in a relational database is stored in tables. Tables are the most important objects in the relational model—-except they are called **relations**.

By convention, each row of the table represents one observation or data record (such as one individual's data). It is called a **tuple**.

Each column of the table contains the data for a field—a specific type of data (such as age or telephone number) for each of the rows. Each column or field is called an **attribute**

In most tables one (or more) of the columns contains identifying information a value that is unique for each row. This might be a persons social security number, a name (although names are often not unique), a serial number, or any other information that uniquely identifies that row. This is called a **primary key**.

Should you wish to complete your overview of the objects in the relational model, you can add these definitions:

- A domain is the universe of legal values for an attribute (or field or column). All valid part numbers, all valid social security numbers, and so forth are valid domains. Domains are a concept that is implemented only indirectly in most relational databases—but it is' a very important concept when it comes to ensuring data integrity. (”Green” is not normally considered to be in the domain of birth dates.)
- The number of rows in a table (relation) is its cardinality.
- The number of columns in a table (relation) is its degree.

This part of the relational model deals with rules of integrity that relate to the use of keys to identify (and therefore to retrieve) rows of data within a table. You may hear talk of unique keys, candidate keys, foreign keys, and referential integrity: all refer to database integrity and to the identification of records within tables.
For the purposes of almost every database that you deal with on a Web site, it is sufficient to identify a unique key for each row—some value that can identify that row when you need to locate it.

Operators are of two types in the relational model: algebraic and calculus‘ For your purposes, it is sufficient to remember that all operations on tables (relations) in the relational model produce other tables. The resultant tables may be larger than the original table (as is the case when you combine several tables into one), or they may be smaller (as is the case when the resultant table has a single row and column—one value).

In the cases where you create databases to drive your Web site, it is useful to know something about normalization—the process of structuring a database’s tables in such a way that they are most efficient. Normalization has a significant basis in database theory; database software is usually optimized to handle the processes needed to store and retrieve normalized data.
There are five forms of normalization. You probably normalize data now without knowing it; you need not normalize your data (and if you do, you need not say that you have done so). Nevertheless, it is useful to know the correct terminology. lt is very easy to set up all of the database fields shown in this module.

Imagine a database record layout that lets you store your appointments for a given day as shown in Table.

Module-104

There is a specific (if changing) conceptual model underlying today's databases—the relational model. Databases differ from some other contemporary software in that they are based on such a model—and it is a model that is studied extensively in universities and other institutions that are not necessarily engaged in the development of commercial products. (There is absolutely nothing wrong with the development of commercial products; however, there is a distinction between a conceptual foundation based on mathematics and computer science and one based on marketing strategies.)

Imagine a database record layout that lets you store your appointments for a given day as shown in TABLE-1. This may look simple (and in fact it is the type of database layout that almost everyone creates when they are starting). Note, however, that there is a repeating group of fields—the time, title: and location of each appointment. This is almost always a problem for the following reasons:

- Since each record contains all of the fields, if you ever need as many as 10 appointments for one day, every record must have space for 10 appointments. This can be an enormous waste of space, as in TABLE-1.
- It is very difficult to use search mechanisms efficiently on this structure. If you are looking for a specific appointment, you wind up searching for a condition in which Appointment 1 name = X or Appointment 2 name = X, etc.
- Getting around this problem results in most imaginative ideas by users and database designers. In this example given, typical work—arounds include having two records for the same day (so as to double the limit of appointments), using two names for one person (for the same reason), piggybacking two appointments in one entry etc.

The correct implementation of this structure is to use two related tables as in TABLE-2 which are linked by a sequence number that is unique for each record in the Name /Date database. Note that this structure doesn't waste space—there are only as many appointments for a given person on a given date as actually exist (if any). Furthermore, there is no limit to the number of appointments that can exist.

Violating the first normal form is the most common mistake that people make in designing databases. However remember two important points:

1. Not all databases support relations such as that used to link the sequence number in the two databases shown in TABLE-2 (last slide). If you are working with old databases—or databases

derived from old databases do not automatically assume that people didn't knew what they were doing.

2. The definition of a repeating group is far from clear. Table-3 shows two sets of data fields. Do you think that the fields in the first column represent a repeating group? The same information can be presented in a way that appears to be a repeating group (the first column) and that appears to be unique data fields (the second column). This is a common situation.
In general, the trade-off is simply stated: repeated groups take up more disk space and are harder to work with, while nonrepeating groups use slightly more processing time (to perform the relationship) and use less disk space. Although you may need to do some experiments for yourself in very time-critical applications, in most cases the cost of normalizing data is well worth it.

The most common reason given for violating this rule is efficiency: "Why should I have to look up the name from another database, when I can just as easily keep a copy in the Appointment database?" The answer is that as soon as you have redundant data in your design, you need to worry about keeping it synchronized. Violation of the second normal form is what lets you have two different names at your bank, two different account numbers for a single credit card, or a multitude of different mailing addresses for a single mass marketer.

There is a reason for violating this rule—and it is a very important consideration. The advantages of storing each data value once (and only once) include not only reduced storage space but also the fact that a change to the data value—such as a change of address or change of name—need be entered only once; it is propagated through the entire database project as various related databases pick up the revised data.

If you are using related data in an invoice database, it is important that the values not change after the invoice has been accepted. A change to your mailing address a week
after you have ordered an item should not affect the records of a shipment that has already been dispatched. More important, a change in the price of an item that you have ordered and paid for should not be reflected on the invoice when it is subsequently printed and stuffed in the carton in which the goods are shipped.

There may be relationships within the data that you store in a database record (the most common is the relationship between a ZIP or postal code and a town name). In this case, data is dependent on (varies in accordance with) another field in the record. Storing both elements is redundant and wasteful; in addition, it can lead to incomplete data in the database.

In a database that stores both postal code and town names, you can search for one and find the other—given the postal code you can find the town and vice versa. This is often useful, but it works only for the towns and postal codes that are used in addresses in the database. lf you order merchandise over the telephone, you will learn that most telemarketers have a separate database of all postal codes for all towns regardless of which codes and towns are used in their customer database; you provide your postal code and they confirm the town that you are in.
Violating the third normal form can create databases in which people rely on incomplete data. On the other hand, violating the third normal form when the data items are very small (as in postal codes) can sometimes simplify the design of a database project sufficiently that is worth the risk.

Fourth normal form requires that there not be two types of relationships within the same database. Example meeting with two people at the same place and time i.e. a joint meeting or conflict of two meetings; thus this must be two separate and conflicting meetings.

vulearninghub.xyz | by VU Learning HUB

Module-105

Originally, SQL was an acronym for “Structured Query Language”; it was designed to be a language to manipulate relational database data. Over time, SQL has become a non-acronym, and its goal has become that of manipulating some-thing called SQL-data—that subset of relational data that it handles properly (it cannot properly handle all aspects of the relational model).

SQL is very powerful; even so, it does not fully implement all of the features needed for the relational model. But you need not worry: you need only a small subset of SQL (in fact, no more than six commands).

Remember the basics of the relational model:

- Everything is a table.
- The results of all operations are tables.

These basics give rise to a very important point: operations within a relational database occur on groups of records (i.e. tables) and result in further groups of records (i.e., tables) that may incorporate changes. In other words, you do not use SQL (or the relational model) to handle individual records.

How This Works in Traditional Programs

In traditional programming (such as that which preceded the adoption of relational databases in the 1980s), programmers wrote code like this:

- Get the first record which is...
- Do something to it.
- Get the next record which is...
- Do that to it.
- Repeat until there are no more records which are…

These loops formed the basis of well-structured programs in the 1980s and 1990s. However, their code mixes two operations: the retrieval of data and its processing. With relational databases, the code becomes much simpler:

- Get all the records which are...
- For each of them, do something to it.

As per basic SQL model, there are no intermediate tests (except to see if all the records have been used), and in fact the relational database supports very efficient retrieval of each of these records after the major database retrieval ("all the records which are...") has been done.

You will see this model over and over in products that support the use of databases on Web sites. A query (the data retrieval) is specified in one way or another. Then, for each of the records that is returned, HTML is generated to display that record.

This works very well when what you are doing is retrieving a set of data. However, there are times when you need to retrieve an individual data item (such as the name that will be shown at the top of your HTML page). In such cases, you need to do exactly the same thing: retrieve all the records that conform to a certain criterion and process each of them in turn.

It is your responsibility to create a table which has a unique identifier so that when you specify that identifier you get only one row. The code that you will write to display the results of the database query will be capable of displaying multiple records—and that is probably not what you want to do if you are simply trying to retrieve the title to place on your Web Page.

Cursors are SQL constructs that are used to implement the logic described in the previous module. A cursor is associated with a table—which is often created as the result of some database operation (such as finding all records which are...). The cursor can be positioned on, after, or before any row in the table. When it is so positioned, the data for that row can be manipulated by the program, and the cursor is typically advanced to the next row.
The most significant database access occurs when the cursor is opened—that is, when its associated table is constructed via a query. Subsequent accesses (retrieving each row) typically involve much less database horsepower.

The issue of **sensitive cursors** can now be explained. If the data to which a cursor points changes (due to another user's activities), those changes may or may not be reflected in the data that you are retrieving. (Remember that the major data retrieval occurs when the cursor is opened.) If you declare a cursor to be sensitive, it reflects such changes; insensitive cursors do not.

Why this matters

Since the major processing occurs once, it is most efficient to get the most out of that processing. You can certainly write perfectly good code which uses cursors to evaluate the data that is retrieved (if the data to which the cursor points has certain characteristics, do X); however, in almost all cases databases work more efficiently if you move those evaluations into the original query.

In other words, if you want to find all customer records with balances that are over two months old and subsequently mark the customers that have a history of late payments, it is best to

form a basic query that retrieves all customers with balances over two months old who also had such late balances in the past year. The alternative (retrieving the customers with late payments and checking each one for previous late payments) is less efficient.

Because databases are often designed to work in this way, you may find that you are working with a process whose timing is very uneven. The time it takes to perform the initial—and complex—query may be noticeable to the user; the time to retrieve each additional row from the cursor is often unnoticeable. In such cases, it makes sense to plan ahead in designing your Web site: know when the major database accesses occur and present messages to users so that they know time will pass before. they get a response.
The most basic SQL statement is the Select statement; it operates on one or more tables and creates a result—a new table. Although the Select statement can be very complicated, its basis is quite simple:

SELECT FROM table(S)
<WHERE Condition>
<GROUP BY condition
<HAVING grouped characteristic>>

Here are some typical Select statements:
Select Names from Customers
This could select all customer names from a Customers table.
Select Names from Customers where Balance > l00.00
This could select all customers with balances greater than 100 from a Customers table.

Select Names from Customers where Balance > 100.00 ORDER BY Balance

This query is the same as the last except that the results will be sorted from smallest to largest balance.

Queries can operate against several tables at the same time; in such cases, column names need to be qualified with table names as in this query and the relationship between records in the two tables needs to be expressed.

vulearninghub.xyz | by VU Learning HUB

Module-106

All of this is background for you: most database-driven Web sites use only the simplest SQL queries. However, you do need to pin down a few facts covered in this module.

You Need the Queries: You need the queries that will generate your Web pages. These may already be known to database designers in your organization; if you are creating your own databases, you will need to create the queries. As you design your Web pages, look at every item that will come from a database and make certain that you have the query (or can create it). If you cannot construct the query that gets your data, you cannot construct the Web site. Two types of queries are typically used on database-driven Web sites:

1. Standard queries that return a variable number of rows are used to answer user queries or to present dynamic database information (as in, ”Here are all of today’s messages”).
2. Specialized queries that return a specific number of items are used to build Web pages.

Table and Field Names: If you are going to be writing the query yourself, you will need table and field names to use. They may not be self-evident from their names; also, it is common to find fields that appear to be similar (or even identical) but are not. If you are new to the world of databases, you may be expecting a degree of uniformity that does not exist; if you are an old hand at databases, you probably know that users and managers have a perverse desire to "refine" database designs.

Both situation can occasionally result in counterintuitive database structures.

Common Keys Linking two tables together using a common key is called a join. While building a Web site on existing database tables, you may be looking for data to display that has not been gathered and displayed in the applications running elsewhere in the organization. It is not surprising to discover that shared keys don’t exist —the ID numbers in the customer table differ from those in the accounting table. This is a very frequent problem in large (and not so large) organizations. If you are using preexisting databases, don’t worry about asking what the shared keys are—and be prepared to discover that the accounting systems do not really talk to the customer support systems and that they in turn do not communicate easily with analytical systems.

Stand Your Ground Creating database-driven Web sites is often a collaborative effort, and the parties to the collaboration often come from different worlds (databases on the one side, Web design on the other). The database queries are the interface between the two worlds. Whichever side of the fence you are, you must understand the query what the Web page needs and what the database can provide.

It is in no way a poor reflection on anyone if they do not understand the language from the other side of the fence. If you are the Web page designer, you must get the query that will get the data you want; if you are the database designer, you must generate the query that will give the Web page designer what is required. This may take some time, but it is key to the success of the project. Get the query, run it interactively if you can, and examine the results. Make certain that everyone is happy.

Insert

The SQL Insert command inserts a row of data into a table. The data must often adhere to certain edits (such as having a unique key, having numeric values for certain fields). You will often use an Insert command to add data to your database-driven Web sites, although the application server you use will often generate it for you automatically. Commonly, the data that is inserted into the database is collected from a user using an HTML form.

Delete

The SQL Delete command deletes a row of data that is specified using the appropriate unique key. Again, edits may come into play to prevent you from deleting part of related records in several tables (referential integrity). As with Insert, you often do not see this code—it is generated for you by your application server software.

Update

The SQL Update command modifies data in an existing row of a table. You can use variations of the Update command to perform mass updates (such as multiplying all values by a constant). "The difference between Insert and Update is that Insert creates a new row and Update modifies an existing row.

Create

A variety of SQL Create statements let you create the objects of relational databases. These include tables as well as subsidiary structures such as views and indexes.
Views

Views can be considered temporary tables. The result of a Select is always a table; such a table can be saved as a view. It does not actually exist as a table, but you can refer to the view —as if it were a table. Each time you refer to the view, the Select statement that created it is effectively re-executed, along with its conditions and criteria.

Indexes

Database software uses whatever information it has at hand to fulfill your queries. It can create indexes to the data in the database—and you can create indexes yourself. This can be useful when you know how the data will be retrieved.

Create statements (as well as views and indexes) are frequently used in database-driven Web sites, but you rarely worry about them. Your database administrator usually takes care of creating them as needed; you may get involved if performance is sluggish—views and indexes can speed up processing.

Module-107

Although files do a great job in many circumstances, they ' re pretty inflexible as data storage solutions go. For example, if you need to filter or sort the data retrieved from a file, you have to write your own PHP code to do it. Not only is this tiresome, but if you ' re working with large sets of data — for example, hundreds of thousands of user records — your script will probably grind to a halt. Not good if you ' re hoping to build a popular Web site. Databases are specifically designed to get around this problem.

This is the first in a series of five modules in which you explore databases and learn how you can use them to create powerful, efficient PHP applications. The aim of this module is to get you started with databases.

Whenever you start work on a data - driven application, one of your first design decisions should be: how will the application store and access its data? The answer will depend on the application ' s requirements. At the simplest level, you should be asking questions like:

- How much data will the application use?
- How often will it need access to the data?
- How often will it need to modify the data?
- How many users are likely to want access to the data at once?
- How much will the data grow over time?
- How much do I stand to lose if the data is broken, stolen, or lost?

If the answer to any of these questions is " a lot, " you probably want to steer clear of using plain text files to store your data. That ' s not to say that text files are useless — in fact, if all you want to do is read a large amount of unfiltered or unsorted data, text files can often be the fastest approach — but generally speaking, if you need to store and access structured data quickly and reliably, plain text files aren ' t a good bet.

Often, the most efficient alternative to text files is to use a database engine — commonly known as a *Database Management System* (DBMS) — to store, retrieve, and modify the data for you. A good database engine serves as a smart go - between for you and your data, organizing and cataloging the data for quick and easy retrieval.

So where does all the data go? Well, it depends to some extent on the database engine you ' re using. Chances are, though, it ' ll end up being stored in a number of files — yes, files! Truth is you can ' t really get away from using files at some point. The trick is in finding ways to use

them as efficiently as possible, and a good database engine has many, many such tricks up its metaphorical sleeves.

In this course, and developers in general, often use the word " database " to refer to the database engine, the data itself, or both. Usually the exact meaning is clear from the context.

Embedded Databases

An embedded database engine, sits inside the application that uses it (PHP in this case). Therefore it always runs — and stores its data — on the same machine as the host application. The database is not networked, and only one program can connect to it at any given time. Moreover, the database can ' t be shared between different machines because each one would simply end up storing and manipulating its own separate version of the data. On the plus side, embedded databases tend to be faster, easier to configure, and easier to work with.

Long-standing examples of embedded database engines include dBase and dbm, and PHP supports both these engines in the form of PHP extensions. A more recent addition is SQLite, which is bundled with the PHP engine itself, making it easy to install.

Client - Server Databases

Client - server databases are, generally speaking, more powerful and flexible than embedded databases. They are usually designed for use over networks, enabling many applications in a network to work simultaneously with the same data. The database engine itself acts as a server, serving up data to its clients (much like Web servers serve pages to Web browsers). In principle it can field requests from just about anywhere with a network connection and a suitable client program. That said, there ' s no reason why you can ' t run both server and client on the same machine; in fact this is a very common setup.

This is the kind of database you're more likely to find in a large company, where large quantities of data need to be shared among many people, where access may be needed from all sorts of different locations, and where having a single centralized data store makes important jobs like administration and backup relatively straightforward. Any applications that need to access the database use specialized, lightweight client programs to communicate with the server.
Most relational databases: Oracle, DB2, and SQL Server — have a client - server architecture.

Database Models

As well as the architecture of the database system, it is worth thinking about the database model that you want to use. The *model* dictates how the data is stored and accessed. Many different database models are used today, but in this module you look at two common ones:

Simple Databases

Simple database engines are, as the name implies, just about the simplest type of database to work with. Essentially, the simple model is similar to an associative array of data. Each item of data is referenced by a single key. It is not possible to define any relationships between the data in the database. For smaller applications there can often be advantages to using a simple database model. For example, if all you need to do is look up data based on keys, simple databases are lightning fast. Common examples of simple - model databases include dbm and its variants, of which Berkeley DB is the most popular these days.

Relational Databases

Relational databases offer more power and flexibility than simple databases, and for this reason they tend to be a more popular choice. They are also known as RDBMSs (Relational Database Management Systems). RDBMSs are often expensive and complex to set up and administer. The widely acknowledged big three in this field are Oracle, DB2 (from IBM), and SQL Server (from Microsoft). All three are massive, feature - rich systems, seemingly capable of just about any kind of data storage and processing that a modern business could need. The flip side of the coin is that these systems are big and expensive, and may contain more functionality than you will ever require.

Fortunately, alternatives are available, such as PostgreSQL and MySQL, which are both open source relational database systems that have proven very popular with PHP developers for many years. They ’ re fast, stable, easily meet the needs of most small - to - medium sized projects, and, to top it all off, they ’ re free!

In principle, you can use any of these database systems in your PHP applications. You can even hook one application up to several different database engines. To keep these chapters to a reasonable length, however, you ’ ll focus on just one database engine: MySQL.

Compared to the other choices, it offers several advantages:

- ❑ It is one of the most popular databases being used on the Web today
- ❑ It is freely available as a download to install and run on your own machine
- ❑ It is easy to install on a wide range of operating systems (including UNIX, Windows, and Mac
- ❑ OS X)
- ❑ It is available as a relatively cheap feature in many Web hosting packages
- ❑ It is simple to use and includes some handy administration tools
- ❑ It is a fast, powerful system that copes well with large, complex databases, and should stand you in good stead when it comes to larger projects

If you ’ re not too concerned about the last criterion (and particularly if you don ’ t want to pay extra for database functionality on your Web hosting account!) you might well find that an embedded database such as SQLite does a perfectly good job. PostgreSQL is also a great choice, and is similar in performance and features to MySQL.

Although these three chapters focus on MySQL, many of the techniques you learn can easily be transferred to other database systems.

In simple terms, a *relational database* is any database system that allows data to be associated and grouped by common attributes. For example, a bunch of payroll records might be grouped by employee, by department, or by date. Typically, a relational database arranges data into tables, where each table is divided into rows and columns of data.

In database parlance, each *row* in a table represents a data *record* : a set of intrinsically connected pieces of data, such as information relating to a particular person. Likewise, each *column* represents a *field* : a specific type of data that has the same significance for each record in the table, such as “ first name ” or “ age. ”

The terms “ row ” and “ record ” are often interchangeable, as are “ column ” and “ field. ”
Here ’ s an example of a database table. Suppose that the manager of a football team sets up a database so that she can track the matches in which her players compete. She asks each player to enter his details into the database after each match. After two matches the manager ’ s table, called matchLog , looks like this:

In this table, you can see that each row represents a particular set of information about a player who played on a certain date, and each column contains a specific type of data for each person or date. Notice that each column has a name at the top of the table to identify it; this is known as the *field name* or *column name*.

vulearninghub.xyz | by VU Learning HUB

Module-108

The manager soon realizes that this matchLog table is going to be huge after everyone on the team has played an entire season's worth of games. As you can see, the structure of the table is inefficient because each player ' s details — number, name, phone number, and so on — are entered every time he plays a match.

Such redundancy is undesirable in a database. For example, say that the player with the number 6 keeps dropping the ball, and his teammates decide to give him a new nickname. To update the table, every one of this player's records need to be modified to reflect his new nickname.

In addition, every time a player enters his details after a match, all of that duplicate information is consuming valuable space on the hard drive. Redundancy is terribly inefficient, wasting a great deal of time and space.

In early 1970s, Dr. E. F. Codd came up with a set of rules that (modules 66 & 67), when applied to data, ensure that your database is well designed. These are known as *normal forms* , and normalizing your data — that is, making sure it complies with these normal forms — goes a long way to ensuring good relational database design. This module doesn't go into normalization details as it is part of course on databases. However, the basic idea is to break up your data into several related tables, so as to minimize the number of times you have to repeat the same data. The matchLog table contains a lot of repeating data. You can see that most of the repeating data is connected with individual players. For example, the player with the nickname " Witblitz " is mentioned twice in the table, and each time he ' s mentioned, all of his information — his player number, name, and phone number — is also included.

Therefore, it makes sense to pull the player details out into a separate players table, as follows:

Observe that each player has just one record in this table. The playerNumber field is the field that uniquely identifies each player (for example, there are two Davids, but they have different playerNumber fields). The playerNumber field is said to be the table ' s *primary key* . Now that the player fields have been pulled out into the players table, the original matchLog table contains just one field — datePlayed — representing the date that a particular player participated in a match.

Here comes the clever bit. First, add the playerNumber column back into the matchLog table:
Now, by linking the values of the playerNumber fields in both the player and matchLog tables, you can associate each player with the date (or dates) he played. The two tables are said to be

joined by the playerNumber field. The playerNumber field in the matchLog table is known as a *foreign key* , because it references the primary key in the players table, and you can ’ t have a playerNumber value in the matchLog table that isn ’ t also in the players table.

Because the only repeating player information remaining in the matchLog table is the playerNumber field, you have saved some storage space compared to the original table. Furthermore, it is now easy to change the nickname of a player, because you only have to change it in one place: a single row in the players table.

This type of connection between the two tables is known as a *one-to-many* relationship, because one player record may be associated with many matchLog records (assuming the player plays in more than one match). This is a very common arrangement of tables in a relational database.

SQL, the Structured Query Language, is a simple, standardized language for communicating with relational databases. SQL lets you do practically any database - related task, including creating databases and tables, as well as saving, retrieving, deleting, and updating data in databases.

As mentioned previously, this module concentrates on MySQL. The exact dialect of SQL does vary among different database systems, but because the basic concepts are similar, the SQL skills you learn on one system can easily be transferred to another. In this section you examine some basic features of SQL: data types, indexes (keys), statements, and queries.

Module-109

When you create a database table, the type and size of each field must be defined. A field is similar to a PHP variable except that you can store only the specified type and size of data in a given field. For example, you can ' t insert characters into an integer field. MySQL supports three main groups of data types — numeric, date/time, and string — which are outlined in the following modules.

The descriptions here are fine for everyday use, but they're not complete. For full details see the MySQL manual at http://dev.mysql.com/doc/ .

You can store numbers in MySQL in many ways, as shown by the following table. Generally speaking, you should pick the data type most suited for the type of numbers you need to store.
Why not just always use the data types that can hold the biggest range of numbers, such as BIGINT and DOUBLE ? Well, the bigger the data type, the more storage space it takes up in the database. For example, an INT field takes up four bytes, whereas a SMALLINT field only requires two bytes of storage. If you end up storing millions of records, those extra two bytes can really make a difference! So use the smallest data type that will comfortably hold the range of values you expect to use.

You can add the attribute UNSIGNED *after a numeric data type when defining a field. An unsigned data type can only hold positive numbers. In the case of the integer types, an unsigned type can hold a maximum value that ' s around twice the size of its equivalent signed type. For example, a* TINYINT *can hold a maximum value of 127, whereas an unsigned* TINYINT *can hold a maximum value of 255. However, for the unsigned* FLOAT *,* DOUBLE *, and* DECIMAL *types, the maximum values are the same as for their signed equivalents.*

As with numbers, you can choose from a range of different data types to store dates and times, depending on whether you want to store a date only, a time only, or both:
[…]

When you need to specify a literal DATE , DATETIME , or TIMESTAMP value in MySQL, you can use any of the following formats:

MySQL lets you store text or binary strings of data in many different ways […]:

The difference between a CHAR and a VARCHAR field is that CHAR stores data as a fixed - length string no matter how short the actual data may be, whereas VARCHAR uses exactly as many

characters as necessary to store a given value. Suppose you insert the string " dodge " into the following fields:

- ❑ char_field defined as CHAR(10)
- ❑ varchar_field defined as VARCHAR(10)

They will store the same string slightly differently, as follows:
char_field: "dodge " // Right-padded with five spaces
varchar_field: "dodge" // No padding

VARCHAR character fields save you disk space. Don ' t be tempted to use VARCHAR fields for storing every string, because that has drawbacks, too. The MySQL server processes CHAR type fields much faster than VARCHAR type, because their length is predetermined. If your strings don ' t vary in length much, or at all, better using CHAR type fields. When your strings are all the same length, VARCHAR takes up more disk space, because it has to store the length of each string in one or two additional bytes.

With the character types — CHAR , VARCHAR , TEXT etc. the amount you can store may be less than the maximum shown, depending on the character set used. E.g. the UTF - 8 (Unicode) character set commonly uses up to 3 bytes per character, so a VARCHAR field may only be able to store up to 21,844 UTF - 8 characters.

Inexperienced database designers sometimes complain about their database engines being slow — a problem that ' s often explained by the lack of an *index* . An index is a separate sorted list of the values in a particular column (or columns) in a table. Indexes are also often called *keys* ; the two words are largely interchangeable. You can optionally add indexes for one or more columns at the time you create the table, or at any time after the table is created.

To explain why indexing a table has a dramatic effect on database performance, first consider a table without indexes. Such a table is similar to a plain text file in that the database engine must search it sequentially. Rows in a relational database are not inserted in any particular order; the server inserts them in an arbitrary manner. To make sure it finds all entries matching the information you want, the engine must scan the whole table, which is slow and inefficient, particularly if there are only a few matches.

Now consider an indexed table. Instead of moving straight to the table, the engine can scan the index for items that match your requirements. Because the index is a sorted list, this scan can be performed very quickly. The index guides the engine to the relevant matches in the database table, and a full table scan is not necessary.

So why not just sort the table itself? This might be practical if you knew that there was only one field on which you might want to search. However, this is rarely the case. Because it is not possible to sort a table by several fields at once, the best option is to use one or more indexes, which are separate from the table.

A *primary key* is a special index that, as you saw earlier, is used to ID records and to relate tables to one another, providing the relational database model. Each related table should have one (and only one) primary key.

You can also create an index or primary key based on combinations of fields, rather than just a single field. For a key to be formed in this way, the combination of values across the indexed fields must be unique.

Because an index brings about a significant boost in performance, you could create as many indexes as possible for maximum performance gain, right? Not always. An index is a sure - fire way to increase the speed of searching and retrieving data from a table, but it makes updating records slower, and also increases the size of the table. This is because, when you insert a record into an indexed table, the database engine also has to record its position in the corresponding index or indexes. The more indexes, the slower the updating process and the larger the table.

So when creating indexes on a table, don ’ t create more than you need. Limit indexed columns to those that will be searched or sorted frequently. If required, you can create additional indexes on a table as you need them to increase performance.

Module-110

Now that you've set up the MySQL root user, you can start working with databases. In the following sections, you create a new database, add a table to the database, and add data to the table. You also learn how to query databases and tables, update data in tables, and delete data, tables, and databases.

Most of the examples in the following sections show commands, statements, and other SQL keywords being entered using all - uppercase letters. Though SQL keywords are traditionally in uppercase, MySQL also lets you enter keywords in lowercase. So use lowercase if you prefer.
To create a new database, all you have to do is use the CREATE DATABASE command.

Type the following to create a new database called mydatabase :
CREATE DATABASE mydatabase;
Press Enter, and MySQL creates your new database. You can see a list of all the databases in the system — including your new database — by typing the command SHOW DATABASES :
[…]

Don ' t forget to type a semicolon at the end of a command or statement before pressing Enter.
You can see that this system has three databases. information_schema and mysql are databases connected with the operation of MySQL itself, and mydatabase is the database you just created.

As you know, tables are where you actually store your data. To start with, you ' ll create a very simple table, fruit , containing three fields: id (the primary key), name (the name of the fruit), and color (the fruit is color). The first thing to do is select the database you just created. Once you ' ve selected a database, any database - manipulation commands you enter work on that database.

Type the following:
USE mydatabase;
Press Enter, and you should see:
Database changed
mysql >

Now create your table. Type the following at the mysql > prompt: […]
Press Enter at the end of each line. Don ' t enter the " - > " arrows; MySQL displays these automatically each time you press Enter, to inform you that your statement is being continued on a new line.

If all goes well, you should see a response similar to the following: […]

By the way, if you ever want to create a regular key (as opposed to a primary key) for a field in a table, use the keyword KEY or INDEX instead of PRIMARY KEY . So if you wanted to add an index for the name field (because your table contained a large number of fruit records and you frequently wanted to look up fruit by name), you could use (again, don ' t type the arrows):

Now try adding some fruit to your table. To add a new row to a table, you use the SQL INSERT statement. In its basic form, an INSERT statement looks like this:
INSERT INTO *table* VALUES (*value1* , *value2* , ...);

This inserts values into each of the fields of the table, in the order that the fields were created. Alternatively, you can create a row with only some fields populated. The remaining fields will contain NULL (if allowed), or in the case of special fields such as an AUTO_INCREMENT field, the field value will be calculated automatically. To insert a row of partial data, use:
INSERT INTO *table* (*field1* , *field2* , ...) VALUES (*value1* , *value2* , ...);

So you can add three rows to the fruit table by inserting data into just the name and color fields (the id field will be filled automatically):

To read data in SQL, you create a query using the SELECT statement. Thanks to the flexibility of SQL, it is possible to run very complex queries on your data (for example, " Give me a list of all transactions over $500 sent from John Smith to Henry Hargreaves between 13 October and 17 November last year "). For now, though, you ' ll stick with couple of simple examples. To retrieve a list of all the data in your fruit table, you can use:
[…]

To retrieve a selected row or rows, you need to introduce a WHERE clause at the end of the SELECT statement. A WHERE clause filters the results according to the condition in the clause. You can use practically any expression in a WHERE condition. Here are some simple WHERE clauses in action:
[…]

You change existing data in a table with the UPDATE statement. As with the SELECT statement, you can (and usually will) add a WHERE clause to specify exactly which rows you want to update. If you leave out the WHERE clause, the entire table gets updated.
Here ' s how to use UPDATE to change values in your fruit table:

Deleting works in a similar way to updating. To delete rows, you use the DELETE statement. If you add a WHERE clause, you can choose which row or rows to delete; otherwise all the data in the table are deleted (though the table itself remains). Here ' s an example:
To delete a table entirely, use the DROP TABLE statement. Similarly, you can delete an entire database with DROP DATABASE .
First, here ' s how to use DROP TABLE :
DROP DATABASE works in a similar fashion:

Be careful with statements such as DELETE and DROP , because you can ' t undo the deletion process. Make sure you back up your MySQL databases regularly, and before carrying out any operation that could potentially wipe a lot of data. You can also alter the definition of a table, even if it already has data within it. To do this, you use the ALTER TABLE statement.

Module-111

PHP provides you with two main ways to connect to MySQL databases:

mysqli (MySQL improved) — This extension is specifically tied to MySQL, and provides the most complete access to MySQL from PHP. It features both procedural (function - oriented) and object - oriented interfaces. Because it has quite a large set of functions and classes, it can seem overwhelming if you ' re not used to working with databases. However, if you know you ' re only ever going to work with MySQL, and you want to squeeze the most out of MySQL ' s power from your PHP scripts, then mysqli is a good choice

PDO (PHP Data Objects) — This is an object - oriented extension that sits between the MySQL server and the PHP engine. It gives you a nice, simple, clean set of classes and methods that you can use to work with MySQL databases. Furthermore, you can use the same extension to talk to lots of other database systems, meaning you only have to learn one set of classes and methods in order to create applications that can work across MySQL, PostgreSQL, Oracle, and so on
Text book uses PDO, mainly because it is easier and quicker to learn, but once you ' ve learned PDO you should find that you can transfer your skills to mysqli if needed.

If you ' ve installed PHP and MySQL using Synaptic on Ubuntu, WampServer on Windows, or MAMP on the Mac, you should find that both the mysqli and PDO extensions are already installed.

To make a connection to a MySQL database in your PHP script, all you need to do is create a new PDO object. When you create the object, you pass in three arguments: the DSN, which describes the database to connect to; the username of the user you want to connect as; and the user ' s password. The returned PDO object serves as your script ' s connection to the database:

A DSN , or Database Source Name, is simply a string that describes attributes of the connection such as the type of database system, the location of the database, and the database name. For example, the following DSN can be used to connect to a MySQL database called mydatabase running on the same machine as the PHP engine:

So, putting it all together, you could connect to your mydatabase database as follows (replacing mypass with your real root password of course):

When you ' ve finished with the connection, you should close it so that it is freed up for other scripts to use. Although the PHP engine usually closes connections when a script finishes, it is a

good idea to close the connection explicitly to be on the safe side. To close the connection, just assign null to your connection variable. This effectively destroys the PDO object, and therefore the connection:

Database errors can be notoriously difficult to track down and deal with. One of the nice things about PDO is that you can get it to return MySQL errors in the form of highly descriptive PDOException objects. You can then use the PHP keywords try and catch to handle these exceptions easily and deal with them appropriately.

To set PDO to raise exceptions whenever database errors occur, you use the PDO::SetAttribute method to set your PDO object ' s error mode, as follows:
[…]

Now you can capture any error that might occur when connecting to the database by using a try ...catch code block. If you were writing a sophisticated application, you ' d probably log the error message to a file, and possibly send an email to the Webmaster informing him of the details of the error. For the sake of these examples, though, you ' ll just display the error message in the Web page:
[…]

PHP runs the code within the try block. If an exception is raised by PDO, the catch block stores the PDOException object in $e , then displays the error message with $e - > getMessage() .
For example, if the $password variable in the script contained an incorrect password, you ' d see a message like this appear when you ran the script:
[…]

Now that you ' ve connected to your database in your PHP script, you can read some data from the database using a SELECT statement. To send SQL statements to the MySQL server, you use the query method of the PDO object:
$conn- > query ($sql);

If your SQL statement returns rows of data as a result set, you can capture the data by assigning the result of $conn - > query to a variable:
$rows = $conn- > query ($sql);

The result returned by $conn - > query is actually another type of object, called a PDOStatement object. You can use this object along with a for each loop to move through all

vulearninghub.xyz | by VU Learning HUB

the rows in the result set. Each row is an associative array containing all the field names and values for that row in the table. For example:

```
$conn->query ( $sql );
```

```
$rows = $conn->query ( $sql );
```

```
$sql = "SELECT * FROM fruit";
$rows = $conn->query( $sql );
foreach ( $rows as $row ) {
  echo "name = " . $row["name"] . "<br />";
  echo "color = " . $row["color"] . "<br />";
}
```


vulearninghub.xyz | by VU Learning HUB

Module-112

It does no good to put records in a database unless you retrieve them eventually and do something with them. That's the purpose of the SELECT statement—to help you get at your data. SELECT probably is used more often than any other in the SQL language, but it can also be the trickiest; the constraints you use to choose rows can be arbitrarily complex and can involve comparisons between columns in many tables.

To create a new database, all you have to do is use the CREATE DATABASE command. Type the following to create a new database called mydatabase :

CREATE DATABASE mydatabase;

Press Enter, and MySQL creates your new database. You can see a list of all the databases in the system — including your new database — by typing the command SHOW DATABASES :

[…]

Don ’ t forget to type a semicolon at the end of a command or statement before pressing Enter.

You can see that this system has three databases. information_schema and mysql are databases connected with the operation of MySQL itself, and mydatabase is the database you just created.

Everything in this syntax is optional except the word SELECT and the *selection_list* part that specifies what you want to retrieve. Some databases require the FROM clause as well. MySQL does not, which allows you to evaluate expressions without referring to any tables:

This section covers an aspect of SELECT that is often confusing—writing joins; that is, SELECT statements that retrieve records from multiple tables. We'll discuss the types of join MySQL supports, what they mean, and how to specify them. This should help you employ MySQL more effectively because, in many cases, the real problem of figuring out how to write a query is determining the proper way to join tables together.

One problem with using SELECT is that when you first encounter a new type of problem, it's not always easy to see how to write a SELECT query to solve it. However, after you figure it out, you can use that experience when you run across similar problems in the future. SELECT is probably the statement for which past experience plays the largest role in being able to use it effectively, simply because of the sheer variety of problems to which it applies.

As you know, tables are where you actually store your data. To start with, you ’ ll create a very simple table, fruit , containing three fields: id (the primary key), name (the name of the fruit), and color (the fruit is color). The first thing to do is select the database you just created. Once

you ’ ve selected a database, any database - manipulation commands you enter work on that database.

Type the following:

USE mydatabase;

Press Enter, and you should see:

Database changed

mysql >

Now create your table. Type the following at the mysql > prompt: […]

Press Enter at the end of each line. Don ’ t enter the “ - > “ arrows; MySQL displays these automatically each time you press Enter, to inform you that your statement is being continued on a new line.

If all goes well, you should see a response similar to the following:

```
Table t1:   Table t2:
+----+----+  +----+----+
| i1 | c1 |  | i2 | c2 |
+----+----+  +----+----+
| 1  | a  |  | 2  | c  |
| 2  | b  |  | 3  | b  |
| 3  | c  |  | 4  | a  |
+----+----+  +----+----+
```

By the way, if you ever want to create a regular key (as opposed to a primary key) for a field in a table, use the keyword KEY or INDEX instead of PRIMARY KEY . So if you wanted to add an index for the name field (because your table contained a large number of fruit records and you frequently wanted to look up fruit by name), you could use (again, don ’ t type the arrows):

Module-113

The simplest join is the trivial join, in which only one table is named. In this case, rows are selected from the named table:

Some people don't consider this form of SELECT a join at all and use the term only for SELECT statements that retrieve records from two or more tables. I suppose it's a matter of perspective.

If a SELECT statement names multiple tables in the FROM clause with the names separated by commas, MySQL performs a full join. For example, if you join t1 and t2 as follows, each row in t1 is combined with each row in t2:

A full join is also called a *cross join* because each row of each table is crossed with each row in every other table to produce all possible combinations. This is also known as the *cartesian product*. Joining tables this way has the potential to produce a very large number of rows because the possible row count is the product of the number of rows in each table. A full join between three tables that contain 100, 200, and 300 rows, respectively, could return 100x200x300 = 6 million rows. That's a lot of rows, even though the individual tables are small. In cases like this, a WHERE clause will normally be used to reduce the result set to a more manageable size.

The JOIN and CROSS JOIN join types are equivalent to the ',' (comma) join operator. For example, the following statements are all the same:

```
SELECT t1.*, t2.* FROM t1, t2 WHERE t1.i1 = t2.i2;
SELECT t1.*, t2.* FROM t1 JOIN t2 WHERE t1.i1 = t2.i2;
SELECT t1.*, t2.* FROM t1 CROSS JOIN t2 WHERE t1.i1 = t2.i2;
```

Normally, the MySQL optimizer considers itself free to determine the order in which to scan tables to retrieve rows most quickly. On occasion, the optimizer will make a non-optimal choice. If you find this happening, you can override the optimizer's choice using the STRAIGHT_JOIN keyword. A join performed with STRAIGHT_JOIN is like a cross join but forces the tables to be joined in the order named in the FROM clause.

STRAIGHT_JOIN can be specified at two points in a SELECT statement. You can specify it between the SELECT keyword and the selection list to have a global effect on all cross joins in the statement, or you can specify it in the FROM clause. The following two statements are equivalent:

```
SELECT STRAIGHT_JOIN ...
FROM t1, t2, t3 ... ;

SELECT ...
FROM t1 STRAIGHT_JOIN t2
STRAIGHT_JOIN t3 ... ;
```

References to table columns throughout a SELECT statement must resolve unambiguously to a single table named in the FROM clause. If only one table is named, there is no ambiguity because all columns must be columns of that table. If multiple tables are named, any column name that appears in only one table is similarly unambiguous. However, if a column name appears in multiple tables, references to the column must be qualified by the table name using *tbl_name.col_name* syntax to specify which table you mean. Suppose a table mytbl1 contains columns a and b, and a table mytbl2 contains columns b and c. In this case, references to columns a or c are unambiguous, but references to b must be qualified as either mytbl1.b or mytbl2.b:

SELECT a, mytbl1.b, mytbl2.b, c

FROM mytbl1, mytbl2 ... ;

Sometimes a table name qualifier is not sufficient to resolve a column reference. For example, if you're joining a table to itself, you're using it multiple times within the query and it doesn't help to qualify a column name with the table name. In this case, table aliases are useful for communicating your intent. You can assign an alias to any instance of the table and refer to columns from that instance as *alias_name.col_name*. The following query joins a table to itself, but assigns an alias to one instance of the table to allow column references to be specified unambiguously:

SELECT mytbl.col1, m.col2 FROM

mytbl, mytbl

AS m

WHERE mytbl.col1 > m.col1;

Module-114

An equi-join shows only rows where a match can be found in both tables. Le and right joins show matches, too, but also show rows in one table that have no match in the other table. The examples in this section use LEFT JOIN, which identifies rows in the le table that are not matched by the right table. **RIGHT JOIN is the same except that the roles of the tables are reversed.** (RIGHT JOIN is available only as of MySQL 3.23.25.)

A LEFT JOIN works like this: You specify the columns to be used for matching rows in the two tables. When a row from the left table matches a row from the right table, the contents of the rows are selected as an output row. When a row in the let table has no match, it is still selected for output, but joined with a "fake" row from the right table in which all the columns have been set to NULL. In other words, a LEFT JOIN forces the result set to contain a row for every row in the let table whether or not there is a match for it in the right table. The rows with no match can be identified by the fact that all columns from the right table are NULL.

Consider once again our two tables, t1 and t2:

A left join produces output for every row in t1, whether or not t2 matches it. To write a left join, name the tables with LEFT JOIN in between (rather than a comma) and specify the matching condition using an ON clause (rather than a WHERE clause):

Now there is an output row even for the value 1, which has no match in t2.

LEFT JOIN is especially useful when you want to find *only* those left table rows that are unmatched by the right table. Do this by adding a WHERE clause that looks for rows in the right table that have NULL values—in other words, the rows in one table that are missing from the other:

LEFT JOIN actually allows the matching conditions to be specified two ways. ON is one of these; it can be used whether or not the columns you're joining on have the same name:

SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2;

The other syntax involves a USING() clause; this is similar in concept to ON, but the name of the joined column or columns must be the same in each table. For example, the following query joins mytbl1.b to mytbl2.b:

SELECT mytbl1.*, mytbl2.* FROM mytbl1 LEFT JOIN mytbl2 USING (b);

LEFT JOIN has a few synonyms and variants. LEFT OUTER JOIN is a synonym for LEFT JOIN. There is also an ODBC-style notaon for LEFT JOIN that MySQL accepts (the OJ means "outer join"):

{ OJ *tbl_name1* LEFT OUTER JOIN *tbl_name2* ON *join_expr* }

NATURAL LEFT JOIN is similar to LEFT JOIN; it performs a LEFT JOIN, matching all columns that have the same name in the left and right tables.

One thing to watch out for with LEFT JOIN is that if the columns that you're joining on are not declared as NOT NULL, you may get problematic rows in the result. For example, if the right table contains columns with NULL values, you won't be able to distinguish those NULL values from NULL values that identify unmatched rows.

For the grade-keeping project, we have a student table listing students, an event table listing the grade events that have occurred, and a score table listing scores for each student for each grade event. However, if a student was ill on the day of some quiz or test, the score table wouldn't have any score for the student for that event, so a makeup quiz or test should be given. How do we find these missing records so that we can make sure those students take the makeup?

The problem is to determine which students have no score for a given grade event and to do this for each grade event. Another way to say this is that we want to find out which combinations of student and event are not represented in the score table.

Note that the ON clause allows the rows in the score table to be joined according to matches indifferent tables. That's the key for solving this problem. The LEFT JOIN forces a row to be generated for each row produced by the cross join of the student and event tables, even when there is no corresponding score table record. The result set rows for these missing score records can be identified by the fact that the columns from the score table will all be NULL. We can select these records in the WHERE clause. Any column from the score table will do, but because we're looking for missing scores, it's probably conceptually clearest to test the score column:

WHERE score.score IS NULL

We can put the results in order using an ORDER BY clause. The two most logical orderings are by event per student or by student per event. Choose the first:

ORDER BY student.student_id, event.event_id

Here's a subtle point. The output displays the student IDs and the event IDs. The student_id column appears in both the student and score tables, so at first you might think that the selection list could name either student. student_id or score.student_id. That's not the case because the entire basis for being able to find the records we're interested in is that all the score table fields are returned as NULL. Selecting score.student_id would produce only a column of NULL values in the output. The same principle applies to deciding which event_id

column to display. It appears in both the event and score tables, but the query selects event.event_id because the score.event_id values will always be NULL.

vulearninghub.xyz | by VU Learning HUB

Module-115

One of the features that MySQL 4.1 introduces is subselect support, which is a long-awaited capability that allows one SELECT query to be nested inside other.

The following is an example that looks up the IDs for event records corresponding to tests ('T') and uses them to select scores for those tests:

SELECT * FROM score

WHERE event_id IN (SELECT event_id FROM event WHERE type = 'T');

In some cases, subselects can be rewritten as joins. Will show how to do that later. You may find subselect rewriting techniques useful if your version of MySQL precedes 4.1.

A related feature that MySQL supports is the ability to delete or update records in one table based on the contents of another. For example, you might want to remove records in one table that aren't matched by any record in another, or copy values from columns in one table to columns in another.

There are several forms you can use to write subselects; this section surveys just a few of them.

Using a subselect to produce a reference value. In this case, you want the inner SELECT to identify a single value to be used in comparisons with the outer SELECT. For example to identify the scores for the quiz that took place on '2002-09-23', use an inner SELECT to determine the quiz event ID, and then match score records against it in the outer SELECT:

With this form of subselect, where the inner query is preceded by a comparison operator, it's necessary that the inner join produce no more than a single value (that is, one row, one column). If it produces multiple values, the query will fail. (In some cases, it may be appropriate to satisfy this constraint by liming the inner query result with LIMIT 1.)

This form of subselect can be handy for situations where you'd be tempted to use an aggregate function in a WHERE clause. For example, to determine which student was born first, you might try the following:

SELECT * FROM student WHERE birth = MIN(birth);

That doesn't work because you can't use aggregates in WHERE clauses. (The WHERE clause determines which records to select, but the value of MIN() isn't known until after the records have already been selected.) However, you can use a subselect to produce the minimum birth date as follows:

SELECT * FROM student WHERE birth = (SELECT MIN(birth) FROM student);

EXISTS and NOT EXISTS subselects. These forms of subselects work by passing values from the outer query to the inner one to see whether they match the conditions specified in the inner query. For this reason, you'll need to qualify column names with table names if they are ambiguous (appear in more than one table). EXISTS and NOT EXISTS subselects are useful for finding records in one table that match or don't match records in another.

Refer once again to our t1 and t2 tables:

The following query identifies matches between the tables—that is, values that are present in both:

NOT EXISTS identifies non-matches—values in one table that are not present in the other:

With these forms of subselect, the inner query uses * as the output column list. There's no need to name columns explicitly because the inner query is assessed as true or false based on whether or not it returns rows, not based on the particular values that the rows may contain. In MySQL, you can actually write prey much anything for the column selection list, but if you want to make it explicit that you're returning a true value when the inner SELECT succeeds, you might write the queries like this:

The IN and NOT IN forms of subselect should return a single column of values from the inner SELECT to be evaluated in a comparison in the outer SELECT. For example, the preceding EXISTS and NOT EXISTS queries can be written using IN and NOT IN syntax as follows:

Rewring Subselects as Joins

For versions of MySQL prior to 4.1, subselects are not available. However, it's often possible to rephrase a query that uses a subselect in terms of a join. In fact, even if you have MySQL 4.1 or later, it's not a bad idea to examine queries that you might be inclined to write in terms of subselects; a join is sometimes more efficient than a subselect.

Rewring Subselects That Select Matching Values

The following is an example query containing a subselect; it selects scores from the score table for all tests (that is, it ignores quiz scores):

Rewring Subselects as Joins

For versions of MySQL prior to 4.1, subselects are not available. However, it's often possible to rephrase a query that uses a subselect in terms of a join. In fact, even if you have MySQL 4.1 or later, it's not a bad idea to examine queries that you might be inclined to write in terms of subselects; a join is sometimes more efficient than a subselect.

Rewring Subselects That Select Matching Values

The following is an example query containing a subselect; it selects scores from the score table for all tests (that is, it ignores quiz scores):

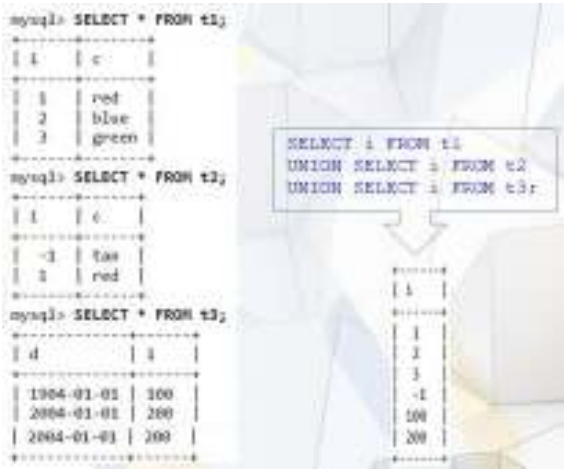
vulearninghub.xyz | by VU Learning HUB

Module-116

If you want to create a result set by selecting records from multiple tables one after the other, you can do that using a UNION statement. UNION is available as of MySQL 4, although prior to that you can use a couple of workarounds (shown later).

For the following examples, assume you have three tables, t1, t2, and t3 that look like this: […]

Tables t1 and t2 have integer and character columns, and t3 has date and integer columns. To write a UNION statement that combines multiple retrievals, just write several SELECT statements and put the keyword UNION between them. For example, to select the integer column from each table, do this: […]



UNION has the following properties:

The names and data types for the columns of the UNION result come from the names and types of the columns in the first SELECT. The second and subsequent SELECT statements in the UNION must select the same number of columns, but they need not have the same names or types. Columns are matched by position (not by name), which is why these two queries return different results: […]

In both cases, the columns selected from t1 (i and c) determine the types used in the UNION result. These columns have integer and string types, so type conversion takes place when selecting values from t3. For the first query, d is converted from date to string. That happens to result in no loss of information. For the second query, d is converted from date to integer (which *does* lose information), and i is converted from integer to string.

UNION has the following properties:

By default, UNION eliminates duplicate rows from the result set:

t1 and t2 both have a row containing values of 1 and 'red', but only one such row appears in the output. Also, t3 has two rows containing '2004-01-01' and 200, one of which has been eliminated.

If you want to preserve duplicates, follow the first UNION keyword with ALL:

To sort a UNION result, add an ORDER BY clause a□er the last SELECT; it applies to the query result as a whole. However, because the UNION uses column names from the first SELECT, the ORDER BY should refer to those names, not the column names from the last SELECT, if they differ.

You can also specify an ORDER BY clause for an individual SELECT statement within the UNION. To do this, enclose the SELECT (including its ORDER BY) within parentheses:

LIMIT can be used in a UNION in a manner similar to that for ORDER BY. If added to the end of the statement, it applies to the UNION result as a whole:

If enclosed within parentheses as part of an individual SELECT statement, it applies only to that SELECT:

You need not select from different tables. You can select different subsets of the same table using different conditions. This can be useful as an alternative to running several different SELECT queries, because you get all the rows in a single result set rather than as several result sets.

Module-117

If you want to create a result set by selecting records from multiple tables one after the other, you can do that using a UNION statement. UNION is available as of MySQL 4, although prior to that you can use a couple of workarounds (shown later).

The example queries and scripts in this module and the next work with two tables: a members table of book club members, and an accessLog table to track each member ' s visits to the book club Web site. So that you can work through these examples, first create these tables and a database to hold them, in MySQL, and populate the tables with some sample data.

If you don ' t fancy typing all these lines directly into the MySQL command - line tool, you can create a text file — say, book_club.sql — and enter the lines in there. Save the file in the same folder as you run the MySQL command - line tool from. Run the tool, then type:

source book_club.sql;

This command reads the lines of the text file and executes them, just as if you ' d manually entered the SQL statements into the tool line - by - line.

Without further ado, here are the SQL statements to create and populate the two tables:

Why is the password field exactly 41 characters long? Further down in the code, you can see that you insert the members ' passwords in encrypted form by calling MySQL ' s password() function. The encrypted password strings returned by password() are always 41 characters long, so it makes sense to use CHAR(41) for the password field.

All character data types have a *collation* that is used to determine how characters in the field are compared. By default, a character field ' s collation is case insensitive. This means that, when you sort the column alphabetically, " a " comes before both " b " and " B ". It also means that queries looking for the text " banana " will match the field values " banana " and " Banana " .

However, by adding the BINARY attribute after the data type definition, you switch the field to a binary collation, which is case sensitive; when sorting, " a " comes before " b ", but " B " comes before " a " (because, generally speaking, uppercase letters come before lowercase letters in a character set). Furthermore, this means that matches are case sensitive too; " banana " will only match " banana ", not " Banana ".

In this case, you created the username field of the members table with the BINARY attribute, making it case sensitive:

username VARCHAR(30) BINARY NOT NULL UNIQUE,

This ensures that there ' s no ambiguity over the case of the letters in each user ' s username; for example, " john " is a different username than " John ". This is important because many people choose usernames where the case of the username ' s characters is significant to them. If they created their account with a username of " john ", and later found out they could also login using " John ", they might wonder if they were working with one account or two!

You ' ve already seen how you can use the keywords PRIMARY KEY to create an index on a column that uniquely identifies each row in a table. The UNIQUE constraint is similar to PRIMARY KEY in that it creates an index on the column and also ensures that the values in the column must be unique. The main differences are:

- You can have as many UNIQUE keys as you like in a table, whereas you can have only one primary key
- The column(s) that make up a UNIQUE key can contain NULL values; primary key columns cannot contain NULL s

In the members table, you add UNIQUE constraints for the username and emailAddress columns because, although they ' re not primary keys, you still don ' t want to allow multiple club members to have the same username or email address.

You can also create a unique key for a column (or columns) by using the keywords UNIQUE KEY at the end of the table definition. So:

An ENUM (enumeration) column is a type of string column where only predefined string values are allowed in the field. For the members table, you created two ENUM fields:

ENUM fields serve two purposes. First, by limiting the range of values allowed in the field, you ' re effectively validating any data that is inserted into the field. If a value doesn ' t match one of the values in the predefined set, MySQL rejects the attempt to insert the value. Second, ENUM fields can save storage space. Each possible string value — " crime ", " horror ", and so on — is associated with an integer, and stored once in a separate part of the table. Each ENUM field can then be stored as an integer, rather than as a string of characters.

As you can imagine, the ENUM data type is only useful in a situation in which there are a small number of possible values for the field. Although you can define up to 65,535 allowed values for an ENUM type, practically speaking, things start to get a bit unwieldy after 20 or so values!

MySQL lets you store dates and times using a number of different data types, such as DATE , DATETIME , TIME , YEAR , and TIMESTAMP . A TIMESTAMP field is a bit different from the other date/time types in that it can automatically record the time that certain events occur. For example, when you add a new row to a table containing a TIMESTAMP column, the field stores

the time that the insertion took place. Similarly, whenever a row is updated, the TIMESTAMP field is automatically updated with the time of the update.

The other point to remember about TIMESTAMP fields is that they store the date and time in the UTC (Universal Coordinated Time) time zone, which is essentially the same as the GMT time zone. This probably won't affect you much, because MySQL automatically converts TIMESTAMP values between UTC and your server's time zone as required. However, bear in mind that if you store a TIMESTAMP value in a table, and you later change the server's time zone, the value that you get back from the TIMESTAMP field will be different.

A TIMESTAMP field is great for tracking things such as when a record was last created or updated, because you don ' t have to worry about setting or changing its value; it happens automatically. In this example, you created a TIMESTAMP field in the accessLog table to track when the last access was made:

lastAccess TIMESTAMP NOT NULL,

Module-118

By default, LIMIT counts from the first row of the results. However, by including two numbers after the LIMIT keyword, separated by a comma, you can specify both the row from which to start returning results, as well as the number of results to return:

You might be wondering what the point of LIMIT is, because you can always just loop through the result set in PHP to extract only the rows you ' re interested in. The main reason to use LIMIT is that it reduces the amount of data that has to flow between MySQL and your PHP script.

Imagine that you want to retrieve the first 100 rows of a million - row table of users. If you use LIMIT 100 , only 100 rows are sent to your PHP script. However, if you don ' t use a LIMIT clause (and your query also contains no WHERE clause), all 1,000,000 rows of data will be sent to your PHP script, where they will need to be stored inside a PDOStatement object until you loop through them to extract the first 100. Storing the details of a million users in your script will quickly bring the script to a halt, due to the large amount of memory required to do so.

LIMIT is particularly useful when you ' re building a paged search function in your PHP application. For example, if the user requests the second page of search results, and you display 10 results per page, you can use SELECT ... LIMIT 10, 10 to retrieve the second page of results.

One of the powerful features that really separate databases from text files is the speed and ease with which you can retrieve data in any order. Imagine that you have a text file that stores the first and last names of a million book club members, ordered by first name. If you wanted to retrieve a list of all the members ordered by *last* name, you ' d need to rearrange an awful lot of rows in your text file.

With SQL, retrieving records in a different order is as simple as adding the keywords ORDER BY to your query, followed by the column you want to sort by:

You can even sort by more than one column at once by separating the column names with commas:

You can read this ORDER BY clause as: " Sort the results by favoriteGenre , then by firstName . " Notice how the results are ordered by genre, but where the genre is the same (" crime "), the results are then sorted by firstName (" Jane " then " John ").

By default, MySQL sorts columns in ascending order. If you want to sort in descending order, add the keyword DESC after the field name. To avoid ambiguity, you can also add ASC after a field name to explicitly sort in ascending order:

Remember that ORDER BY works faster on a column that has an index, because indexes are already sorted in order.

So far, all the WHERE clauses you've looked at have been fairly precise:

Although this approach is good if you know the exact column values you ' re after, sometimes it is useful to be a bit less specific in your queries. For example, say you wanted to get a list of book club members that have travel among their interests. Each otherInterests field in the members table is free - form, consisting of a plain - English list of topics. How can you find out which otherInterests fields contain the word " travel " ?

The answer is to use the LIKE operator. This operator allows you to specify a string in the form of a pattern to search for, rather than an exact string:

SELECT ... WHERE fieldName LIKE pattern ;

Within the pattern string, you can include the following wildcard characters in addition to regular characters:

- % matches any number of characters (including no characters at all)
- _ (underscore) matches exactly one character

So to retrieve a list of members that list travel as one of their interests, you could use:

vulearninghub.xyz | by VU Learning HUB

Module-119

Just as PHP contains a large number of built - in functions, MySQL also gives you many functions to assist you with your queries. In this module we look at some of MySQL ' s aggregate functions. Rather than returning the actual data contained in a table, these functions let you summarize a table ' s data in different ways:

- count() — Returns the number of rows selected by the query
- sum() — Returns the total of all the values of a given field selected by the query
- min() — Returns the minimum value of all the values of a given field selected by the query
- max() — Returns the maximum value of all the values of a given field selected by the query
- avg() — Returns the average of all the values of a given field selected by the query

You can use count() in two slightly different ways:

- count(fieldname) — Returns the number of rows selected by the query where fieldname isn ' t NULL
- count(*) — Returns the number of rows selected by the query, regardless of whether the rows contain any NULL values

Here are a couple of count() examples. The first example counts all the rows in the members table:

Occasionally a query returns more data than you actually need, even when using WHERE and LIMIT clauses. Say your accessLog table contains the following data:

Now, imagine you want to get a list of the IDs of users that have accessed the site since November 7. You might create a query as follows:

Now there ' s a slight problem: the value 3 appears twice in the result set. This is because there are two rows in the accessLog table with a memberId of 3 and a lastAccess date later than November 7, representing two different pages viewed by user number 3. If you were displaying this data in a report, for example, user number 3 would appear twice. You can imagine what would happen if that user had visited 100 different pages!

To eliminate such duplicates, you can place the keyword DISTINCT after SELECT in the query:

DISTINCT removes any rows that are exact duplicates of other rows from the result set. For example, the following query still contains two instances of 3 in the memberId column, because the pageUrl column is different in each instance:

You have seen how to use functions such as count() and sum() to retrieve overall aggregate data from a table, such as how many female members are in the book club. What if you wanted to get more fine - grained information? For example, say you want to find out the number of different page URLs that each member has viewed. You might try this query:

That ' s no good. All this query has given you is the total number of rows in the table! Instead, you need to *group* the pageUrl count by member ID. To do this, you add a GROUP BY clause. For example:

That ' s better. By combining an aggregate function, count() , with a column to group by (memberId), you can view statistics on a per - member basis. In this case you can see that members 1 and 6 have each viewed one distinct page, whereas member 3 has visited two different pages.

You can combine GROUP BY and ORDER BY in the same query. Here ' s how to sort the previous data so that the member that has viewed the highest number of distinct pages is at the top of the table:

So far, all your queries have worked with one table at a time. However, the real strength of a relational database is that you can query multiple tables at once, using selected columns to relate the tables to each other. Such a query is known as a *join* , and joins enable you to create complex queries to retrieve all sorts of useful information from your tables.

In the previous examples that retrieved statistics from the accessLog table, your result sets contained a list of integer member IDs in a memberId column. For instance, let ' s say you want a list of all members that have accessed the Web site:

Now, of course, the member ID on its own isn ' t very helpful. If you want to know the *names* of the members involved, you have to run another query to look at the data in the members table:

Now you can see that member number 1 is in fact John Sparks, member number 3 is Jo Scrivener, and member number 6 is Bill Swan.

However, by using a join, you can combine the data in both tables to retrieve not only the list of member IDs that have accessed the site, but their names as well, all in the one query:

Module-120

As you start to work with many tables, things can start to get unwieldy. For example, in the preceding section you used this query to retrieve a list of names of members who have accessed the Web site:

[…]

There ’ s a lot of repetition of the table names accessLog and members in this query. Fortunately, SQL lets you create short table aliases by specifying an alias after each table name in the FROM clause. You can then use these aliases to refer to the tables, rather than using the full table names each time:

[…]

You can also use the AS keyword to create aliases for the columns returned by your query. Consider this query that you looked at earlier:

[…]

Notice that the second column in the result set is called count(pageUrl) . Not only is this not very descriptive, but you ’ ll find it is awkward to refer to in your PHP script. Therefore, it is a good idea to rename this column to something more meaningful:

[…]

MySQL contains a wealth of operators and functions that you can use to build more complex queries. You ’ ve already used a few of these in this chapter. Here you explore some other common operators and functions. Bear in mind that this is nowhere near a complete list (you can find such a list in the MySQL manual at http://dev.mysql.com/doc/).

Much like PHP, MySQL features various comparison operators that you can use to compare column values and other expressions in your queries. Here are some common ones:

[…]

By using the null - safe operator < = > , you ensure that any NULL value isn ’ t propagated through to the result:

[…]

You can also use the Boolean operators AND , OR , and NOT to build more complex expressions. For example:

[…]

MySQL ’ s functions can be broken down into many categories. For example, there are many date and time functions, such as now() , that retrieves the current date and time (useful when comparing dates and times against the current moment). You can also use curdate() to retrieve just the date portion of now() , and curtime() to get just the time portion:

[…]

```
mysql> SELECT * FROM accessLog WHERE lastAccess > '2008-11-04' AND lastAccess
< '2008-11-09';
+----------+-------------+-----------+---------------------+
| memberId | pageUrl     | numVisits | lastAccess          |
+----------+-------------+-----------+---------------------+
|        3 | books.php   |         2 | 2008-11-08 19:47:34 |
|        3 | contact.php |         1 | 2008-11-08 14:52:12 |
+----------+-------------+-----------+---------------------+
```

```
mysql> SELECT now(), curdate(), curtime();
+---------------------+------------+-----------+
| now()               | curdate()  | curtime() |
+---------------------+------------+-----------+
| 2008-11-09 12:17:08 | 2008-11-09 | 12:17:08  |
+---------------------+------------+-----------+
1 row in set (0.08 sec)
```

TOPIC-12: JavaScript

Module-121

Programmers, developers, and internet users have always been confused between Java and JavaScript. Many people *still* think that JavaScript is part of the Java platform, which is not true. In truth, JavaScript has nothing to do with Java, the only common thing between them is the word "Java", much like in Car and Carpet, or Grape and Grapefruit.

JavaScript is a client-side scripting language for HTML, developed by Netscape, Inc, while Java is a programming language, developed by Sun Microsystems. While in today's world calling JavaScript just a client-side scripting language would not be good, as it's now been used in servers also using node.js and people are doing object-oriented development in JavaScript, but that was what it was originally developed.

There are several differences between Java and JavaScript, from how they are written, compiled, and executed. Even the capability of Java and JavaScript vary significantly. Java is a full feature Object-oriented programming language, used in almost everywhere, starting from programming credit card to server-side coding.

Android uses Java as a programming language for creating Android apps, Swing is a Java API used to create desktop applications and Java EE is a Java platform for developing web and enterprise applications.

On the other hand, JavaScript is primarily used to bring interactivity into web pages, though there are other alternatives like Flash, JavaScript is the most popular one and regaining lots of ground lost earlier with the introduction of powerful and easy to use libraries like jQuery and jQuery UI.

1) Execution Environment

The first difference between Java and JavaScript is that Java is compiled + interpreted language, Java code is first compiled into class files containing byte code and then executed by JVM, on the other hand, JavaScript code is directly executed by the browser. One more difference that comes to from this fact is that Java is run inside JVM and needs JDK or JRE for running, on there other hand JavaScript runs inside the browser and almost every modern browser supports JavaScript.

2) Static vs Dynamic Typed language

Another key difference between JavaScript and Java is that, JavaScript is a dynamic typed language, while Java is a statically typed language. Which means, variables are declared with

type at compile time, and can only accept values permitted for that type, other hand variables are declared using vary keyword in JavaScript, and can accept different kinds of value e.g. String, numeric and boolean etc. When one variable or value is compared to other using == operator, JavaScript performs type coercion. Though it also provides === operator to perform strict equality check, which checks for type as well. See here for more differences between == and == operator in JavaScript.

3) Support of Closures

JavaScript supports closures, in form of anonymous function. In simple words, you can pass a function as an argument to another function. Java doesn't treat method as first class citizen and only way to simulate closure is by using anonymous class. By the way Java 8 has brought real closure support in Java in form of lambda expression and this has made things much easier. It's very easy to write expressive code without much clutter in Java 8.

4) OOP

Java is an Object Oriented Programming language, and though JavaScript also supports class and object, it's more like an object oriented scripting language. It's much easier to structure code of large enterprise application in Java then JavaScript. Java provides packages to group related class together, provides much better deployment control using JAR, WAR and EAR as well.

Module-122

5) Right Once Run Anywhere

Java uses byte code to achieve platform independence, JavaScript directly runs on browser, but code written in JavaScript is subject to browser compatibility issue i.e. certain code which work in Mozilla Firefox, may not work in Internet Explorer 7 or 8. This is because of browse based implementation of JavaScript. This was really bad until jQuery comes. Its a JavaScript library which helps to free web developers from this browser compatibility issues. This is why I prefer to write code using jQuery rather than using plain old JavaScript code, even if its as simple as calling getElementById() or getElementByName() methods to retrieve DOM elements.

6) Block vs Function based Scoping

Java mainly uses block based scoping i.e. a variable goes out of scope as soon as control

comes out of the block, unless until its not a instance or class variable. On the other hand JavaScript mainly uses function based scoping, a variable is accessible in the function they are declared. If you have a global variable and local variable with same name, local will take precedence in JavaScript.

7) Constructors

Java has concept of constructors, which has some special properties e.g. constructor chaining and ensuring that super class constructor runs before sub class, on the other hand JavaScript constructors are just another function. There is no special rules for constructors in JavaScript e.g. they cannot have return type or their name must be same as class.

8) NullPointerException

JavaScript is much more forgiving than Java, you don't have NullPointerException in JavaScript, your variable can accept different kinds of data because of JavaScript is dynamically typed language.

9) Applicability

JavaScript has it's own space, along with HTML and CSS in Web development, while Java is everywhere. Though both has a good number of open source libraries to kick start development, jQuery has certainly brought JavaScript on the forefront.

10) Inheritance

Java has instances and classes as separate concepts of inheritance. For doing inheritance, you need to use the base class for forming a new class and then using this new class to produce

derived instances. While JavaScript is an object-oriented language like Java, it does not use classes. You do not define classes or create objects from them. In fact, JavaScript is not class-based, but prototype-based. For doing inheritance, you can use any object instance for a prototype.

Module-123

JavaScript is largely a complementary language, meaning that it's uncommon for an entire application to be written solely in JavaScript without the aid of other languages like HTML and without presentation in a web browser. Some Adobe products support JavaScript, and Windows 8 begins to change this, but JavaScript's main use is in a browser.

JavaScript is also the J in the acronym AJAX (Asynchronous JavaScript and XML), the darling of the Web 2.0 phenomenon. However, beyond that, JavaScript is an everyday language providing the interactivity expected, maybe even demanded, by today's web visitors.

JavaScript can perform many tasks on the client side of the application. For example, it can add the needed interactivity to a website by creating drop-down menus, transforming the text on a page, adding dynamic elements to a page, and helping with form entry.

JavaScript relies on another interface or host program for its functionality. This host program is usually the client's web browser, also known as a *user agent*. Because JavaScript is a client-side language, it can do only what the client allows it to do.

Some people are still using older browsers that don't support JavaScript at all. Others won't be able to take advantage of many of JavaScript's fancy features because of accessibility programs, text readers, and other add-on software that assists the browsing experience. And some people might just choose to disable JavaScript because they can, because of security concerns (whether perceived or real), or because of the poor reputation JavaScript received as a result of certain annoyances like pop-up ads.

When you build a web application that gets served from Microsoft Internet Information Services (IIS) 6.0, you can assume that the application will usually work when served from an IIS 6.0 server anywhere. Likewise, when you build an application for Apache 2, you can be pretty sure that it will work on other Apache 2 installations. However, the same assumption cannot be made for JavaScript. When you write an application that works fine on your desktop, you can't guarantee that it will work on somebody else's. You can't control how your application will work after it gets sent to the client.

Because JavaScript is run wholly on the client, the developer must learn to let go. As you might expect, letting go of control over your program has serious implications. After the program is on the client's computer, the client can do many undesirable things to the data before sending it back to the server. As with any other web programming, you should never trust any data coming back from the client. Even if you've used JavaScript functions to validate the contents of forms, you still must validate this input again when it gets to the server. A client with JavaScript disabled might send back garbage data through a web form. If you believe, innocently enough,

that your client-side JavaScript function has already checked the data to ensure that it is valid, you might find that invalid data gets back to the server, causing unforeseen and possibly dangerous consequences.

The JavaScript developer also must be aware of the *Same-Origin Policy*, which dictates that scripts running from within one domain neither have access to the resources from another Internet domain, nor can they affect the scripts and data from another domain.

For example, JavaScript can be used to open a new browser window, but the contents of that window are somewhat restricted to the calling script. When a page from your website (*mydomain.com*) contains JavaScript, that page can't access any JavaScript executed from a different domain, such as *microsoft.com*. This is the essence of the Same-Origin Policy: JavaScript has to be executed in or originate from the same location.

The Same-Origin Policy is frequently a restriction to contend with in the context of frames and AJAX's *XMLHttpRequest* object, where multiple JavaScript requests might be sent to different web servers. With the introduction of Windows Internet Explorer 8, Microsoft introduced support for the *XDomainRequest* object, which allows limited access to data from other domains.

Module-124

Because JavaScript isn't a compiled language, you don't need any special tools or development environments to write and deploy JavaScript applications. Likewise, you don't need special server software to run the applications. Therefore, your options for creating JavaScript programs are virtually limitless.

You can write JavaScript code in any text editor; in whatever program you use to write your Hypertext Markup Language (HTML) and cascading style sheet (CSS) files; or in powerful integrated development environments (IDEs) such as Visual Studio. You might even use all three approaches. You might initially develop a web application with Visual Studio but then find it convenient to use a simple text editor such as Notepad to touch up a bit of JavaScript. Ultimately, you should use whatever tool you're most comfortable with.

Creating a web project with JavaScript in Visual Studio 2012

1. Within Visual Studio, select New Web Site from the File menu. This opens the New Web Site dialog box

2. Select ASP.NET Empty Web Site (the language selection—Visual Basic or Visual C#—is not important), as shown here. Change the name to jsbs, with a path appropriate to your configuration. When the information is correct, click OK. Visual Studio creates a new project.

Creating a web project with JavaScript in Visual Studio 2012

3. Visual Studio 2012 creates an empty project for you... really empty, with not even so much as a default page. Create a new file by selecting New File from the File menu. The Add New Item dialog box opens, as shown in the following graphic. Select HTML Page, change the name to index.html, and then click Add. Visual Studio opens the new file and automatically enters the DOCTYPE and other starting pieces of an HTML page for you.

4 & 5. In the index.html page, place your cursor between the and tags, and change the title to My First Page. Your environment should look like the one shown here:

6. Select Save All from the File menu. The finished script and page should resemble the screen shown here:

7. To view the page, select Start Debugging from the Debug menu. This starts the ASP.NET Development Server (if it's not already started) and takes you to the page in your default browser. You might see a dialog, like the following, indicating that the debugging isn't enabled in your web.config. Click OK to dismiss this dialog (and enable debugging).

Now you should receive a page with an alert, similar to the alert shown here:

8. Click OK, and then close the browser.

The script works as follows. First, the script tag is opened and declared to be JavaScript, as shown

by this code:

<script type="text/javascript">

vulearninghub.xyz | by VU Learning HUB

Module-125

JavaScript is case sensitive. You must be aware of this when naming variables and using the language keywords. A variable named *remote* is not the same as a variable named *Remote* or one named *REMOTE*. Similarly, the loop control keyword *while* is perfectly valid, but naming it *WHILE* or *While* will result in an error.

Keywords are lowercase, but variables can be any mix of case that you'd like. As long you are consistent with the case, you can create any combination you want. For example, all the following examples are perfectly legal variable names in JavaScript:

Button

One

Txt1

A

C

Tip You'll typically see JavaScript coded in lowercase except where necessary—for exam-ple, with function calls such as *isNaN()*, which determines whether a value is Not a Number (the *NaN* in the function name).

For the most part, JavaScript ignores *white space*, which is the space between statements in JavaScript. You can use spaces, indenting, or whatever coding standards you prefer to make the JavaScript more readable. However, there are some exceptions to this rule. Some keywords, such as *return*, can be misinterpreted by the JavaScript interpreter when they're included on a line by themselves. Making programs more readable is a good enough reason to include white space.

Consider the following code sample. It includes minimal white space and indenting.

The second code sample performs just like the first, but it's easier to read and follow—at least it appears so to me! I find that it takes a short amount of time to actually write code but several years to work with it. When I visit the code a year later, I'm much happier when I've made the code more readable and easier to follow.

Speaking of creating more readable code and maintaining that code over the long term: Comments are your friends. Code that seems obvious now won't be nearly so obvious the next time you look at it, especially if a lot of time has passed since you wrote it. Comments can be placed into JavaScript code in two ways: multiline and single-line.

A multiline comment in JavaScript will look familiar to you if you've coded in the C programming language. A *multiline* comment begins and ends with /* and */, respectively, as the following code example shows: **(A)**

A *single-line* comment begins with two front slashes (//) and has no end requirement because it spans only a single line. An example is shown here: **(B)**

Using multiple single-line comments is perfectly valid, and I use them for short comment blocks rather than using the multiline comment style previously shown. For example, look at this block of code: **(C)**

Tip You might find it quicker to use the two-slash method for small comments that span one line or a few lines. For larger comments, such as those at the beginning of a program or script, the multiline comment style is a better choice because it makes adding or deleting information easier.

Semicolons are used to delineate expressions in JavaScript. Technically, semicolons are not required for most statements and expressions. However, the subtle problems that you can encounter when you don't use semicolons add unnecessary errors and hence unnecessary debugging time. In some instances, the JavaScript interpreter inserts a semicolon when you might not have wanted one at all. For example, consider this statement: **(A)**

In all likelihood, you wanted to write: **(B)**

But JavaScript, acting on its own, inserts a semicolon after the *return* statement, making the code appear like this to the JavaScript interpreter: **(C)**

This code won't work. If you used this code in a function, it would return *undefined* to the caller, which is unlikely to be what you want. This is an example where free use of white space is not allowed—you can't successfully use line breaks to separate the *return* keyword from the value that it's supposed to return. But you definitely shouldn't use semicolons in one instance: when using loops and conditionals, for example: **(D)**

In this case, you wouldn't use a semicolon at the end of the *if* statement. The reason is that the statement or block of statements in opening and closing braces that follows a conditional is part of the conditional statement—in this case, the *if* statement. A semicolon marks the end of the *if* statement, and if improperly placed, dissociates the first part of the *if* statement from the rest of it. For example, the following code is wrong (the code within the braces will execute regardless of whether *a* equals 4): **(E)**

Module-126

Related closely to white space and even to semicolons in JavaScript are line breaks, sometimes called *carriage returns*. Known in the official ECMA-262 standard as “Line Terminators,” these characters separate one line of code from the next. Like semicolons, the placement of line breaks matters. As you saw from the example in the previous module, placing a line break in the wrong position can result in unforeseen behavior or errors.

Not surprisingly, the most common use of line breaks is to separate individual lines of code for readability. You can also improve readability of particularly long lines of code by separating them with line breaks. However, when doing so, be aware of issues like the one illustrated by the *return* statement cited earlier module, in which an extra line break can have unwanted effects on the meaning of the code.

JavaScript can be placed in a couple of locations within a Hypertext Markup Language (HTML) page: in the *<HEAD> </HEAD>* section or between the *<BODY>* and *</BODY>* tags. The most common location for JavaScript has traditionally been between the *<HEAD>* and *</HEAD>* tags near the top of the page. However, placing the *<SCRIPT>* stanza within the *<BODY>* section is becoming more common. Be sure to declare what type of script you’re using. Although other script types can be used, because this is a JavaScript book, I’ll declare the following within the opening *<SCRIPT>* tag:

One important issue to note when you use JavaScript relates to pages declared as Extensible Hypertext Markup Language (XHTML). Therefore, JavaScript used within strict XHTML should be declared as follows:

Older browsers might not parse the CDATA section correctly. This problem can be worked around by placing the CDATA opening and closing lines within JavaScript comments, like this:

When you place the actual JavaScript code in a separate file, you don’t need to use this ugly CDATA section at all. You’ll probably discover that for anything but the smallest scripts, defining your JavaScript in separate files—usually with the file extension .js—and then linking to those scripts within the page, is desirable. Here’s a reminder of how you link to a file using the *src* attribute of the *<SCRIPT>* tag:

When you place the actual JavaScript code in a separate file, you don’t need to use this ugly CDATA section at all. You’ll probably discover that for anything but the smallest scripts, defining your JavaScript in separate files—usually with the file extension .js—and then linking to those scripts within the page, is desirable. Here’s a reminder of how you link to a file using the *src* attribute of the *<SCRIPT>* tag:

Placing JavaScript in an external file has several advantages, including the following:

Separation of code from markup Keeping the JavaScript code in a separate file makes maintaining the HTML easier, and it preserves the structure of the HTML without you having to use a CDATA section for XHTML.

Easier maintenance Using JavaScript in a separate file, you can make changes to the JavaScript code in that separate file without touching the HTML on the site.

Caching Using a separate file for JavaScript enables web browsers to cache the file, thus speeding up the webpage load for the user.

Like programs written in other languages, JavaScript programs consist of statements put together that cause the JavaScript interpreter to perform one or more actions. And like statements in other languages, JavaScript statements can be simple or compound. This section briefly examines JavaScript statements, with the assumption that you've already seen several examples in the previous chapters and that you'll see others throughout the book.

What's in a statement?

JavaScript is more than you might think. A JavaScript statement, or expression, is a collection of tokens of various categories including keywords, literals, separators, operators, and identifiers that are put together to create something that makes sense to the JavaScript interpreter. A statement usually ends with a semicolon, except in special cases like loop constructors such as *if*, *while*, and *for*, coverage beyond the scope of this course.

Here are some examples of basic statements in JavaScript:

The two types of JavaScript statements

JavaScript statements come in two basic forms, simple and compound. I won't spend a lot of time discussing statements because you don't really need to know much about them. However, you should know the difference between simple and compound statements. A *simple statement* is just what you'd expect—it's simple, like so:

A *compound statement* combines multiple levels of logic. An *if*/*then*/*else* conditional such as the one given here provides a good example of this:

Certain words in JavaScript are *reserved*, which means you can't use them as variables, identifiers, or constant names within your program because doing so will cause the code to have unexpected results, such as errors. For example, you've already seen the reserved word *var* in previous examples. Using the word *var* to do anything but declare a variable can cause an error or other unexpected behavior, depending on the browser. Consider this statement:

```
// Don't do this!var var = 4;
```

The code example won’t result in a direct error to a browser, but it also won’t work as you intended, possibly causing confusion when a variable’s value isn’t what you expect.

The following table includes the words that are currently reserved by the ECMA-262 edition 5.1 specification: **(A)**

Several other words (shown in the following table) are reserved for future use and therefore shouldn’t be used in your programs: **(B)**

The following table shows the words that are reserved for the future when in strict mode: **(C)**

vulearninghub.xyz | by VU Learning HUB

Module-127

JavaScript has several *built-in functions*, which are functions that are defined by the language itself. Which built-in functions are available depends on the language version you're using. Some functions are available only in later versions of JavaScript, which might not be supported by all browsers. Detecting a browser's available functions (and objects) is an important way to determine whether a visitor's browser is capable of using the JavaScript that you created for your webpage.

Tip You can find an excellent resource for compatibility on the QuirksMode website *(http://www.quirksmode.org/compatibility.html)*.

JavaScript is similar to other programming languages in allowing user-defined functions. An earlier example in this module defined a function called *cubeme()*, which raised a given number to the power of 3. That code provides a good opportunity to show the use of JavaScript in both the *<HEAD>* and *<BODY>* portions of a webpage.

Using Microsoft Visual Studio, Eclipse, or another editor, edit the file example1.html in the sample code. Within the webpage, add the code in bold type:

Save the page, and then run the code or view the webpage in a browser. You'll receive an alert like the following:

The code in this example incorporates the code from the earlier example into a full HTML page, including a DOCTYPE declaration. The code declares a function, *cubeme()*, within the *<HEAD>* section of the document, like this:

This code accepts an argument called *incomingNum* within the function. An *if/then* decisional statement is the heart of the function. When the incoming number equals *1*, the function returns the text string, "What are you doing?" When the incoming number is not equal to *1*, the *Math.pow* method is called, passing the *incomingNum* variable and the integer *3* as arguments. The call to *Math.pow* raises the incoming number to the power of 3, and this value is then returned to the calling function.

All the previous code was placed within the *<HEAD>* portion of the document so that it can be called by other code, which is just what we're going to do. The browser then renders the *<BODY>* section of the document, which includes another bit of JavaScript code.

Strict mode, enhances error checking and security. To help fight against mistyped variable names, variable declarations require the use of the var keyword. Additionally, changes to the eval() function and other areas help improve the code. Strict mode is enabled with the following syntax, which is very similar to syntax used in Perl:

Strict mode is locally scoped, meaning that it can be enabled globally by placing the *use strict* line at the beginning of the script; or it can be enabled only within a function by placing the line within the function itself, like so:

Strict mode helps catch typographical errors by preventing undeclared variables. All variables in strict mode need to be instantiated prior to use. Example, code:

When used in strict mode, the preceding code would create an error condition because the variable *x* hasn't been declared with the *var* keyword, as in the following example:

Strict mode changes how the *eval()* function is handled. The *eval()* function executes a string as if it were regular JavaScript code and can lead to security issues in certain cases. In strict mode, *eval()* cannot instantiate a new variable or function that will be used outside the *eval()* statement. For example, consider the following code:

In the preceding code example, a syntax error would be produced because strict mode is enabled and the *testVar* variable isn't available outside the *eval()* statement.

Strict mode also prevents duplication of variable names within an object/function call:

Module-128

The *data types* of a language describe the basic elements that can be used within that language. You're probably already familiar with data types, such as strings or integers, from other languages. Depending on who you ask, JavaScript defines anywhere from three to six data types. (The answer depends largely on the definition of a data type.) You work with all these data types regularly, some more than others.

The six data types in JavaScript discussed in this module are as follows:

- Numbers
- Strings
- Booleans
- Null
- Undefined
- Objects

The first three data types—numbers, strings, and Booleans—should be fairly familiar to programmers in any language. The latter three—null, undefined, and objects—require some additional explanation.

Additionally, JavaScript has several reference data types, including the *Array*, *Date*, and *RegExp* types.

Numbers in JavaScript are just what you might expect them to be: numbers. However, what might be a surprise for programmers who are familiar with data types in other languages like C is that integers and floating point numbers do not have special or separate types. All these are perfectly valid numbers in JavaScript:

4 51.50 -14 0xd

The last example, *0xd*, is a hexadecimal number. Hexadecimal numbers are valid in JavaScript, and you won't be surprised to learn that JavaScript allows math to be performed using all of the listed number formats. Try the following exercise.

It's interesting to note that even though you multiplied two hexadecimal numbers, the output in the alert dialog box is in base 10 format.

JavaScript has some built-in functions and objects for working with numeric values. The European Computer Manufacturers Association (ECMA) standard defines several of them. One of them is the *isNaN()* function.

NaN is an abbreviation for *Not a Number,* and it represents an illegal number. You use the *isNaN()* function to determine whether a number is legal or valid according to the ECMA-262 specification.

For example, a number divided by zero would be an illegal number in JavaScript. The string value "This is not a number" is obviously also not a number. Although people might have a different interpretation of what is and isn't a number, the string "four" is not a number to the *isNaN()* function, whereas the string "4" is.

The *isNaN()* function requires some mental yoga at times because it attempts to prove a negative—that the value in a variable *is not* a number. Here are a couple of examples that you can try to test whether a number is illegal.

1. In Microsoft Visual Studio, Eclipse, or another editor, create a new HTML file or edit the isnan.html file in the companion content.

2. In the file, place the following markup. If you've created a new file with Vision Studio, delete any existing contents first.

3. View this page in a browser. In Visual Studio, press F5. You'll see a page like this one:

4. The function *isNaN()* returns *false* from this expression because the integer value 4 *is* a number. Remember that the meaning of this function is, "Is 4 Not a Number?" Well, 4 *is* a number, so the result is *false*.

If you're running through Microsoft Visual Studio, stop the project. For those not running Visual Studio, close the web browser.

Edit isnan.html.

Change the *isNaN()* function line to read:

document.write("Is Not a Number: " + isNaN("four"));

View the page in a browser, or rerun the project in Visual Studio. You'll now see a page like this:

In second test case, because the numeral 4 is represented as a string of nonnumeric characters (*four*), the function returns *true*: the string *four* is not a number. I purposefully used double quotation marks in each code example (that is, *"4"* and *"four"*) to show that the quotation

vulearninghub.xyz | by VU Learning HUB

marks don't matter for this function. Because JavaScript is smart enough to realize that *"4"* is a number, JavaScript does the type conversion for you. However, this conversion can sometimes be a disadvantage, such as when you're counting on a variable or value to be a certain type.

The *isNaN()* function is used frequently when validating input to determine whether something—maybe a form variable—was entered as a number or as text.

Other numeric constants are available in JavaScript, some of which are described in Table 4-1. These constants might or might not be useful to you in your JavaScript programming, but they exist if you need them.

The *Math* object is a special built-in object used for working with numbers in JavaScript, and it has several properties that are helpful to the JavaScript programmer, including properties that return the value of pi, the square root of a number, a pseudo-random number, and an absolute value.

Several other properties of the *Math* object can be helpful to your program. Some of them act as functions or methods on the object, several of which are listed in Table 4-2.

Module-129

The *data types* of a language describe the basic elements that can be used within that language. You're probably already familiar with data types, such as strings or integers, from other languages. Depending on who you ask, JavaScript defines anywhere from three to six data types. (The answer depends largely on the definition of a data type.) You work with all these data types regularly, some more than others.

The six data types in JavaScript discussed in this module are as follows:

- Numbers
- Strings
- Booleans
- Null
- Undefined
- Objects

The first three data types—numbers, strings, and Booleans—should be fairly familiar to programmers in any language. The latter three—null, undefined, and objects—require some additional explanation.

Additionally, JavaScript has several reference data types, including the *Array*, *Date*, and *RegExp* types.

Strings are another basic data type available in JavaScript. They consist of one (technically zero) or more characters surrounded by quotation marks. The following examples are strings:

The last example in the preceding list requires some explanation. Strings are surrounded by either single or double quotation marks. Strings enclosed in single quotation marks can contain double quotation marks. Likewise, a string enclosed in double quotation marks, like the ones you see in the preceding example, can contain single quotation marks. So basically, if the string is surrounded by one type of quotation mark, you can use the other type within it. Here are some more examples:

Escaping quotation marks

If you use the same style of quotation mark both within the string and to enclose the string, the quotation marks must be *escaped* so that they won't be interpreted by the JavaScript engine. A single backslash character (\) escapes the quotation mark, as in these examples:

Other escape characters

JavaScript enables other characters to be represented with specific escape sequences that can appear within a string. Table 4-3 shows those escape sequences.

Open your sample page.

Within the <*SCRIPT*> section, place the following line of JavaScript:

document.write("hello\t\t\"hello\"goodbye");

View the page in a browser.

You'll see a page like the following. Notice that the tab characters don't show through because the browser interprets HTML and not tab characters.

This rather contrived example shows escape sequences in action. In the code, the word *hello* is followed by two tabs, represented by their escape sequence of *t*, followed by an escaped double-quote \" and then the word *hello* followed by another escaped double-quote \", finally followed by the word *goodbye*.

The *length* property on a *string* object gives the length of a string, not including the enclosing quotation marks. The *length* property can be called directly on a string literal:

However, it's much more common to call the *length* property on a variable, like this:

Both examples give the same result.

Some commonly used *string* methods, besides *substring,* include *slice*, *substr*, *concat*, *toUpperCase*, *toLowerCase*, and the pattern matching methods of *match*, *search*, and *replace*. I discuss each of these briefly.

The *slice* and *substring* methods return string values based on another string. Accept two arguments: the beginning position and optional end position. Some examples:

A subtle difference between *slice* and *substring* is how they handle arguments with negative values. The *substring* method will convert any negative values to 0, while *slice* will treat negative arguments as the starting point from the end of the string (counting backwards from the end, essentially).

The *substr* method also accepts two arguments: the first is the beginning position to return, and, in contrast to *substring/slice*, the second argument is the number of characters to return, not the stopping position. Therefore, the code examples for *substring/slice* work a little differently with *substr*:

The *concat* method concatenates two strings together:

You don't work with Booleans in the same way that you work with strings and numbers; you can define and use a Boolean variable, but typically you just use an expression that evaluates to a Boolean value. Booleans have only two values, you rarely set variables as such. Rather, you use Boolean expressions within tests, such as an *if*/*then*/*else* statement. Consider this statement:

A Boolean expression is used within the *if* statement's condition to determine whether the code within the braces will be executed. If the content of the variable *myNumber* is greater than the integer 18, the Boolean expression evaluates to *true*; otherwise, the Boolean evaluates to *false*.

Null is another special data type in JavaScript (as it is in most languages). *Null* is, simply, nothing. It represents and evaluates to *false*. When a value is *null*, it is nothing and contains nothing. However, don't confuse this nothingness with being empty. An empty value or variable is still full; it's just full of emptiness. Emptiness is different from *null*, which is just plain nothing. For example, defining a variable and setting its value to an empty string looks like this:

The variable *myVariable* is empty, but it is not null.

Undefined is a state, sometimes used like a value, to represent a variable that hasn't yet contained a value. This state is different from *null*, although both *null* and *undefined* can evaluate the same way. You'll learn how to distinguish between a null value and an undefined value.

Module-130

Declaring variables

Variables are declared in JavaScript with the *var* keyword. The following are all valid variable declarations:

Variable names can contain uppercase and lowercase letters as well as numbers, but they cannot start with a number. Variables cannot contain spaces or other punctuation, with the exception of the underscore character (_). Following variable names invalid:

Take a look at the preceding example. Whereas the first three variable names are invalid because characters are used that aren't valid at all (or aren't valid in that position, as is the case with the first example), the last variable name, *var,* is invalid because it uses a keyword. For more information about keywords or reserved words in JavaScript.

You can declare multiple variables on the same line of code, as follows:

var x, y, zeta;

These can be initialized on the same line, too:

Variable types

Variables in JavaScript are not strongly typed. It's not necessary to declare whether a given variable will hold an integer, a floating point number, or a string. You can also change the type of data being held within a variable through simple reassignment. Consider this example, where the variable *x* first holds an integer but then, through another assignment, it changes to hold a string:

A variable's *scope* refers to the locations from which its value can be accessed. Variables are *globally scoped* when they are used outside a function. A globally scoped variable can be accessed throughout your JavaScript program. In the context of a webpage—or a document, as you might think of it—you can access and use a global variable throughout.

Variables defined within a function are scoped solely within that function. This effectively means that the values of those variables cannot be accessed outside the function. Function parameters are scoped locally to the function as well.

Here are some practical examples of scoping, which you can also find in the companion code in the scope1.html file:

The code defines two variables: a global variable called *aNewVariable* and a variable called *incomingBits,* which is local to the *doSomething()* function. Both variables are passed to

respective *alert()* functions within the *doSomething()* function. When the *doSomething()* function is called, the contents of both variables are sent successfully and displayed on the screen, as depicted in Figures.

1. Using Visual Studio, Eclipse, or another editor, edit the file scoping.html in the Chapter04 sample files folder, which you can find in the companion content.

2. Within the page, replace the TODO comment with the boldface code shown here (the new code can be found in the scoping.txt file in the companion content):

3. Save the file.

4. View the file in a web browser. The result is three alerts on the screen.

The first alert is this:

But wait a minute—examine the code. How many calls to the *alert()* function do you see? Hint: two are in the *<HEAD>* portion, and another two are within the *<BODY>* portion, for a total of four calls to the *alert()* function. So why are there only three alerts on the screen when four calls are made to the *alert()* function in the script?

Because this is a module on variable scoping, you might already have figured it out. But this example demonstrates well how to troubleshoot JavaScript problems when the result isn't what you expect

Module-131

In this module we will first build a web form that a pizza company might use to take orders. The company makes just a few special pizzas: one with vegetables; one with a variety of meats; and one that is Hawaiian style, with ham and pineapple toppings. The company would like a webpage with three buttons to help their pizza makers keep track of the pizza types ordered. The buttons preselect the main topping on the pizza.

Subsequently we will use another type of box—a check box—allows users to select multiple items. The pizza-ordering scenario introduced serves as a good example for illustrating the check box.

The heart of the example is twofold, the click event handler and the *flip()* function. Each input element that begins with the string special is selected with a jQuery selector. These are then looped through with the jQuery *each()* function, and a click event handler is added to each using the jQuery *on()* function. The click event handler calls the *flip()* function. The resulting code looks like this:

The flip() function looks like this:

This function examines the value of the *pizzatype* variable that gets passed into the function and then, using the conditional, changes the value of the select box, called *topping,* accordingly.

The preceding example shows how to obtain information from a form and how to set information within a form. Although the form doesn't look like much, and the pizza company isn't making many pizzas right now, it's growing because of the popularity of its pizzas. Future examples in this module expand on this form.

Recall that in the initial pizza ordering system, when the pizza order taker selected one of three pizza types, the *"Main Topping"* select box changed to reflect the main ingredient of the pizza. However, allowing more flexibility, such as more pizza types, would be nice.

The previous example showed select boxes, and you saw text boxes used earlier in this chapter, too. Another type of box—a check box—allows users to select multiple items. The pizza-ordering scenario introduced serves as a good example for illustrating the check box.

Figure at left shows a new pizza prep form. The order taker can now select from a variety of ingredients, in any combination.

Selecting the various ingredients and clicking the Prep Pizza button displays the selected pizza toppings on the screen, as shown in Figure at right.

The code for this functionality is shown in next slide.

The heart of the page is the function *prepza()*, which starts by gathering the number of check boxes contained within the form *pizzaform*. These are selected using the *name* attribute *toppingcheck* along with the *:checked* filter, all part of a jQuery selector, as follows:

$("input[name=toppingcheck]:checked").each(function() {

Each of the checked elements is looped through, and a new <*P*> element is created. Like the previous example, a click event handler is added using jQuery's *on()* function.

Keep this example in mind, because you can use it to combine with functionality that automatically selects toppings when a user presses a button, as in the select box example you saw earlier.

Radio buttons also create a group of options, but unlike check boxes, only one radio button from the group can be selected at any given time. In the context of the pizza restaurant example, visitors might use a radio button to select the type of crust for the pizza: thin, deep dish, or regular. Because a pizza can have only one kind of crust, using radio buttons for this selection type makes sense. Adding radio buttons to select a crust type results in a page like that shown in Figure.

The code that processes the radio buttons is similar to the code you saw that processed the check boxes. The main difference is that radio buttons all share the same name and logical grouping, meaning that they are grouped together and only one can be checked at a time. The code for processing the radio buttons is added to the *prepza()* function, like this:

Module-132

You should never assume that what gets to the server is valid. Many web developers known to have said, “We have a JavaScript validation on the data, so we don’t need to check it on the server.” This assumption couldn’t be further from the truth. People can and do have JavaScript disabled in their browsers; and people also can send POST-formatted and GET-formatted data to the server-side program without having to follow the navigation dictated by the browser interface. No matter how many client-side tricks you employ, they’re just that—tricks. Someone will find a way around them.

The bottom line is that you can and should use JavaScript for pre-validation. Pre-validation is a small sanity check that can be helpful for providing quick feedback to users when your code notices something blatantly wrong with the input. But you *must* perform the actual validation of all input on the server side, after users have submitted their input completely.

This module uses a server-side program to create a catalog order system that has three simple elements: a product, a quantity, and a price. The items to be sold are blades of weed from Saad’s lawn. Because blades of weed are from my lawn are so rare, orders are limited to three blades per household, and the price is high. I limit the order quantity by using some JavaScript code.

I created a page to sell the blades of weed. When viewed in a browser, the page looks like Figure 15-9.

Here’s the HTML and JavaScript to produce the page. Note also that you won’t be able to submit the form because the form action, catalog.php, doesn’t actually exist. The action of the form isn’t that important to this example.

With JavaScript enabled in my browser, the user’s attempt to order a quantity of three or fewer blades of weed is acceptable, so the form gets submitted to the server-side script, which handles the request and returns an order total, shown in Figure-A.

If the user goes back to the page, still with JavaScript enabled, and attempts to order a quantity of four blades of grass, he or she sees an *alert()* dialog box, like the one shown in Figure-B

So far, so good. Now imagine that I disabled JavaScript in my browser. There’s no noticeable change in the page when I go to the order form, so the page looks exactly like the one in Figure-A. However, I’m now able to order a quantity of 1,500. Simply entering 1500 into the quantity and clicking Place Order results in the server-side web form happily receiving and processing the order, as shown in Figure-C

Because no validation existed on the server side, this input was perfectly valid, and the order could be processed. The only problem is that I don't have 1,500 blades of grass on my lawn (I counted), so I can't possibly fulfill this order.

With JavaScript enabled in my browser, the user's attempt to order a quantity of three or fewer blades of weed is acceptable, so the form gets submitted to the server-side script, which handles the request and returns an order total, shown in Figure-A.

If the user goes back to the page, still with JavaScript enabled, and attempts to order a quantity of four blades of grass, he or she sees an *alert()* dialog box, like the one shown in Figure-B

So far, so good. Now imagine that I disabled JavaScript in my browser. There's no noticeable change in the page when I go to the order form, so the page looks exactly like the one in Figure-A. However, I'm now able to order a quantity of 1,500. Simply entering 1500 into the quantity and clicking Place Order results in the server-side web form happily receiving and processing the order, as shown in Figure-C

Because no validation existed on the server side, this input was perfectly valid, and the order could be processed. The only problem is that I don't have 1,500 blades of grass on my lawn (I counted), so I can't possibly fulfill this order.

Because no validation existed on the server side, this input was perfectly valid, and the order could be processed. The only problem is that I don't have 1,500 blades of grass on my lawn (I counted), so I can't possibly fulfill this order.

You might be tempted to dismiss this scenario as contrived, but it represents an all-too-common occurrence in web applications. In fact, this example is relatively tame compared to some situations in which a site actually lets a visitor change the price of an item during the ordering process and never bothers to validate that input—because "no one will ever do that." Well, people have done that before, and they will again—if you don't stop them.

You might be tempted to try to solve the problem by requiring that all visitors have JavaScript enabled in their browsers before they can place an order—but that doesn't work. You can attempt to figure out if JavaScript is enabled, but you can never be 100 percent certain.

The only correct way to solve this issue is to validate and to enforce valid rules on the server side. The back-end script should check the business rule of the quantity limitation. Doing this won't be a problem the vast majority of the time, but it takes only that one time—and then I'd be outside trying to dig up 1,500 blades of grass for my customers.

This module showed how easy it is to bypass JavaScript validation by simply turning off JavaScript in the browser. JavaScript should be used only for pre-validation and never as the sole means of ensuring that input is valid.

TOPIC-13: Server side operations

Module-133

This is a checklist of the tools that you need. Most have been discussed at length in previous modules; you should now have enough information to make your choices. These steps need to be done both sequentially and simultaneously—almost everything depends on everything else. Fortunately, you do it only once.

Whether it is a commercial Internet service provider, an in-house intranet, or some combination, you need a basic Web site and the means to get to it. If you are setting up your own Web server, you need to select the software (and hardware) to run the Web site; for most people, the choice of Web server hardware and software does not matter, whatever your ISP offers is what you get.

Compatibility

Remember, however, that your database must be able to interact properly with the Web server hardware and software. If the database is to run on the Web server (rather than on another machine), it must run on the hardware and software involved: Microsoft Access runs only on Windows platforms, and so forth.

If you rely on application server software to run on your Web server (either as a plug—in or as an extension make certain that it can run and that it does run—the fact that an ISP can run a product like does not mean that they will do so for one customer.

Domain name

This is also the time to get your domain name resolved. If you do not have one, obtain one now from the appropriate agency. If you do not use a domain name, you leave yourself open to having to reprint stationery and business cards as well as to change Web pages whenever your location changes. This may be due to your changing ISPs—or even to the ISP changing part of its configuration. Your domain name is yours and you can rely on it staying unchanged.

Your application server—the interface between the Web server and your database can be part of the Web server (as is the case with Microsoft Internet Information Server), a plug-in to that server, or a separate application that is called by the Web server as needed (as with a Perl script or other).

The application server needs to be compatible with everything: your choice here may limit your choices with regard to ISP or database in some cases to one option. Because this choice can be so limiting, make certain that you understand your options. If you are selecting a package that

includes a database and application server, understand that that combination may be yours forever: you may be able to change your database only if you change your application server (and vice versa).

Your database must run on the hardware available and must interact correctly with your application server (which is why so many application server / database combinations are available). Most databases today are based on SQL, and for that reason your database design will probably be transportable to another database (together with all of your data); however, your scripts, transactions, and processes defined in your application server may not be transportable.

In fact, this can be a blessing. If you find that you have made the wrong choice of application server (too complicated, not well supported, or not reliable enough), you can pick another one and move your data—if necessary—to another database. Rewriting your application server scripts, processes, and transactions is much less of a problem than redesigning a database.

Your database may not be your choice: in a large organization, there may be an enterprise-wide database (or database standards) to which you must adhere. In all other cases, however, it is a good idea to keep up with at least one database other than the one you use. You do not have to duplicate your work—or even do any work in the other database, but you should spend a little time thinking about how you would use another database to accomplish your goals. As noted previously, databases today are remarkably similar. If you find something that you are doing that really is possible only with one database, make a note of it, and remember that you are sowing the seeds of nonportable solutions.

With your ISP, Web server, application server, and database in place, you can start to actually develop your Web site. Again, you have a variety of choices, and prudence suggests that you try to avoid tying yourself to a single product that may in turn tie you to a specific database, Web server, or application server—or even to particular versions of those products.

In addition to those products, remember that you are normally designing Web pages to be used by people who use different browsers and different versions of those browsers. Even within a private intranet, it is rarely the case that every user uses the same version of the same browser.

When it comes to authoring tools, the situation is the reverse of that with application‘ servers and databases: you do not have to make a choice. Remember that the benefits of the Web derive in large part from reliance on the international HTML standard. Creating your website with a text-based editor or some tool you are creating the same material. You will increase your options if you do not tie yourself to a single product. (Furthermore, you may decrease your costs if you need to hire temporary or full-time web developers and you can let them use

vulearninghub.xyz | by VU Learning HUB

whatever authoring tools they choose.) If you do make deliberate choices that limit your other options, make certain to document them for future reference.

Your database, Web pages, scripts etc. need to get onto your Web and application servers for which you need FTP (file transfer protocol). FTP consists of two basic connections: a control connection and a data transfer connection. There are programs that implement FTP by itself; in addition Web authoring tools and Web site management tools often include FTP functionality so that you can manage a site.

Be careful of integrated tools: they often assume that you are working with a simple site located on a single Web server—and that often is not the case with database-driven Web sites. Some of your site's files may need to be uploaded to different directories or different servers, and the complexity of that structure may best be handled by an FTP program that lets you manipulate files and directories explicitly.

Since FTP is used relatively infrequently, choose the tool that you are most comfortable with—one whose operation is most intuitive. Note that changes to your Web pages can frequently be made without reuploading the pages; one of the benefits of creating Web pages from database data.

E-mail software is not actually part of your Web site, but you need a good e-mail program to manage your site. You may generate e-mail messages automatically when things happen (this involves the mail server at your ISP, not an e—mail program on your computer). In order to deal with the e-mail that you receive from your site, you need a program with which you are comfortable; it should also be able to filter messages based on subject and address fields. That way, messages to info@yourdomain.com can wind up in a different folder on your personal computer than messages sent to you@yourdomain.com—even though both are delivered to your e-mail account.

vulearninghub.xyz | by VU Learning HUB

Module-134

Your site is very special to you—you know it and understand it, and you probably know your way around it better than anyone else in the world. Although it may shock you to think about it, some people who use your site will care only about a small section of it—perhaps only a single page. People need to be able_ to maneuver through your site to the information that they want without undue difficulty.

It is very hard to redesign an entire site after it has been put up—you may need to make changes on every single page. Before you set your first page in HTML (a less mutable medium than you might think), consider what your site should look like. There are three structures in common use (and many combinations of them are used, too):

1. Unified sites, where each page has a guide to the entire

2. Distributed sites, where each page follows certain design standards but where the navigation is facilitated within each subsite rather than across the site as a whole

3. Fragmented sites, where each page or subsite obeys its own rules

Designing a Unified Site A unified site has its entire structure visible on every page: from every page you can go to every section or subsite of the main site.

The Implementation of this particular site relies on frames: there are actually two separate HTML pages shown here-one is the page with the navigation buttons at the left (with its own scrollbar) and the other is in the center and right of the implemented without using frames: in such a case, the navigation buttons on the left are repeated on each of the site's Pages)

lf you click on the Events button, you go to the corresponding page. That page is also actually two HTML pages—the frame at the left contains the same HT'ML page as in Homepage; the frame in the center contains new content for this page. (Again, although this implementation uses frames, the design can be carried out without using frames.)

At first glance, a unified site might appear to be the ne plus ultra in site design—until you start to think about the mechanics of designing, implementing, and using it. That unchanging set of buttons at the left of each page takes up precious space, and for someone who is interested only in one area (or even one page of the site) they are a distraction. Furthermore, their space could be used for navigation within a subsite, and that is not possible.

A unified design works best when the number of navigation buttons (subsites) is relatively small—under 10, when the subsites are not complex. If the Events of Y2K in this example were to need their own set of navigation buttons, you would quickly have a page with more navigation tools than content. (The page shown in Figure 14-2 is a good example of the type of

subsite that works with this design: the subsite page has a number of links to its individual pages, and they have no further links within the site.)

Designing a Distributed Site A distributed site keeps certain design and navigation elements constant across all pages, but it takes liberties with them—the most common being to use the navigation tools to apply to each subsite rather than to the site as a whole. Here you see another home page, that of the Mid-Hudson Library System: it uses this technique.

This site does not use frames, but it is similar to the page shown in last slide: the sections of the site are shown in the navigation bar at the top. (This page uses tables, rather than frames.

If you click on the Online catalogue link under the For the public link, you can go to the corresponding page, which lets you choose from a variety of sources. Note that this page uses the same logical layout as the home page, but its navigation bar reflects options for this section rather than for the site as a whole.

A site such as this that allows each of its subsites to set its own design and navigation rules can be considered a distributed rather than a unified site.

Distributed sites avoid many of the problems with unified sites: changes to the layout of the site affect navigation tools only within the affected subsite, and there is more space on each page for customized navigation tools. In designing such a site, it should be clear what design and navigation elements are required to be used on all pages, which elements are required on all subsite main pages, and what (if any) design navigation elements may not be used.

Using Fragmented Sites The rules for subsites can be very loose: they can copy the general style and navigation rules (as the page at the left hand-side in last slide does from the page on the right-hand side), or the subsite pages can be totally delinked to the main site—as shown on this slide, the page that appears if you click the Calendar link the page at the left hand-side in last slide.

You would not know that this page is part of the same site, and it may not be surprising to learn that this page could be located on a different computer and that it uses totally different software from that used in pages on last slide. It is the calendar page for the library catalog that could be using a database (Informix) to store its data‘ It is integrated with the pages on previous slide because you arrive at it through their links, but it looks very different.

Making Your Choice Many people think that a unified site structure is the most elegant and efficient until they consider these points:

- It may be impossible to decide on what the overall site structure—which appears on every page—should be. Whether this is because a multinational enterprise has trouble

vulearninghub.xyz | by VU Learning HUB

reaching consensus or because you keep changing your mind about how to present your personal Web site, the problem is essentially the same.

- The unified structure is actually quite un Web-like. The essence of the Web is that people can jump around from place to place, clicking on links or entering URLs. Most people do not stay within a single site. (Exceptions to this are corporate intranet sites that are designed to allow no access to the outside or to make a very clear distinction between inside—the unified corporate site—and the Web in general.)

- Most important, for database-driven Web sites, you may not be able to modify pages that are generated dynamically to match a predetermined template. Often the use of frames can solve this problem, but in many cases, a prepackaged design comes along with a commercial database.

vulearninghub.xyz | by VU Learning HUB

Module-135

Beneath the logical and well-organized site that the user sees, you may need to manage a site that is organized according to a very different logic: pages requiring the services of your database or application server may be on one (or more) servers—regardless of the section of the site that they are in—while static HTML pages may be on different servers. As noted previously, each of these servers may be located at a different physical location; one or more of them may be under the control of your organization, and one or more of them may be managed by an ISP or DSP.

As if that were not complex enough, you need to establish a way to develop, test, and use your static and dynamic HTML pages. In traditional Web sites, you may establish different environments—each is a complete copy of your Web site, and each represents a different version of the site.

Production/server

Ultimately, your files must be placed on a Web server where people can access them. You typically do not have control over the directory in which these files are placed, but you do have control over the subdirectories.

These files are the ones that people access. No files other than your finished files should be in this directory. You may think that a file called private.htm will be invisible to everyone if you do not have a link to it; but it is very easy to list the Files in a directory on a Web server.

All of these files should be backed up as part of the web servers' routine maintenance. Unless you are the administrator, you should not be responsible for backing up the files that you have placed on the server.

Production/mirror

This is an identical copy of your production/ server environment. You may set up your Web server to distribute user requests among the production/ server and production/ mirror environments. Frequently, a number of production / mirror environments are set up—often at various locations. lt is usually cheaper to have a number of mirror servers set up than to have one massive server that can fulfill all of your user's needs. In addition, it is more reliable to have a number of redundant systems than to have one critical system. Of course, as with all aspects of complex Web sites, mirroring adds to complexity at the same time that it reduces operational costs.

Production/backup

If it is at all possible, you should have a complete copy of the files and folders on your Web site on a single computer that is not the Web server. This is your production/ backup site the site that you can use for final testing.

This site is on your computer and is under your control. You should be responsible for backing it up whenever changes are made; you should also be responsible for moving files from it to the Web server.

Do not allow your production/ backup site to differ from the production / server site. Sometimes a few files accumulate on one site that are not duplicated on the other one. Deal with these either by deleting them or by duplicating them to the site from which they are missing. Keep these two sites identical.

When a file needs to be modified or added to your site, you move that file into the production/ backup environment. You should run' through your test procedures there; when you are satisfied, move the file onto your Web server.

Note that you move files into your production/backup environment: you do not create them or make changes to them there.

Test environments

A test environment lets you experiment with Web pages and your databases. It differs from the production/ backup environment in that it may not have your entire site on it but only the files on which you are working. .

Depending on your security needs and the nature of your site, you may or may not allow modifications to files directly in the test environment. You certainly should never allow such modifications directly in either the production/ backup or production/ server environment, and you always allow that behavior in your development environments.

Your test environment may need to have multiple servers if your production environment has multiple servers. If it does not, you will wind up with different file naming conventions in the two environments; things will work in one case but not in the other. (Typically they will work in test and fail in production see standard references on Luck and Fate for further information.

Development environments

A development environment is just that—an area where you and your colleagues can work on databases and Web pages. All bets are off in these environments: you can rename files, you can move them, and you can change the structure of the site as your ideas evolve. This is the only way to be productive in developing a Web site.

Because you need this kind of freedom you will need to establish a development environment where this can happen: you cannot do these things in the production or test environment, if you make exceptions (even for yourself) before you know it you will have different file structures in each environment and you will not know what files belong where.

Archives

Another common environment that you may have consists of archives. You should have regular backups of all of the files on your computer, but you may want to make special backups (perhaps on removable media) of your production / mirror environment every time you change it.

If you have not worked in a controlled environment before, this may seem like a lot of overhead to you. In fact, it is the standard way to control large systems involving multiple files. Before long it will become second nature to you.

If you cut corners, you will soon find yourself with incompatible versions of files all over the place; even worse, you will not know what is the correct combination of files to make your Web site function. Unfortunately, as with disk backups, it usually takes an accident to convince people of the need for such preventive actions. Remember that your Web site—even if on an intranet—is a very public area. Do you really want your boss to ask you why the site is all messed up?

It is rare that you can establish a full panoply of environments such as these when you are dealing with database driven Web sites: the number of servers involved in each of the major environments is large, and items such as the cost of additional licenses for test versions of databases can quickly add up. Figure out what you need, what you can afford, and what you can maintain.

vulearninghub.xyz | by VU Learning HUB

Module-136

Beneath the logical and well-organized site that the user sees, you may need to manage a site that is organized according to a very different logic: pages requiring the services of your database or application server may be on one (or more) servers—regardless of the section of the site that they are in—while static HTML pages may be on different servers. As noted previously, each of these servers may be located at a different physical location; one or more of them may be under the control of your organization, and one or more of them may be managed by an ISP or DSP.

As if that were not complex enough, you need to establish a way to develop, test, and use your static and dynamic HTML pages. In traditional Web sites, you may establish different environments—each is a complete copy of your Web site, and each represents a different version of the site.

Do not rename files

The cardinal rule is never to rename a file. Once you have given it a name, that is its name forever and ever. Adhering to this rule will prevent the broken links that are generated when you rename a file and leave HTML pages pointing to the old name.

Renaming a file includes moving it to a new folder or directory: that changes its name for some purposes. Thus, this rule means that you must put the file wherever it is going to be with the name it is going to carry-forever.

File structures for easy maintenance

Managing a Web site quickly becomes a difficult task as files are modified, added, and removed. Managing a Web site that also involves databases is even more complex.

On your desktop, you can pretty much put files anywhere you want and call them anything you want. To a large extent that is true on the Web, but in practical terms you will find substantial limits on the freedom you may be used to.

Naming conventions and filing strategies differ from computer to computer—as they do from person to person and from organization to organization. Using a very standard structure will often make it easier for other people to access your files (either for maintenance or with Internet browsers).

Eight dot three

The most restrictive naming convention is that used in DOS (and later in the first versions of Windows)——an eight—character filename followed by a three-character suffix that identifies the file. Examples are schedule.doc, program.exe, and autoexec.bat

Some operating systems (Mac OS, for instance) incorporate the information from the suffix in the file itself; thus, the file-name is not used to identify the kind of file it is. Other operating systems (Windows 95, for instance) use the suffix to help identify the file type and determine the kind of icon to represent it with, but they normally do not display the suffix. Other operating systems (Unix, for example) allow multiple suffixes that can be interpreted in various ways.

Of all of these, the eight-dot—three conventions is the most restrictive and therefore will work on most platforms. Although you need only a naming convention that will work on your Web server, adhering to this convention will ultimately make it easier to support your files.

Capitalization

Some systems (like Unix) distinguish between upper— and lowercase characters: the file schedule.doc is not the same as Schedule.doc. Others (like Windows and Mac OS) do not make this distinction. For this reason, it is best not to rely on capitalization to distinguish between files. Whatever conventions you use, remember that capitalization should be used only to provide extra information to people and not as a distinguishing characteristic of files.

Keep track of files and folders

Because you are not going to be moving or renaming folders, it makes sense to keep track of what you have. Files and folders have been known to disappear from Web servers: you need to be able to restore what should be there.

The easiest way to keep track of your Web site is to keep an updated list of each file, together with its contents, its update date, and the folder in which it belongs. You can create a small database with this information and publish it on your Web site (or on a section of your site that is available only to your project team).

Note the distinction between a database listing the files that you think should be on your Web site and the directory listing of your Web site, which is a listing of what is actually there. If you have a housekeeping accident and accidentally delete files or move them to the wrong directory, the database will help you reconstruct what should be where.

Establishing directories

vulearninghub.xyz | by VU Learning HUB

The principle of not renaming files extends to directories. As a consequence, it will not do to name a directory ClientTst and another one ClientPrd (for test and production environments). Create your directories at a higher level—such as Test and Production—and then place all of the files and folders for your Web site in each folder. That way, nothing will ever have to be renamed. Within the Test, Production, and other folders, you may have duplicate files and duplicated directory structures, but since they are in separate environments, nothing should be confused.

Web servers are very good at accessing other files within the same directory as the file that they are currently processing. Most people get into trouble when they try to mix files from other directories.

If you think of each site and area of your site as its own site and place' it within its own directory, all will be well. Within each directory, place a default file—called default.htm, index.html, or home.htm according to your Web server's standards.

Special-purpose folders

Within your Web site, you can have any number of subfolders. As noted previously, you can use such subfolders (or subdirectories) for self-contained portions of your site—sub—sites, in fact. Within each site (or subsite), there are often special folders for all files, but leave incremental revision numbers to fluctuate separately for each file.

Versioning the web pages

It is normally a good idea to place a version number at the bottom of each Web page. You may also place a date there, but for infrequently updated pages that may suggest to users that the site is getting a bit old to be relied on (even though with dynamic HTML that displays data from a database the content of the page may be much more recent than the date that is displayed).

You may want to consider placing the modification date of each Web page as a hidden field that you can view but that is not shown to users. (You can place a version number as part of the text to be placed in the HEAD element of each page on your site.)

Versioning the databases

Many databases allow you to set a version explicitly or to set a global field for a table or database into which you can place a version number.

Note that a database version typically applies to the database format—its fields and layouts. A database timestamp shows the date and time of last update.

vulearninghub.xyz | by VU Learning HUB

Module-137

Managing a database-driven Web site is often much easier than managing a traditional Web site: once it has been set up, it chugs along by itself, and the Webmaster or other coordinator often has less work to do. This is because pages do not need to be updated whenever any piece of information on the site changes: the underlying databases are updated, and the revised data automatically flows onto Web pages as necessary.

Nevertheless, you have to set up the site properly to make this dream of simple management a reality. This module covers the major issues:

1. Managing databases from afar
2. Database housekeeping
3. Scheduling maintenance and updates
4. Managing change (testing)

What actually happens when you have a database-driven site?

Although some people maintain their own Web server and application server, for many people it is an Internet service provider (ISP) that maintains these computers and keeps the Internet and database software running. In real life, only a few people have access to their Web server (which provides all Internet services), their application server (which may be the same computer or may provide only database services), and all aspects of their databases (with maximum access privileges).

Databases That Move and Those That Do Not

There are two basic scenarios for managing a database on the Web: you can move the database to and from your Web site as you do your HTML pages or you can move data to and from the database on your database service provider's computer.

In the first case, you have a great deal of control; in the second, the database service provider maintains the database and you work within it. Typically, personal computer—based databases (such as Microsoft Access and Ninox database) are managed in the first way, databases such as Oracle, DB2, and Intermix in the second.

Your DSP should provide routine backup (at least daily). These routine backups are for all files on the database and Web servers. The DSP takes the precautions necessary to ensure that in

case of a disaster the files can be restored from a reasonable point in time. However, there are several reasons you might want your own backup schedule:

- You may want to keep backups from specific points in time: ends of months, years, semesters, etc.
- You may want more than one previous backup.
- You may want more frequent backups—if you have a class registration database, you may want to back it up hourly during the days of semester registration.
- You may need to place backups at a certain location for Security purposes.

With regard to archiving and partial archiving of data. You may want to remove old records from your online database but keep them in a backup. In the case of a guest book, you might want to upload a totally empty database periodically and retrieve the previous one for integration with your master database on your local computer.

If doing database maintenance in this way, remember that you will temporarily be causing links to fail. The best way to manage this is to have an alternate home page that announces that the site is undergoing maintenance. Start by replacing your normal home page with this page, and then move databases back and forth. When you are done, restore the original home page.

Many people do not spend much time browsing their own Web sites: they know them well. As a result, it is not uncommon to get a telephone call or email message remarking that your site (or part of it) has been in disrepair for some time. Make certain that you have a routine schedule to check the site.

Furthermore, you need a schedule for database maintenance. This may take the form of a wholesale database cleaning and reorganization or just a relatively minor check that everything looks OK.

A database—driven Web site should take care of itself to a large extent; however, you cannot wait until a disaster has happened to implement a maintenance schedule. Be particularly sensitive to the fact that such a site often involves a variety of people: at least one Internet service provider, a database administrator, Web designers, authors, editors, and testers. You can implement a relatively permanent schedule of maintenance (such as database maintenance on the first of the month, Web page maintenance on the fifth, and so on); alternatively, you can schedule a wholesale maintenance effort on a certain routine basis (such as database, Web page, and content updates all during the last week of the month).

Whatever you do, do not think that this is a maintenance-free effort. The results will show!

vulearninghub.xyz | by VU Learning HUB

Many are the tales of minor changes to computer programs that have brought down major systems—for hours or days. There is no such thing as a minor change in an integrated digital world. -

There is also no such thing as a Web site that does not change. You need procedures to manage change—and those procedures must include testing. Depending on the nature of your site, you may need several sets of procedures for managing change: if you may have emergency changes; you need a streamlined procedure that saves time but does not bypass security and other restrictions. (Many security breaches come in the aftermath of emergencies)

In general, the person who has made a change is not the best person to conduct final testing. The person making a change should do preliminary testing to make certain that everything is working properly, and in fact everyone should assume that there will be no further testing—do not rely on someone else catching problems.

vulearninghub.xyz | by VU Learning HUB

Module-138

Whether it is on the Internet, an intranet, or a local area network, your database is now public. The typos and misspellings in the data records are visible; the incorrect information is published. The interface to your data becomes part of your data both explicitly (if you switch the "Length" and “Width” data field labels, people will receive invalid information) and implicitly (as the quality of your data is judged based on the quality of your interface).

Not only are the contents of your database and Web site public: your maintenance of them is also public. “Temporarily unavailable" is not the best response to a search request from a user.

The principal issues to consider are as follows:

What are you doing on the Web?

What your site says about you?

Letting people know about your site?

The Internet is a mass medium in terms of its reach (scores of millions of people); however, its performance from moment to moment as an individual user clicks from link to link is less that of a mass medium than that of a highly personalized and individual communications medium. What an individual sees—through links traversed—is almost always a unique sequence. Indeed, with dynamically built Web pages based on database searches, the very pages that a user sees are unreproducible and transitory.

In addition to being simultaneously a mass and a personal medium, the Internet (as is true of all networks to a greater or lesser extent) brings together people from a wide variety of backgrounds and areas. Your site will most likely be used by people who share certain interests (mountain trekking, for example) that are relevant to your site; at the same time, these people will bring different regional, cultural, linguistic, and social perspectives to your site.

The look and feel of a site often betray unwanted and unnecessary information about its developers and sponsors. In an international context, references to “abroad” are ambiguous or insulting. It is not a matter of political correctness to make certain that the manner in which you present your site and its information does not make people feel unwelcome: it is a matter of good manners (and often good business).

By the same token, the‘ look and feel of a site can provide a vast array of desirable information about its developers and sponsors. A database of movies is perceived very differently if it is encountered on a site sponsored by a church, by a women's group, by a school, or by a minority association—even if the database contains exactly the same information in each context.

It is a mistake to believe that your site's information is unaffected by its context; it is an even bigger mistake to believe that you have somehow or other managed to construct a neutral environment for your information. Its mere presence on a network makes a statement about the potential users and the information: the users are computer literate (or semiliterate) and the information is more or less public.

Functionality across Devices

Functionality can follow the size of the device. Sometimes different aspects of a website are lost when it is viewed on Mobile or pad. You should take every step possible to make every part of your website accessible.

User Experience

The way an individual uses a mobile device and a pad are different. Different platforms offer different layouts. You should consider the different ways an individual might access your website and where to give them an appropriate experience.

More than Layout

Layout is important. However the different functions of a device a website is seen on should be considered. If a person makes a phone call or is watching a movie at the same time on another screen should all be taken into account during the design phase.

Considering Device Dimensions

Since a designer or programmer cannot control the device and dimensions our website is viewed. Therefore, have a flexible and adjustable design that includes break points.

Utilizing Fancy Script

If you use complicated scripts like Flash or JS it might provide an excellent experience for a desktop user. This can provide a complication for many mobile users. It can take longer to load and might end up being inaccessible.

http://www.weviokorea.com/responsive-web-design-concerns/

There are two ways of publicizing your site:

1. Place references to it on other Internet resources; people can place links to your site on their sites.

2. Place these references in other media-magazine articles, corporate newsletters, books, etc. This is the best way to reach people who do not normally use the Internet.

Achieve one goal: do not change your address. This will invalidate all your efforts. Follow as many of the following guidelines as possible.

Do not change your address: Make certain that your site is (or is part of) a named domain that you control. For a modest fee, your Internet service provider (ISP) can help you obtain an address such as mycompany.com. It is a simple matter to change the Internet routing tables if you should move mycompany.com to another Internet service provider. Make certain that you are named as one of the contacts for your site.

Identify a site in relation to known address: If you do not have a domain of your own (for example, if you are a department of a corporation), give out your site's address in a context that will not change. It is easy to place a button on a corporation's home page that links to your site. Then, give out mycorporation.com as your address (if necessary, telling people to click on Division X).

The site is part of your address: Once you' are satisfied that you have a site address that will not change, make certain that it is part of your return address in e—mail, regular mail, and on letterheads.

vulearninghub.xyz | by VU Learning HUB

TOPIC-14: NoSQL

Module-139

New and often crucial information is generated hourly, from simple tweets about what people have for dinner to critical medical notes by healthcare providers. As a result, systems designers no longer have the luxury of closeting themselves in a room for a couple of years designing systems to handle new data. Instead, they must quickly create systems that store data and make information readily available for search, consolidation, and analysis. All of this means that a particular kind of systems technology is needed.

The good news is that a huge array of these kinds of systems already exists in the form of NoSQL databases. The not-so-good news is that many people don't understand what NoSQL databases do or why and how to use them. Not to worry, though. That's why this course is for you. In this and subsequent modules, will introduce you to NoSQL and help you understand why you need to consider this technology further now.

Google Bigtable paper

In 2006, Google released a paper that described its Bigtable distributed structured database. Google described Bigtable as follows: "Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers."

Similar to an RDBMS model at first sight, Bigtable stores rows with a single key and stores data in the rows within related column families. Therefore, accessing all related data is as easy as retrieving a record by using an ID rather than a complex join, as in relational database SQL.

This model also means that distributing data is more straightforward than with relational databases. By using simple keys, related data — such as all pages on the same website (given as an example in Google's paper) — can be grouped together, which increases the speed of analysis. You can think of Bigtable as an alternative to many tables with relationships. That is, with Bigtable, column families allow related data to be stored in a single record.

Bigtable is designed to be distributed on commodity servers, a common theme for all NoSQL databases created after the information explosion caused by the adoption of the World Wide Web. A *commodity server* is one without complex bells and whistles — for example, Dell or HP servers with perhaps 2 CPUs, 8 to 16 cores, and 32 to 96GB of RAM. Nothing fancy, lots of them, and cheaper than buying one big server (which is like putting all your eggs in one expensive basket).

The first documented use of the term NoSQL was by Carlo Strozzi in 1998. He was visiting San Francisco and wanted to get some people together to talk about his lightweight, relational database.

Carlo used the term NoSQL because his database was accessed via shell scripts, rather than through use of the standard Structured Query Language (SQL). The original meaning was "No SQL." That is, instead of using SQL, it used a query mechanism closer to the developer's source environment — in Carlo's case, the UNIX scripting world.

Carlo's meeting in San Francisco came and went. Developers continued to experiment

with alternate query mechanisms. Technology appeared to abstract complex queries away from the developer.

There's a cost to using SQL. Complex queries are hard to debug, and it's even harder to make them perform well, which increases the cost of development, administration, and testing. Finding an alternative mechanism, or a library to hide the complexities at least, looked like a good way to reduce costs and make it easier to adopt best practices.

Amazon Dynamo paper

Amazon released a paper of its own in 2007 describing its Dynamo data storage application. In Amazon's words: "Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance."

The paper goes on the describe how a lot of Amazon data is stored by use of a primary key, how consistent hashing is used to partition and distribute data, and how object versioning is used to maintain consistency across data centers.

The Dynamo paper basically describes the first globally distributed key-value store used at Amazon. Here the keys are logical IDs, and the values can be any binary value of interest to the developer. A very simple model, indeed.

Many open-source NoSQL databases had emerged by 2009. Riak, MongoDB, HBase, Accumulo, Hypertable, Redis, Cassandra, and Neo4j were all created between 2007 and 2009. These are just a few NoSQL databases created during this time, so as you can see, a lot of systems were produced in a short period of time. However, even now, innovation moves at a breakneck speed.

The #NoSQL hashtag is the first modern use of what we today all regard as the term NoSQL. The description from the meeting is well worth reading in full — as the sentiment remains accurate today. "*This meetup is about 'open source, distributed, non relational databases'.*

This meetup included speakers from LinkedIn, Facebook, Powerset, Stumbleupon, ZVents, and couch.io who discussed Voldemort, Cassandra, Dynamite, HBase, Hypertable, and CouchDB, respectively.

This explosion of databases happened because non-relational approaches have been applied to a wide range of problems where an RDBMS has traditionally been weak. NoSQL databases were also created for data structures and models that in an RDBMS required considerable management or shredding and the reconstitution of data in complex plumbing code.

Each problem resulted in its own solution — and its own NoSQL database, which is why so many new databases emerged. Similarly, existing products providing NoSQL features discovered and adopted the NoSQL label, which makes the jobs of architects, CIOs, and IT purchasers difficult because it's unlikely that one NoSQL database can solve all the issues in a particular business area.

So, how can you know whether NoSQL will help you, or which NoSQL database to choose? The answer to these questions will be covered in some of the upcoming modules by discussing the variety of NoSQL databases and the business problems they address.

Module-140

Schema agnostic: A database schema is the description of all possible data and data structures in a relational database. With a NoSQL database, a schema isn't required, giving you the freedom to store information without doing up-front schema design.

Nonrelational: Relations in a database establish connections between data tables. For example, a list of transaction details can be connected to a separate list of delivery details. With a NoSQL database, this information is stored as an aggregate — a single record with everything about the transaction, including the delivery address.

Commodity hardware: Some databases are designed to operate best (or only) with specialized storage and processing hardware. With a NoSQL database, cheap off-the-shelf servers can be used. Adding more of these cheap servers allows NoSQL databases to scale to handle more data.

Highly distributable: Distributed databases can store and process a set of information on more than one device. With a NoSQL database, a cluster of servers can be used to hold a single large database.

NoSQL databases are schema agnostic. You aren't required to do a lot of up-front design work before you can store data in NoSQL databases. You can start coding and store and retrieve data without knowing how the database stores and works internally. (If and when you need advanced functionality, then you can manually add further indexes or tweak data storage structures.) Schema agnosticism may be the most significant difference between NoSQL and relational databases.

An alternative interpretation of schema agnostic is *schema on read.* You need to know how the data is stored only when constructing a *query* (a coded question that retrieves information from the database), so for practical purposes, this feature is exactly what it says: You need to know the schema on read.

The great benefit to a schema agnostic database is that development time is shortened. This benefit increases as you go through multiple development releases and need to alter the internal data structures in the database. In RDBMS if you were to make a change, you'd have to spend a lot of time deciding how to re-architect the existing schema. In NoSQL databases, you simply store a different data structure. There's no need to tell the database beforehand.

Note that not all NoSQL databases are fully schema agnostic. Some, such as HBase, require you to stop the database to alter column definitions. They're still considered NoSQL databases because not all defined fields (columns in this case) are required to be known in advance for each record — just the column families.

NoSQL databases don't have this concept of relationships between their records. They instead denormalize data. This means that in a NoSQL database would have an Order structure with the Delivery Address embedded. This means the delivery address is duplicated in every Order row that uses it. This approach has the advantage of not requiring complex query time joins across multiple data structures (tables) though.

NoSQL databases don't store information about how individual records relate to other records in the database, which may sound like a limitation. However, NoSQL databases are more flexible in terms of the data structures you can store.

In relational database theory, the goal is to *normalize* your data (that is, to organize the fields and tables to remove duplicate data). In NoSQL Databases — especially Document or Aggregate databases — you often deliberately denormalize data, storing some data multiple times.

Relational views and NoSQL denormalizations are different approaches to the problem of data spread across records. In NoSQL, you may have to maintain multiple denormalizations representing different views of the same data. This approach increases the cost of storage but gives you much better query time.

The main advantage of distributed approach is in the case of very large datasets, because for some storage requirements, even the largest available single server couldn't store or process all the data you need. Consider all the messages on Twitter and Facebook. You need a distributed mechanism to effectively manage all that data, even if it's mostly about what people had for breakfast and cute cat videos.

An advantage of distributing your database is that you can use cheaper servers, called *commodity servers*, which are cheaper than single very powerful servers. (However, a decent one will still cost you $10,000!) Even for smaller datasets, it may be cheaper to buy three commodity servers instead of a single, higher-powered server.

Another key advantage is that adding high availability is easier; you're already halfway there by distributing your data. If you replicate your data once or twice across other servers in the cluster, your data will still be accessible, even if one of the servers crashes, burns, and dies.

Module-141

Although some features are fairly common to NoSQL databases (for example, schema agnosticism and non-relational structure), it's not uncommon for a database to lack one or more of the following features and still qualify as a modern NoSQL database.

NoSQL software is unique because the open-source movement has driven development rather than follow a set of commercial companies. You therefore can find a host of open-source NoSQL products to suit every need. When developers couldn't find a NoSQL database for their needs, they created one, and published it initially as open-source.

Earlier modules did not covered "Common features" because the majority of popular NoSQL solutions are driven by commercial companies, with the open source variant lacking the key features required for mission critical use in large enterprises.

Buyer beware! When it comes to selecting a NoSQL database, remember "total cost of ownership." Many organizations acquired open-source products only to find that they need a high-priced subscription in order to get the features they want.

ACID-compliant transaction means the database is designed so it absolutely will not lose data:

- Each operation moves the database from one valid state to another (***A*tomic**).
- Everyone has the same view of the data at any point in time (***C*onsistent**).
- Operations on the database don't interfere with each other (***I*solation**).
- When a database says it has saved data, you know the data is safe (***D*urable**).

Prior to 2014, the majority of NoSQL definitions didn't include ACID transaction support as a defining feature of NoSQL databases. This is no longer true.

Not many NoSQL databases have ACID transactions. Exceptions to that norm are FoundationDB, Neo4j, and MarkLogic Server, which do provide fully serializable ACID transactions.

The following list identifies the requisite features that large enterprises look for (or should look for) when investing in software products that run the core of their system.

High availability: Fault tolerance when a single server goes down

Disaster recovery: For when a datacenter goes down, or more likely someone digs up a network cable just outside the datacenter

Support: Someone to stand behind a product when it goes wrong (or it's used incorrectly!)

Services: Product experts who can advise on best practices and help determine how to use a product to address new or unusual business needs

Ecosystem: Availability of partners, experienced developers, and product information — to avoid being locked into a single vendor's expensive support and services contract

Many NoSQL databases are used by enterprises. Just visit the website of any of the NoSQL companies, and you'll see a list of them. But there is a difference between being used by an enterprise, and being a piece of mission critical *enterprise software*.

NoSQL databases are often used as high-speed caches for web-accessible data on mission-critical systems. If one of these NoSQL systems goes down, though, you lose only a copy of the data — the mission-critical store is often an RDBMS! Seriously question enterprise case studies and references to be sure the features mentioned in the preceding list of enterprise features exist in a particular NoSQL product.

Module-142

Not everything fits well into rows and columns — for example, a book with a tree structure of cover, parts, chapters, main headings, and subheadings. Likewise, what if a

particular record has a field that could contain two or more values? Breaking this out into another sheet or table is a bit of overkill, and makes it harder to work with the data as a single unit.

There are also scenarios in which the relationships themselves can hold their own metadata. An RDBMS doesn't handle those situations at all; an RDBMS just relates records in tables using structures about the relationships known at design time.

If you're wondering whether NoSQL is just a niche solution or an increasingly mainstream one, the answer lies in this module. So, it's time to talk about recent trends and how you can use NoSQL databases over and above the traditional RDBMS approach.

Since the advent of the World Wide Web and the explosion of Internet connected devices, information sharing has dramatically increased. Details of our everyday lives are shared with friends and family, whether they're close or continents away. Much of this data is unstructured text; moreover the structures of data are constantly evolving, making it hard to quantify. There are simply no end of things to keep track of (for example, you can't predict when a website or newsfeed will be updated, or in what format).

It's true that search engines help you find potentially useful information; however, search engines are limited because they can't distinguish the distinctions of how you search or what you're aiming for.

Furthermore, simply storing, managing, and making use of this information is a massive task. What's needed is a set of database solutions that can handle current and emerging data problems, which leads us back to NoSQL, the problems, and the possibilities.

Although there's been an outpouring of enthusiasm by the development community about NoSQL databases, not many killer applications have been created and put on the market. These applications will take time to emerge — right now, NoSQL databases are being used to solve problems that emerge in conventional approaches.

Consider a retail website. The original design has a single order with a single set of delivery information. What if the retailer now needs to package the products into potentially multiple deliveries?

With a relational system, you now have to spend a lot of time deciding how best to handle this redesign. Do you create an Order Group concept, with each group related to a different delivery

schedule? Do you instead create a Delivery Schedule containing delivery information and relate that to Order Items?

Managing feeds of external datasets you cannot control is a similar issue. Consider the many and varied ways Twitter applications create tweets. Believe it or not, a simple tweet involves a lot of data, some of it application specific and some of it general across all tweets.

Or perhaps you must store and manage XML documents across different versions of the same XML schema. It's still a variety problem. You may have to support both structures at the same time. This is a common situation in financial services, insurance and public sectors (including federal government metadata catalogues for libraries), and information-sharing repositories.

A decade ago 80% of organizations' data was typically unstructured, and this percentage has increased. That statistic is still used today, nine years later, though the proportion is bound to be more now.

Increasingly the focus of organizations has been to use publicly available data alongside their own to gain greater business insight — for example, using government-published open data to discover patterns of disease, research disease outbreak, or to mine Twitter to find how well a particular product is received. Whatever the motivation, there is a need to bring together a variety of data, much of which is unstructured, and use it to answer business questions.

For storing this data and discovering relevant information presents issues, too. Databases evaluate queries over indexes. Search engines do the same thing. In NoSQL, there is an ever-increasingly blurred line between where the database ends and the search engine begins. This enables unstructured information to be managed in the same way as more regular (albeit rapidly changing) information.

Consider a person tweeting about a product. You may have a list of products, list of medical issues, and list of positive and negative phrases. Being able to write "If a new tweet arrives that mentions Ibuprofen, flag it as a medication" enables you to see how frequently particular medications are used or to specify that you only want to see records mentioning the medication Ibuprofen. This process is called *entity extraction.*

Relational databases can suffer from a sparse data problem — this is where it's possible for columns to have particular values, but often the columns are blank. Consider a contact management system, which may have a field for home phone, cell phone, twitter ID, email, and other contact fields. Usually you have only one or two of these fields present.

Using an RDBMS requires a null value be placed into unused columns. Potentially, there could be 200 different fields, 99 percent with blank null values.

An RDBMS will still allocate disk space for these columns, though, because they potentially could have a value after future edits of the contact data. This is a great waste of resources. It's also inefficient to retrieve 198 null values over SQL in a result set.

NoSQL databases are designed to bypass this problem. They store and index only what is provided by the client application. No nulls stored, and no storage space previously allocated, but unused. You just store what you need to use.

You may discover facts and relationships over time. Consider LinkedIn where someone may be a second-level connection (a friend of a friend). You realize you know the person, so you add her as a first level relationship by inserting a single fact or relationship in the application.

Using an RDBMS for this would require an ever-increasing storm of many-to-many relationships and linking tables, one table schema for each relationship class. This approach would be hard to keep up with and maintain.

Another aspect of complex relationships is on the query side. What if you want to know all people within three degrees of separation of a person? This is a common statistic on LinkedIn. Just writing the SQL gives you a headache. "Return all people who are related to Person1, or have a relationship with Person2 who is related to Person1, or is related to Person3, who is related to Person4, who is related to Person1. Oh, and make sure there are no duplicates.

Triple and graph store NoSQL databases are designed with dynamically changing relationships in mind. They specifically use a simpler data model but at terrific scale to ensure these questions can be answered quickly.

NoSQL vendors have focused strongly on ease of development. A technology can be adopted rapidly only if the development team views it as a lower-cost alternative. This perspective results in streamlined development processes or quicker ways to beat traditionally knotty problems, like those in traditional approaches mentioned in this module.

Lower total cost of ownership (TCO) is always a favorite with chief information officers. Being able to use commodity hardware and rapidly churn out new services and features are core features of a NoSQL implementation.

It's not all about lower cost or making developers' lives easier though. A whole new set of data types and information management problems can be solved by applying NoSQL approaches.

Module-143

NoSQL databases aren't restricted to a rows-and-columns approach. They are designed to handle a great variety of data, including data whose structure changes over time and whose interrelationships aren't known yet.

If you're wondering whether NoSQL is just a niche solution or an increasingly mainstream one, the answer lies in this and subsequent modules. So, it's time to talk about recent trends and how you can use NoSQL databases over and above the traditional RDBMS approach.

Column stores are similar at first appearance to traditional relational DBMS. However, column stores organize data differently than relational databases do. Instead of storing data in a row for fast access, data is organized for fast column operations. This column-centric view makes column stores ideal for running aggregate functions or for looking up records that match multiple columns.

Perhaps the key difference between column stores and a traditional RDBMS is that, in a column store, each record (think *row* in an RDBMS) doesn't require a single value per column. Instead, it's possible to model column families. A single record may consist of an ID field, a column family for "customer" information, and another column family for "order item" information.

Each one of these column families consists of several fields. One of these column families may have multiple "rows" in its own right. Order item information, for example, has multiple rows — one for each line item. These rows will contain data such as item ID, quantity, and unit price.

A key benefit of a column store over an RDBMS is that column stores don't require fields to always be present and don't require a blank padding null value like an RDBMS does. This feature prevents the sparse data problem, thus preserving disk space. An example of a variable and sparse data set is shown in Figure.

The great thing about column stores is that you can retrieve all related Information using a single record ID, rather than using the complex SQL join as in an RDBMS. Doing so does require a little upfront modeling and data analysis, though.

Key-value stores also have a record with an ID field — the key in key-value stores — and a set of data. This data can be one of the following:

- An arbitrary piece of data that the application developer interprets (as opposed to the database)
- Any set of name-value pairs (called *bins*)

Some key-value stores support typing (such as integers, strings, and Booleans) and more complex structures for values (such as maps and lists). This setup aids developers because they don't have to hand-code or decode string data held in a key-value store.

Maps are a simple type of key-value storage. A unique key in a map has a single arbitrary value associated with it. The value could be a list of another map. So, it's possible to store tree structures within key-value stores, if you're willing to do the data processing yourself.

Key-value stores are optimized for speed of ingestion and retrieval. If you need very high ingest speed on a limited numbers of nodes and can afford to sacrifice complex ad hoc query support, then a key-value store may be for you.

Under the hood of these approaches is a simple concept: every *fact* (or more correctly, *assertion*) is described as a triple of subject, predicate, and object:

- A *subject* is the thing you're describing. It has a unique ID called an IRI. It may also have a type, which could be a physical object (like a person) or a concept (like a meeting).
- A *predicate* is the property or relationship belonging to the subject. This again is a unique IRI that is used for all subjects with this property.
- An *object* is the intrinsic value of a property (such as integer or Boolean, text) or another subject IRI for the target of a relationship.

More accurately, though, such triple information is conveyed with full IRI information in a format such as Turtle, like this:

You can quickly build this simple data structure into a web of facts, which is called a *directed graph* in computer science. Ismail could be a *friend_of* Altaf or *married_to* Noshaba. Noshaba may or may not have a *knows* relationship with Altaf.

These directed graphs can contain complex and changing webs of relationships, or triples. Being able to store and query them efficiently, either on their own or as part of a larger multi-data structure application, is very useful for solving particular data storage and analytics problems.

Module-144

Document databases are sometimes called aggregate databases because they tend to hold documents that combine information in a single logical unit — an aggregate. You might have a document that includes a TV episode, series, channel, brand, and scheduling and availability information, which is the total set of result data you expect to see when you search an online TV catch-up service.

Although an online store's orders and the related delivery and payment addresses and order items can be thought of as a tree structure, you may instead want to use a column store for these. This is because the data structures are known up front, and it's likely they won't vary and that you'll want to do column operations over them. Most of the time, a column store is a better fit for this data.

A table, for example, can be modeled as a very flat XML document — that is, one with only a single set of elements, and no sub-element hierarchies. A set of triples (aka subgraph) can be stored within a single document, or across documents, too. The utility of doing so depends, of course, on the indexing and query mechanisms supported. There's no point storing triples in documents if you can't query them.

It may seem strange to include search engines in this module, but many of today's search engines use an architecture very similar to NoSQL databases. Their indexes and query processing are highly distributed. Many search engines are even capable of acting as a key-value or document store in their own right.

Storing many structures in a single database necessitates a way to provide a standard query mechanism over all content. Search engines are great for that purpose. Consider search as a key requirement to unstructured data management with NoSQL Document databases.

Search technology is different from traditional query database interface technology. SQL is not a search technology; it's a query language. Search deals with imperfect matches and relevancy scoring, whereas query deals with Boolean exact matching logic (that is, all results of a query are equally relevant).

Given the range of data types being managed by NoSQL databases, you're forgiven if you think you need three different databases to manage all your data. However, although each NoSQL database has its core audience, several can be used to manage two or more of the previously mentioned data structures. Some even provide search on top of this core all-data platform.

Hybrid databases can easily handle document and key-value storage needs, while also allowing fast aggregate operations similar to how column stores work. Typically, this goal is achieved by

using search engine term indexes, rather than tabular field indexes within a table column in the database schema design itself.

There are more than 250 databases described by analysts in the NoSQL field. With so many there, only few can be selected. Here is a condensed list of the leaders in providing NoSQL databases:

- **Columnar:** DataStax, Apache Cassandra, HBase, Apache Accumulo, Hypertable
- **Key-value:** Basho Riak, Redis, Voldemort, Aerospike, Oracle NoSQL
- **Triple/graph:** Neo4j, Ontotext's GraphDB (formerly OWLIM), MarkLogic, OrientDB, AllegroGraph, YarcData
- **Document:** MongoDB, MarkLogic, CouchDB, FoundationDB, IBM Cloudant, Couchbase
- **Search engine:** Apache Solr, Elasticsearch, MarkLogic
- **Hybrid:** OrientDB, MarkLogic, ArangoDB

Module-145

If you studied databases, you may have been trained 2in a relational way of thinking. Mention database to most people, and they think relational database management system. This is natural because during the past 30 years, the RDBMS has been so dominant.
To aid you on this journey, will introduce some key terms that are prevalent, as well as what they mean when applied to NoSQL databases.

Here is a rather **ambitious** list of topics to be covered in this module.

Database Construction

• Database: A single logical unit, potentially spread over multiple machines, into which data can be added and that can be queried. The relational term tablespace could also be applied to a NoSQL database or collection.
• Data farm: A term from RDBMS referring to a set of read-only replica sets stored across a managed machines cluster. In an RDBMS, these typically can't have machines added without down time. In NoSQL clusters, it's desirable to quickly scale up.
• Partition: A set of data to be stored together on a single node for processing efficiency, or to be replicated. Could also be used for querying. In this case, it can be thought of as a collection.

Database Structure

• Collection: A set of records, typically documents, that are grouped together. This is based not on a property within the record set, but within its metadata. Assigning a record to a collection is usually done at creation or update time.
• Schema: In RDBMS and to a certain extent column stores. The structure of the data must be configured in the database before any data is loaded.

Record

• Record: A single atomic unit of data representation in particular database. In RDBMS, this would be a row, as it is in column stores. This could also be a value in a key-value store, a document in a document store, or a subject (not triple) in a triple store.
• Row: Atomic unit of record in an RDBMS or column store. Could be modeled as an element within a document store or as a map in a key-value store.
• Field: A single field within a record. A column in an RDBMS. May not be present in all records, but when present should be of the same type or structure.
• Table: A single class of record. In Bigtable, they are also called tables. In a triple store, they may be called subject RDF types or named be graphs, depending on the context. In a document store, they may be collections.

Associations

• Primary key: A guaranteed unique value in a particular table that can be used to always reference a record. A key in a key-value store, URI in a document store, or IRI in a triple or graph store.
• Foreign key: A data value that indicates a record is related to a record in a different table or record set. Has the same value as the primary key in the related table.
• Relationship: A link, or edge in graph theory, that indicates two records have a semantic link. The relationship can be between 2 records in same or different tables.

Storage Organization

• *Server:* A single computer node within a cluster. Typically runs a single instance of a database server's code.
• *Cluster:* A physical grouping or servers that are managed together in the same data center to provide a single service. May replicate its databases to clusters in other data centers.
• *Normal form:* A method of normalizing, or minimizing duplication, in data in an RDBMS. NoSQL databases typically lead to a denormalized data structure in order to provide faster querying or data access.

Replication Technology

• *Disk replication:* Transparent replication of data between nodes in a single cluster to provide high-availability resilience in the case of a failure of a single node.
• *Database replication:* Replication between databases in different clusters. Replicates all data in update order from one cluster to another. Always unidirectional.
• *Flexible replication:* Provides application controlled replication of data between databases in different clusters. Updates may not arrive in the same order they were applied to the first database. Typically involves some custom processing, such as prioritization of data updates to be sent next. Can be bi-directional with appropriate
update conflict resolution code.

Search tools

• *Index:* An ordered list of values present in a particular record.
• *Reverse index:* An ordered list of values (terms), and a list of primary keys of records that use these terms. Provides for efficient unstructured text search and rapid aggregation functions and sorting when cached in memory.
• *Query:* A set of criteria that results in a list of records that match the query exactly, returned in order of particular field value(s).
• *Search:* A set of criteria that results in a relevancy-ordered list that match the query. The search criteria may not require an exact match, instead returning a relevancy calculation weighted by closeness of the match to the criteria. This is what Google does when you perform a search.

Module-146

The consistency property of a database means that once data is written to a database successfully, queries that follow are able to access the data and get a consistent view of the data. In practice, this means that if you write a record to a database and then immediately request that record, you're guaranteed to see it. It's particularly useful for things like Amazon orders and bank transfers.

Consistency is a sliding scale, though, and a subject too deep to cover here. However, in the NoSQL world, consistency generally falls into one of two camps:

- ACID Consistency (ACID stands for Atomicity, Consistency, Isolation, Durability): ACID means that once data is written, you have full consistency in reads.
- Eventual Consistency (BASE): BASE means that once data is written, it will eventually appear for reading..

ACID is a general set of principles for transactional systems, not something linked purely to relational systems, or even just databases, so it's well worth knowing about. ACID basically means, "This database has facilities to stop you from corrupting or losing data," which isn't a given for all databases. In fact, the vast majority of NoSQL databases don't provide ACID guarantees.

Atomicity: Each operation affects the specified data, and no other data, in the

database.

Consistency: Each operation moves the database from one consistent state to

another.

Isolation: One operation in-flight does not affect the others.

Durability: The database will not lose your data once the transaction reports success.

ACID transactional consistency can be provided in various ways:

- In the locking model, you stop data from being read or written on the subset of information being accessed until the transaction is complete, which means that during longer-running transactions, the data won't be available until all of the update is committed.

- An alternative mechanism is multiversion concurrency control (MVCC), which bears no resemblance to document versioning; instead, it's a way of adding new data without read locking.

- BASE means that rather than make ACID guarantees, the database has a tunable balance of consistency and data availability. To ensure that every client sees all updates (that is, they have a consistent view of the data), a write to the primary node holding the data needs to lock until all read replicas are up to date. This is called a *two-phase commit* — the change is made locally but applied and confirmed to the client only when all other nodes are updated.

- BASE relaxes this requirement, requiring only a subset of the nodes holding the same data to be updated in order for the transaction to succeed. Sometime after the transaction is committed, the read-only replica is updated. The advantage of this approach is that transactions are committed faster. Having readable live replicas also means you can spread your data read load, making reading quicker.

- The downside is that clients connecting to some of the read replicas may see out-of-date information for an unspecified period of time. In some scenarios, this state is fine. If you post a new message on Facebook and some of your friends don't see it for a couple of minutes, it's not a huge loss. If you send a payment order to your bank, though, you may want an immediate transaction.

- Some NoSQL databases have ACID-compliance on their roadmap, even though they are proponents of BASE, which shows how relevant ACID guarantees are to *enterprise*, *mission-critical* systems. Many companies use BASE-consistency products when testing ideas because they are free but then migrate to an ACID-compliant paid-for database when they want to go live on a mission-critical system.

- The easiest way to decide whether you need ACID is to consider the interactions people and other systems have with your data. For example, if you add or update data, is it important that the very next query is able to see the change?

- In financial services, the need for consistency is obvious. Think of traders purchasing stock. They need to check the cash balance before trading to ensure that they have the money to cover the trade. If they don't see the correct balance, they will decide to spend money on another transaction. If the database they're querying is only eventually consistent, they may not see a lack of sufficient funds, thus exposing their organization to financial risk.

- Similar cases can be built for ACID over BASE in health care, defense, intelligence, and other sectors. It all boils down to the data, though, and the importance of both timeliness and data security.

Consistency is a sliding scale, not an absolute. Many NoSQL databases allow tuning between levels of consistency and availability, which relates to the *CAP theorem.* The CAP theorem shows the list of options as to how the balance between consistency, availability, and partitioning can be maintained in a BASE database system.

CAP stands for *C*onsistency, *A*vailability, and *P*artitioning, which are aspects of data management in databases. Here are some questions to consider when considering a BASE and thus CAP approach:

- **Consistency:** Is the database fully (ACID) consistent, or eventually consistent, or without any consistency guarantees?
- **Availability:** During a partition, is all data still available (that is, can a partitioned node still successfully respond to requests)?
- **Partitioning:** If some parts of the same database cluster aren't communicating with each other, can the database still function separately and correct itself when communication is restored?

The CAP theorem states you cannot have all three features at the same time. Most of the time, this means you can have only two of the three. The reality is that each is a sliding scale. You may be able to trade off a little of one for more of another.

Typically, the tradeoff is between consistency and partitioning. A particular NoSQL database generally provides either

- CA (consistency and availability)
- AP (availability and partition tolerance)

Module-147

MongoDB is the poster child for the NoSQL database movement. If asked to name a NoSQL database, most people will say MongoDB, and many people start with MongoDB when looking at NoSQL technology. This popularity is both a blessing and a curse. It's obviously good for MongoDB, Inc. (formerly 10gen). On the flip side, though, people try to use MongoDB for purposes it was not designed for, or try to apply relational database approaches to this fundamentally different database model.

JavaScript Object Notation is an open standard format and data interchange format, that uses readable text to store and transmit data objects consisting of attribute-value pairs and array types.

MongoDB natively handles JSON documents. Like XML, JSON documents' property names can be quite verbose text. MongoDB uses its own BSON (short for Binary JSON) storage format to reduce the amount of space and processing required to store JSON documents. This binary representation provides efficient serialization that is useful for storage and network transmission.

One of the main strengths of MongoDB is the range of official programming language drivers it supports. In fact, it officially supports ten drivers. These drivers are released under Apache License v2.0, allowing you to extend the drivers, or fix them as needed, and to redistribute the code.

Also, more than 32 unofficial drivers (the code is not reviewed by MongoDB) under a variety of licenses are available, which is by far the most language drivers I've come across for any NoSQL database.

Official: C, C++, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala

Unofficial: ActionScript 3, Clojure, ColdFusion, D, Dart, Delphi, Entity, Erlang, Factor, Fantom, F#, Go, Groovy, JavaScript, Lisp, Lua, MATLAB, Node.js, Objective C, OCaml, Opa, Perl, PHP, PowerShell, Prolog, Python, R, REST, Ruby, Scala, Racket, Smalltalk

Storing data is one thing, finding it again is quite another! Retrieving a document using a document ID, or (primary) key, is supported by every NoSQL document database.

In many situations, you may want a list of all comments on a web page or all recipes for *Haleem*. This requires retrieving a list of documents based not on their key but on other information within the document — for example, a page_id JSON property. These indexed

fields are commonly referred to as *secondary indexes.* Adding an index to these fields allows you to use them in queries against the MongoDB database.

Sometimes you may want to search by several of fields at a time, e.g. all *Haleem* recipes containing meat but are cholesterol free. MongoDB solves this issue by allowing to create a *compound index,* which is basically an index for all three fields (recipe type, ingredients, is cholesterol free), maybe ordered according to name of recipe.

You can create a compound index for each combination of query fields and sort orders you need. The flip side is that you need an index for every single combination. If you want to add a query term for only five-star recipes, then you need yet another compound index, maybe several for different sorting orders, too.

Other document NoSQL databases and search engines solve this matter by allowing an intersection of the results of each individual index. In this way, there's no need for compound indexes, just a single index per field and an intersection on each index lookup's document id list. This approach reduces the amount of administration required for the database and the space needed for the index on disk and in memory.

MongoDB, Inc., provides advice on running MongoDB on a wide variety of cloud platforms, which isn't surprising because MongoDB emerged from the 10gen company's cloud application requirement.

- MongoDB supports the following public cloud platforms:
- Amazon EC2
- dotCloud
- Google Compute Engine
- Joyent Cloud
- Rackspace Cloud
- Red Hat OpenShift
- VMWare Cloud Foundry
- Windows Azure

There is, of course, nothing stopping you from downloading MongoDB and installing it on your private cloud. This, too, is supported.

Not all functionality is available on the free download version of MongoDB. If you want any of the following functionality, you must buy MongoDB Enterprise from MongoDB, Inc.

- **MongoDB Management Service (MMS):** Enables disaster recovery replication to a second cluster and is a systems monitoring tool.
- **Security integrations:** Includes Kerberos, LDAP authentication, and auditing.
- **Enterprise software integration:** Integrates MongoDB with your organization's monitoring tools through SNMP (Simple Network Management Protocol).
- **Certified operating system support:** Includes full testing and bug fixes for operating systems.
- **On-demand training:** Provides access to online training portal.
- **24/7 support:** Includes software support and bug fixes.
- **Commercial license:** Enables you to use MongoDB as an embedded database in a commercial product or service you sell.

In practice, disaster recovery replication to a second or subsequent site is required for enterprise application software. If you're a large enterprise thinking about betting part of your business on a NoSQL document database, you need disaster recovery replication, which means spending money for an Enterprise License Agreement.

Module-148

A given business problem can be solved different ways in each of the databases (key-value, Bigtable, graph / triple stores, and document databases) covered in this book. storing a document for a unique ID, for example, is a feature of both key-value stores and document databases. Various databases can, therefore, technically be called hybrid in that they support multiple paradigms of data management.

Document databases aren't automatically classified as key-value stores even though they can technically store values against a unique key. Likewise, not all databases that provide in-memory caching of property values are classified as column stores.

Polyglot persistence is basically a data integration technique for an application from various data sources. When relational database management systems first became mainstream, they tended to offer different advantages. Over time, such features became standard in all relational database management systems.

One product many features

Customers may find that they prefer a single-product approach because one rich product makes training developers and administering the IT landscape easier than using multiple databases would. A single product also means that you don't have to become a coding plumber. You don't have to figure out how to join two different systems together. With a single product, vendors generally do these things themselves.

Best of breed solution

The main risk with a single-product approach is that the product may provide weak functionality in every area rather than doing one thing well. Sometimes you do need advanced features, in which case, you want to use multiple products.

As an IT professional seen as your clients' "trusted adviser," it's your job to figure out where to draw the line between using multiple products and a single product.

Hybrid approaches provide important advantages, including the following:

✓ **Single strategic tech stack:** Implements a single data layer to power all your applications. As an IT professionals you've probably unknowingly been using relational database management systems to do this, but NoSQL means there's no up-front schema design, which gives you the flexibility to create an operational database and achieve fast application builds.

✓ **Common indexes / no duplication:** Storing a single index rather than having an index of the same data in multiple products is advantageous. Storing a document in an enterprise content

management (ECM) platform means indexes are held in an RDBMS. Separate indexes will also exist in a search engine that indexes content held in that repository. A hybrid NoSQL system that supports search means a single set of indexes, which results in lower costs for storage and faster reindexing.

✓ **More real-time data through the stack (fewer moving parts):** Because indexes are updated as information is added to a hybrid NoSQL document database and search engine, fewer indexes as well as nearly real-time indexes are produced, or at least they're transactionally consistent. This real-time indexing powers alerting and messaging applications, such as the backbone of HealthCare.gov.

✓ **Easy administration (fewer moving parts):** Database admins need to be absolute experts on the systems they manage. The level of complexity in all products is great, and it increases over time. Therefore, having multiple products typically means the need for multiple administrators, each with different skillsets.

A single product offers a number of advantages. If you add them up, the following cost-saving measures are huge. They can easily mean half the cost of implementing a new database layer:

✓ Less integration code between your application and its persistence layer

✓ Less ETL code to convert data formats between products

✓ Lower software license, maintenance, and consulting costs

✓ Lower training costs for developers and administrators (and a single API to access all your data)

✓ Lower salary costs because fewer experts for each system are needed

✓ Fewer moving parts with backups and maintenance, such as patches and security updates

You gain some of these benefits by adopting any kind of NoSQL technology. The ability to load data "as is" into a schema-less document store, for example, means lower ETL (Extract, Transform, and Load) costs. As soon as you introduce two NoSQL stores, though, moving data between them or merging data from each of them still entails more ETL costs than adopting a single NoSQL database.

A column store database performs rapid aggregate calculations, and it returns sets of atomic data (column families) of a whole row (record). Using column stores requires transforming data into a row and column structure, and supporting multiple instances of data within some column families (avoiding cross table joins like in an RDBMS).

Because document NoSQL databases with search also update their indexes during the transaction that updates a document, these indexes are also updated in real-time, which is great for an analytics platform.

All common aggregation algorithms are present: mean, mode, median, standard deviation, and more — plus support for user-defined aggregate functions written in fast C++ that work next to the data, processing the data throughout the cluster.

Understanding context, is important in navigating directly to the most appropriate data. The way to describe these contexts is to use an ontology, which is a set of terms and definitions that applications use to describe a unique information domain.

This technology is associated with the semantic web and triple stores. It's not a graph store problem because you're not interested in analyzing the links or the minimum distance between subjects; you're just using the links.

Often, when publishing data, you know a lot about its context. Adding this information into a database helps later on when the data is queried. Understanding what people searched for previously and linking those queries to subjects as triples may also be advantageous.

If you have a similar requirement for rapid discovery of content or for context-aware search, then investigate a hybrid document or triple store NoSQL database with search capabilities.

Module-149

NoSQL databases support storing data “as is.” Key-value stores give you the ability to store simple data structures, whereas document NoSQL databases provide you with the ability to handle a range of flat or nested structures. Typically, the data takes one of these formats:

✓ A binary object to be passed through a set of layers

✓ An XML document

✓ A JSON document

Being able to handle these formats natively in a range of NoSQL databases lessens the amount of code you have to convert from the source data format to the format that needs storing. Using this approach, you greatly reduce the amount of code required to start using a NoSQL database. Moreover, because you don’t have to pay for updates to this “plumbing” code, ongoing maintenance costs are significantly decreased.

Being able to manage unstructured text greatly increases information and can help organizations make better decisions. For example, advanced uses include support for multiple languages with facetted search, snippet functionality, and word-stemming support. Advanced features also include support for dictionaries and thesauri.

Furthermore, using search alert actions on data ingest, you can extract named entities from directories such as those listing people, places, and organizations, which allows text data to be better categorized, tagged, and searched.

Because of the schema agnostic nature of NoSQL databases, they’re very capable of managing change — you don’t have to rewrite ETL routines if the XML message structure between systems changes.

Some NoSQL databases take this a step further and provide a universal index for the structure, values, and text found in information. Microsoft DocumentDB and MarkLogic Server both provide this capability.

Structured Query Language (SQL) is the predominant language used to query relational database management systems. Being able to structure queries so that they perform well has over the years become a thorny art. Complex multi-table joins are not easy to write from memory.

Although several NoSQL databases support SQL access, they do so for compatibility with existing applications such as business intelligence (BI) tools. NoSQL databases support their own

access languages that can interpret the data being stored, rather than require a relational model within the underlying database.

Application developers don't need to know the inner workings and vagaries of databases before using them. NoSQL databases empower developers to work on what is required in the applications instead of trying to force relational databases to do what is required..

NoSQL databases handle partitioning (*sharding*) of a database across several servers. So, if your data storage requirements grow too much, you can continue to add inexpensive servers and connect them to your database cluster (*horizontal scaling*) making them work as a single data service.

Contrast this to the relational database world where you need to buy new, more powerful and thus more expensive hardware to scale up (*vertical scaling*). If you were to double the amount of data you store, you would easily quadruple the cost of the hardware you need.

Providing durability and high availability of a NoSQL database by using inexpensive hardware and storage is one of NoSQL's major assets. Being able to do so while providing generous scalability for many uses also doesn't hurt!

Most relational databases support the same features but in a slightly different way, so they are all similar. NoSQL databases, in contrast, come in four core types: key-value, columnar, document, and triple stores. Within these types, you can find a database to suit your particular (and peculiar!) needs. With so much choice, you're bound to find a NoSQL database that will solve your application woes.

Many applications need simple object storage, whereas others require highly complex and interrelated structure storage. NoSQL databases provide support for a range of data structures.

- Simple binary values, lists, maps, and strings can be handled at high speed in key-value stores.
- Related information values can be grouped in column families within Bigtable clones.
- Highly complex parent-child hierarchal structures can be managed within document databases.
- A web of interrelated information can be described flexibly and related in triple and graph stores.

The NoSQL industry is awash with databases, though many have been around for less than ten years. For example, IBM, Microsoft, and Oracle only recently dipped their toes into this market.

Consequently, many vendors are targeting particular audiences with their own brew of innovation.

Because they are so new, NoSQL databases don't have legacy code, which means they don't need to provide support for old hardware platforms or keep strange and infrequently used functionality updated.

Queries and processing work now pass to several servers, which provides high levels of parallelization for both ingest and query workloads. Being able to calculate aggregations next to the data has also become the norm.

You no longer need a separate data warehouse system that is updated overnight. With fast aggregations and query handling, analysis is passed to the database for execution next to the data, which means you don't have to ship a lot of data around a network to achieve locally combined analysis.

Module-150

NoSQL is a catch-all term for a variety of database types that exhibit common architectural approaches. These databases aren't intended for related table, rows, and columns data. They are highly distributed, which means data is spread across several servers, and they're tolerant of data structure changes (that is, they're schema agnostic).

You can find several types of databases under the NoSQL banner:

- Key-value stores provide easy and fast storage of simple data through use of a key.
- Column stores provide support for very wide tables but not for relationships between tables.
- Document stores support JSON and/or XML hierarchical structures.
- Triple (and graph) stores provide the same flexibility to relationships that document NoSQL databases provide to record structures.

Many NoSQL databases provide full ACID support across clusters. MarkLogic Server, OrientDB, Aerospike, and Hypertable are all fully ACID-compliant, providing either *fully serializable* or *read-commit* ACID compliance.

Many other NoSQL databases can provide ACID-like consistency by using sensible settings in client code. This typically involves a *Quorum* or *All* setting for both read and write operations. These databases include Riak, MongoDB, and Microsoft DocumentDB.

Data loss happens when NoSQL databases are used incorrectly or when less mature products are used. Some NoSQL products are less mature, having only been around for fewer than five years, so they haven't developed data loss prevention features yet.

The guarantee of durability in ACID compliance is vital for enterprise systems, and ACID-compliant NoSQL databases provide this guarantee. Therefore, you're assured that no data is lost once the database confirms the data is saved.

On the contrary, many organizations are using NoSQL databases for mission-critical workloads, including the following:

- Defense and intelligence agencies storing and sharing information
- Media companies storing all their digital assets for publication and purchasing in NoSQL databases

- Media companies providing searchable metadata catalogs for their video and audio media
- Banks using NoSQL databases as primary trade stores or back office anti-fraud and risk-assessment systems
- Government agencies using NoSQL databases as the primary back ends for their health care systems

These are not small systems or simple caches for relational systems. They are cases for which NoSQL is well suited. Of course, some NoSQL databases are more ready for enterprise systems than others.

Many NoSQL databases now provide record-level and even data-item level (cell) security. Microsoft DocumentDB, MarkLogic Server, OrientDB, AllegroGraph, and Accumulo all provide fine-grained role-based access Control (RBAC) to access records stored within these NoSQL databases.

Some NoSQL databases are even accredited and used by defense organizations. Accumulo came from a National Security Agency (NSA) project. MarkLogic Server is independently accredited under the U.S. Department of Defense's (DoD) Common Criteria certification.

There are numerous open-source databases in the NoSQL world. Many commercial companies have attempted to replicate Red Hat's success by offering a subset of their products' capabilities to be used for free under an open-source license.

There are many fully commercial companies in the NoSQL space. Microsoft, MarkLogic, Franz (Allegrograph), Hypertable, and Aerospike are all major commercial companies successfully offering NoSQL databases.

Their use in new web and mobile application stacks have made NoSQL databases popular. They're easy to use from the start, and many operate under a for-free license agreement, making them attractive to startups.

Social media applications commonly use NoSQL databases. Social media applications bring in web published data and aggregate it together in order to discover valuable information.

The vast majority of use cases, though, aren't Web 2.0-type applications. They're the same applications that have been around a long time, but where relational databases no longer provide an adequate solution. This includes scenarios where the data being stored is very sparse, with many blank (null) values, or where there is frequent change over time of the structure of the information being stored.

Microsoft, Oracle, and IBM each have their own NoSQL database on the market right now. Although susceptible to bluster, these companies invest in technology only when they see a profit.

Established players like MarkLogic with years on the market have also proved that NoSQL technology isn't just hype and is valuable to a range of real-world customers across industries in mission-critical systems.

There is a common misconception that NoSQL is used because developers don't have a grasp on the fundamentals needed to configure relational databases so that they perform well. This is completely incorrect. NoSQL comprises a range of approaches brought together to answer fundamentally different data problems than a relational database management system (RDBMS) solves.

If you're comparing an RDBMS to a NoSQL database, then you're comparing apples to motorbikes! NoSQL databases will not replace RDBMS. They are intended for data that's structured fundamentally different, as well as for different data problems.

Many of the highly distributed approaches of NoSQL are being blended with RDBMS technology, which has resulted in the emergence of many NewSQL databases. Although NewSQL is helping to deal with NoSQL developers' criticisms of RDBMS technology, NewSQL is organized around the same data structures as an RDBMS is.

NoSQL databases are for different data problems, with different data structures and use cases.

Module-151

The popularity of NoSQL databases arises from the sheer number of developers who are excited about using them. Developers see NoSQL as an enabling and liberating technology. Unlike the traditional relational approach, NoSQL gives you a way to work with data that is closer to the application than the relational data model.

Developers adopt NoSQL technologies for many reasons, some of which are highlighted in this chapter.

No Need to Write SQL

Writing Structured Query Language (SQL) — and doing it well — is the bane of many enterprise developers' existence. This pain is because writing very complex queries with multiple joins across related tables isn't easy to do. Moreover, in light of regular database changes over time, maintaining complex query code is a job in and of itself.

Enterprise developers have invented a number of ways to avoid writing SQL. One of the most popular ways is through the use of the Object-Relational Mapping (ORM) library, Hibernate. Hibernate takes a configuration file and one or more objects and abstracts away the nasty SQL so that developers don't have to use it. This comes at a cost in terms of performance, of course, and doesn't solve all query use cases. Sometimes you have to fall back to SQL.

NoSQL databases provide their own query languages, which are tuned to the way the data is managed by the database and to the operations that developers most often perform. This approach provides a simpler query mechanism than nested SQL statements do. Some NoSQL databases also provide an SQL interface to query NoSQL databases, in case developers can't break the SQL habit!

No long time designing schema

Schema agnosticism in NoSQL databases allows you to load data quickly without having to create a relational schema over a period of months. You don't have to analyze up front every single data item you need to store in NoSQL, as you do with an RDBMS.

Less Data Transform Code (ETL)

A common problem with relational databases due to upfront schema design is forcing nonrelational data into rows and columns. This shredding mechanism, along with other code methods that preprocess information for storage and post-process it for retrieval is referred to as *extract, transform, and load (ETL).*

This code forces developers to take their nice shiny object and document models and write code to store every last element. Doing so is nasty and also leads to highly skilled developers writing poor performing and uninteresting plumbing code.

NoSQL databases allow you to keep the stored data structures much closer to their original form. Data flowing in between systems is typically in an XML format, whereas when it comes to web applications, data is formatted in a JSON document. Being able to natively store, manage, and search JSON is a huge benefit to application developers.

Easier to Maintain Code

All the code that you write must be maintained. By keeping database structures close to the application code's data formats, you minimize the amount of code, which in turn minimizes code maintenance and regression testing that you need to do over time.

When data structures change on an RDBMS, you have to review all SQL code that may use the changed tables. In NoSQL, you simply add support for the new elements, or just ignore them! Much easier to maintain due to NoSQL databases being schema-agnostic.

Code Execution Close to Data

Unlike RDBMS, many NoSQL databases allow code to be distributed across all servers that store relevant data, which allows for greater parallelization of the workload. This approach is especially important for large ingestions of data that need processing and for complex aggregation analytics at query time.

User-defined functions (UDFs) and server-side scripting in a variety of NoSQL databases provide this distributed capability. UDFs are similar to Hadoop's MapReduce capability, except UDFs can happen in real time rather than in batch mode and doesn't require the same outlay in infrastructure that Hadoop plus a database would require

Lots of Open-Source Options

In many enterprise software areas, the choice of a solid open-source solution is lacking. Only one or two widespread options may exist. Availability of skills and local in-country support are even bigger problems.

However, there are a myriad of open-source NoSQL databases. Many of these have full-fledged commercial companies that offer support and have offices globally. So, if you do need support or more features, you can move to those versions eventually.

This reduces the cost of adopting NoSQL technology and allows you to "try before you buy." This availability of open-source alternatives has caused commercial companies in the NoSQL

space to offer free but well-featured versions of their software or to offer special startup licenses to small organizations.

Easy to Scale

You don't need to get a costly DBA to spend days refactoring SQL and creating materialized views in order to eek every inch of performance out of NoSQL systems.

Key-value stores can handle hundreds of thousands of operations per server. All types of NoSQL can scale horizontally across relatively cheap commodity servers. So, it's much easier to scale your database cluster with NoSQL than with traditional relational databases.

In addition, because of their ability to scale, NoSQL databases also fit well into public and private clouds. NoSQL databases are designed to be flexible and expand and contract as the uses for your application change. This capability is often called elasticity.

Eventual Consistency Data Model

Although I believe that mission-critical cases require ACID compliance, not every application needs to do so. Being able to relax consistency across very large clusters can be useful for some applications.

NoSQL databases allow you to relax these constraints or to mix and match strong consistency and weak consistency in the same database, for different record types.

Esoteric Language Support

Pretty much all databases support the main programming languages such as Java and C# .NET. Many databases support the likes of PHP, Python, and Ruby on Rails.

NoSQL has a flourishing set of language drivers for an even wider range of programming languages. Currently more than 34 different programming languages and platforms are supported by NoSQL databases. If your organization has a domain-specific language, you may well find support for it in a NoSQL database.

JavaScript End-to-End

JavaScript use has exploded in recent years. It's a convenient scripting language both on the web and, thanks to Node.js, on the server-side.

Many NoSQL databases now support full end-to-end JavaScript development. This means your organization can now use the same pool of programming language skills to craft web

applications and middle tier data APIs and business logic, as well as handle back-end database triggers and MapReduce-based analytical processing next to the data.

As a result, in comparison to other database technologies, the total cost of ownership (TCO) of NoSQL is lower.

Module-152

MySQL is used in the SQL database management system, a product from Microsoft corporation, where in NoSQL is a database type where SQL is necessary to access the document-based contents of the non-relational database management systems. Structuring and standardizing the database is essential for a relational database with MySQL. NoSQL, on the other hand, allows the unformatted and non-related data to be placed and operated as per the user's requirements.

MySQL

a. MySQL development project has made its source code available under the terms of GNU General Public License, also under a variety of proprietary agreements. Initially, MySQL was owned and sponsored by a Swedish company called MySQL AB, now owned by Oracle Corporation.

b. MySQL is relational in nature since all the data is stored in different tables and relations are established using primary keys or other keys known as foreign keys.

c. MySQL is fast, easy to use a relational database that is being utilized by big and small businesses equally well. There are a plethora of reasons behind the popularity of relational databases like MySQL. It is a very powerful program in its own right, by handling a large subset of the functionality of the most expensive and powerful database packages.

d. A standard form of language being used is a well-known data language called SQL for MySQL database. It can work on a multitude of operating systems and with many languages like C++, PHP, Java, C, etc. One of the key advantages of MySQL is that it is customizable since open source GPL license allows programmers to modify the MySQL software to fit their own specific environments

NoSQL

a. A data structure used by the NoSQL database is vastly different from those used in a relational database. Some operations are faster in NoSQL than relational databases like MySQL. Data structures used by NoSQL databases can also be viewed as more flexible and scalable than relational databases.

b. NoSQL databases are primarily used in big data and real-time web applications. These types of databases gain a surge in their popularity in the early twenty-first century which was primarily triggered by companies such as Facebook, Amazon, and Google.

c. Most of the NoSQL databases are driven by eventual consistency which means database changes are propagated to all nodes within milliseconds, so queries of data might not return updated data immediately, which is a problem called stale reads. A central concept of the NoSQL database revolves around “document”.

d. While every document-oriented database implementation differs on the details of the definition, but they all assume that documents encapsulate and encode data in some standard formats of encodings.

e. Documents are addressed in the database via a unique key that represents the document. In addition to the key lookup performed by a key-value store, the database also offered API or a query language that retrieves the documents based on their contents.

Differences

2 MySQL is a relational database that is based on tabular design whereas NoSQL is non-relational in nature with its document-based design.

3 MySQL is not so easily scalable with their rigid schema restrictions whereas NoSQL can be easily scaled with their dynamic schema nature.

4 The detailed database model is required before database creation in MySQL whereas no detailed modeling is required in the case of NoSQL database types.

5 MySQL database with its settled market encompasses a huge community whereas the NoSQL database with the short span arrival has a comparatively short community.

6 MySQL is being used with a standard query language called SQL whereas NoSQL like databases misses a standard query language.

7 MySQL is less flexible with its design constraint whereas NoSQL being non-relational in nature, provides a more flexible design as compared to MySQL.

8 MySQL is available with a wide array of reporting tools help application’s validity whereas NoSQL databases lack reporting tools for analysis and performance testing.

Conclusion

In MySQL vs NoSQL, we have seen that NoSQL databases are becoming a major part of the database landscape today. There are being shipped with multiple advantages, like performance at a big data level, scalability, and flexibility of design, etc. Hence, they can be a real game changer in the upcoming IT market. Other attributes like lower cost and open source features

make NoSQL an appealing option for many companies looking to integrate big data. However, NoSQL is still a young technology without the set of standards that SQL databases like MySQL offer.

As with any major business decision, IT leaders also need to weigh their options and thus conclude the difference between both, what are the features which are important to them in the database. Some people may argue that NoSQL is the way to the future, whereas other people are concerned with its lack of standardization. At the end of the day, the choice depends on the complex business needs of the organization and the volume of data it consumes.

TOPIC-15: Search Engine Optimization (SEO)

Module-153

Basic Definition & SEO now

But no single definition of SEO.

60+ SEO experts asked how they define SEO now

All answers and perspectives are unique - no SEO definition is wrong.

What SEO professionals say "what is SEO?".

The process of optimizing a website - as well as all the content on that website - so it will appear in prominent positions in the organic results of search engines. SEO requires an understanding of how search engines work, what **people** search for, and why and how **people** search. Successful SEO makes a site appealing to **users** and search engines. It is a combination of technical and marketing.

Module-154

Search engine optimization, like any specialized industry, has its own unique set of terminology, definitions, and abbreviations. Although in the text SEO glossary compiles more than 200 of the most common terms you are likely to hear and will definitely need to know during your SEO career; but in this module around 50 are considered.

Above the Fold: Content that appears on a website before the user scrolls. Google created the Page Layout Algorithm in 2012 to lower the rankings of websites featuring too many ads in this space.

Algorithm: Steps coded to be used by search engines to retrieve data and deliver results. Search engines use a combination of algorithms to deliver ranked webpages via a results page based on a number of ranking factors and signals.

Alt Attribute: HTML code that provides information used by search engines and screen readers (for blind and visually-impaired people) to understand the contents of an image.

B2B: Short for business-to-business. In B2B SEO, the buying cycle is longer, products and services are more expensive, and the audience is professional decision-makers.

Baidu: The most popular search engine in China, Baidu was founded in January 2000 by Robin Li and Eric Xu.

Bing: Microsoft's search engine launched in June 2009, replacing Microsoft Live Search (previously MSN and Windows Live Search). Since 2010, Bing has powered Yahoo's organic search results as part of a search deal Microsoft and Yahoo struck in July 2009.

Broken Link: A link that leads to a 404 not found. Typically, a link becomes broken when: A website goes offline or A webpage is removed without implementing a redirect or the destination URL is changed without implementing a redirect.

Bounce Rate: The percentage of website visitors who leave without visiting another page on that website. Bounce rates range widely depending on industry and niche. Although bounce rate can indicate potential content or website issues, it is not a direct ranking factor, according to Google.

Click Bait: Content that is designed to entice people to click, typically by overpromising or being intentionally misleading in headlines, so publishers can earn advertising revenue.

Click-Through Rate (CTR): The rate (expressed in a percentage) at which users click on an organic search result. This is calculated by dividing the total number of organic clicks by the total number of impressions then multiplying by 100.

Cloaking: Showing different content or URLs to people and search engines. A violation of Google's Webmaster Guidelines.

Conversion Rate: The rate (expressed in a percentage) at which website users complete

a desired action. This is calculated by dividing the total number of conversions by traffic, then multiplying by 100.

Crawl Budget: The total number of URLs search engines can and want to crawl on a website during a specific time period.

Crawler: A program search engines use to crawl the web. Bots visit webpages to collect information and add or update a search engine's index.

Dead-End Page: A webpage that links to no other webpages. So called because once a user or bot arrives on this page, there is no place to move forward.

Deep Link: A link pointing to any webpage other than the homepage.

DuckDuckGo: A search engine that was founded September 28, 2008. It is often praised for its heavy focus on user privacy and a lack of filter bubbles (search personalization).

Engagement Metrics: Methods to measure how users are interact with webpages and content.

Findability: How easily the content on a website can be discovered, both internally (by users) and externally (by search engines).

Google Analytics: A free web analytics program that can be used to track audience behavior, traffic acquisitiaon sources, content performance, and much more.

Googlebot: The web crawling system Google uses to find and add new websites and webpages to its index.

Google Trends: A website where you can explore data visualizations on the latest search trends, stories, and topics.

Guest Blogging: A popular link building tactic that involves developing content for other

websites in exchange for a backlink pointing at your own pages.

Hidden Text: Any text that can't be seen by a user that is intended to manipulate search rankings by loading webpages with content-rich keywords and copy.

Inbound Link: A link to a webpage that originates from an external website.

Indexability: How easily search engine bot can understand and index a webpage.

Keyword Stuffing: Adding irrelevant keywords, or repeating keywords beyond what is natural, to a webpage in the hopes of increasing search rankings.

Landing Page: Any webpage that a visitor can navigate to.

Link Bait: Intentionally provocative content that is meant to grab people's attention and attract links from other websites.

Link Farm: When a group of websites link to each other, usually using automated programs, in the hopes of artificially increasing search rankings. A spam tactic.

Link Velocity: How quickly (or slowly) a website accumulates links. A sudden increase in link velocity could potentially be a sign of spamming,

Log File: A file that records users' information, such as IP addresses, type of browser, Internet Service Provider (ISP), date/time stamp,

Meta Keywords: A tag that can be added to the "head" section of an HTML document.

Noindex Tag: A meta tag that tells search engines not to index a specific webpage in its index.

Organic Search: The natural, or unpaid, listings that appear on a SERP. Organic search results, which are analyzed and ranked by algorithms, are designed to give users the most relevant result based on their query.

Orphan Page: Any webpage that is not linked to by any other pages on that website.

PageRank: According to Google: "PageRank is the measure of the importance of a page based on the incoming links from other pages.

Paid Search: Pay-per-click advertisements that appear above (and often below) the organic results on search engines.

PPC (Pay Per Click): A type of advertising where advertisers are charged a certain amount every time a user clicks on the ad.

Redirect: A technique that sends a user (or search engine) who requested one webpage to a different (but equally relevant) webpage.

Rich Snippet: Structured data can be added to the HTML of a website to provide contextual information to the search engines during crawling.

robots.txt: A Standard is a text file, accessible at the root of a website, that tells search engine crawlers which areas of a website should be ignored.

Search Engine Results Page (SERP): The page search engines display to users after conducting a search.

Status Codes: The codes sent by a server whenever a link is clicked, a webpage or file is requested, or a form is submitted. Common HTTP status codes important to SEO: • 200 (OK) • 404 (Not Found) • 500 (Internal Service Error) etc.

User Experience (UX): The overall feeling users are left with after interacting with a brand, its online presence, and its product/services.

Visibility: The prominence and positions a website occupies within the organic search results.

Voice Search: A type of voice-activated technology that allows users to speak into a device (usually a smartphone) to ask questions or conduct an online search.

WordPress: A popular blogging and content management system.

Yandex: The most popular search engine in Russia, Yandex was founded Sep. 1997 by Arkady Volozh and Ilya Segalovich.

Module-155

Search engine optimization (SEO) is essential because organic search is arguably the most valuable marketing channel there is.

Every company that operates online needs SEO. Period.

Small. Medium. Large.

Startup. Local business. Global brand.

Whenever someone is using a search engine to find information that's relevant to your product, service, or website, you need to be there.

Yet, there remains a constant struggle, to justify the value of SEO.

In the early days, people searched to find a list of documents that contained the words they typed in. That's no longer the case.

Today's searchers search to solve problems, to accomplish tasks, and to "do" something. They might be searching to book a flight, buy something, learn the latest lyrics of their favorite artist, or browse cat photos – but these are all actions. Or, as Gates referred to them, verbs.

When a user starts a search, they're really starting a journey. Marketers love to talk about something called "the consumer journey." It's just a fancy way of referencing a user's path from the inception of their task to the completion – and most of these journeys start with a search.

The consumer journey has been gradually playing a larger role in search over the last decade. Originally depicted as a funnel wherein users move from awareness to consideration to purchase, this old consumer journey has become outdated (although we still use this model for illustrative purposes and to make persona research easier).

Search touches all three of these areas 1. Attract. 2. Engage.3. Convert.

But search concentrates heavily on the first phase: Attract.

"If you build it, they will come" may apply to a carnival, but it doesn't work with websites. It's no longer enough to have an awesome product. You must actively attract customers via multiple channels and outlets.

If a webpage is involved in attracting visitors, engaging visitors, or converting them, there should be an important SEO component to that page. This is why, despite some claims to the contrary from clients or design agencies, every webpage is, in fact, an SEO page.

Today's websites are more application than they are a website, and applications come with lots of fancy features that don't always play nicely with search engines

Sticking with the crazy straw model, today's consumer journey no longer happens on a single device. Users may start a search on their mobile device, continue researching on their tablet or work laptop, and ultimately purchase from their desktop at home.

As technology continues to evolve, SEOs will constantly deal with new ways of searching, new devices to search on, and new types of searches (like voice search, or searches done by my oven) but the one thing that will remain constant is why people search.

3. Orgnic Traffic Statistics

32.5 percent: Average traffic share first Google organic search result gets.

91.5 percent: The average traffic share generated by the sites listed on the first Google search results page.

51 percent of all website traffic comes from organic search, **10 percent** from paid search, **5 percent** for social, and **34 percent** from all other sources.

Over **40 percent** revenue captured by organic traffic.

73 billion: The estimated number of phone calls generated from mobile search alone by the end of 2018.

Module-156

Search intent is the reason behind a searcher's query on **search** engines. It represents the objective the searcher is trying to accomplish. For example, someone might want to learn about something, find something, or buy something.

When we choose target keywords, there is the tendency and appeal to go after those with the highest search volumes, but much more important than the keyword's search volume is the intent behind it. This is a key part of the equation that is often overlooked when content is produced, it's great that you want to rank for a specific term but the content has to not only be relevant but also satisfy the user intent.

Google has worked to improve its algorithm to be able to determine people's search intent. Google wants to rank pages that best fit the search term, as well as the search intent behind a specific search query. Therefore, make sure your post or page fits the search intent of your audience.

1. Informational intent

Lots of searches on the internet are done by people looking for information. That could be information about the weather, information about educating children, information about SEO, you name it. People with informational intent have a specific question or want to know more about a certain topic.

2. Navigational intent

People with this intent want to visit a specific website. For example, people who search for [Facebook] are usually on their way to the Facebook website.

3. Transactional intent

Lots of people buy stuff on the internet and browse the web to find the best purchase. People are searching with transactional intent when their purpose is to buy something.

4. Commercial investigation

Some people have the intention to buy in the (near) future and use the web to do their research. What washing machine would be best? Which SEO plugin is the most helpful? These people also have transactional intent but need some more time and convincing. These types of search intents are usually called commercial investigating intents.

Due to the diversity of language, many queries have more than one meaning – for example, [Apple] can either be a consumer electrical goods brand or a fruit.

Google handles this issue by classifying the query by its interpretation. The interpretation of the query can then be used to define intent. Query interpretations are classified into the following three areas:

1. Dominant Interpretations

The dominant interpretation is what most users mean when they search a specific query. Google search raters are told explicitly that the dominant interpretation should be clear, even more so after further online research.

2. Common Interpretations

Any given query can have multiple common interpretations. The example given by Google in their guidelines is [mercury] – which can mean either the planet or the element. In this instance, Google can't provide a result that Fully Meets a user's search intent but instead, produces results varying in both interpretation and intent (to cover all bases).

3. Minor Interpretations

A lot of queries will also have less common interpretations, and these can often be locale dependent.

Mobile search surpassed desktop search globally in May 2015 in the greater majority of verticals. In fact, a recent study indicates that 57percent of traffic comes from mobile and tablet devices.

Google has also moved with the times – the two mobile-friendly updates and the impending mobile-first index being obvious indicators of this. Increased internet accessibility also means that we are able to perform searches more frequently based on real-time events.

As a result, Google is currently estimating that 15 percent of the queries it's handling on a daily basis are new and have never been seen before. This is in part due to the new accessibility that the world has and the increasing smartphone and internet penetration rates being seen globally.

According to ComScore, mobile is gaining increasing ground not only in how we search but in how we interact with the online sphere. In a number of countries, including the United States, United Kingdom, Brazil, Canada, China, and India, more than 60 percent of our time spent online is through a mobile device.

One key understanding of mobile search is that users may not also satisfy their query via this device. In my experience, working across a number of verticals, a lot of mobile search queries

tend to be more focused on research and informational, moving to desktop or tablet at a later date to complete a purchase.

Do, Know, Go is a concept that search queries can be segmented into 3 categories. These classifications to an extent determine type of results Google delivers to its users.

Do (Transactional Queries)

When a user performs a "do" query, they are looking to achieve a specific action, such as purchasing a specific product or booking a service. These are important to e-commerce websites for example, where a user may be looking for a specific brand or item. Device action queries are also a form of do query and are becoming more and more important given how we interact with our smartphones and other technologies.

Know (Informational Queries)

A "know" query is an informational query, where the user is wanting to learn about a particular subject. Know queries are closely linked to micro-moments. In September 2015, Google released a guide to micro-moments, which are happening due to increased smartphone penetration and internet accessibility. Micro-moments occur when a user needs to satisfy a specific query there and then, and these often carry a time factor, such as checking train times or stock prices.

Go (Navigational Queries)

"Go" queries are typically brand or known entity queries, where a user is looking to go to a specific website or location. If a user is specifically searching for Adidas, serving them Puma as a result wouldn't meet their needs. Likewise, if your client wants to rank for a competitor brand term, you need to make them question why would Google show their site when the user is clearly looking for the competitor.

For a long time, the customer journey is a staple activity in planning and developing both marketing campaigns and websites. While mapping out personas and planning how users navigate the website is important, it's necessary to understand how a user searches and at what stage of their own journey they are at.

The internet has redefined the traditional marketing funnel—because of the amount of information *literally* available at their fingertips, people no longer adhere to a linear buyer's journey.

The new buyer's journey is a looping, continuous cycle, with liquid starting and ending points. Consumers want to do their own research, and they use that research to make informed

purchase decisions. If you want to convert more customers, you need to provide content for every single stage of this new looping buyer's journey, and you can do it by optimizing your content for search intent.

The word journey often sparks connotations of a straight path and a lot of basic user journeys usually follow the path of landing page > form or homepage > product page > form. We assume that users know exactly what they want to do, but mobile and voice search have introduced a new dynamic to our daily lives and shape our day-to-day decisions in a way like no other.

Module-157

Content and SEO.

At their best, they form a bond that can catapult any website to the top of search engine rankings.

But that's only when they're at their best. Because, when they're at their worst, they can cause Google penalties that are near impossible to recover from.

The purpose of this module is simple; to provide you with an understanding of why content is important for SEO and show you what you can do to make sure they work together in harmony.

As we dive in, we'll gain a better understanding of what content means, what its SEO value is, and how to go about creating optimized content that lands you on the search engine radar.

"High quality, useful information that conveys a story presented in a contextually relevant manner with the goal of soliciting an emotion or engagement. Delivered live or

asynchronously, content can be expressed using a variety of formats including text, images, video, audio, and/or presentations."

If we avoid a description of "quality" content, we can take a more direct approach by looking at the dozens of different types of digital content.

"Content comes in any form (audio, text, video), and it informs, entertains, enlightens, or teaches the people who consume it."

The reason optimized content is important is simple… you won't rank in search engines without it.

But, as we've already touched on briefly, it's important to understand that there are multiple factors at play here. On one side, you have content creation.

Optimizing content during creation is done by ensuring that your content is audience-centric and follows the recommendations laid out in the previous section.

But what does audience-centric mean and how does it differ from other types of content? This graphic does a great job of explaining the difference:

Audience-centric simply means that you're focusing on what audiences want to hear rather than what you want to talk about.

And, as we've identified, producing useful and relevant content is the name of the game if you're looking to rank in search engines.

On the other side of the optimization equation is the technical stuff. This involves factors like keywords, meta titles, meta descriptions, and URLs.

And that's what we're going to talk about next as we dive into how to actually create optimized content.

1. While we've already identified that your main goal should be to create audience-centric content, keyword research is necessary to ensure that the resulting content can be found through search engines.

2. Online readers have incredibly short attention spans. And they're not going to stick around if your article is just one ginormous paragraph. It's best to stick with paragraphs that are 1-2 sentences in length, although it's alright if they stretch to 3-4 shorter sentences.

3. Don't try to write about everything and anything within a single piece of content. And don't try to target dozens of keywords. Doing so is not only a huge waste of time, but it prevents you from creating the most "useful and relevant" content on your topic.

4. Since Google has made it clear that credibility is an important SEO factor, linking to relevant, trustworthy, and authoritative sites can help ensure that search engines see your content as credible. Be sure, however, that the words you're using for the link are actually relevant to the site the user will be sent to.

1. Title tags help search engines understand what your page is about. In addition, they can be a determining factor for which search result a user chooses. To optimize your title tag, you'll want to be sure of the following:

• Keep it under 60 characters.

• Don't stuff multiple keywords into the title.

• Be specific about what the content is about.

• Place target keywords at the beginning.

2. Your meta description is the small snippet of text that appears under the

title tag and URL. As far as meta description best practices, you should:

• Keep it under 160 characters.

• Provide a short, specific overview of what the content is about.

• Include relevant keywords (they will be highlighted when a user sees search results).

3. Your URL structure is another component of SEO that has an indirect impact on rankings as it can be a factor that determines whether a user clicks on your content.

Readability is most important here, as it ensures that search users aren't scared off by long and mysterious URLs.

The image provides a great example of how URL readability can affect the way a user sees results.

Module-158

I want to make one thing perfectly clear: When it comes to SEO, every page is a landing page. And when I say “every page,” I mean every single page that is crawlable and indexable by the search engines.

So if you don’t want search engines indexing a page, be sure to block them from it. Everything else – and I mean everything – needs to be treated as a landing page.

But not all pages are created equal. Every page on a website will have a unique purpose, will attract different audiences, and will direct visitors to different goals.

So what then makes a good landing page? There are a lot of answers to that question that has to do with design, usability, conversions, etc. Let me give you some quick hits that cover it all.

A good landing page:

- Captures the visitors’ attention.
- Addresses their needs, wants, and requirements.
- Answers important/relevant questions.
- Entices them to want the solution you provide.
- Directs them to take a particular action.
- Compels them to do so.

The home page must provide a global view of what the website offers. It should give visitors the “big picture” of the products and services you offer and why they should do business with you. It acts as a doorway for the visitor to enter and begin their journey into your site where they’ll find more details about what you offer.

Many SEOs make the mistake of trying to optimize home pages for the business’s primary product or service. This strategy can be just fine if you’re a singular product or service company. But the moment you offer something outside of the one product/category scope, the optimization of the home page becomes irrelevant.

The better and more sustainable strategy for optimizing a home page is to focus on the company brand name. And in that sense, that makes optimizing the home page easy because when you type in the name of the company, the home page of that business should be more relevant than any page on a competitor’s website.

About US Page

Studies have proven that visitors who have seen a site's About Us page are more likely to convert than those that don't. This statistic can prove either a symptom or a result.

• **Symptom:** Visitors who are close to converting check out a site's About Us before they commit.

• **Result:** Visitors who visit an About Us page are heavily influenced by the content and become more likely to convert, if the page satisfies what they wanted to learn.

The About Us page may seem like an odd one to try to optimize, but in reality, there are a lot of keywords that are tailor made for these pages. Any industry- or product-related keywords that are qualified with company, business, agency, firm, office, bureau, or similar types of keywords are ready-made fits for the About Us page.

Contact Us page

Only one reason a visitor navigate to this page: They want your contact information.

What they actually do with that information is anyone's guess. Maybe they'll send you an email, maybe they'll call, or maybe they just want to know where you're located. And it's this last option that provides us with prime optimization fodder.

Whether you're a national or a local company, inevitably, some people prefer to do business with someone close by. A quick bit of keyword research will likely prove this out for your industry. While local business may not be your bread and butter, there is no reason to ignore it either.

Product category and sub-category pages provide fantastic optimization opportunities. In the buying cycle, these pages most frequently serve those who are in the shopping phase. That means those visitors have a good idea of what they want but are looking to learn more about the options available to them.

The goal of the page is to give the visitor access to those options, which are usually the actual product detail pages themselves. For the most part, the product category pages are nothing more than pass-through pages. Visitors may revisit the page frequently, but only so you can pass them through to the products.

On an SEO level, these pages are an optimization gold mine. The keywords that these pages cover are generally not so broad that they lose all value, but not so specific that they lose all search volume. Consider them the SEO sweet spot.

These pages do, however, present something of a problem for many SEOs. Pages need content in order to be optimized, but visitors on these pages don't want content, they just want to see the products. At least that's what many believe. However, not everybody subscribes to this theory.

When a visitor is in the shop phase of the sales cycle, they will visit a lot of product detail pages. When they move on to the buy phase, that means they have gathered enough information to know fairly precisely what they want. Now they are just looking at the fine details and deciding which version of the product they want and who to buy it from.

When it comes to optimizing product pages, keyword research almost becomes irrelevant. That's because there are so many variables that it's impossible to focus the content of these pages on every potential variable in any traditional way. But, as odd as that sounds, that actually makes optimization of these pages that much easier. And it has less to do with the keywords and more about the construction of the page's content.

As with any page, you want to optimize the tags: title, description, alt, headings, etc. But where most other pages require a custom approach, product pages can easily be optimized en masse by using dynamic keyword insertion.

FAQs Page

In the age of Google answer boxes, help and FAQ pages have become more important than ever. While you always want to make sure you are answering questions throughout your website, FAQ pages provide a good catch-all for the often requested information. And they are ready-made for getting your content to appear in the coveted answer box.

Not sure what your most asked questions are? Your keyword research will tell you. Search for your keyword and then pull out any phrases that start with who, what, when, where, why, and how. Decide which questions are worth answering and which aren't and you have yourself the start of a FAQ page!

Blog Posts

Every site has a limit to the number of pages that can be added before it gets overly cluttered and begins to interfere with the conversion process. But there is almost no limit to the number of relevant topics you can optimize pages for. This is where blog posts come into play.

Any topic that you can't explore – or can't explore as in depth – on your main site, can be explored in great detail in a blog post. Or a series of blog posts. Every post can be targeted for a specific searcher's need and be used to drive relevant traffic to your site.

TOPIC-16: Amazon Web Services (AWS)

Module-159

✓ **On-demand self-service:** A consumer can unilaterally provision computing capabilities, such as server time and network storage, automatically as needed without requiring human interaction with each service provider.

✓ **Broad network access:** Capabilities are available over the network and accessed via standard mechanisms that promote use by heterogeneous thin or thick client platforms (such as mobile phones, tablets, laptops, and workstations).

✓ **Resource pooling:** The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. L*ocation independence* i.e. the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (by country, state, or data center, for example).

✓ **Rapid elasticity:** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

✓ **Measured service:** Cloud systems automatically control and optimize resource use by leveraging a metering capability at a level of abstraction that's appropriate to the type of service (e.g. storage, processing, bandwidth, or active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

✓ **Infrastructure as a Service (Iaas):** Offers users the basic building blocks of computing: processing, network connectivity, and storage. (Of course, you also need other capabilities in order to fully support IaaS functionality — such as user accounts, usage tracking, and security.) You would use an IaaS cloud provider if you want to build an application from scratch and need access to fairly low-level functionality within the operating system.

✓ **Platform as a Service (PaaS):** Instead of offering low-level functions within the operating system, offers higher-level programming frameworks that a developer interacts with to obtain

computing services. For example, rather than open a file and write a collection of bits to it, in a PaaS environment the developer simply calls a function and then provides the function with the collection of bits. The PaaS framework then handles the grunt work, such as opening a file, writing the bits to it, and ensuring that the bits have been successfully received by the file system. The PaaS framework provider takes care of backing up the data and managing the collection of backups, for example, thus relieving the user of having to complete further burdensome administrative tasks.

✓ **Software as a Service (SaaS):** Has clambered to an even higher rung on the evolutionary ladder than PaaS. With SaaS, all application functionality is delivered over a network in a pretty package. The user need do nothing more than use the application; the SaaS provider deals with the hassle associated with creating and operating the application, segregating user data, providing security for each user as well as the overall SaaS environment, and handling a myriad of other details.

If you find the mix of *I, P,* and *S* in the preceding module confusing, wait 'til you hear about the whole private-versus-public cloud computing distinction. Note the sequence of events:

1. Amazon, as the first cloud computing provider, offers *public cloud computing* — anyone can use it.

2. Many IT organizations, when contemplating this new Amazon Web Services creature, asked why they couldn't create and offer a service like AWS to their own users, hosted in their own data centers. This on-premise version became known as *private cloud computing.*

3. Continuing the trend, several hosting providers thought they could offer their IT customers a segregated part of their data centers and let customers build clouds there. This concept can also be considered private cloud computing because it's dedicated to one user. On the other hand, because the data to and from this private cloud runs over a shared network, is the cloud truly private?

4. Finally, after one bright bulb noted that companies may not choose only public or private, the term *hybrid* was coined to refer to companies using both private and public cloud environments.

Amazon Web Services was officially in March, 2006 by offering Simple Storage Service, its first service i.e. S3. The idea behind S3 was simple: It could offer the concept of object storage over the web, a setup where anyone could put an object — essentially, any bunch of bytes — into

S3. Those bytes may comprise a digital photo or a file backup or a software package or a video or audio recording or a spreadsheet file etc.

In its first six years, S3 has grown in all dimensions. The service is now offered throughout the world in a number of different regions. Objects can now be as large as 5 terabytes. S3 can also offer many more capabilities regarding objects. An object can now have a termination date, for example: You can set a date and time after which an object is no longer available for access.

S3 did not remain the lone AWS example for long. Just a few months after it was launched, Amazon began offering Simple Queue Service (SQS), which provides a way to pass messages between different programs. SQS can accept or deliver messages within the AWS environment or outside the environment to other programs (your web browser, for example) and can be used to build highly scalable distributed applications.

The overall pattern of AWS has been to add additional services steadily, and then quickly improve each service over time. AWS is now composed of more than 25 different services, many offered with different capabilities via different configurations or formats. This rich set of services can be mixed and matched to create interesting and unique applications, limited only by your imagination or needs.

Amazon is the pioneer of cloud computing and, because you'd have to have been living under a rock not to have heard about "the cloud," being the pioneer in this area is a big deal. The obvious question is this: If AWS is the big dog in the market and if cloud computing is the hottest thing since sliced bread, how big are we talking about?

That's an interesting question because Amazon reveals little about the extent of its business. Rather than break out AWS revenues, the company lumps them into an Other category in its financial reports.

Amazon itself provides a proxy for the growth of the AWS service. Every so often, it announces how many objects are stored in the S3 service. Take a peek at Figure 1-1, which shows how the number of objects stored in S3 has increased at an enormous pace, jumping from 2.9 billion at the end of 2006 to over 2 trillion objects by the end of the second quarter of 2012. Given that pace of growth, it's obvious that the business of AWS is booming.

Other estimates of the size of the AWS service exist as well. A very clever consultant named Huan Liu examined AWS IP addresses and projected the total number of server racks held by

vulearninghub.xyz | by VU Learning HUB

AWS, based on an estimate of how many servers reside in a rack. Table 1-1 breaks down the numbers by region.

vulearninghub.xyz | by VU Learning HUB

Module-160

Unlike most of its competitors, Amazon builds its hardware infrastructure from commodity components. *Commodity*, here refers to using equipment from lesser-known manufacturers who charge less than their brand-name competitors. For components for which commodity offerings aren't available, Amazon negotiates rock-bottom prices.

The brand-name hardware providers assert that one benefit of paying premium prices is higher-quality. It may be true that premium priced equipment (traditionally called enterprise equipment because of the assumption that large enterprises require more reliability and are willing to pay extra to obtain it) is more reliable in an apples-to-apples comparison. That is, an enterprise-grade server lasts longer and suffers fewer outages than its commodity-class counterpart. But, how much more reliable the enterprise gear, and how much that improved reliability is worth. In other words, it needs to know the cost-benefit ratio of enterprise-versus-commodity. At the scale of million servers *whoever* provides it — is breaking all the time.

The scale at which Amazon operates affects other aspects of its hardware infrastructure as well. Besides components such as servers, networks, and storage, data centers also have power supplies, cooling, generators, and backup batteries. Depending on the specific component, Amazon may have to use custom-designed equipment to operate at the scale required.

✓ **Based on virtualization:** A technology that abstracts software components from dependence on their underlying hardware is the basis of AWS. Allowing creating virtual machines, start them, terminate them, and restart them quickly makes AWS possible.

✓ **Operated as a service:** Amazon created a software infrastructure in order to be able to offer its computing capability as a service. For example, Amazon had to create a way for users to operate their AWS resources from a distance and with no requirement for local hands-on interaction. And it had to segregate a user's resources from everyone else's resources to ensure security, because no one wants other users to be able to see, access, or change his resources.

✓ **Designed for flexibility:** Computing services available at a moment's notice to support application needs and constantly changing business conditions. In other words, just as Amazon can't predict what its computing requirements will be in a year or two, neither can the market for which Amazon built AWS.

✓ **Highly resilient:** Hardware is inherently unreliable, there is no way to implement hardware resiliency. AWS are made highly resilient by implementing resource redundancy — essentially using multiple copies of a resource to ensure failure of a single resource does not cause service

to fail. For example, instead of storing just one copy of each of your objects in its S3 service, that object may sometimes be unavailable due to disk failure. Instead, AWS keeps multiple copies of an object, ensuring that even if one — or two! — objects become unavailable because of hardware failure, users can still access the object, thereby improving S3 reliability and durability.

vulearninghub.xyz | by VU Learning HUB

Module-161

✓ **AWS computing services provided by Amazon:** As noted earlier, Amazon currently provides more than 25 AWS services and is launching more all the time. AWS provides a large range of cloud computing services — you'll be introduced to many of them in this course.

✓ **Computing services provided by third parties that operate on AWS:** These services tend to offer functionality that enables you to build applications of a type that AWS doesn't strictly offer. For example, AWS offers some billing capability to enable users to build applications and charge people to use them, but the AWS service doesn't support many billing use cases — user-specific discounts based on the size of the company, for example. Many companies (and even individuals) offer services complementary to AWS that then allow users to build richer applications more quickly. (If you carry out the AWS exercises I set out for you later in this book, you'll use one such service offered by Bitnami.)

✓ **Complete applications offered by third parties that run on AWS:** You can use these services, often referred to as SaaS (Software as a Service), over a network without having to install them on your own hardware. (Check out the "IaaS, Paas, SaaS" section, earlier in this chapter, for more on SaaS.) Many, many companies host their applications on AWS, drawn to it for the same reasons that end users are drawn to it: low cost, easy access, and high scalability. An interesting trend within AWS is the increasing move by traditional software vendors to migrate their applications to AWS and provide them as SaaS offerings rather than as applications that users install from a CD or DVD on their own machines.

Here are some benefits of being able to leverage the network effects of the AWS ecosystem in your application:

✓ **The service is already up and running within AWS.** You don't have to obtain the software, install it, configure it, test it, and then integrate it into your application. Because it's already operational in the AWS environment, you can skip directly to the last step — perform the technical integration.

✓ **The services have a cloud-friendly licensing model.** Vendors have already figured out how to offer their software and charge for it in the AWS environment. Vendors often align with the AWS billing methodology, charging per hour of use or offering a subscription for monthly access. But one thing you don't have to do is approach a vendor that has a large, upfront license fee and negotiate to operate in the AWS environment — it's already taken care of.

✓ **Support is available for the service.** You don't have to figure out why a software component you want to use doesn't work properly in the AWS environment — the vendor takes

responsibility for it. In the parlance of the world of support, you have, as the technology industry rather indelicately puts it, a throat to choke.

✓ **Performance improves.** Because the service operates in the same environment that your application runs in, it provides low latency and helps your application perform better.

✓ **The focus is on the concerns of IT operations rather than on the concerns of developers.** Often, this concern translates as, "The service is not easy to use." For example, an enterprise cloud provider may require a discussion with a sales representative before granting access to the service and then impose a back-and-forth manual process as part of the account setup. By contrast, AWS allows anyone with an e-mail address and a credit card access to the service within ten minutes.

✓ **The service itself reflects its hosting heritage, with its functionality and use model mirroring how physical servers operate.** Often, the only storage an enterprise cloud service provider offers is associated with individual virtual machines — no object storage, such as S3, is offered, because it isn't part of a typical hosting environment.

✓ **Enterprise cloud service providers often require a multiyear commitment to resource use with a specific level of computing capacity.** Though this strategy makes it easier for a cloud service provider to plan its business, it's much less convenient for users — *and* it imposes some of the same issues that they're trying to escape from!

✓ **The use of enterprise equipment often means higher prices when compared to AWS.** I have seen enterprise cloud service providers charge 800 percent more than AWS. Depending on organization requirements and the nature of the application, users may be willing to pay a premium for these providers; on the other hand, higher prices and the long-term commitment that often accompanies the use of an AWS competitor may strike many users as unattractive or even unacceptable.

Module-162

The AWS environment acts as an integrated collection of hardware and software services designed to enable the easy, quick, and inexpensive use of computing resources.

Now, sitting atop this integrated collection is the AWS application programming interface (API, for short): In essence, an *API* represents a way to communicate with a computing resource. With respect to AWS, nothing gets done without using the AWS API.

The term API traditionally referred to the programming interface offered by one or more routines that were bundled into a library of functions. Someone would supply a library that, say, performs date-and-time manipulation functions. A software engineer would bundle that library into a program and could then call those functions via the API that the library offers. The API represents the "contract" that the library offers.

The API defines the functional interface, the format of any information supplied to the functions (commonly called arguments or parameters) within the library, the operation to be performed, and the output that each function would return to the calling program.

The meaning of the term *API* has been extended: Rather than be used solely to discuss libraries that are directly attached to other programs, it's now used to refer to software environments in which the different software programs run on different servers and communicate across a network.

Furthermore, that network may be contained within a single data center or, quite commonly, extend across the Internet. This network-based API approach is often referred to as a *web services environment* — notice how Amazon's cloud computing offering is named Amazon Web Services?.

Innovation: Just as musical mash-ups let people combine musical resources into new creations, so, too, do web services foster innovation. Though I may not be able to see the value in a combination of, say, vehicle gas mileage ratings, local gas prices, and state park reviews, someone else may conclude that an application allowing someone to enter the make and model of her automobile to find out which parks one can visit for less than $25 in gas costs would o the trick and many people may agree.

Niche market support: In a non-web-services world, the only people who can develop applications are those working for organizations. Only they have access to the computing resources or data — so the only applications that are developed are ones that the company

deems useful. However, once those resources and data are made available via web services, anyone can create an application, which allows the development of applications targeted at niche markets. For example, someone can combine Google Maps with a municipal bus schedule in a mobile app to allow users to see when and where the next bus will be available nearby.

New sources of revenue: Companies can provide a web services interface into their business transaction systems and allow outside entities to sell their goods. For example, the large retailer Sears has made it possible for mobile app developers and bloggers to sell Sears goods via a Sears web service. These developers and bloggers reach audiences that Sears may not be able to reach — but Sears can prosper without having to be involved.

SOAP (short for Simple Object Access Protocol), had widespread industry support, complete with a comprehensive set of standards. Those standards were too comprehensive. The people designing SOAP set it up to be extremely flexible — it can communicate across the web, e-mail, and private networks. To ensure security and manageability, a number of supporting standards that integrate with SOAP were also defined.

SOAP is based on a document encoding standard known as Extensible Markup Language (XML, for short), and the SOAP service is defined in such a way that users can then leverage XML no matter what the underlying communication network is.

REST, or Representational State Transfer is far less comprehensive than SOAP, aspires to solve fewer problems. It doesn't address some aspects of SOAP that seemed important but that, in retrospect, made it more complex to use — security, for example.

AWS originally launched with SOAP support for interactions with its API, but it has steadily *deprecated* (reduced its support for, in other words) its SOAP interface in favor of REST.

Module-163

The AWS environment acts as an integrated collection of hardware and software services designed to enable the easy, quick, and inexpensive use of computing resources.

Now, sitting atop this integrated collection is the AWS application programming interface (API, for short): In essence, an *API* represents a way to communicate with a computing resource. With respect to AWS, nothing gets done without using the AWS API.

REST is designed to integrate with standard web protocols so that REST services can be called with standard web verbs and URLs. For example, a valid REST call looks like this:
http://search.examplecompany.com/CompanyDirectory/EmployeeInfo?empname=IsmailSiddiq
i
A query to the REST service of examplecompany to see an employee's personnel information. The HTTP verb that accompanies this request is GET, asking for information to be returned.
If you take a quick look at the following example of an AWS API call, you'll quickly see that it closely resembles the REST example above:
https://ec2.amazonaws.com/?Action=RunInstances
&ImageId=ami-60a54009
&MaxCount=3
&MinCount=1
&Placement.AvailabilityZone=us-east-1b
&Monitoring.Enabled=true
&AUTHPARAMS

The call, which is straightforward, instructs AWS to run between one and three instances based on an Amazon machine image of ami-60a54009 and to place them in the us-east-1b availability zone. AWS provides monitoring capabilities, and this call instructs AWS to enable this monitoring. The AUTHPARAMS part is a stand-in for the information that AWS uses to implement security in its API.

At this point, you might not feel confident about your ability to successfully use AWS. Never fear. That's because many clever people have recognized that the API is difficult to use and have created tools to make AWS simpler to use. Figure shows the four major categories of AWS interaction mechanisms that spare you from the burden of interacting with the AWS API directly.

✓ **AWS management console:** Amazon offers a graphical web interface that allows you to interact with service (and your own) computing resources. For many people, the AWS

management console is the primary mechanism they use to operate AWS. Even people who use the other two mechanisms to interact with AWS also make heavy use of the management console.

✓ **CLI/SDK:** Many software engineers write applications that need to interact with AWS services directly. Now, calling the web services API directly is complicated and error-prone. To help them, Amazon and other companies have created language libraries (commonly called SDKs, standing for Software Development Kits) and a command line interface (commonly called a CLI), which allows commands to be entered in a terminal connected to AWS. A software engineer can more easily incorporate library routines into an application, making it easier and faster to build AWS-based applications.

✓ **Third-party tools:** Many companies build tools that incorporate AWS. Some of these tools extend or simplify AWS itself, similar to what the language libraries do for software engineers. Other tools are products that offer separate functionality or even entire applications. What these tools have in common is that they provide functionality to shield users from interacting with the AWS API, making AWS easier and faster to use.

When you sign up for an account with AWS, you create an access key and a secret access key sent to you. Each one is a lengthy string of random characters, and the secret access key is the longer of the two. When you download the secret access key, you should store it somewhere secure. After you do this, both you and Amazon have a copy of the access key and the secret access key. Retaining a copy of the secret access key is crucial because it's used to encrypt information sent back and forth between you and AWS, and if you don't have the Secret Access Key, you can't execute any service calls on AWS. In addition to this encryption, AWS has two other methods it uses to ensure the legitimacy of the service call:

✓ The first is based on the date information included with the service call payload, which it uses to determine whether the time associated with the making of the service call is appropriate; if the date in the service call is much different from what it should be AWS concludes that it isn't a legitimate service call and discards it.

✓ The second additional security measure involves a checksum you calculate for the payload. (A *checksum* is a number that represents the content of a message.) AWS computes a checksum for the payload; if its checksum doesn't agree with yours, it disallows the service call and doesn't execute it. If someone tampers with the message, when AWS calculates a checksum, that checksum no longer matches the one included in the message, and AWS refuses to execute the service call.

Module-164

The AWS environment acts as an integrated collection of hardware and software services designed to enable the easy, quick, and inexpensive use of computing resources.

Now, sitting atop this integrated collection is the AWS application programming interface (API, for short): In essence, an *API* represents a way to communicate with a computing resource. With respect to AWS, nothing gets done without using the AWS API.

Here are two important points to take away from this initial account setup:

✓ **Your account is now set up as a general AWS account.** You can use AWS resources anywhere in the AWS system — the US East or either of the two US West regions, Asia Pacific (Tokyo, Singapore, or Australia), South America (Brazil), and Europe (Ireland). Put another way, your account is scoped over the entirety of AWS, but resources are located within a specific region.

✓ **You have given AWS a credit card number to pay for the resources you use.** In effect, you have an open tab with AWS, so *be careful* about how much computing resource you consume. For the purposes of this book, you don't have to worry much about costs — your initial sign-up provides a free level of service for a year that should be sufficient for you to perform the steps in this course as well as experiment on your own without spending any money.

Module-165

✓ **Storage is an increasingly important topic to IT because of the recent staggering increase in the amount of data that businesses use in their day-to-day operations.** Though traditional structured data (the database) is growing quite rapidly, the use of digital media (video) by businesses is exploding. IT organizations are using more and more storage, and they often look to communication service providers (CSPs) such as Amazon to provide storage. Another driver of storage consumption is the recent rise of *big data,* which refers to analyzing very large datasets. Companies are drowning in data, and many are finding it nearly impossible to keep up with managing their own, on-premises storage systems.

✓ **Storage is the first AWS offering that Amazon offered.** Storage therefore holds a significant place in the AWS ecosystem, including some extremely innovative uses of its storage services by AWS customers over the years.

✓ **A number of AWS offerings rely on AWS storage, especially Simple Storage Service (S3).** Understanding AWS storage services helps you better understand the operation of the AWS offerings that rely on AWS storage.

✓ **AWS continues to innovate and deliver new storage services.** Glacier, for example, provides a fresh twist on addressing a historic IT issue: archival storage. Glacier is discussed later in this module as well (in case you need something to look forward to).

Why does Amazon offer four different AWS storage services? Put simply, the enormous growth of storage makes traditional approaches (local storage, network-attached storage, storage-area networks, and the like) no longer appropriate, for these three reasons:

✓ **Scaling:** Traditional methods simply can't scale large enough to handle the volume of data that companies now generate. The amounts of data that companies must manage outstrip the capabilities of almost all storage solutions.

✓ **Speed:** They can't move data fast enough to respond to the demands that companies are placing on their storage solutions. To be blunt, most corporate networks cannot handle the level of traffic required to shunt around all the bits that companies store.

✓ **Cost:** Given the volumes of data being addressed, the established solutions aren't economically

For these reasons, the issue of storage has long since moved beyond local storage (for example, disk drives located within the server using the data). Over the past couple decades, two other forms of traditional storage have entered the market — network-attached storage (NAS) and storage-area networks (SAN) — which move storage from the local server to within the network on which the server sits. When the server requires data, rather than search a local disk for it, it seeks it out over the network.
Object storage provides the ability to store objects — which are essentially collections of digital bits. Those bits may represent a digital photo, an MRI scan, a structured document such as an XML file — or a video.

Object storage offers the reliable (and highly scalable) storage of collections of bits, but imposes no structure on the bits. The structure is chosen by the user, who needs to know, for example, whether an object is a photo (which can be edited), or an MRI scan (which requires a special application for viewing it). The user has to know both the format as well as the manipulation methods of the object. The object storage service simply provides reliable storage of the bits.

Object storage differs from file storage, which you may be more familiar with from using a PC. File storage offers *update functionality,* and object storage does not. By contrast, using file storage allows you to continuously update the file by *appending* new information to it — in other words, you update the file as the program creates new log records.

Object storage offers no such update ability. You can insert or retrieve an object, but you can't change it. Instead, you update the object in the local application and then insert the object into the object store.
Distributed key-value storage, in contrast to object storage, provides structured storage that is somewhat akin to a database but different in important ways in order to provide additional scalability and performance. Though key-value storage systems vary in different ways, they have these common characteristics:

✓ **Data is structured with a single key that's used to identify the record in which all remaining data resides.** The key is almost always unique — such as a user number, a unique username (title_1795456, for example), or a part number. This ensures that each record has a unique key, which helps facilitate scale and performance.

✓ **Retrieval is restricted to the key value.** For example, to find all records with a common address (where the address is not the key), every record has to be examined.

✓ **No support exists for performing searches across multiple datasets with common data elements.** Don't support joins, the two tables would have to be matched at the application level rather than by the storage systems. Using this concept, commonly described as "The intelligence resides in the application," executing joins requires application "smarts" and lots of additional coding. Key-value storage represents a trade-off between ease of use and scalability, and the trade-off is biased toward scalability (and less ease of use).

vulearninghub.xyz | by VU Learning HUB

Module-166

Simple Storage Service (fondly known as S3) is one of the richest, most flexible, and, certainly, most widely used AWS offerings. It's no exaggeration to call S3 "the filing cabinet of the Internet." Its object storage is used in an enormous variety of applications by individuals and businesses, such as

✓ **Dropbox:** This file storage and syncing service uses S3 to store all of the documents it stores on behalf of its users.

✓ **Netflix:** This popular online consumer video service uses S3 to store videos before they go out to its Content Delivery Network. In fact, Netflix operates almost 100 percent on AWS, making it somewhat of a poster child for the service.

✓ **Medcommons:** This company stores customers' health records online in S3 — and, by the way, it complies with the strict requirements of the Health Insurance Portability and Accountability Act (HIPAA).

S3 objects are treated as *web objects* — that is, they're accessed via Internet protocols using a URL identifier.

A *bucket* in AWS is a group of objects. The bucket's name is associated with an account — for example, the bucket named aws4me is associated with aws4me account. The bucket name doesn't need to be the same as the account name; it can be anything. However, the bucket namespace is completely *flat:* Every bucket name must be unique among all users of AWS. If you try to create a bucket named test within your account, you'll see an error message because someone else has already claimed that name.

A *key* in AWS is the name of an object, and it acts as an identifier to locate the data associated with the key. In AWS, a key can be either an object name or a more complex arrangement that imposes some structure on the organization of objects within a bucket (as in bucketname/photos/myphotos/My+Photo.JPG, where /photos/myphotos is part of the object name).

This convenient arrangement provides a familiar directory-like or URL-like format for object names; however, it doesn't represent the actual structure of the S3 storage system — it's merely a comfortable and memorable method of naming objects, making it easy for humans to keep track. Even though many tools present S3 storage as though it's in a familiar file folder organization (including the AWS Management Console itself), they imply nothing about how the objects are stored within S3.

You probably won't use the (not particularly user-friendly) API to post (create), get (retrieve), or delete S3 objects. You may access them via a programming library that encapsulates the API calls and offers higher-level S3 functions that are easier to use.

More likely, however, you'll use an even higher-level tool or application that provides a graphical interface to manage S3 objects. You can be sure, however, that somewhere down in the depths of the library or higher-level tool, are calls to the S3 API.

In addition to the most obvious and useful actions for objects (such as post, get, and delete), S3 offers a wide range of object management actions — for example, an API call to get the version number of an object. Recall that object storage disallows updating an object (unlike a file residing within a file system). S3 works around this issue by allowing versioning of S3 objects — you can modify version 2 of an S3 object, for example, and store the modified version as version 3. This gets around the process to update objects outlined earlier: Retrieve old object, modify object in application, delete old object from S3, and then insert modified object with original object name.

AWS offers fine-grained access controls to implement S3 security: You can use these controls to explicitly control who-can-do-what with your S3 objects. The mechanism by which this access control is enforced is, naturally enough, the Access Control List (ACL).

These four types of people can access S3 objects:

✓ **Owner:** The person who created the object; he can also read or delete the object.

✓ **Specific users or groups:** Particular users, or groups of users, within AWS. (Access may be restricted to other members of the owner's company.)

✓ **Authenticated users:** People who have accounts within AWS and have been successfully authenticated.

✓ **Everyone:** Anyone on the Internet (as you may expect).

S3 provides a rich set of actions in the S3 API. Several functions, for example, allow the manipulation of object versions to retrieve a certain version of an object.

The access controls specify who, and the actions specify what — who has the right to do what with a given object. The interaction between the S3 access controls and the object actions gives S3 its fine-grained object management functionality.

Companies store content files used by their partners in S3. Most consumer electronics and appliance manufacturers now offer their user manuals in digital format; many of them store those files in S3. Many companies place images and videos used in their corporate websites in

S3, which reduces their storage management headaches — and ensures that in conditions of heavy web traffic, website performance isn't hindered by inadequate network bandwidth.

The most common S3 actions revolve, naturally enough, around creating, retrieving, and deleting objects. Here's the common lifecycle of an S3 object: Create the object in preparation to use it; set permissions to control access to the object; allow applications and people to retrieve the object as part of an application's functionality; and delete the object when the application that uses the object no longer requires it. Of course, many objects are never removed, because they have an ongoing purpose over a long time span.

As you get more familiar with S3, you'll undoubtedly start exploring additional S3 functionality. S3 offers encryption of objects stored in the service, securing your data from anyone attempting to access it inappropriately. You can log requests made against S3 objects to audit when objects are accessed and by whom. S3 can even be used to host *static* websites: They don't dynamically assemble data to create the pages served up as part of the website — removing the need to run a web server.

AWS as a whole is organized into regions, each of which contains one or more availability zones, or AZs. Although S3 locates buckets within regions, keep in mind that S3 bucket names are unique across all S3 regions, even though buckets themselves reside in particular regions. For example, if you create a bucket named after your company, you have to choose in which region to locate the bucket.

When an AWS virtual machine (VM) needs to access an S3 object, and the VM and the object reside in the same AWS region, Amazon imposes no charge for the network traffic that carries the object from S3 to EC2. If the VM and the object are in different regions, however (the traffic is carried over the Internet), AWS charges a few cents per gigabyte — which can be costly for very large objects or heavy use.

One way around this problem is to locate multiple buckets with duplicate objects in each region and tweak the bucket names to avoid conflicts — for example, by renaming my aws4me to aws4me_us_west and creating similarly named buckets in all other regions. I can then create duplicate objects in each of the similarly named buckets, to eliminate network traffic charges no matter where I run an EC2 instance (albeit at somewhat greater complexity and somewhat higher charges to pay for storing all the duplicate objects).

You're also charged for API calls to S3, which don't vary by volume. Finally, you pay for the network traffic caused by the delivery of S3 objects. Storage costs start at $.095 per gigabyte

per month for the first terabyte, and they trend downward as total storage increases to $.055 per gigabyte per month for more than 5000 terabytes of storage.

The API call costs vary from $.01 per 1,000 requests (for PUT, COPY, POST, or LIST calls) to $.01 per 10,000 requests (for GET and all other requests). DELETE requests are free.

Data transfer pricing — for transfers into or out of an AWS region — varies (as you can surmise) by volume. Transferring data in is a gift — there's no charge for inbound network traffic placing data into S3 storage. For outbound traffic, there's no charge for the first gigabyte of traffic. Then the charge becomes $.12 per gigabyte up to 10TB, with pricing lowered based on scale. The price is reduced to $.05 per gigabyte for traffic between 150TB and 500TB.

Amazon also offers *reduced redundancy* for S3 storage, which retains fewer copies of your data — and trades reliability for cost. Reduced redundancy storage starts at $.076 per gigabyte of storage and decreases to $.037 per gigabyte at volumes higher than 5,000TB.

Module-167

There are many users of AWS who struggle to describe why they adopted it. Still others are interested in AWS, but aren't sure about exactly what it is. And others who know what it is, and why they adopted it, but get tongue-tied when asked to justify their decision by higher management. To solve all those problems in one fell swoop, here is a list of the ten best reasons to use AWS.

IT has a reputation as the "Department of No." Though it's true that in some IT organizations innumerable and inexplicable roadblocks are placed in the way of anyone seeking access to the of "infrastructure," others are frustrated by the sheer complexity of coordinating many different resources, each with its own interface and configuration rules, all of which must be successfully stitched together to provide access to computing resources. Most of these multidepartment, manual, time-consuming efforts are the result of the years-long build-up of established processes executed in serial fashion, resulting in IT provisioning cycles that commonly require weeks to months to deliver computing resources. The result of all this: It's slower than snail and widely despised.

Amazon, rethought the provisioning process as though it were being designed from scratch and implemented it as an integrated and automated service. Because every part of the infrastructure is managed via an API, no human interaction is necessary for the installation or configuration of resources. And, because the services are offered in a fine-grained fashion (IP addresses are managed separately from storage, as one example), resources can be defined and started in parallel, rather than having to be done one step at a time. The result: IT resources are available in minutes, not in weeks or months.

Consequently, AWS enables the presence of IT agility — the ability of IT organizations to respond rapidly to resource requirements — and allows it to stop being thought of as the "Department of No."

Sales or human resources, for example — also calls IT the "Department of No." And being faced with a slow-moving IT organization today isn't only inconvenient — it's also dangerous to your business.

This danger results from the changing nature of IT applications. In the past, IT applications primarily automated internal company processes @— payroll, invoicing, document management — commonly referred to as systems of record because they, well, recorded information. Applications are now far more likely to be used to interact with customers or, indeed, to enable customers to "self-service" their needs. These applications are often called

systems of engagement because they foster engagement with parties outside the organization. And the rise of smartphones, tablets, and sophisticated websites raises the bar on customer expectations. When I realize that a company I use offers me a way to check the status of my orders online, I quickly expect the other companies I use to offer the same capability. And if those other companies don't provide it, I'm likely to look for other providers that do.

To satisfy today's customers, businesses have to roll out new applications quickly — to be agile, in other words. And AWS enables business agility enormously. It's no secret that a significant part of the AWS user base is made up of business units that adopt AWS as a way around IT and its protracted provisioning processes. This business unit adoption is sometimes called shadow IT or, even more pejoratively, rogue IT. No matter what you call it, this adoption occurs because business units feel the need to roll out new applications quickly in order to respond to market demands — and AWS helps businesses quicken their response time by being more agile.

Building an application can take a long time, because the developer needs to install and configure the software components, all of which takes time. For components that require commercial licensing, you have to arrange for payment — which can take a long time, given the complexities of budget approval and contract negotiation.
Amazon makes application development faster and less difficult with a rich services ecosystem:

- **A range of services as part of its AWS offering:** These services range from foundation building blocks, such as object and volume storage, to platform services, such as queues and e-mail, upto full applications, such as Elastic MapReduce and Redshift.
- **Services hosted on AWS by many third-party companies:** For example, both Informatica and Dell Boomi offer application integration services within AWS. AWS users can integrate applications running in AWS via these services and never have network traffic exit AWS, causing lower network latency and better application performance.
- **All home-grown (and most third-party) AWS services are offered with the same pricing model as AWS:** Pricing is standardized with simple contracts that can be executed online. Therefore, users can avoid protracted contract negotiations and large upfront payments, which aligns with the user of AWS itself.

The rich AWS ecosystem is most valuable aspects of AWS. Quickness of response (agility, in other words) is critical today, for business in general *and* IT in particular. The rich AWS ecosystem fosters agility, and it's an important reason to use AWS.

IT operations are thankless — and endless — tasks. In fact, the term **Sisyphean** may have been coined to describe the eternal job of administering IT resources. Earlier in this module, I outline how AWS makes resource provisioning easier, but AWS also makes ongoing operations simpler.

First, because AWS takes responsibility for much of the traditional IT infrastructure — buildings, power, network, and physical servers, for example — a huge amount of work is taken off the IT plate, and the burden for IT operations shrinks immediately.

"But wait," as the infomercials say, "there's more!" Beyond taking responsibility for the physical infrastructure, AWS also takes on much of the IT administrative burden associated with systems operations. For example, the relational database service (RDS) takes on responsibility for running databases, backing them up, and restarting failed instances to ensure necessary uptime. All these tasks are important, and all of them occupy people's time and attention.

By simplifying IT operations, AWS allows its users to focus on the truly important part of IT: applications. In effect, AWS allows a user to devote more of the IT budget to the qualities that differentiate the business while letting users reduce investment in the important, but nondifferentiating tasks associated with "keeping the lights on."

AWS is organized into regions, and Amazon has regions throughout the world: the United States, Europe, South America, and a number of locations in the Asia Pacific region.
Because AWS is a global service, users can take advantage of a service located nearby, which results in lower network latency and better application performance. The global nature of AWS also leads to local services ecosystems, in the form of native consultancies and system integrators, making it easy for users to obtain help delivered in native languages and with local expertise.

Amazon continues to roll out new regional locations, so you're likely to have access to a nearby service location as well as to the rich AWS ecosystem.

Module-168

Most services, once they gain popularity, inevitably decline based on overwhelmed staff, resource shortages, and competition between users. AWS, on the other hand, is popular, but its popularity has the effect of making the service better. Today, Amazon has a reinforcing cycle occurring:

- Having more users creates a greater volume of use, which increases the amount of hardware Amazon buys, which reduces its costs via economies of scale, which are passed on to users in the form of lower prices.
- Because of the large number of users, companies that offer complementary services (online application integration, for example) decide to place their services in AWS first, which makes the overall service better, which attracts more users.
- As more people and companies use AWS, more knowledge is made available in the form of human capital and other resources. This knowledge makes it easier for new users to get started and to be productive quickly, making AWS more attractive.

Recognize that the AWS status as the largest cloud service provider brings enormous benefits, moreover, those benefits will continue to grow as the service expands.

People recognize that innovation makes life better and that it can improve the future for generations to come. Cloud computing wouldn't existed without the presence of Amazon. All of the incumbent technology market leaders had no incentive to change the way they did business. It took an outsider like Amazon, which had no legacy business to protect, to rethink the way technology is delivered.

AWS has transformed how technology is offered to customers and, as a result, has enabled an explosion of innovation. The innovation and low cost associated with AWS allow small and large companies alike to launch new offerings quickly and inexpensively. As one innovation consultant put it: "AWS has reduced the cost of failure. AWS lets you easily try out a new product to see whether it "gets traction." Moreover, if a new offering gets traction and starts to accelerate, AWS lets you easily scale it up. On the other hand, if the service doesn't achieve adoption, that's no problem — the ease of shutting down AWS resources means that not much is lost if a potential innovative offering doesn't pan out."

The kinds of things that AWS enables range from the useful-but-not-life-changing (Netflix video streaming) to, well, life-changing (enhanced drug discovery via genetic analysis from companies like Eli Lilly).

AWS could be to the information age what Henry Ford's mass production was to the industrial age — and we all know how that turned out!

Commenters who analyze Silicon Valley trends note that the cost of starting an Internet business is now less than 10 percent of what it cost a mere decade ago. Much of that cost reduction is due to AWS: its on-demand low pricing and easy termination with no penalties make it possible to use and pay for exactly as much computing capacity as you need, when you need it.

The cost effectiveness of AWS isn't limited to start-ups, though. Every company can benefit from access to inexpensive computing that doesn't require a lengthy commitment. It's a sign of the powerful benefits of AWS that much of the existing vendor community is terrified of what will happen when their customers begin to demand AWS-like prices and convenience from them.

If you're a part of any company small or large, Amazon can make your IT dollars go further. It's significantly more cost effective than the traditional mode of obtaining IT resources: large up-front payments with little certainty about whether the amount provisioned is too small (or too much).

Every 10 to 15 years, the IT industry is profoundly disrupted by the emergence of a new platform — a new form of computing that changes the way applications are built and used. In the 1980s, the rise of networked PCs (the client-server architecture) transformed mainframes into a legacy environment — and led to Microsoft becoming the dominant player in the software industry. Likewise, in the 1990s the Internet made the web (and the HTTP protocol) the de facto architecture for all applications — and led to the dominance of companies such as Google and, yes, Amazon.

Cloud computing is the next-generation platform for computing. Its characteristics of highly scalable, on-demand computing services that are available within minutes and carrying no requirement for long-term commitment will become the foundation for all future applications. As the saying goes, resistance is futile.

Amazon Web Services, by far the leading cloud computing provider in the industry, is growing at rates of more than 100 percent. Its record of innovation and price competitiveness is unmatched in the industry. Ten years from now, AWS could be the Microsoft or Google of its era. Your organization *must* become familiar with AWS and figure out how to use it effectively — otherwise, it may find itself the IT equivalent of a buggy whip manufacturer after Henry Ford invented the assembly line.

vulearninghub.xyz | by VU Learning HUB

Great careers are built on being the right person in the right place at the right time. Being the right person is all about you — your capacity for hard work, productive work relationships, and intelligence, for example. These characteristics will help you be successful no matter which field or role you work in.

But being in the right place at the right time — that has a lot to do with insight about where a new market, made possible by some type of innovation, is emerging and planting your flag there. People who moved into the automobile industry in the 1920s or into the television business in the 1950s or into the Internet in the 1990s all encountered enormous opportunities as a new market searched for expertise to enable great companies to be built.

Technology innovation creates huge skills gaps in the industry and makes those with knowledge and experience invaluable. If you believe that AWS is the next-generation platform, it too can represent “the right place at the right time” for you.

Module-169

As per CTO Amazon “Everything fails all the time”. IT departments have traditionally attempted to render both infrastructure and applications impervious to failure: A hardware resource or an application component that “fell down on the job” increased the urgency of the search for perfection in order to banish failure. Unfortunately, that search was never successful — the failure of resources and applications has been part of the IT world from the beginning.

Amazon starts from a different perspective, it is the world’s largest online retailer and one of *the* largest web-scale companies worldwide. When you run data centers containing thousands of servers and tens of thousands of disk drives, resource failure is a daily occurrence. And when a hardware resource fails, the software or data residing on that resource suddenly stops working or becomes unavailable.

One cant also rely and continuous functioning of software components or external services; they fail too. An element of a software package configuration or an unforeseen program execution path or an excessive load on an external service means that, even if hardware continues operating properly, portions of an application can fail.

Thus, the single most important cloud application design principle is to accept that perfect system doesn’t exist; failure is a constant companion. Rather than become frustrated by this state of affairs, you should recognize this principle and embrace it. Having recognized that failure is inevitable, be sure to adopt cloud application measures to mitigate circumstances and insulate yourself from failure. The rest of this module is all about insulating yourself from failure.

If you can’t count on individual resources to always work properly, what can you do? The best insurance against resource failure is to use redundant resources, managed in such a way that if a single resource fails, the remaining resource (or resources — you can have more than one additional resource in a redundant design) can seamlessly pick up the workload and continue operating with no interruption.

Amazon has adopted this principle in its AWS offering. Many of its services use redundant resources. For example, every S3 object has three copies, each stored on a single machine. Likewise, the AWS Queue service spreads user queues across multiple machines, using redundancy to maintain availability.

Design your applications to operate with two (or more!) instances at each tier in the application. Every tier should be cross-connected to all instances in any adjacent tier. In this

way, if a resource (either hardware or software) becomes unavailable, the remaining resources can accept all of the application traffic and maintain application availability.
Of course, if resource failure brings your application to a state in which only a single resource is still operating at a given tier, redundancy is no longer protecting you — launch a new resource to ensure that redundancy is retained.

Okay, you recognize the need to protect yourself against resource failure, whether it's hardware or software, and you resolve to use multiple instances to avoid application failure in the event of a server crash, disk breakdown, or even software or service unresponsiveness. But that still doesn't help if a problem occurs at a higher level, such as the entire data center that your application runs going dark from a power outage or natural disaster.

Well, just as you use redundancy at the individual-component level, you use redundancy at the data-center level to avoid this problem. Rather than run your application on multiple instances within a single data center, you run those instances in *different* data centers. Fortunately, Amazon makes it easy with its Regional Availability Zone architecture. Every region has at least two availability zones, which are essentially separate data centers, to provide higher-level redundancy for applications.

Availability zones are located far enough apart to be resistant to natural disasters, so even if one is knocked off the air by a storm or an earthquake, another one remains operating so that you can continue to run your application.

And availability zones are connected by high-speed network connections to ensure that your application's performance doesn't suffer if it spans multiple availability zones.

Redundancy is good, and it's important to avoid a situation in which your application, with redundant resources, becomes non-redundant through the failure of a redundant resource. The question then is, how to *know* when the formerly redundant application is no longer so because of failure? However, this cant be done manually, because this is incredibly boring and it's a huge waste of money.

A much more efficient mode of operation is to have the system *itself* tell you when something fails — a process known as *monitoring.* You set up an automated resource to take the place of a human, and whenever something important happens, it notifies you (*alerts* you, in other words). Automated monitoring has two virtues i.e. computers don't get bored, and you don't pay salaries to computers. Fortunately, AWS offers two excellent services to support automated monitoring:

✓ **CloudWatch:** You can set it up to monitor many AWS resources, including EC2 instances, EBS volumes, SQS queues, and more. CloudWatch is free for certain capabilities, and it's inexpensive for additional capabilities.

✓ **Simple Notification Service (SNS):** It can deliver alerts to you via e-mail, SMS, and even HTTP so that you can publish alerts to a web page. You can easily wire CloudWatch into SNS so that alerts from CloudWatch are automatically and immediately delivered to you, thereby enabling you to take quick action to resolve system deficiencies, including resource failure resulting in a lack of redundancy. Monitoring is a critical companion to redundant application design, and to be integrated into your application from dy-one.

It's an unfortunate fact that many, many AWS users fail to keep track of the resources they use, which can lead to underused, or even *unused,* resources running in AWS.

This problem is significant because AWS resources continue to run up charges, even if the resources aren't performing useful work. Here's the short version of how to avoid:

- Use the AWS Trusted Advisor service or a commercial utilization and cost tracking services like Cloudyn.
- Design your application so that it can have individual resources added or subtracted so that resource utilization rates stay high and resources don't sit around idle or lightly utilized.
- Use AWS EC2 reserved instances to reduce the cost of the computing side of your application.
- Regularly review your AWS bills to see if there are resources or applications being used that you don't know about — and then go find out about them!

Module-170

In the last module "Monitoring Prevents Problems," it was pointed out that, rather than dedicate a person's efforts to monitoring an application 24/7, monitoring and alerts allow the system to track an application's behavior and then notify a human that intervention is required. The drawback to this setup is that you still need a human to implement the intervention.

Wouldn't it be great if no human was required in order to take action, based on the specific situation? The good news is that AWS management systems have this capability. Amazon offers three: CloudWatch, Auto Scaling, and Elastic Beanstalk, and commercial offerings have management capability that extends beyond the type that Amazon itself offers.

Common to all these management systems is a set of monitoring capabilities, along with the ability to execute AWS instructions to perform tasks such as restarting resources when failure occurs or starting and adding resources to an application when the user load increases to the extent of requiring more computing capacity.

As applications become more complex, sophisticated management systems are practically a prerequisite. Trying to track and respond to application issues thrown up by a six- or seven-tiered application that uses a number of AWS services is quite challenging. It makes sense to seek out tools to help reduce the burden.

The number-one concern expressed about cloud computing in general, and AWS in particular, is security. The common questions being how well Amazon manages its data center security measures or to what extent Amazon can prevent its personnel from improperly accessing user systems. (Answers: Nothing can prevent someone from improperly using their administrative permissions, although Amazon has measures in place to monitor improper access.)

First, Amazon does a good job of securing its offering, at least as well as the best in the industry. Second (and this point is crucial), users retain significant responsibility for their application's security when using AWS.

You *must* recognize your security responsibility and take measures to implement and support it. Your application design can help prevent security breaches and potential access to critical data. Here are some guidelines, boiled down to the basics:

✓ **Use multiple security groups to partition your application.** Doing so ensures that malicious actors cannot gain direct access to application logic and data.

✓ **Use Amazon Virtual Private Cloud (VPC) to shield EC2 instances that don't require external access from the Internet.** VPC is an outstanding way to increase application security and will become the default operating environment for AWS, so learn how to use it.
✓ **Implement the specific application security measures.** Patch software packages quickly, implement intrusion prevention software, and manage security keys carefully.

One concern that potential users of AWS often raise is what can be done to prevent inappropriate data access by AWS personnel. The answer is "nothing." The best-designed security systems in the world have too often fallen vulnerable to malicious insiders. Amazon screens its employees, and methodically tracks all employee access to AWS infrastructure, but at least theoretically it is possible for an Amazon employee to access your data, whether on disk or during transit across the AWS network.

Rather than attempt to prevent access to the resources on which your important data is stored or transmitted, follow this approach: Recognize the potential for such access, and make it useless if it occurs i.e. by encryption. With data privately encrypted by the user and available only to those with the private key associated with the data encryption, whoever attempt to access the data — it will appear to be meaningless garbage from the perspective of the intruder. Encryption can be applied in two ways to protect data security certainly as secure as it will be running in your own data center.
:
✓ **Encrypt network traffic.** Network traffic — often referred to as "data in transit" — can easily be encrypted using the Secure Sockets Layer (SSL). SSL ensures that no one can gain useful information from accessing network traffic. This approach can also be used for network traffic across the Internet, preventing outside intruders from accessing network traffic.
✓ **Encrypt data residing on storage.** Data residing on storage is commonly called *data at rest* — it refers to data that's written to and read from disk storage in encrypted fashion. The private keys to access disk data can be held secure on your own premises, preventing access to your data by any Amazon personnel.

A tiered design makes it possible to improve security by partitioning security groups. It may not be as obvious that using a tier-based application design, particularly one that uses redundant, *scalable* tiers (tiers that can grow and shrink by the addition or subtraction of instances to the tier) can also improve the efficiency of your application.

The reason is that tiered, scalable applications can adjust the number of computing resources assigned to an application, growing and shrinking dynamically in response to user load. This approach ensures that all running resources are being used to support user traffic and not

sitting idle. The idea is that these resources should be available for use in case the application load grows sufficiently to require the processing of the idle instance.

Moreover, partitioning your application into tiers allows you to work on improving one portion of it while leaving the rest undisturbed. You can improve the efficiency of the entire application while methodically moving through the tiers, improving performance and reducing resource consumption one tier at a time.

Even if your application begins life as a single instance, with all software packages contained in a single, integrated code base, you should design it so that it may have portions removed and moved to other tiers. This approach supports incremental, gradual improvement to ensure that high resource consumption is reduced over time.

Technical debt refers to a concept in which software code, having been implemented earlier in a project's lifespan, ends up poorly written and not efficient. Technical debt, like its financial counterpart, imposes a cost and hampers efficiency.

The obvious way to reduce technical debt is to periodically revisit and rethink application design and implementation, with an eye toward updating the design and reimplementing important portions of the code.

The most effective method for completing this task is to have all portions of the application designed with an input-and-output interface that defines how an application portion (or package) is called by others and how it calls on other application portions to fulfill their responsibilities. When you use this design approach, different components or portions of an application can be updated or replaced without disturbing the other portions of the application or the overall application itself — as long as the interface "contracts" are adhered to (in other words, the interface operates as advertised).

Updating the functionality of an application as needed is easier when the section that the functionality resides in can be easily modified without disturbing other portions of the application. Without this approach, an application that consists of a large, mingled code base is nearly impossible to modify, if for no other reason than no single software engineer is likely to be able to understand all the different portions of the application design or code to enable you to avoid the dreaded technical debt.

TOPIC-17: Web development frameworks

Module-171

Web frameworks have transformed the world of programming and become vitally important in every development process. Even the smallest unit of an application is comprised of coding, and a web framework simply automates it. You might try browsing different sites, books and articles about it, but find only general and ambiguous information – nothing but endless definitions and difficult terms. Well, it's time to handle this issue and get a clear understanding of web frameworks.

A web framework is a software tool that provides a way to build and run web applications. As a result, you don't need to write code on your own and waste time looking for possible miscalculations and bugs.

In the early days of web development, all applications were hand-coded, and only the developer of a certain app could change or deploy it. Web frameworks introduced a simple way out of this trap. Since 1995, all the hassle connected with changing an application's structure has been put in order because of the appearance of a general performance. And that's when web-specific languages appeared. Their variety is now working well for both static and dynamic web

Types of Web Frameworks

There are two main functions of frameworks: to work on the server side (backend), or on the client-side (frontend), corresponding to their type. This division is not complicated and looks like this:

Frontend frameworks deal mostly with the external part of a web application. Briefly, it's what a user sees when they open the app. The inside stuff is the work of the backend.

Server-side frameworks The rules and architecture of these frameworks allows you to create simple pages, landings and forms of different types. However, in order to build a web application with a well-developed interface, you should have a wider functionality. These frameworks can also form the output data and improve security in case of web attacks. All of these can definitely simplify the development process. Server-side frameworks work mostly on particular but important details without which an application can't work properly.

Client-side frameworks. Unlike the server side, client-side frameworks have nothing to do with business logic. Their work takes place inside the browser. Thus, one can improve and implement new user interfaces. Numerous animated features can be created with frontend

frameworks as well as SPA (single-page applications). Each of the client-side frameworks differs in function and use. For comparison purposes, here they are:

Cross-functional frameworks. Meteor is known as a full-stack web framework. Meaning it covers almost all the needs of both the client and server sides, which makes Meteor extremely popular and widely used. You don't have to waste your time merging two frameworks; you can simply choose this and accelerate your development process.

But this is not the main feature of Meteor. The two sides work in one language, so you can create and use the same code for both. The next thing is "real-time mode" – when a change is made to one interface, it happens to all the others, too. One example is a shareable document or a spreadsheet. When you add some comments to pages you read or edit, other users will also see them.

That is all about type division, but dimensions are important as well. The "size" of different frameworks is also different. There are some "monsters" in the framework world that provide all-in-one solutions.

But some lightweight solutions focus on a narrow specialization; these are called micro-frameworks. These solutions won't provide everything you need, but sometimes it's better to decompose the functionality across several approaches (frameworks, micro-frameworks, libraries). You can extend micro-framework functionality with third-party applications and build some small projects on top of it, or combine micro-frameworks with your main "big" framework.

The architecture of almost all most popular web development frameworks is based on the decomposition of several separate layers (applications, modules, etc.), which means that you can extend functionality according to your requirements and integrate your changes with framework code, or use third-party applications designed by external vendors. This flexibility is another key benefit of frameworks. There are a lot of open-source communities and commercial organizations that produce applications or extensions for popular frameworks e.g., Django REST Framework, ng-bootstrap, etc.).

The MVC – that is, a Model, View and Controller – are the three things each web framework is made of. It is considered to be a basic structure, but there can be several contrasts among them.

The **model** contains all the data and business logic layers, its rules and functions.

The **view**, on the other hand, is responsible for all visual representations of data, like diagrams, charts etc.

As for the **controller**, it simply converts the input data into the scope of commands of the previous ones.

They are inseparable, and it's extremely important to put the process in order to avoid troubles or mistakes while running an application.

Web Caching: Web caching simply helps store different documents and avoids annoying phenomenon of the server overload. Users can use it in various systems if several conditions are met. It also works on the server side. For example, you may notice cached content links on the SERP (Search Engine Results Page) of a search engine like Google.

Scaffolding: This is another important technique to know and use, which is supported by some MVC frameworks. Typical parts of application or the entire project structure (in case of initialization) can be generated by the framework automatically. This approach increases the speed of the development cycle and standardizes the codebase.

Web template system: A web template system is a set of different methodologies and software implemented to construct and deploy web pages. Template engines are used to process web templates. They are a tool for web publishing in a framework.

Security: Online security has plenty of criteria for identifying and permitting or rejecting access to different functions in a web framework. It also helps recognize the profiles that use the application to avoid click jacking. As a result, the framework itself is authentic and authorized.

URL Mapping: If you want to simplify the indexing of your website by search engines while creating a clear and eye-catching site name, this web frameworks' feature is custom-made for it. URL Mapping can also facilitate access to your sites' URLs.

Applications: Numerous types of web applications are supported by web frameworks. The most common and best frameworks for app development support the construction of blogs, forums, general-purpose websites, content management systems, etc.

Module-172

https://gearheart.io/blog/top-10-web-development-frameworks-2019-2020/

Now, there are various forms of web frameworks and models available in the market for both static and dynamic web applications and pages. Form validation is beneficial if people require the user to input data structures that are estimated to meet definite necessities. Depending on your business requirements and goals, you can take the decision of choosing which type of framework is suitable from the below-mentioned example for your web application development. Based on popularity, there are two kinds of web frameworks.

- **Server-side (Backend)**
- **Client-side (Frontend)**

Basically, the inside topics of the application are the work of the backend framework, and what a user sees when they open the application, called front end, which is the external part of a web application framework. In this module list and comparison of web frameworks is given in detail.

https://www.monocubed.com/web-development-framework-comparison/

Django Around 12,000 known web projects were built using Django. This alone can tell you a lot about the framework's popularity. Although it's one of the older website development frameworks, released in 2005, it's still one of the top-picks due to its modern view on problem-solving and constant improvements. Django is arguably the most popular web application framework, based on Python, one of the most used programming language in the world.

Express.js Speed and simplicity are the main principles of Express.js which is a Node.js API and web app development framework. One of the many open-source frameworks, it includes a great number of out-of-the-box tools, and many solutions can be made with just a couple of code lines.

Ruby on Rails framework (RoR), written in the Ruby language, is used today by more than 826,000 live websites, and such companies as Airbnb, YellowPages, Groupon, and many others. This web framework has a wide spectrum of uses, including solving very complicated development problems.

Spring is based on Java, this web application framework is very popular in the world of backend web development. Practically, any professional working with this language will sooner or later use Spring. Many famous companies appreciate the advantages of Spring. Wix, TicketMaster, and Billguard are among them

Symphony is a well-known framework among the community of PHP developers. It significantly reduces the time required for the creation of complicated PHP-based web apps. Here are some of its major features. Symfony framework is appreciated for its stability, high speed, flexibility, and a possibility for code reuse. Also, when it comes to creating high-performing apps, it offers a very convenient event dispatcher together with dependency injection, and possibilities for code optimization. In addition, it consumes a comparatively little amount of memory. However, Symfony is a bit slow for real-time apps.

Angular can be considered the best framework for web applications, and it definitely takes the lead among the products of the Google company for developers. AngularJS, it's predecessor, has bee first released in 2009, and completely rewritten in 2016.

Ember is one of the most entrusted and mature Javascript web dev frameworks. Released in 2011, it has been rapidly growing and gaining more and more influence in the world of professional web development.

IFlutter is absolutely JS-free as it's written in Dart, a programming language also created by Google for developing server-side and web applications for both desktop and mobile platforms. This lets Flutter interact with the platform without passing through the JavaBridge which, in turn, allows you to work way faster than otherwise.

MVP: Minimum Viable Product definition i.e. most initial definition of project

React is not exactly a web app framework, but a JavaScript library. Yet it definitely deserves a spot in this list. React has gained its fame due to the revolutionary component-based architecture that other frameworks began to use much later.

Vue.js is one of the newer frameworks for web development which is growing in popularity very quickly. Its greatest advantage is, if you already have a product, you use Vue.js on its part and everything will function just fine. No lags, no troubles.

DOM: Document Object Model i.e. cross platform language independent interface

TOPIC-18: Database Schema examples

Module-173

Advertising is the form of communication intended to persuade an audience (viewers, readers or listeners) to purchase or take necessary action upon products, ideas or services.

Different types of media can be used to convey these messages, including traditional mass media (newspapers, magazines, television, radio, outdoor, direct mail), or new media (internet and mobile phones).

New media or communication through internet have changed the pace, style and characteristics of communication.

Online or internet advertising is a method of promotion that uses the Internet and World Wide Web to deliver marketing messages to create interest among customers. Examples of online advertising include contextual ads on search engine result pages, banner ads, rich media ads, Social network advertising, blogs, online classified advertising, advertising networks and e-mail marketing, including e-mail spam.

Early Years: First online advertising was initiated when Hot-Wired signed up fourteen advertisers for its online debut (October 27, 1994). After this initiation we saw the emergence and public acceptance of the Web as an interactive medium in the following years.

Growth: The year 1994 saw the first online advertisement that was quickly followed by a period of research on advertiser and publisher ad formats and technology. In late 1990s, billions of money were invested in online advertising.

Current Scenario: Banner ads today, as they were more than a decade ago, are no more effective online advertising medium. Online advertising has been constantly rising since 2004. With the number of hours an internet user spends browsing websites, advertisers have realized the significance and advantage of manipulating user tendency to scour the web.

From SEO marketing, blogs and social media to stylish ads, interactive tools and branding technologies, advertisers are now using a wide array of platforms to increase business visibility

Top-5 Advantages

1) Wider Coverage The online advertising gives advertisements a wider coverage and this globally wider coverage helps in making advertisements reach more audiences, which may ultimately help you in getting better results through online advertising campaign.

2) Affordable Another main advantage of online advertising or marketing is cost effectiveness. It is much more affordable compared with the traditional advertising costs. With a much lesser cost advertiser can advertise on the net for a wider range of audience.

3) Informative In online advertising, the advertiser is able to convey more details about the product/service to the audience and that too at relatively lower cost. Most of the online advertising campaigns are composed of a clickable link to a detailed landing page, where users get more information about the product mentioned in the advertisement.

4) Flexible Payment Payment flexibility is another added advantage of online advertising and marketing. In offline advertising advertisers need to pay the full amount to the advertising agency irrespective of the results. In case of online advertising, advertisers get flexibility of paying for only qualified leads, clicks or impressions.

5) Easy Audience Engagement Online advertising makes it easy for the audience to engage with the ads or products. It helps advertiser to get more feedback from the audience and thereby improve the quality of ads going forward.

Top-5 Disadvantages

1) Customers Ignore Ads Consumers have developed an antipathy to all forms of advertising. This is also the case with online advertising, where consumers can avoid clicking banner advertisements, can bypass ads in online videos close pop-up advertisements as soon as they come up on their screens. Customers are in total control of which advertising messages they want to click and respond to.

2) Viewing Problems/Web snarls Website downtime, lags in website or video loading and browser complications can reduce the number of times consumers see online advertisements and how well they see them. With these technical issues, companies lose the chance to broadcast advertisements for their products or services.

3) Expensive Ad Prices Pricing for advertising online can range from inexpensive to highly expensive ads. The cost for different types online ads vary depending on the amount of traffic and the type of readership for a particular website. Online advertising through payper-click campaigns and social media sites can also create disorder on a company's marketing budget, potentially yielding little to no return on investment.

4) Consumers Get Distracted/Lack of Interest When customers visit a website, they typically have an objective in mind. Websites extend customers with various options that can easily distract them and pull their attention from online advertisements.

5) Too Many Options Internet offers millions of websites on which companies can place their advertisements. It can be overwhelming, especially for small business holders. With this wide range of options, it is very difficult to narrow down the choices to the websites that will attract the most potential customers and sales.

4 Basic Types

1) Sponsorships Traditional forms of advertising. Essentially, an advertiser will sponsor an entire site, or at least a section of a site or a specific page on a site. With this type, it is most usual for the sponsorship to be exclusive for a certain period of time.

2) Pop-up and Pop-under Ads The idea to this ad is borrowed from television. This form of advertisement opens in a separate window when a web page is loading. The more times people click on these ads, the more money can be charged. The future of pop ads must however be considered, when a recent study showed that people mostly find these ads annoying.

3) Flash/DHTML Ads These kinds of advertisements incorporate Flash animation or other moving graphics. This may be animated display ads in more traditional shapes and sizes, or, as of late, they can be stylish ads that function similarly to pop-up ads but with much deeper integration into the overall design of the site.

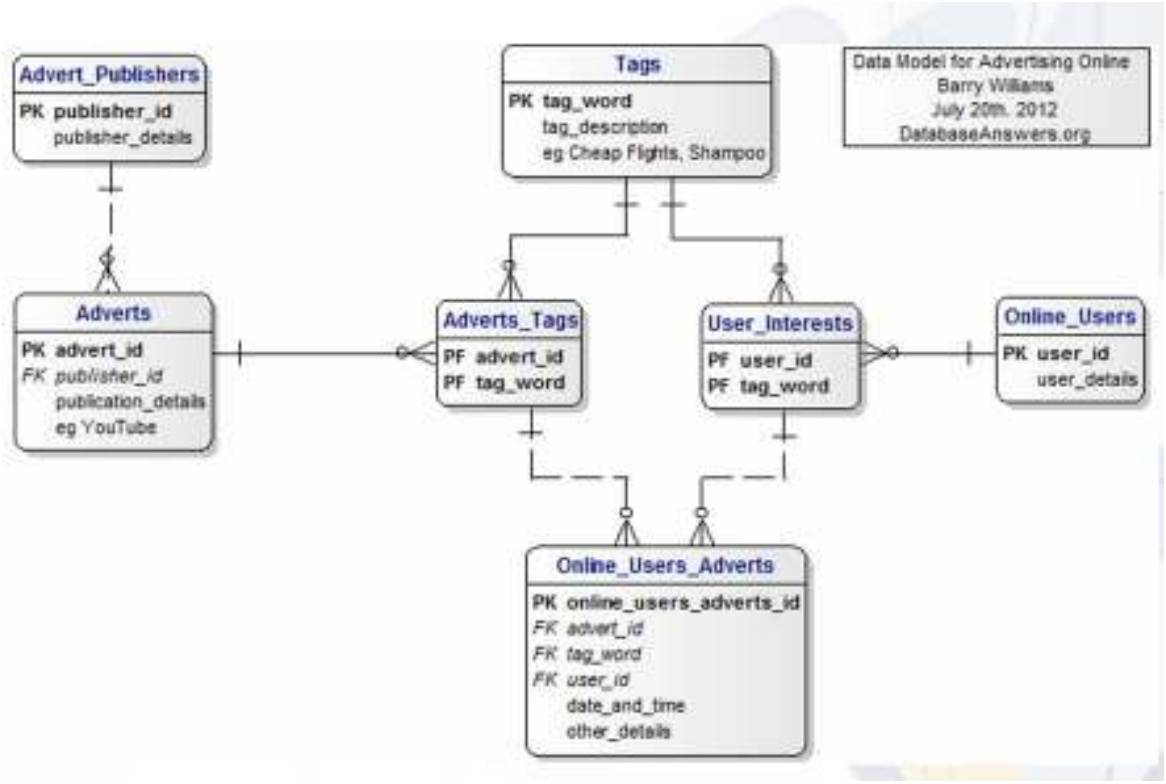
4) Affiliate Ads Affiliate advertising is based on the idea of an advertiser paying a publisher (the affiliate) for any business that is brought in. Normally, the publisher will run ads for an affiliate with special tracking code that helps the advertiser to identify which website a visitor came from.

5) Pay-per-click Ads Most commonly, this type of advertising is associated with search engines and contextual advertising. Generally, the advertiser pays out for each click on an ad. Commonly these ads will be text links, and will be shown either as portion of a search results page, or based on the content of a website.

6) Search Engine Optimization This is type of online advertising service provided by many web media companies. They will look at the target audience, competitors and the keywords for business and optimize advertiser's website content.

Module-174

Logical data model



Module-175

Airline reservation systems are business critical applications, and they are functionally quite complex, the operation of an in-house airline reservation system is relatively expensive.

Prior to deregulation, airlines owned their own reservation systems with travel agents subscribing to them. Today, the GDS are run by independent companies with airlines and travel agencies being major subscribers.

The industry is at 98% electronic ticket issuance today, although electronic processing for MCOs was not available in time for the IATA mandate.

Airline reservation systems incorporate airline schedules, fare tariffs, passenger reservations and ticket records. An airline's direct distribution works within their own reservation system, as well as pushing out information to the GDS. The second type of direct distribution channel are consumers who use the internet or mobile applications to make their own reservations. Travel agencies and other indirect distribution channels access the same GDS as those accessed by the airline reservation systems.

Reservation systems may host "ticket-less" airlines and "hybrid" airlines that use e-ticketing in addition to ticket-less to accommodate code-shares and interlines.

In addition to these "standardized" GDS, some airlines have proprietary versions which they use to run their flight operations. A few examples are Delta's OSS and Deltamatic systems and EDS SHARES.

In the airline industry, available seats are commonly referred to as inventory. The inventory of an airline is generally classified into service classes (e.g. first, business or economy class) and up to 26 booking classes, for which different prices and booking conditions apply. Inventory data is imported and maintained through a schedule distribution system over standardized interfaces.

One of the core functions of inventory management is inventory control. Inventory control steers how many seats are available in the different booking classes, by opening and closing individual booking classes for sale. In combination with the fares and booking conditions stored in the Fare Quote System, the price for each sold seat is determined.

In most cases, inventory control has a real time interface to an airline's Yield management system to support a permanent optimization of the offered booking classes in response to changes in demand or pricing strategies of a competitor.

Users access an airline's inventory through an availability display. It contains all offered flights for a particular city-pair with their available seats in the different booking classes. This display contains flights which are operated by the airline itself as well as code share flights which are

operated in co-operation with another airline. If the city pair is not one on which the airline offers service, it may display a connection using its own flights or display the flights of other airlines.

The availability of seats of other airlines is updated through standard industry interfaces. Reservations for individual passengers or groups are stored in a so-called passenger name record (PNR).

Among other data, the PNR contains personal information such as name, contact information or special services requests (SSRs) e.g. for a vegetarian meal, as well as the flights (segments) and issued tickets. Some reservation systems also allow to store customer data in profiles to avoid data re-entry each time a new reservation is made for a known passenger. In addition, most systems have interfaces to CRM systems or customer loyalty applications (aka frequent traveler systems).

Once a flight has departed, the reservation system is updated with a list of the checked-in passengers (e.g. passengers who had a reservation but did not check in (no shows) and passengers who checked in, but did not have a reservation (go shows)). Finally, data needed for revenue accounting and reporting is handed over to administrative systems.

The Fares data store contains fare tariffs, rule sets, routing maps, class of service tables, and some tax information that construct the price – "the fare". Rules like booking conditions (e.g. minimum stay, advance purchase, etc.) are tailored differently between different city pairs or zones, and assigned a class of service corresponding to its appropriate inventory bucket.

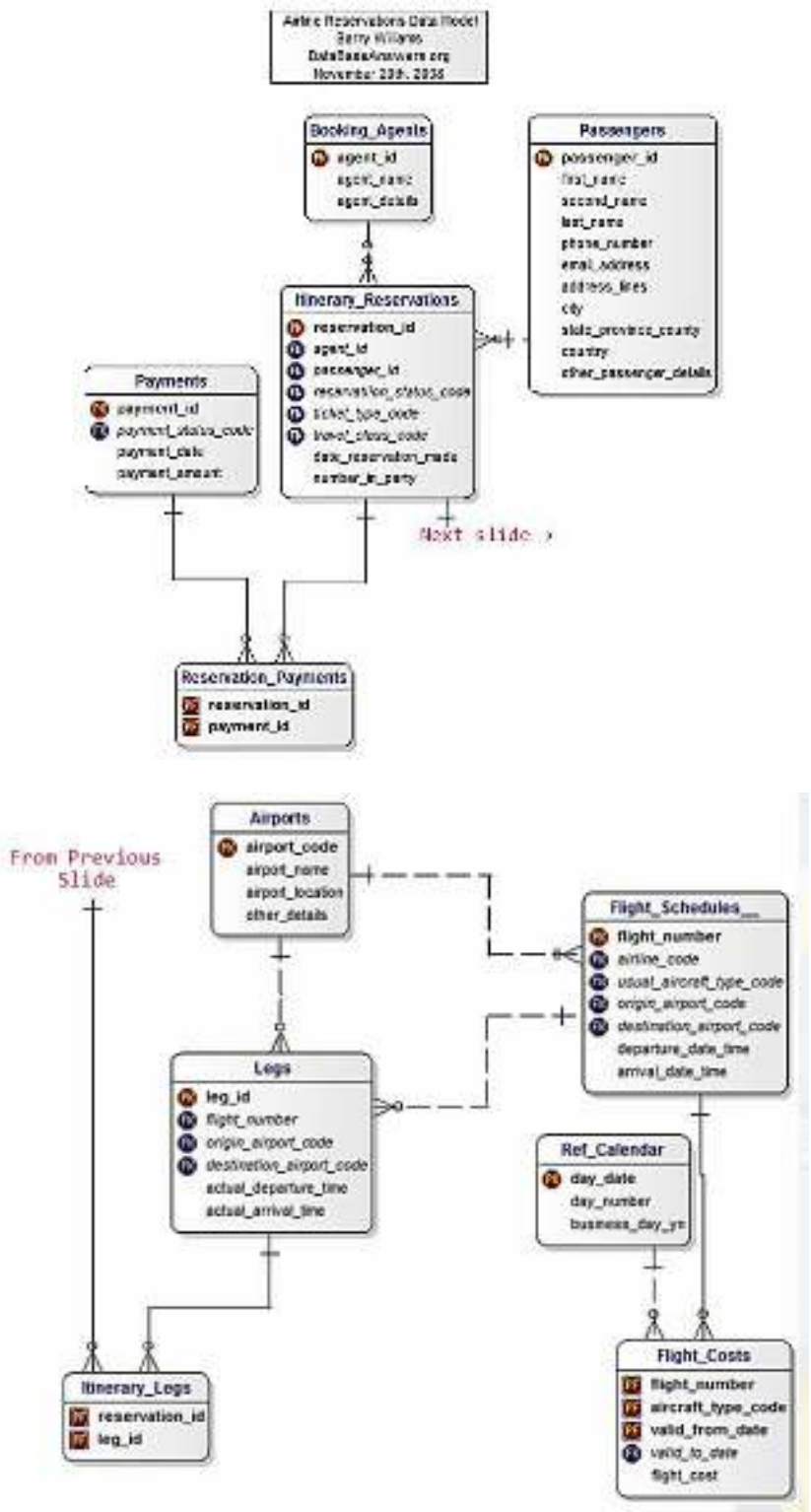
Inventory control can also be manipulated manually through the availability feeds, dynamically controlling how many seats are offered for a particular price by opening and closing particular classes.

The compiled set of fare conditions is called a fare basis code. Every airline employs staff who code air fare rules in accordance with yield management intent. There are also revenue managers who watch fares as they are filed into the public tariffs and make competitive recommendations. Inventory control is typically manipulated from here, using availability feeds to open and close classes of service.

The role of the ticketing complex is to issue and store electronic ticket records and the very small number of paper tickets that are still issued. The electronic ticket information is stored in a database containing the data such as the ticket number, the fare and tax components of the ticket price or exchange rate information. The industry is at 98% electronic ticket issuance today, although electronic processing for MCOs was not available in time for the IATA mandate.

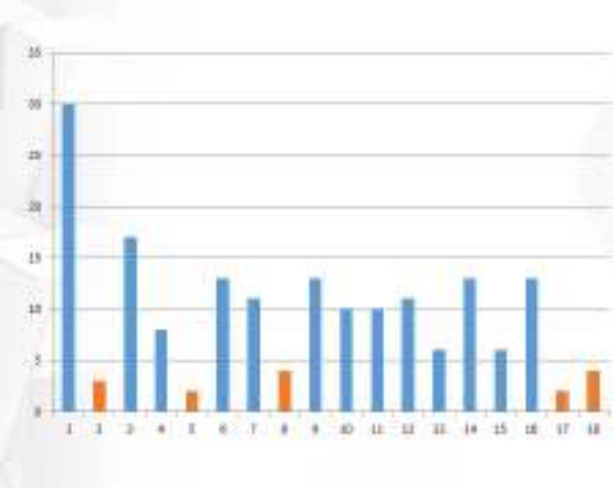
Module-176

Logical data model



Module-177

About the course

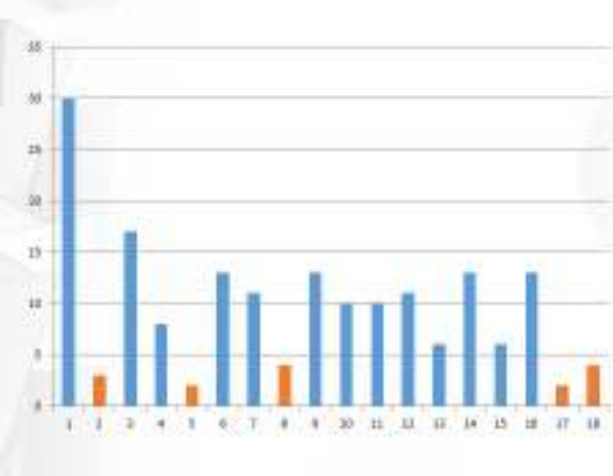


Topics

1. Introduction
2. Web application vs. Web site
3. Building E-Commerce SMB
4. How databases work on web?
5. Database driven website
6. Understanding database schemas
7. Web database connectivity
8. Web database operations
9. Rapid Application Development (RAD)

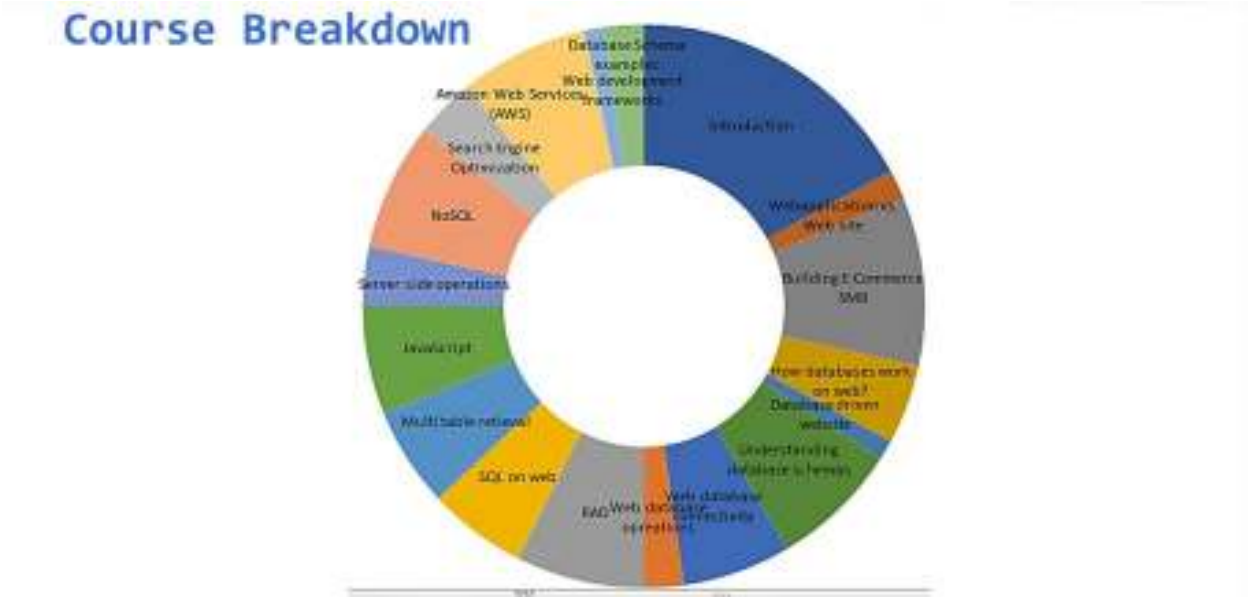
Topics

1. Introduction
2. Web application vs. Web site
3. Building E-Commerce SMB
4. How databases work on web?
5. Database driven website
6. Understanding database schemas
7. Web database connectivity
8. Web database operations
9. Rapid Application Development (RAD)



Topics

10. SQL on web
11. Multi table retrieval
12. JavaScript
13. Server side operations
14. NoSQL
15. Search Engine Optimization
16. Amazon Web Services (AWS)
17. Web development frameworks
18. Database Schema examples


Course Breakdown
Database Schema examples
Web development frameworks
Amazon Web Services (AWS)
Search Engine Optimization
NoSQL
Server side operations
Javascript
Multitable retrieval
SQL on web
RAD
Web database operations
Web database connectivity
Understanding database schemas
Database driven website
How databases work on web?
Building E Commerce SMB
Web site
Introduction