Ali Khan 03334444071

CS403 Database Management System as Favorite Subject for Viva Preparation

Database Management System or DBMS in short refers to the technology of storing and retrieving users' data with utmost efficiency along with appropriate security measures. This tutorial explains the basics of DBMS such as its architecture, data models, data schemas, data independence, E-R model, relation model, relational database design, and storage and file structure and much more.

Database Management System: is the set of programs or system which is used to create and maintain database.

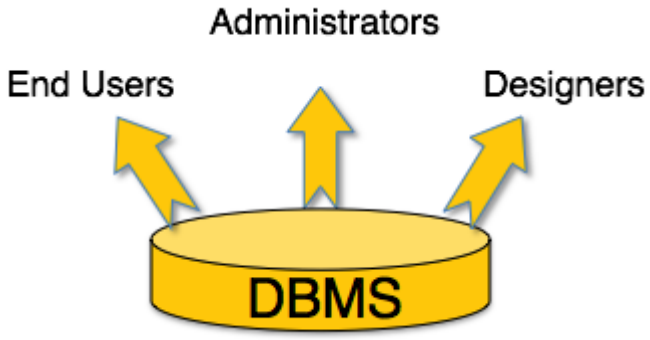
Data: is the collection of raw, facts and figures like college admission form consists of data.

Database: is organized collection of related data.

A database management system stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

Users

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 –



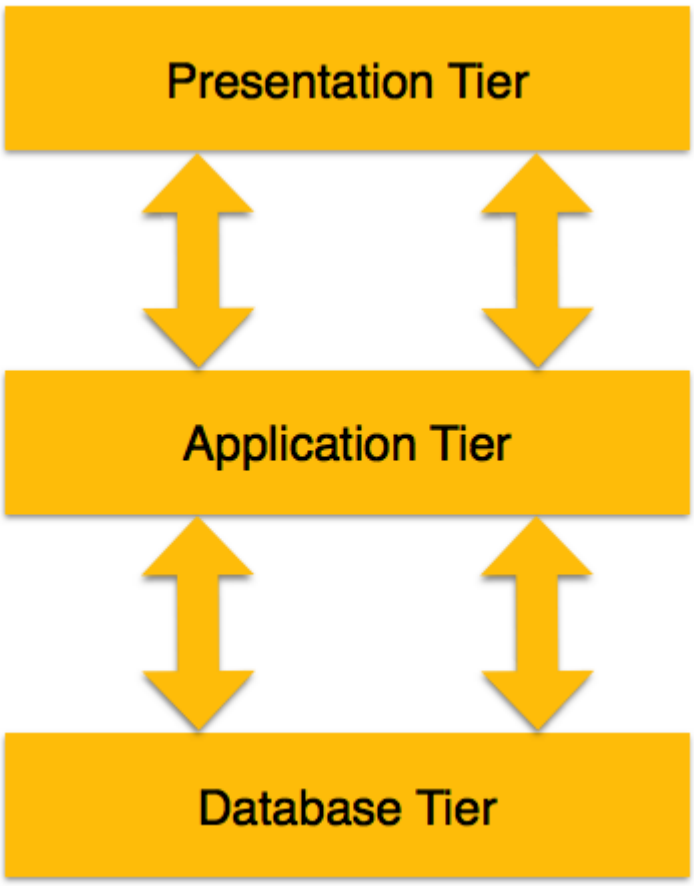
- **Administrators** – Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain isolation and force security. Administrators 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.

3-tier Architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.



- **Database (Data) Tier** – At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.

- **Application (Middle) Tier** – At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.

- **User (Presentation) Tier** – End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Entity-Relationship Model

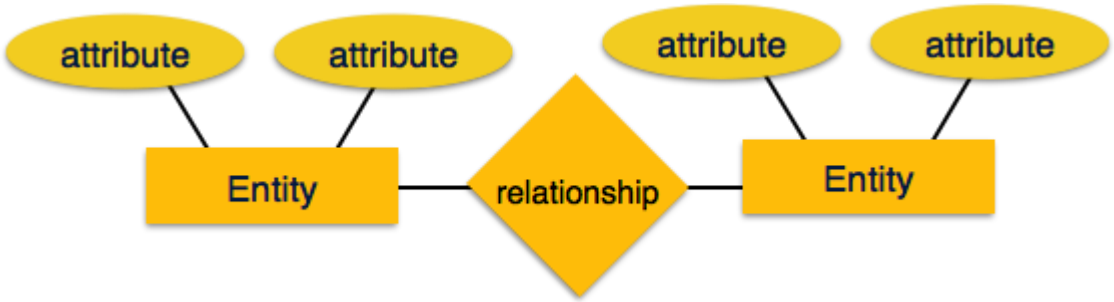
Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of a database.

ER Model is based on –

- **Entities** and their *attributes.*
- **Relationships** among entities.

These concepts are explained below.



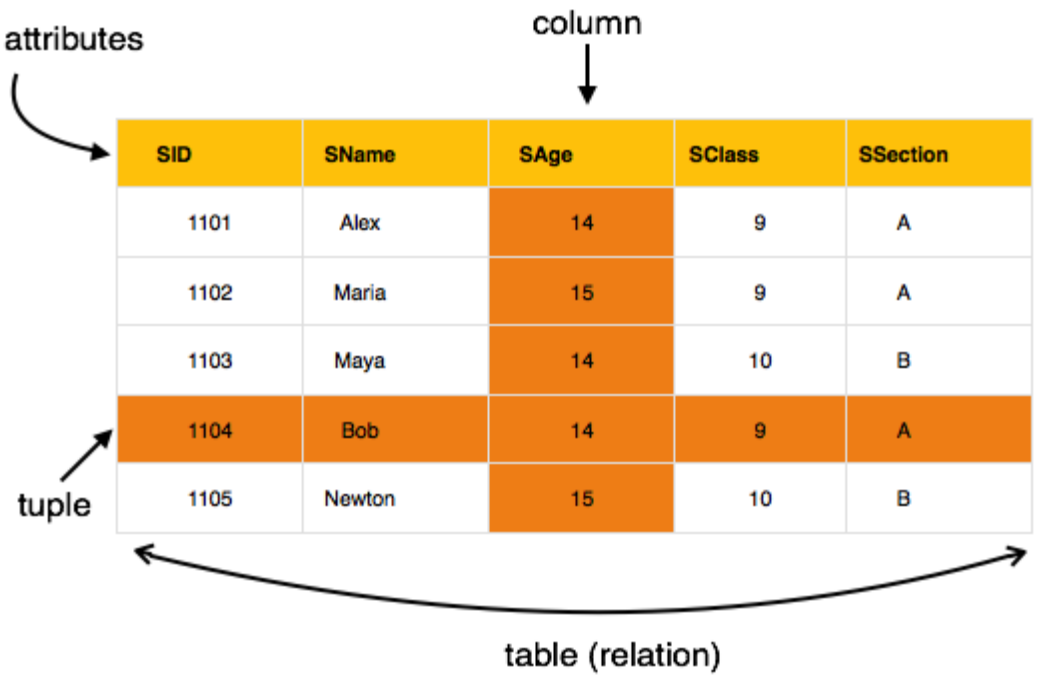
- **Entity** – An entity in an ER Model is a real-world entity having properties called **attributes**. Every **attribute** is defined by its set of values called **domain**. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.
- **Relationship** – The logical association among entities is called***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

  Mapping cardinalities –

  - one to one
  - one to many
  - many to one
  - many to many

Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.



SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

The main highlights of this model are –

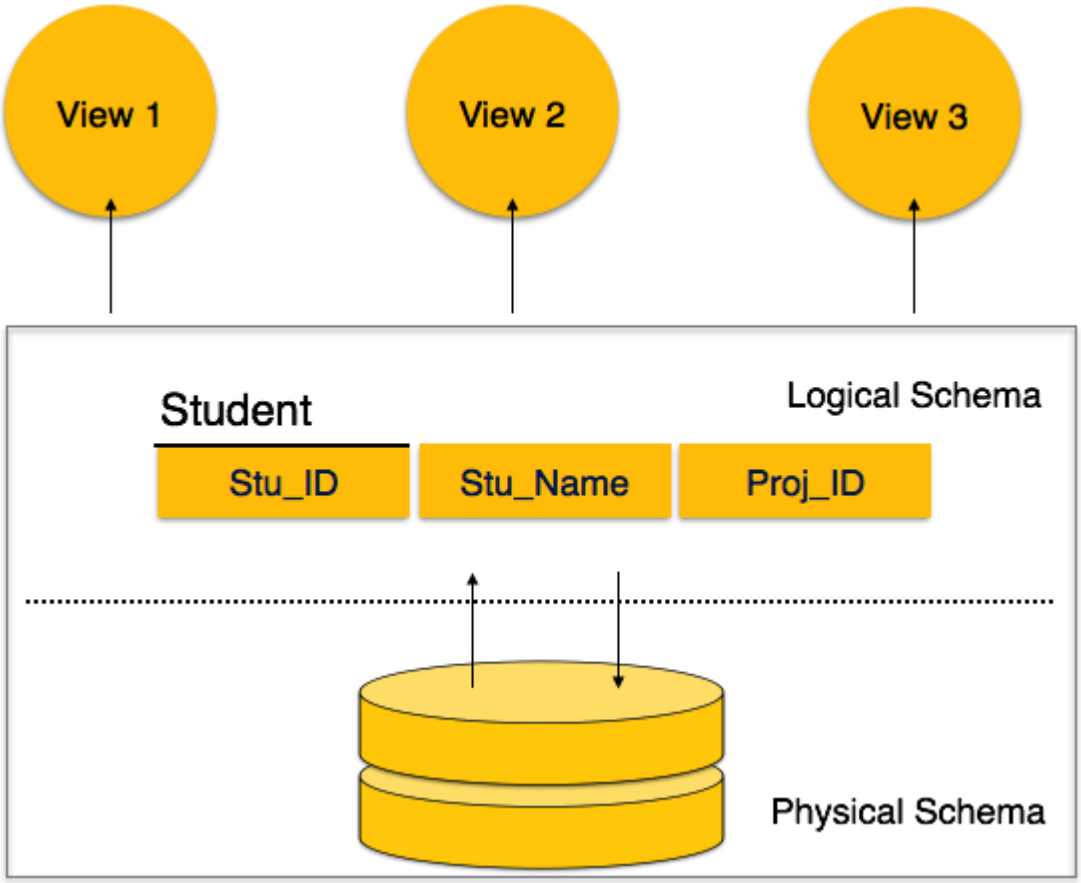
- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.

Database Schema

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by

means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.



A database schema can be divided broadly into two categories –

- **Physical Database Schema** – This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.

- **Logical Database Schema** – This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

ER Model - Basic Concepts

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

- **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 –

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- 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.

Keys

A key is a set of attributes that can be used to identify or access a particular entity instance of an entity type (or out of an entity set).

A key can be simple, that is, consisting of single attribute, or it could be composite which consists of two or more attributes.

Super Key:

A super key is a set of one or more attributes which taken collectively, allow us to identify uniquely an entity instance in the entity set.

Super Key is defined as a set of attributes with in a table that uniquely identifies each record within a table. Super Key is a superset of Candidate key.

An attribute or a combination of attribute that is used to identify the records uniquely is known as Super Key. A table can have many Super Keys.

A combination of one or more columns in a table which can be used to identify a record in a table uniquely, a table can have any number of super keys.

A Super key is any combination of fields within a table that uniquely identifies each record within that table.

Super key is a set of one or more than one keys that can be used to identify a record uniquely in a table. Example: Primary key, Unique key, Alternate key are subset of Super Keys.

Candidate Key:

A candidate key is a super key that contains no extra attribute. It consists of minimum possible attributes.

A candidate is a subset of a super key. A candidate key is a single field or the least combination of fields that uniquely identifies each record in the table.

A Candidate Key is a set of one or more fields/columns that can identify a record uniquely in a table. There can be multiple Candidate Keys in one table. Each Candidate Key can work as Primary Key.

Primary Key:

It is the key selected by the database designer for unique identification.

Primary Key is a candidate key that is most appropriate to become main key of the table. It is a key that uniquely identify each record in a table.

Primary key must hold a unique value for each record.

A Candidate Key that is used by the database designer for unique identification of each row in a table is known as Primary Key. A Primary Key can consist of one or more attributes of a table.

Primary key is a set of one or more fields/columns of a table that uniquely identify a record in database table. It can not accept null, duplicate values. Only one Candidate Key can be Primary Key.

Alternate Key

The candidate keys that are not selected as primary key are known as alternate keys.

Candidate keys which are not chosen as the primary key are known as alternate keys.

Alternate Key can be any of the Candidate Keys except for the Primary Key.

E.g. of Alternate Key is “Name, Address” as it is the only other Candidate Key which is not a Primary Key.

We cannot define the Alternate Key Separately from a Candidate Key, for a table, if there are two Candidate Keys and one is chosen as a Primary Key the other Candidate Key is known as the Alternate Key of that table.

An Alternate key is a key that can be work as a primary key. Basically it is a candidate key that currently is not primary key.

Composite Key

A primary key that consists of two or more attributes is known as composite key.

Key that consists of two or more attributes that uniquely identify an entity occurrence is called Composite key.

Composite key, a key that is composed of two or more attributes.

If we use multiple attributes to create a Primary Key then that Primary Key is called Composite Key.

Composite Key is a combination of more than one fields/columns of a table. It can be a Candidate key, Primary key.

Foreign Key

A foreign key is an attribute or set of attributes in a relation whose values match a primary key in another relation.

A Foreign Key accesses data in some other related table via its Primary Key.

One or more attributes in an entity type that represents a key, either primary or secondary, in another entity type.

A foreign key is an attribute or combination of attribute in one base table that points to the candidate key (generally it is the primary key) of another table.

Foreign Key is a field in database table that is Primary key in another table. It can accept multiple null, duplicate values.

A column of one table points to the Primary Key column of another table to implement referential data integrity.

A foreign key consists of one or more columns in a table whose value in one row uniquely identifies another row in the same or another table.

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

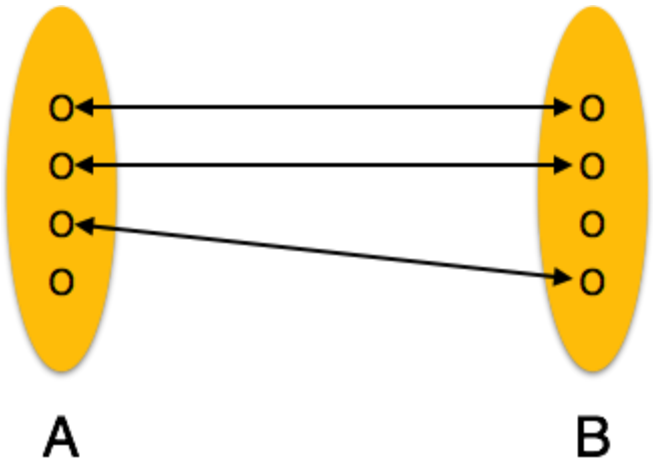
The number of participating entities in a relationship defines the degree of the relationship.

- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

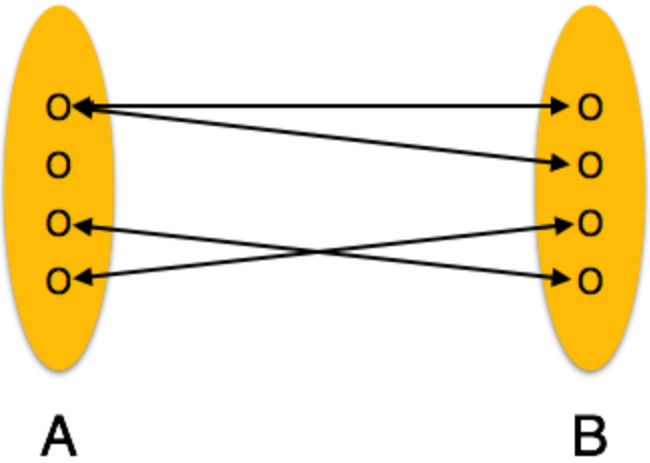
Mapping Cardinalities

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.

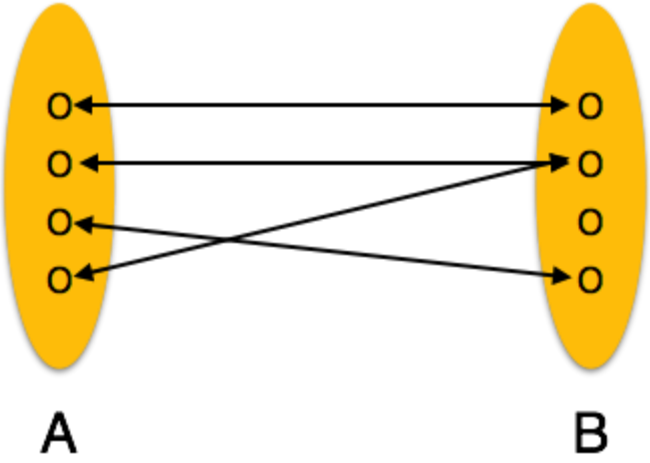
- **One-to-one** – One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



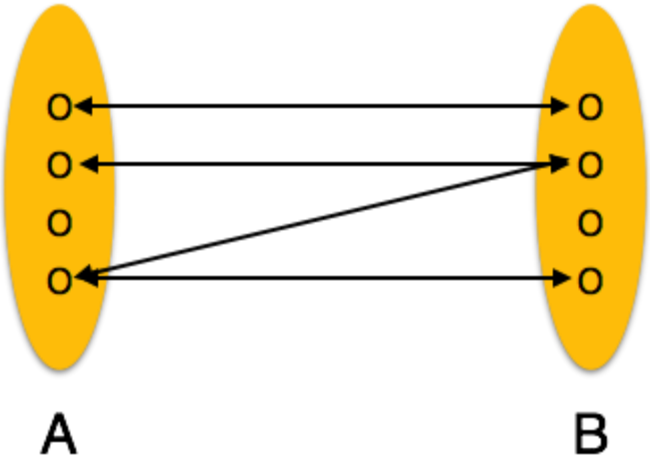
- **One-to-many** – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



- **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.



- **Many-to-many** – One entity from A can be associated with more than one entity from B and vice versa.



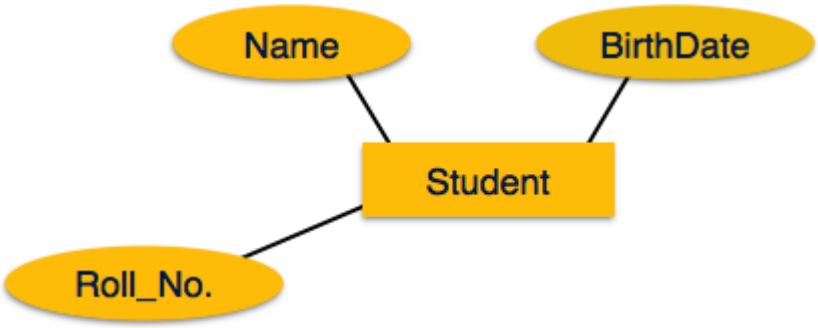
Entity

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

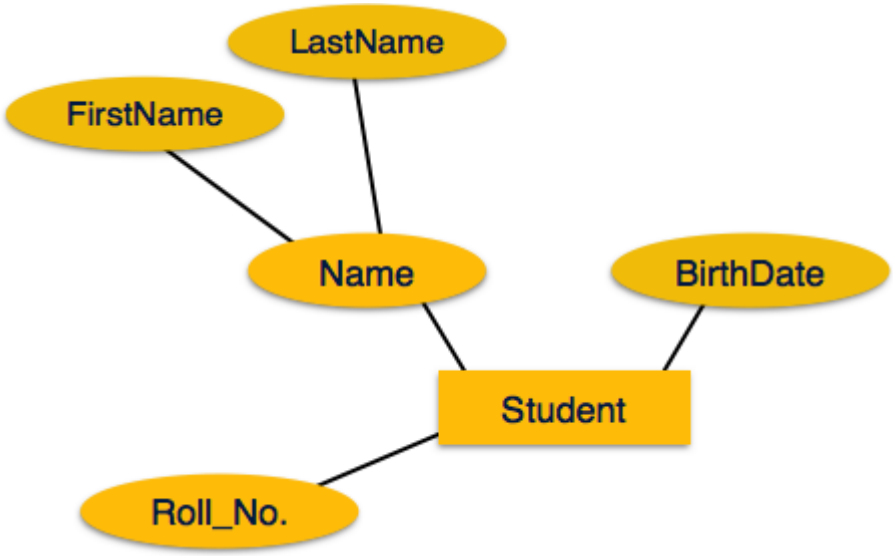


Attributes

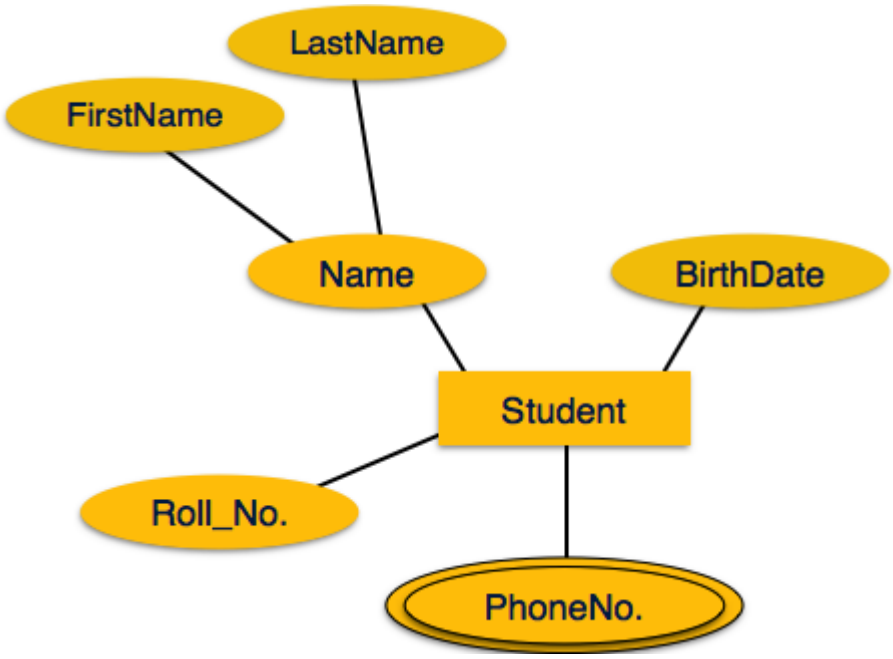
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



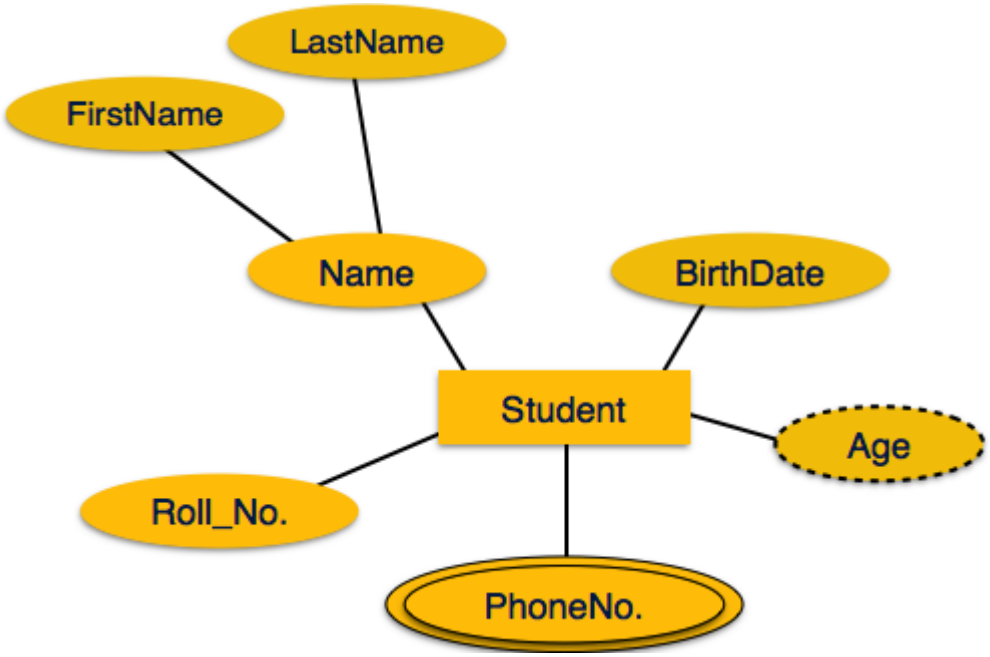
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multivalued attributes are depicted by double ellipse.



Derived attributes are depicted by dashed ellipse.



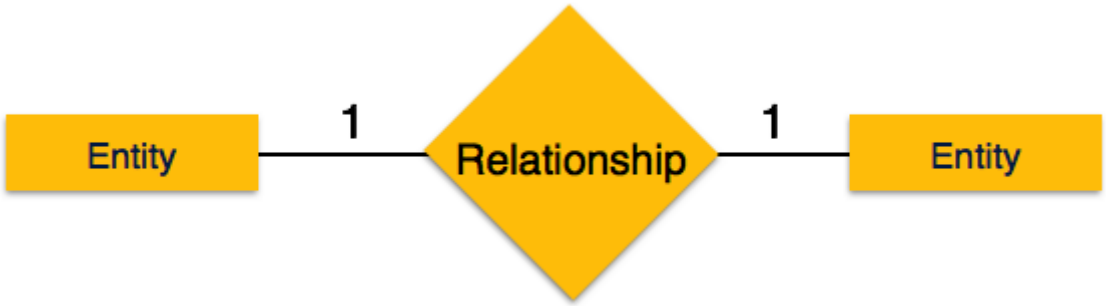
Relationship

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.

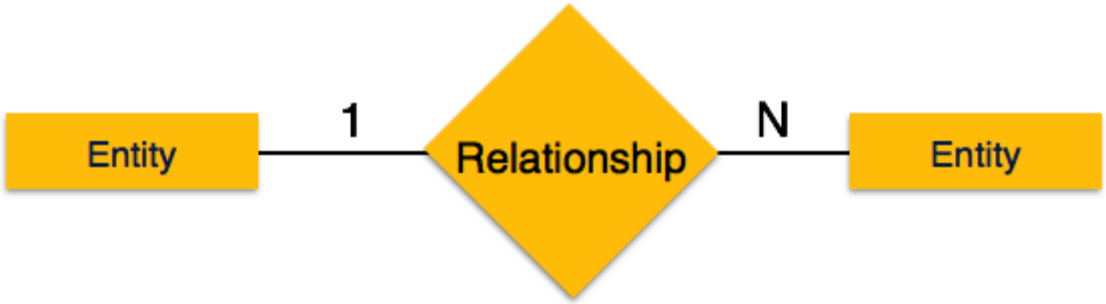
Binary Relationship and Cardinality

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.

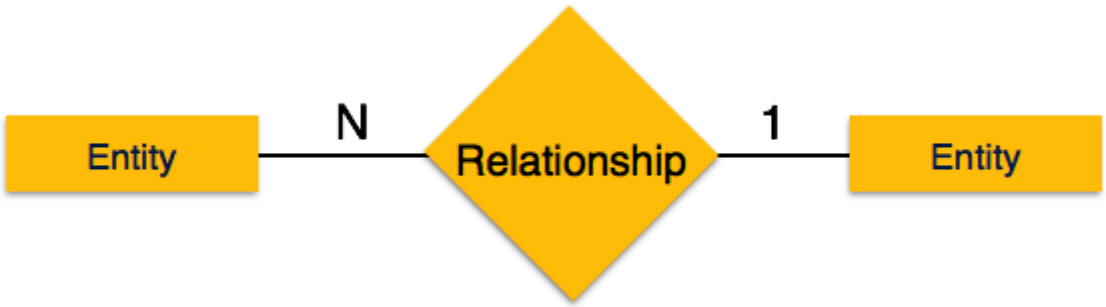
- **One-to-one** – When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



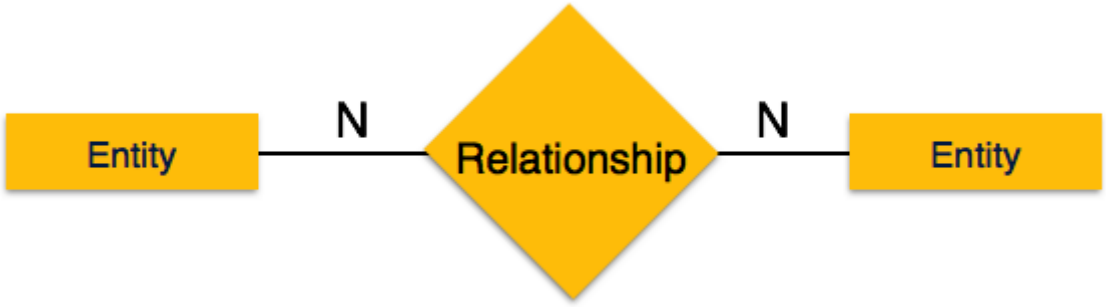
- **One-to-many** – When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



- **Many-to-one** – When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects 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. It depicts many-to-one relationship.



- **Many-to-many** – The following image reflects that more 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. It depicts many-to-many relationship.



Relation Data Model

Concepts

Tables – In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Tuple – A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance – A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema – A relation schema describes the relation name (table name), attributes, and their names.

Relation key – Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain – Every attribute has some pre-defined value scope, known as attribute domain.

SQL Overview

SQL is a programming language for Relational Databases. It is designed over relational algebra and tuple relational calculus. SQL comes as a package with all major distributions of RDBMS.

SQL comprises both data definition and data manipulation languages. Using the data definition properties of SQL, one can design and modify database schema, whereas data manipulation properties allows SQL to store and retrieve data from database.

Data Definition Language

The DDL section is used for creating database objects, such as tables. In practice, people often use a GUI for creating tables and so on, so it is less common to hand-write DDL statements than it used to be.

SQL uses the following set of commands to define database schema –

CREATE

Creates new databases, tables and views from RDBMS.

For example –

```
Create database tutorialspoint;
Create table article;
Create view for_students;
```

DROP

Drops commands, views, tables, and databases from RDBMS.

For example–

```
Drop object_type object_name;
Drop database tutorialspoint;
Drop table article;
Drop view for_students;
```

ALTER

Modifies database schema.

```
Alter object_type object_name parameters;
```

For example–

```
Alter table article add subject varchar;
```

This command adds an attribute in the relation **article** with the name **subject** of string type.

Data Manipulation Language

The DML section is used to manipulate the data such as querying it. While is also common to use a query builder to create queries, people do still hand-craft DML statements, such as queries.

SQL is equipped with data manipulation language (DML). DML modifies the database instance by inserting, updating and deleting its data. DML is responsible for all froms data modification in a database. SQL contains the following set of commands in its DML section –

- SELECT/FROM/WHERE
- INSERT INTO/VALUES
- UPDATE/SET/WHERE
- DELETE FROM/WHERE

These basic constructs allow database programmers and users to enter data and information into the database and retrieve efficiently using a number of filter options.

SELECT/FROM/WHERE

- **SELECT** – This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.

- **FROM** – This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.

- **WHERE** – This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.

For example –

```
Select author_name
From book_author
Where age > 50;
```

This command will yield the names of authors from the relation **book_author**whose age is greater than 50.

INSERT INTO/VALUES

This command is used for inserting values into the rows of a table (relation).

Syntax–

```
INSERT INTO table (column1 [, column2, column3 ... ]) VALUES (value1 [, value2, value3 ... ])
```

Or

```
INSERT INTO table VALUES (value1, [value2, ... ])
```

For example –

```
INSERT INTO tutorialspoint (Author, Subject) VALUES ("anonymous", "computers");
```

UPDATE/SET/WHERE

This command is used for updating or modifying the values of columns in a table (relation).

Syntax –

```
UPDATE table_name SET column_name = value [, column_name = value ...] [WHERE condition]
```

For example –

```
UPDATE tutorialspoint SET Author="webmaster" WHERE Author="anonymous";
```

DELETE/FROM/WHERE

This command is used for removing one or more rows from a table (relation).

Syntax –

```
DELETE FROM table_name [WHERE condition];
```

For example –

```
DELETE FROM tutorialspoints
   WHERE Author="unknown";
```

SQL Commands:

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

DDL - Data Definition Language:

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DML - Data Manipulation Language:

Command	Description
SELECT	Retrieves certain records from one or more tables
INSERT	Creates a record

UPDATE	Modifies records
DELETE	Deletes records

DCL - Data Control Language:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

CRUD

C stand for create

R stand for read

U stand for update

D stand for delete

SQL stand for standard query language

To add new rows to table.... insert query

To retrieve rows from tables in a database.... select query (Retrieve means to pick anything)

To modify the rows of table....update query

To delete rows from table... delete query

Normalization

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies** – If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- **Deletion anomalies** – We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** – We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

First Normal Form

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

We re-arrange the relation (table) as below, to convert it to First Normal Form.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

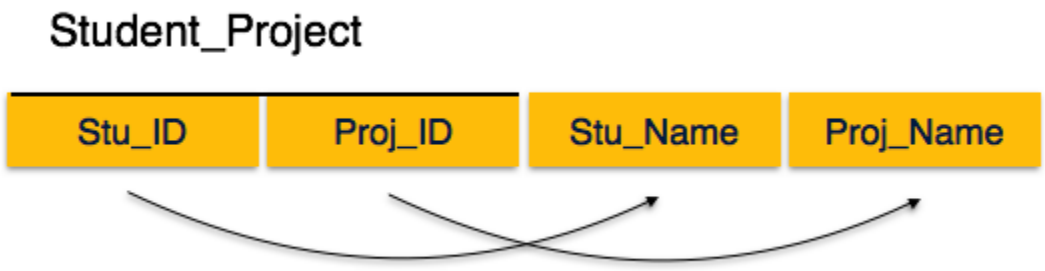
Each attribute must contain only a single value from its pre-defined domain.

Second Normal Form

Before we learn about the second normal form, we need to understand the following –

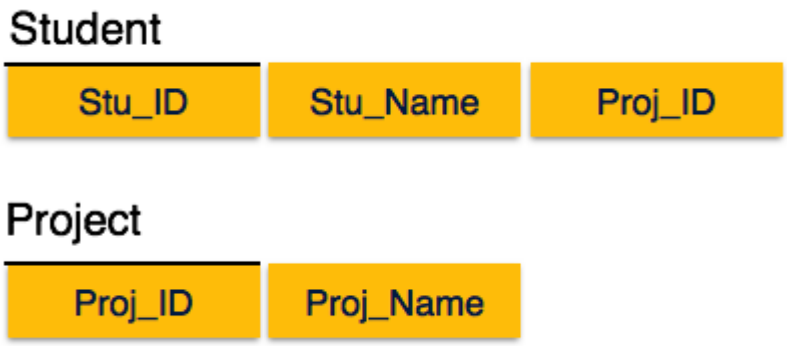
- **Prime attribute** – An attribute, which is a part of the prime-key, is known as a prime attribute.
- **Non-prime attribute** – An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of

the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called**partial dependency**, which is not allowed in Second Normal Form.



We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

Third Normal Form

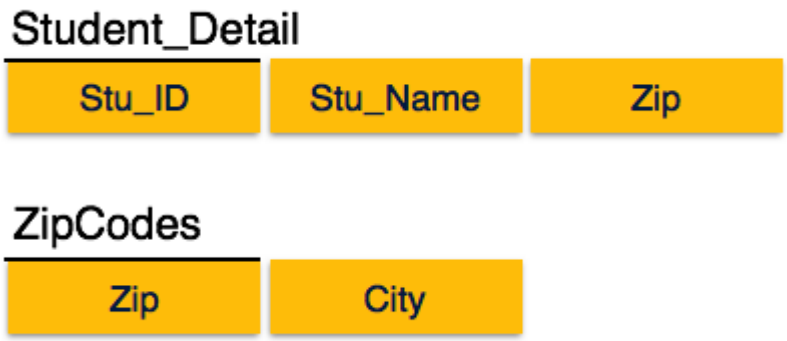
For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy –

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either –
    - X is a superkey or,
    - A is prime attribute.



We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists **transitive dependency**.

To bring this relation into third normal form, we break the relation into two relations as follows –



Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that –

- For any non-trivial functional dependency, X → A, X must be a super-key.

In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,

Stu_ID → Stu_Name, Zip

and

Zip → City

Which confirms that both the relations are in BCNF.

Joins

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

We will briefly describe various join types in the following sections.

Theta (θ) Join

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol **θ**.

Notation

```
R1 ⋈θ R2
```

R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don't have anything in common, that is R1 ∩ R2 = Φ.

Equijoin

When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

Natural Join (⋈)

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Outer Joins

Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins – left outer join, right outer join, and full outer join.

Left Outer Join(R ⟕ S)

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Right Outer Join: (R ⟖ S)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Full Outer Join: (R ⟗ S)

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Indexing

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types –

- **Primary Index** – Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** – Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- **Clustering Index** – Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Difference between Drop, Truncate and Delete

DELETE

1. DELETE is a DML statement.

2. DELETE removes some rows if WHERE clause is used

3. Can be rolled back

4. Can be used with or without WHERE clause

5. Does not reset identity of the table

6. Triggers will be fired.

7. When DELETE operation is performed, all the data get copied into Rollback Tablespace first,

and then delete operation get performed. Hence we can get back the data by ROLLBACK command.

SYNTAX:

To delete a particular row

DELETE FROM table_name

WHERE column_name = column_value

To delete all rows

DELETE FROM table_name

Or

DELETE * FROM table_name

DROP

1. DROP is a DDL statement.

2. Removes a table from the database. Table structures, indexes, privileges, constraints will also be removed.

3. Cannot be rolled back

4. No Triggers will be fired.

SYNTAX:

DROP TABLE table_name

TRUNCATE

1. TRUNCATE is a DDL Statement.

2. Removes all rows from a table, but the table structures and its columns, constraints, indexes remains.

3. Cannot be rolled back

4. Resets the identity of the table

5. Truncate is faster and uses fewer system and transaction log than delete.

6. Cannot use TRUNCATE on a table referenced by a FOREIGN KEY constraint.

7. No Triggers will be fired.

8. Cannot use WHERE conditions

SYNTAX:

TRUNCATE TABLE table_name