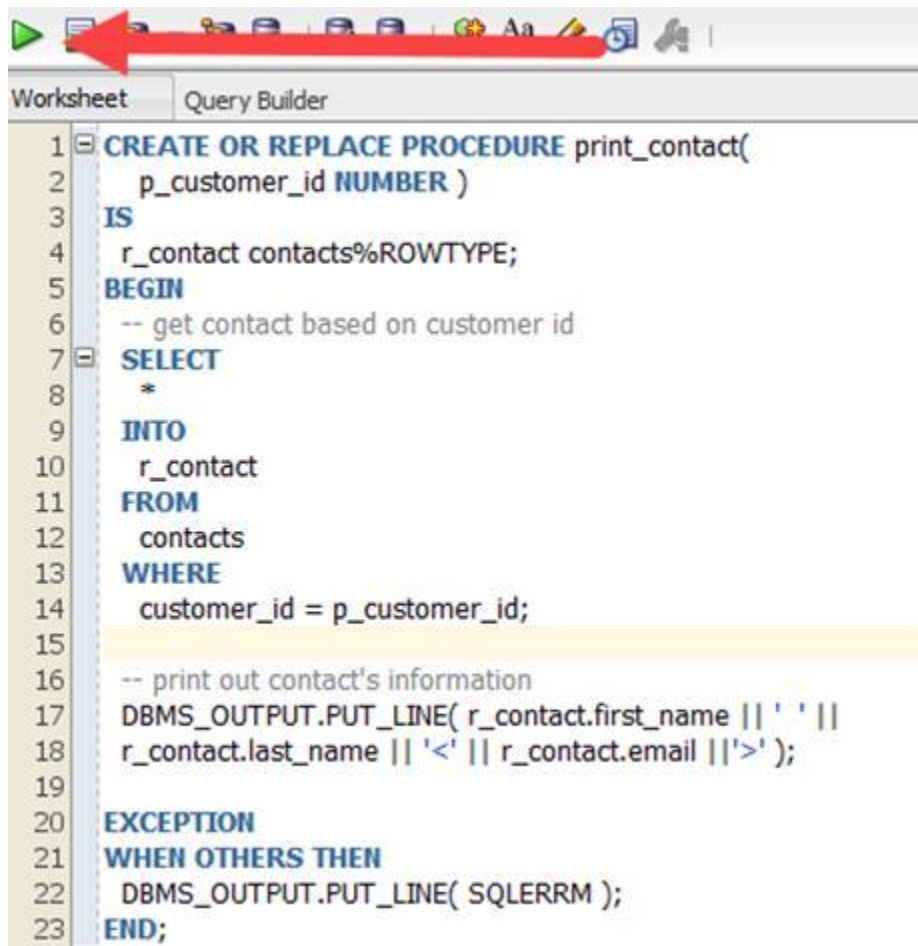




```
1 CREATE OR REPLACE PROCEDURE print_contact(  
2   p_customer_id NUMBER )  
3 IS  
4   r_contact contacts%ROWTYPE;  
5 BEGIN  
6   -- get contact based on customer id  
7   SELECT  
8     *  
9   INTO  
10    r_contact  
11   FROM  
12    contacts  
13   WHERE  
14    customer_id = p_customer_id;  
15  
16   -- print out contact's information  
17   DBMS_OUTPUT.PUT_LINE( r_contact.first_name || ' ' ||  
18     r_contact.last_name || '<' || r_contact.email || '>' );  
19  
20 EXCEPTION  
21 WHEN OTHERS THEN  
22   DBMS_OUTPUT.PUT_LINE( SQLERRM );  
23 END;
```

Introduction to Database Administration

TOPIC 132

PL/SQL Function

PL/SQL Function

PL/SQL Function

What is Function in PL/SQL?

Function - Example

Compiling & Executing Function

Executing a Function & Show Errors

Data Dictionary for Function

PL/SQL Function

What is Function in PL/SQL?

Is a block with a name

Returns one value only.

The DECLARE key word is not used

Parameters can only be IN

Is stored (**USER_SOURCE**)

PL/SQL Function

What is Function in PL/SQL?

Writing PL/SQL Function

```
CREATE OR REPLACE FUNCTION fname(  )
```

```
RETURN datatype IS
```

```
BEGIN
```

```
EXCEPTION
```

```
END;
```

Function - Example

CREATE OR REPLACE FUNCTION

func_test(p_empno IN VARCHAR2) RETURN NUMBER IS v_job EMP.job

PE:

BEGIN

SELECT, job

sal

INTO v_job_sal FROM emp

WHERE empno = p_empno; DBMS_OUTPUT.PUT_LINE('Job sal is

return v_sal;

EXCEPTION

;

GEN OTHERS THEN

'11=job);

END;

DBMS_OUTPUT.PUT_LINE('ERROR.')

SQL Show errors

PL/SQL Function

Compiling & Executing Function

SET SERVEROUTPUT ON;

DECLARE

...

v_sal emp.sal%TYPE;

...

BEGIN

...

v_sal := func_test(3477);

DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_sal)

);

...

END;

/

PL/SQL Function

Executing a Function & Show Errors

```

SQL> CREATE OR REPLACE FUNCTION
  2     func_test(p_empno IN VARCHAR2)  RETURN NUMBER IS
  3         v_job  EMP.job%TYPE;
  4         v_sal  EMP.sal%TYPE;
  5 BEGIN
  6     SELECT job, sal
  7     INTO  v_job,v_sal
  8     FROM  emp
  9         WHERE empno = p_empno;
 10     DBMS_OUTPUT.PUT_LINE('job is '||v_job);
 11     return v_sal;
 12 EXCEPTION
 13     WHEN OTHERS THENv_sal;
 14         DBMS_OUTPUT.PUT_LINE('ERROR...');
 15 END;
 16 /

```

Warning: Function created with compilation errors.

SQL> show error

Errors for FUNCTION FUNC_TEST:

LINE/COL ERROR

```

-----
13/18      PLS-00103: Encountered the symbol "THENV_SAL" when expecting one
           of the following:
           then or
           The symbol "then" was substituted for "THENV_SAL" to continue.

```

PL/SQL Function

Data Dictionary for Function

```

SQL> select text from user_source
  2  where name='FUNC_TEST';

TEXT
-----
FUNCTION
    func_test(p_empno IN VARCHAR2)  RETURN NUMBER IS
        v_job  EMP.job%TYPE;
        v_sal  EMP.sal%TYPE;
BEGIN
    SELECT job, sal
    INTO  v_job,v_sal
    FROM  emp
        WHERE empno = p_empno;
    DBMS_OUTPUT.PUT_LINE('job is '||v_job);
    return v_sal;

TEXT
-----
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('ERROR...');
END;

```

End

TOPIC 133

Calling Procedures & Functions

Calling Procedures & Functions

Calling Procedures & Functions

Calling Procedure

Calling Function

Calling Procedure with Parameters

Calling Procedures & Functions

Calling Procedure

```
DECLARE
...
BEGIN
...
  proc_test('23');
...
END;
/
```

Or

```
SQL> exec proc_test('1123')
```

Procedure can be called after compilation in some other function, procedure, block or package.

For example,

Calling Procedures & Functions

Calling Function

Function can be called after compilation in some other function, procedure, block or package by assigning it to variable.

```
DECLARE
    ...
    v_sal      emp.sal%TYPE;
    ...
BEGIN
    ...
    v_sal := func_test(3477);
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_sal));
    ...
END;
/
```

Calling Procedures & Functions

Calling Procedure with Parameters

```

create or replace procedure proc_inout
    (p_empno in number,
     p_sal out number) is
BEGIN
    select sal
    into p_sal
    from emp
    where empno=p_empno;
END;
/

create or replace procedure proc_out is
    v_sal EMP.SAL%TYPE;
BEGIN
    proc_inout(7369,v_sal);
    DBMS_OUTPUT.PUT_LINE('salary is :'||v_sal);
END;
/

SQL> exec proc_out
salary is :2000

PL/SQL procedure successfully completed.

```

Introduction to Database Administration

TOPIC 134

Exercises of Functions

Exercises of Functions

Exercises of Functions

Compiling & Executing Function

Function - Exercise-1

Function - Exercise-2

Exercises of Functions

Compiling & Executing Function

SET SERVEROUTPUT ON;

DECLARE

...

```

v_sal emp.sal%TYPE;
... BEGIN
...
v_sal := func_test(3477);
DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_sal)
);
... END;
/

```

Exercises of Functions

Function – Exercise-1

```

CREATE OR REPLACE FUNCTION
    func_MaxSal(p_deptno IN NUMBER) RETURN NUMBER IS
    v_MaxSal      EMP.sal%TYPE;
BEGIN
    SELECT max(sal)
    INTO  v_MaxSal
    FROM  emp
    where deptno = p_deptno;
    DBMS_OUTPUT.PUT_LINE('Highest Salary is '||v_MaxSal);
    return v_MaxSal;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('ERROR...' || SQLERRM);
END;
/

```

```

SQL> DECLARE
2   vv_MaxSal EMP.SAL%TYPE;
3 BEGIN
4   vv_MaxSal := func_MaxSal(10);
5   DBMS_OUTPUT.PUT_LINE('Calling from block:'||vv_MaxSal);
6 END;
7 /
Highest Salary is 5000
Calling from block:5000

```

Exercises of Functions

Function – Exercise-2

```

CREATE OR REPLACE FUNCTION fact(x number)
RETURN number IS
    f number;
BEGIN
    IF x=0 THEN
        f := 1;
    ELSE
        f := x * fact(x-1);
    END IF;
RETURN f;
END;
/

DECLARE
    num number;
    factorial number;
BEGIN
    num:= 6;
    factorial := fact(num);
    dbms_output.put_line(' Factorial ' || num || ' is ' || factorial);
END;
/
Factorial 6 is 720
PL/SQL procedure successfully completed.

```

Introduction to Database Administration

TOPIC 135

Exercises of Procedures

Exercises of Procedures

Exercises of Procedures

Compiling & Executing Function

Procedure Exercise-I

Exercises of Procedures

Compiling & Executing Procedure

DECLARE

...

BEGIN

...

proc_test('23');

...

END;

/

Or

SQL> exec proc_test('1123')

Exercises of Procedures

Procedure – Exercise-1

```
CREATE OR REPLACE PROCEDURE
  proc_MaxSal(p_deptno IN NUMBER, p_MaxSal OUT NUMBER) IS
  v_MaxSal      EMP.sal%TYPE;
BEGIN
  SELECT max(sal)
  INTO  v_MaxSal
  FROM  emp
  where deptno = p_deptno;
  DBMS_OUTPUT.PUT_LINE('Highest Salary is '||v_MaxSal);
  p_MaxSal := v_MaxSal;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('ERROR...'||SQLERRM);
END;
/
```

```
CREATE OR REPLACE PROCEDURE proc_CALLING(p_dno IN NUMBER) IS
  vv_MaxSal EMP.SAL%TYPE;
BEGIN
  proc_MaxSal(p_dno, vv_MaxSal);
  DBMS_OUTPUT.PUT_LINE('Calling from block:'||vv_MaxSal);
END;
/
```

SQL> execute proc_CALLING(20);

```
SQL> execute proc_CALLING(20);
Highest Salary is 3400
Calling from block:3400

PL/SQL procedure successfully completed.
```

TOPIC 136

PL/SQL Package-1

PL/SQL Package-1

PL/SQL Package-1

Purpose of PL/SQL Package

PL/SQL Package contains ...

Schematic Diagram of Package

PL/SQL Package Specification

PL/SQL Package Body

Package Specification & its Body

PL/SQL Package-1

Purpose of PL/SQL Package

In PL/SQL, a package is a schema object that contains definitions for a group of related functionalities.

A package

includes **variables, constants, cursors, exceptions, procedures, functions, and subprograms.**

It is compiled and stored in the Oracle Database.

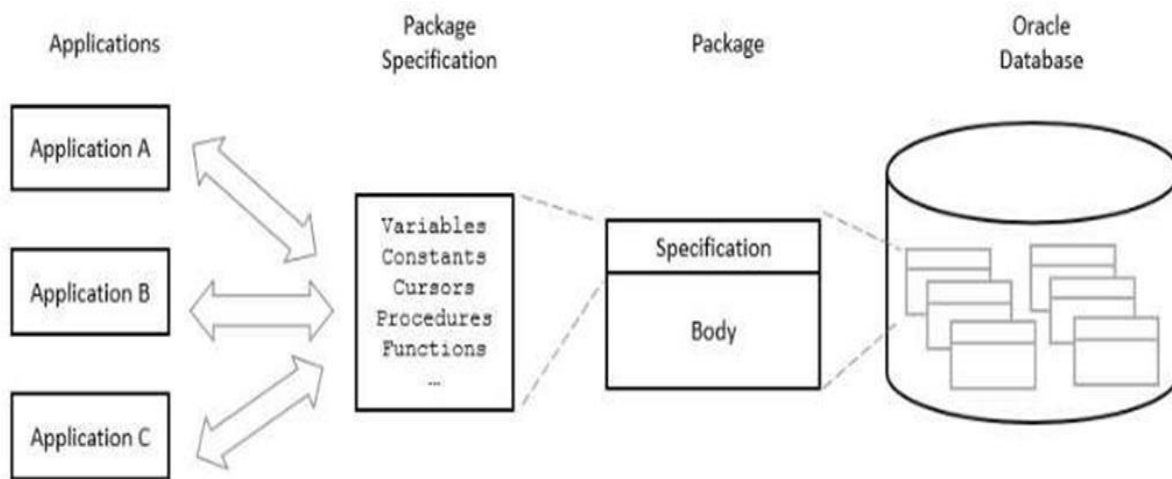
PL/SQL Package-1

PL/SQL Package contains ...

Typically, a package has a specification and a body. A package specification is mandatory while the package body can be required or optional, depending on the package specification.

PL/SQL Package-1

Schematic Diagram of Package



<https://www.oracletutorial.com/plsql-tutorial/plsql-package/>
PL/SQL Package-1

PL/SQL Package Specification

The package specification declares the public objects that are accessible from outside the package.

If a package specification whose public objects include cursors and subprograms, then it must have a body which defines queries for the cursors and code for the subprograms.

PL/SQL Package-1

PL/SQL Package Body

A package body contains the implementation of the cursors or subprograms declared in the package specification. In the package body, you can declare or define private variables, cursors, etc., used only by package body itself.

PL/SQL Package-1

PL/SQL Package Body

A package body can have an initialization part whose statements initialize variables or perform other one-time setups for the whole package.

A package body can also have an exception-handling part used to handle exceptions.
PL/SQL Package-1

```
CREATE OR REPLACE PACKAGE emp_info IS
```

```
END emp_info;  
/
```

```
CREATE OR REPLACE PACKAGE BODY emp_info is
```

```
END emp_info; --end of package body  
/
```

Package Specification & its Body
Introduction to Database Administration

TOPIC 137

PL/SQL Package-2
PL/SQL Package-2

PL/SQL Package-2

What is a Package in PL/SQL?

Package Specification

Package Body

Add in a Package Body by hiding from

User Specification

Invoking Package Subprogram

PL/SQL Package-2

What is a Package in PL/SQL?

Selected and purposeful collection of procedures, functions and variables is managed in package.

Package

Package Specification

Package Body

Invoking Package subprogram

Abstract Programming

To avoid spreading of procedures & functions

Is stored (**USER_SOURCE**)

PL/SQL Package-2

Package Specification



CREATE OR REPLACE PACKAGE emp_info IS

v count **INTEGER;—global variable for all sub programs inside package**



**PROCEDURE insert_record(p_empna IN NUMBER
p_ename IN VARCHAR2**



**, p ob IN VARCHAR2
, p_sal IN NUMBER
, p comm IN NUMBER
, p deptno IN VARCHAR2);**

PROCEDURE delete_record(p_empno IN NUMBER);

FUNCTION sum_dept_sal(p_deptno IN dept.deptno%TYPE) RETURN emp.sal%TYPE;

END emp_info;

/

Package Body

CREATE OR REPLACE PACKAGE BODY emp_info AS

```
PROCEDURE insert_record(p_empno IN NUMBER, p_ename IN VARCHAR2,
p_job IN VARCHAR2, p_sal IN NUMBER,
p_comm IN NUMBER, p_deptno IN VARCHAR2) IS
BEGIN
    INSERT INTO EMP (empno, ename, job, sal, comm, deptno)
    VALUES (p_empno, p_ename, p_job, p_sal, p_comm, p_deptno);
END insert_record;
```

```
PROCEDURE delete_record(p_empno IN NUMBER) IS
BEGIN
    DELETE FROM EMP
    WHERE empno = p_empno;
END delete_record;
```

```
FUNCTION sum_dept_sal(p_deptno IN NUMBER) RETURN emp.sal%TYPE IS
v_sum_sal emp.sal%TYPE;
BEGIN
    SELECT SUM(sal)
    INTO v_sum_sal
    FROM emp
    WHERE deptno = p_deptno;
    RETURN v_sum_sal;
END;
```

END emp_info; —end of package body

PL/SQL Package-2

Purpose of PL/SQL Package

Following are its main purposes:

- Make code more modular
- Hide implementation details
- Improve application performance
- Minimize unnecessary recompiling code
- Manage authorization easily

PL/SQL Package-2

Add in a Package Body by hiding from User Specification

```
Procedure call_from_out is
  v_empno EMP.empno%TYPE;
  v_ename EMP.ename%TYPE;
  v_job    EMP.job%TYPE;
  v_sal    EMP.sal%TYPE;
  v_comm   EMP.comm%TYPE;
  v_deptno EMP.dept%TYPE;
begin

  v_empno := 8899;
  v_ename := 'ALI';
  v_job    := 'MANAGER';
  v_sal    := 8000;
  v_comm   := 300;
  v_deptno := 10;

  insert_record( v_empno, v_ename, v_job, v_sal, v_comm,v_deptno);
  commit;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Except call_from_out:'||SQLERRM);
end;
/
```

PL/SQL Package-2

Invoking Package Subprogram

```
DECLARE
    v_sum_sal emp.sal%TYPE;
BEGIN
    ...
    v_sum_sal := emp_info.sum_dept_sal(10);
    ...
    emp_info.delete_record(1234567);
    ...
END;
/
```

Or

```
SQL> exec emp_info.insert_record(...,...)
```

Introduction to Database Administration

TOPIC 138

Execution of Packages

Execution of Packages

Execution of Packages

Compiling a Package using SQL*Plus

Compiling a Package using Oracle Developer

Compiling a Package using Files on SQL*Plus

Calling Function and Procedure from Package

Execution of Packages

Compiling a Package using SQL*Plus

If you use SQL*Plus for compiling and creating a package body, you type forward slash (/) as follows:

```
44  return in_net_value;  
45  EXCEPTION  
46  WHEN no_data_found THEN  
47  DBMS_OUTPUT.PUT_LINE( SQLERRM );  
48  END get_net_value_by_customer;  
49  
50  END order_mgmt;  
51  /
```

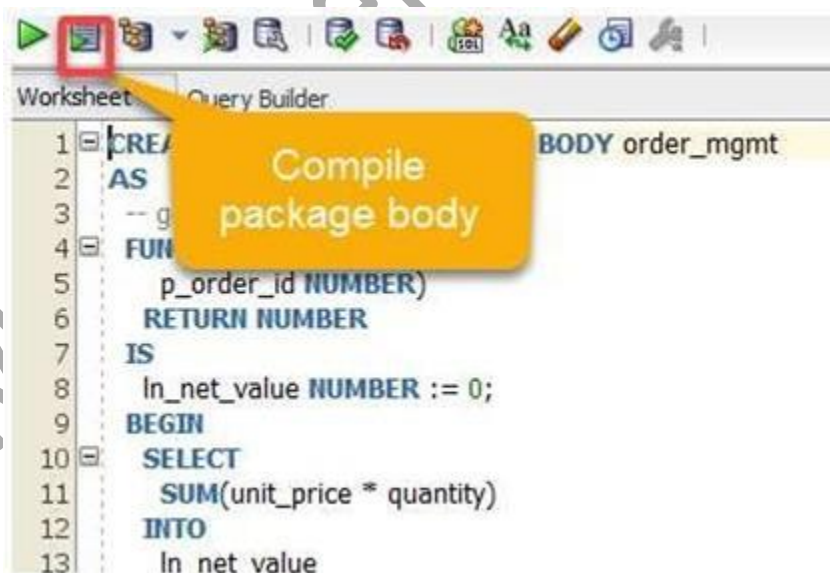
Package body created.

SQL> _

Execution of Packages

Compiling a Package using Oracle Developer

From PL/SQL Developer, you click the Run Script button to compile the package body.



Execution of Packages

Compiling a Package using Files on SQL*Plus

If you manage package specification & body using separate files, you can compile separately using the following commands:

```
SQL> @path_to_package_body_file
```

```
SQL > @path_to_package_body_file 2 /  
Execution of Packages
```

Calling Function and Procedure from Package

```
DECLARE  
    v_sum_sal emp.sal%TYPE;  
BEGIN  
    ...  
    v_sum_sal := emp_info.sum_dept_sal(10);  
    ...  
    emp_info.delete_record(1234567);  
    ...  
END;  
/
```

Or

```
SQL> exec emp_info.insert_record(...,...)
```

End
Introduction to Database Administration

TOPIC 139

PL/SQL Triggers PL/SQL Triggers

PL/SQL Triggers

What is an Oracle Triggers?
Oracle Trigger Usage
Oracle Trigger Syntax
Oracle Trigger Header
Oracle Trigger Body
Create or Replace a Trigger
PL/SQL Triggers

What is an Oracle Triggers?

A trigger is a named PL/SQL block stored in the Oracle Database and executed automatically when a triggering event takes place. The event can be any of the following:

A data manipulation language (DML) statement executed against a table e.g., INSERT, UPDATE, or DELETE.
PL/SQL Triggers

What is an Oracle Triggers?

For example, if you define a trigger that fires before an INSERT statement on the customers table, the trigger will fire once before a new row is inserted into the customers table.
A data definition language (DDL) statement executes e.g., CREATE or ALTER statement.
These triggers are often used for auditing purposes to record changes of the schema.
PL/SQL Triggers

What is an Oracle Trigger?

A system event such as startup or shutdown of the Oracle Database.
A user event such as login or logout.
The act of executing a trigger is also known as firing a trigger. We say that the trigger is fired.
PL/SQL Triggers

Oracle Trigger Usage

Oracle triggers are useful in many cases such as the following:

Enforcing complex business rules that cannot be established using integrity constraint such as UNIQUE, NOT NULL, and CHECK.

Preventing invalid transactions.

PL/SQL Triggers

Oracle Trigger Usage

Gathering statistical information on table accesses.

Generating value automatically for derived columns.

Auditing sensitive data.

PL/SQL Triggers

Oracle Trigger Syntax

Following is its syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER } triggering_event ON table_name
[FOR EACH ROW]
[FOLLOWS | PRECEDES another_trigger]
[ENABLE / DISABLE ]
[WHEN condition]
DECLARE
    declaration statements
BEGIN
    executable statements
EXCEPTION
    exception_handling statements
END;
```

PL/SQL Triggers

Oracle Trigger Header

```
CREATE [OR REPLACE] TRIGGER trigger_name  
    {BEFORE | AFTER } triggering_event ON table_name  
    [FOR EACH ROW]  
    [FOLLOWS | PRECEDES another_trigger]  
    [ENABLE / DISABLE ]  
    [WHEN condition]
```

PL/SQL Triggers

Oracle Trigger Body

```
DECLARE  
    declaration statements  
  
BEGIN  
    executable statements  
  
EXCEPTION  
    exception_handling statements  
  
END;
```

PL/SQL Triggers

Create or Replace a Trigger

The **CREATE** keyword specifies that you are creating a new trigger. The **OR REPLACE** keywords are optional. They are used to modify an existing trigger.

Even though the OR REPLACE keywords are optional, they appear with the CREATE keyword in most cases.

End

Introduction to Database Administration

TOPIC 140

Trigger Examples

Trigger Examples

Trigger Examples

Trigger Example-1

Execution of Trigger Example-1

Trigger Example-2

Trigger & Function Example-3

Trigger Examples

Trigger Example-1

Display difference of salary when salary of employee is changed

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON EMP
FOR EACH ROW
WHEN (NEW.EMPNO > 0)
DECLARE
    sal_diff number;
BEGIN
    sal_diff := :NEW.sal - :OLD.sal;
    dbms_output.put_line('Old salary: ' || :OLD.sal);
    dbms_output.put_line('New salary: ' || :NEW.sal);
    dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

Trigger Examples

Execution of Trigger Example-1

Display difference of salary when salary of employee is changed

```
SQL> CREATE OR REPLACE TRIGGER display_salary_changes
 2 BEFORE DELETE OR INSERT OR UPDATE ON EMP
 3 FOR EACH ROW
 4 WHEN (NEW.EMPNO > 0)
 5 DECLARE
 6     sal_diff number;
 7 BEGIN
 8     sal_diff := :NEW.sal - :OLD.sal;
 9     dbms_output.put_line('Old salary: ' || :OLD.sal);
10     dbms_output.put_line('New salary: ' || :NEW.sal);
11     dbms_output.put_line('Salary difference: ' || sal_diff);
12 END;
13 /

Trigger created.

SQL> set serveroutput on
SQL> update emp
 2 set sal=777
 3 where ename='SMITH';
Old salary: 800
New salary: 777
Salary difference: -23

1 row updated.
```

Trigger Examples

Trigger Example-2

Keeping log of old data into a log table

```

CREATE OR REPLACE TRIGGER del_emp
BEFORE DELETE ON emp
FOR EACH ROW
DECLARE
    x number(2);
BEGIN
    INSERT INTO emp_audit
        VALUES (:NEW.empno, USER, sysdate);
END;
/

--Other forms can be BEFORE INSERT OR UPDATE
--OF salary ON EMP
--Need to create table emp_audit
--select * from emp_audit;

```

Trigger Examples

Trigger & Function Example-3

```

create table emp1
as select * from emp;

alter table emp1
add bonus number(6,2);

CREATE OR REPLACE FUNCTION "BONUS_CALC"
    (p_sal IN NUMBER) RETURN NUMBER IS
BEGIN
    return(p_sal/600);
    dbms_output.put_line(' ***** THE RECORD HAS SUCCESSFULLY BEEN INSERTED *****');
END;
/

CREATE OR REPLACE TRIGGER "UPDATE_BONUS" before INSERT OR UPDATE OF SAL ON "EMP1" FOR
BEGIN
    --AFTER option gives error of mutating
    --select Bonus_Calc(:New.sal) into :New.bonus from dual;
    --Or use the following assignment statement
    :New.bonus := Bonus_Calc(:New.sal);

    dbms_output.put_line(' ***** THE RECORD HAS SUCCESSFULLY BEEN INSERTED *****');
end;
/

Update emp1
set sal=1600
where ename in ('ALLEN');

```

TOPIC 141

Trigger Exercises

Trigger Exercises

Trigger Exercises

Exercise-1 (Off day DML Restriction)

Exercise-2 (for Data Entry)

Trigger's Disable/ Enable

Trigger's Disable/ Enable using SQL

Developer

Exercise-3 (Mutating Error)

Fired Triggers

Updating & Deleting on Table with
specific Columns

Trigger Exercises

Exercise-1 (Off day DML Restriction)

This is a statement trigger on the EMP table, and it fires before an INSERT, UPDATE, or DELETE statement is issued. First, the trigger determines the day of the week. If the day is Saturday or Sunday, an error message is generated.

Trigger Exercises

Exercise-1 (Off day DML Restriction)

```

SQL> CREATE OR REPLACE TRIGGER EMP_ModifyIssue
  2 BEFORE INSERT OR UPDATE OR DELETE ON EMP
  3 DECLARE
  4   v_day VARCHAR2(10);
  5 BEGIN
  6   v_day := RTRIM(TO_CHAR(SYSDATE, 'DAY'));
  7   IF v_day LIKE ('S%') THEN
  8     RAISE_APPLICATION_ERROR (-20000, 'A table cannot be modified during off hours');
  9   END IF;
 10 END;
 11 /

```

Trigger created.

```

SQL>
SQL> update emp
  2 set sal=700
  3 where ename='SMITH';
update emp
  *
ERROR at line 1:
ORA-20000: A table cannot be modified during off hours
ORA-06512: at "SCOTT.EMP_MODIFYISSUE", line 6
ORA-04088: error during execution of trigger 'SCOTT.EMP_MODIFYISSUE'

```

Trigger Exercises

Exercise-2 (for Data Entry)

```

CREATE OR REPLACE TRIGGER Derived
BEFORE INSERT OR UPDATE OF Ename ON Emp

/* Before updating the ENAME field, derive the values for
the UPPERNAME and SOUNDEXNAME fields. Restrict users
from updating these fields directly: */
FOR EACH ROW
BEGIN
  :NEW.Uppername := UPPER(:NEW.Ename);
  :NEW.Soundexname := SOUNDEX(:NEW.Ename);
END;

```

Trigger Exercises

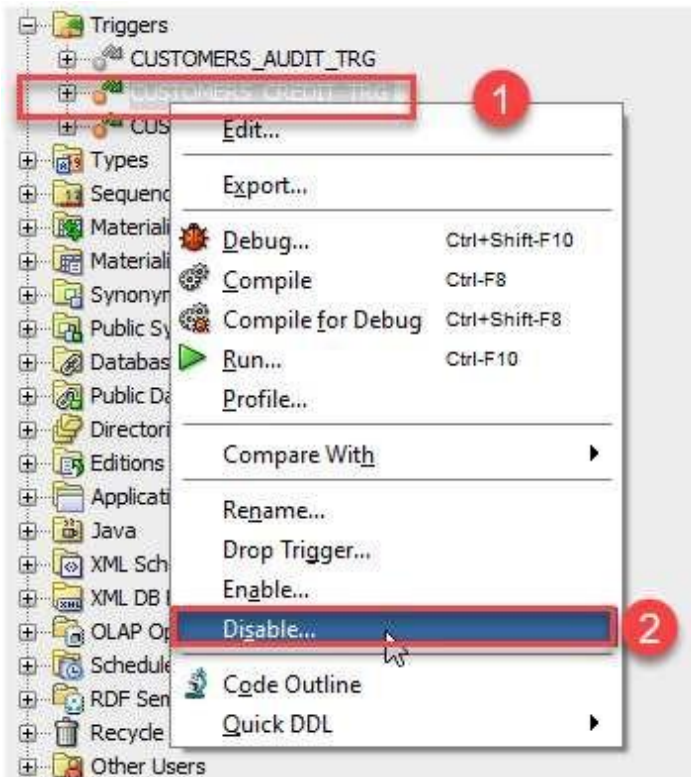
Trigger's Disable/ Enable

Sometimes, you may want to disable a trigger for testing and troubleshooting purposes. To disable a trigger, you use the ALTER TRIGGER DISABLE statement:

```
ALTER TRIGGER trigger_name DISABLE; ALTER TABLE table_name DISABLE ALL TRIGGERS;
```

Trigger Exercises

Trigger's Disable/ Enable using SQL Developer



Trigger Exercises

```
SQL> CREATE OR REPLACE TRIGGER Update_EmpDeptno
  2 BEFORE INSERT OR UPDATE ON EMP
  3 FOR EACH ROW
  4 DECLARE
  5 v_cnt Number(3);
  6 BEGIN
  7 IF :NEW.deptno IS NOT NULL THEN
  8 BEGIN
  9 SELECT count(*)
 10 INTO v_cnt
 11 FROM EMP
 12 WHERE deptno = :NEW.deptno;
 13 EXCEPTION
 14 WHEN NO_DATA_FOUND THEN
 15 RAISE_APPLICATION_ERROR
 16 (-20001, 'This is not a valid deptno');
 17 END;
 18 END IF;
 19 END;
 20 /
```

Trigger created.

```
SQL> Update emp
  2 set deptno = 11
  3 where deptno=10;
Update emp
*
```

ERROR at line 1:

```
ORA-04091: table SCOTT.EMP is mutating, trigger/function may not see it
ORA-06512: at "SCOTT.UPDATE_EMPDEPTNO", line 6
ORA-04088: error during execution of trigger 'SCOTT.UPDATE_EMPDEPTNO'
```

Exercise-3 (Mutating Error)

Trigger Exercises

```

CREATE OR REPLACE TRIGGER Update_EmpDeptno
BEFORE INSERT OR UPDATE ON EMP
FOR EACH ROW
DECLARE
v_cnt Number(3);
BEGIN
    IF :NEW.deptno IS NOT NULL THEN
        BEGIN
            SELECT count(*)
            INTO v_cnt
            FROM dept
            WHERE deptno = :NEW.deptno;
            DBMS_OUTPUT.PUT_LINE (:NEW.deptno);
        EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR (-20001, 'This is not a valid deptno');
        END;
    END IF;
END;
/

```

Exercise-3 (Resolving Mutating Error)

Trigger Exercises

Fired Triggers

Need to see which trigger names are fired on commands such as

```

select distinct name
from user_source
where type='TRIGGER';

```

```

drop trigger DEL_EMP;
drop trigger UPDATE_EMPDEPTNO;

```

```
SQL> update emp
  2   set deptno=11
  3   where deptno=10;
11
Old salary:
New salary:
Salary difference:
11
Old salary: 4000
New salary: 4000
Salary difference: 0
11
Old salary: 2450
New salary: 2450
Salary difference: 0
```

Trigger Exercises

Updating & Deleting on Table with specific Columns

```

CREATE OR REPLACE TRIGGER Updating_EMP
BEFORE
  INSERT OR
  UPDATE OF sal, deptno OR
  DELETE
ON EMP
BEGIN
  CASE
    WHEN INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Inserting');
    WHEN UPDATING('sal') THEN
      DBMS_OUTPUT.PUT_LINE('Updating sal');
    WHEN UPDATING('deptno') THEN
      DBMS_OUTPUT.PUT_LINE('Updating department ID');
    WHEN DELETING THEN
      DBMS_OUTPUT.PUT_LINE('Deleting');
  END CASE;
END;
/

update emp
set sal=2000
where ename='SMITH';

```

Introduction to Database Administration

TOPIC 142

Cursor Programming-1

Cursor Programming-1

Cursor Programming-1

What is a Cursor?

Implicit Cursor

Execution Cycle of Explicit Cursor

Cursor's usage

Explicit Cursor Attributes

Cursor Programming-1

What is a Cursor?

A cursor is a pointer that points to a result of a query.
It is mainly used with tables which return more than one row.
It is handled implicitly and explicitly.
Cursor Programming-1

Implicit Cursor

Whenever Oracle executes an SQL statement such as SELECT INTO, INSERT, UPDATE, and DELETE, it automatically creates an implicit cursor.

Oracle internally manages the whole execution cycle of implicit cursors and reveals only the cursor's information and statuses such as SQL%ROWCOUNT, SQL%ISOPEN, SQL%FOUND, and SQL%NOTFOUND.
Cursor Programming-1

Implicit Cursor

The implicit cursor is not elegant when the query returns zero or multiple rows which cause NO_DATA_FOUND or TOO_MANY_ROWS exception respectively.
Cursor Programming-1

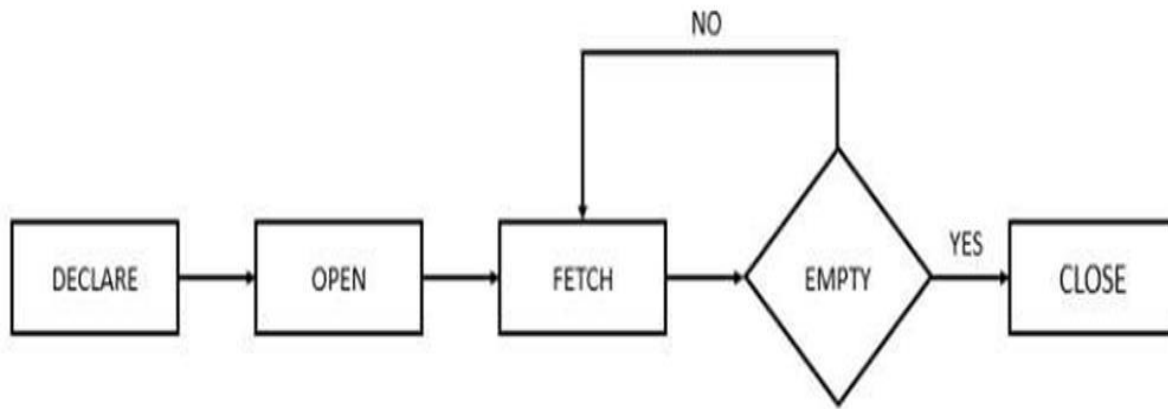
Explicit Cursor

An explicit cursor is an SELECT statement declared explicitly in the declaration section of the current block or a package specification.

For an explicit cursor, you have control over its execution cycle from OPEN, FETCH, and CLOSE.

Oracle defines an execution cycle that executes an SQL statement and associates a cursor with it.
Cursor Programming-1

Execution Cycle of Explicit Cursor



Cursor Programming-1

Cursor's usage

Declare a cursor

```
CURSOR cursor_name IS query;
```

Open a cursor

```
OPEN cursor_name;
```

Cursor Programming-1

Cursor's usage

Fetch from a cursor

The FETCH statement places the contents of the current row into var. `FETCH cursor_name INTO variable_list;`
To retrieve all rows in a result set, you need to fetch each row till the last one.

Closing a cursor

```
CLOSE cursor_name;
```

Cursor Programming-1

Explicit Cursor Attributes

%ISOPEN

%FOUND

%NOTFOUND

%ROWCOUNT

Introduction to Database Administration

TOPIC 143

Cursor Programming-2

Cursor Programming-2

Cursor Programming-2

Explicit Cursor Attributes

Implicit Cursor - Record Type

ROWTYPE Example - 1

ROWCOUNT Example

How to Declare Cursor?

%ISOPEN, %FOUND, FETCH Example

Cursor Programming-2

Explicit Cursor Attributes

%ISOPEN

%FOUND

%NOTFOUND

%ROWCOUNT

Cursor Programming-2

Implicit Cursor – Record Type

A record is a composite data structure, which means that it is composed of one or more elements.

Records are very much like a row of a database table, but each element of the record does not stand on its own. PL/SQL supports three kinds of records: table-based, cursor-based, and programmer-defined.

```
v_EmpRec EMP%TYPE;  
Cursor Programming-2
```

Implicit Cursor – Record Type

```
CURSOR c_emp IS  
SELECT empno, ename, job  
FROM emp  
WHERE deptno = 20; vr_emp c_emp%ROWTYPE;
```

ROWTYPE Example - 1

```
DECLARE  
v_emp EMP%ROWTYPE; BEGIN  
SELECT *  
INTO v_emp  
FROM EMP  
WHERE ename = 'SMITH';  
DBMS_OUTPUT.PUT_LINE('Record number is ' || v_emp.  
ename);  
EXCEPTION  
WHEN NO_DATA_FOUND THEN  
DBMS_OUTPUT.PUT_LINE('The Employee  
'|| 'is  
not in the database');  
END;
```

ROWTYPE Example - 2

```
DECLARE  
CURSOR c_emp IS SELECT * FROM EMP;  
vr_emp c_emp%ROWTYPE;
```

```

BEGIN
OPEN  z_emp i

FETCH  z_emp INTO  vr_emp ; EXIT  WHEN  z_emp 2N0TF OUND ;
DBMS_OUTPUT . PUT_L INE (vr_emp . empno
' ' || vr_emp . ename || ' ' || vr_emp . sa  ;

/
*)
END  LOOP ;
END ;

```

ROWCOUNT Example

```

DECLARE
vpj ob EMP . j ob%type: BEGIN
SELECT j ob INTO vpj ob FROM ERP
HERE empno = 7698 :
I F SQL%ROWCOUNT = 1 THEN
DBMS_OUTPUT . PUT_L INE (v_job || ' has a ' ||
'Empno of 7698'):
ELSIF SQL%ROWCOUNT = 0 THEN DBMS_OUTPUT.PUT_LINE('The Empno 7698 is '
not in the database'):
ELSE
DBMS_OUTPUT.PUT_LINE('Stop... ): END IF:
END:

```

Cursor Programming-2

How to Declare Cursor?

CURSOR c_emp IS

SELECT empno, ename, job

FROM emp

WHERE deptno = 20;



7369	Smith	Clerk
7566	Jones	Manager
7788	SCOTT	Analyst
7876	Adams	Clerk
7906	FORD	Analyst

Cursor Programming-2

%ISOPEN, %FOUND, FETCH Example

```
DECLARE
    v_empno EMP.Empno%TYPE;
    CURSOR c_EMP IS
        SELECT Empno
        FROM EMP
        WHERE Empno > 1000;
BEGIN
    OPEN c_EMP;
    LOOP
        FETCH c_EMP INTO v_empno;
        IF c_EMP%FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Just FETCHED row '
                || TO_CHAR(c_EMP%ROWCOUNT) || ' EMP ID: ' || v_empno);
        ELSE
            EXIT;
        END IF;
    END LOOP;
    CLOSE c_EMP;
    EXCEPTION
    WHEN OTHERS THEN
        IF c_EMP%ISOPEN
        THEN
            CLOSE c_EMP;
        END IF;
END;
```

TOPIC 144

Cursor Programming-3

Cursor Programming-3

Cursor Programming-3

How to Declare Cursor?

Sample Program using Cursor

Cursor for DML Insert

Cursor with Parameters

Cursor Program-1

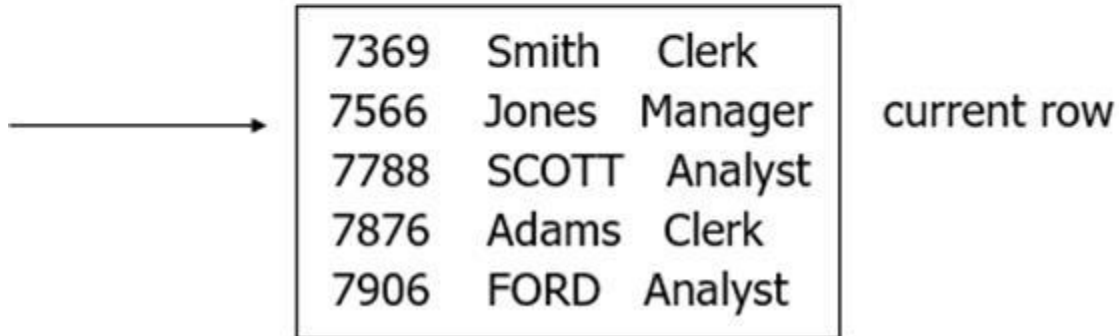
Cursor Program-1 (Solution)

Cursor Programming-3

How to Declare Cursor?

CURSOR c_emp IS

SELECT empno, ename, job
FROM emp
WHERE deptno = 20;



A horizontal arrow points from the left towards the first row of the table below. The table is enclosed in a rectangular box and contains five rows of employee data. The text 'current row' is positioned to the right of the table, aligned with the second row.

7369	Smith	Clerk
7566	Jones	Manager
7788	SCOTT	Analyst
7876	Adams	Clerk
7906	FORD	Analyst

Cursor Programming-3

Sample Program using Cursor

DECLARE

```
CURSOR c_emp IS  
SELECT empno, ename, job  
FROM emp  
WHERE deptno = 20;
```

BEGIN

```
FOR v_c IN c_emp LOOP  
    ...  
    DBMS_OUTPUT.PUT_LINE(v_c.ename);  
    ...  
END LOOP;  
...  
end;  
/
```

Cursor Programming-3

Cursor for DML Insert

```

DECLARE
    CURSOR c_emp IS
        SELECT empno, ename, job
        FROM EMP
        WHERE empno > 1100;
BEGIN
    FOR r_emp IN c_emp
    LOOP
        INSERT INTO table_log
        VALUES(r_emp.ename);
    END LOOP;
END;
/

```

Cursor Programming-3

Cursor with Parameters

```

DECLARE
    CURSOR c_emp (p_job IN EMP.job%TYPE) IS
        SELECT *
        FROM EMP
        WHERE job = p_job;
BEGIN
    FOR r_emp IN c_emp('CLERK')
    LOOP
        DBMS_OUTPUT.PUT_LINE(r_emp.ename || ' -- ' || r_emp.job);
    END LOOP;
END;
/

```

Cursor Programming-3

Cursor Program-1

```
SQL> desc patient_visit
```

Name	Null?	Type
PATIENT_ID	NOT NULL	CHAR(7)
DEPEND_SNO	NOT NULL	VARCHAR2(2)
PATIENT_VISITNO	NOT NULL	NUMBER(4)
EMP_ID		CHAR(7)
DOCTOR_SNO		NUMBER(3)
VISIT_DATE		DATE
VISIT_NOTE		VARCHAR2(50)
CAT_TEMPLATE		LONG
MEDICATION		VARCHAR2(500)
CHECKUP_STATUS		CHAR(1)
SUB_CAT_ID		NUMBER(6)
CAT_TEMPLATE1		VARCHAR2(3000)

Given a table with first three columns are composite keys. Write a PL/SQL program using cursor to update cat_template1 with LONG column cat_template. Assuming this tables contains more than 40,000 records.

Cursor Programming-3

```
declare
  v_count number(7);
cursor c1 is
  select  PATIENT_ID,DEPEND_SNO,PATIENT_VISITNO,CAT_TEMPLATE
  from patient_visit;
begin
  v_count := 0;
  for v_c in c1 LOOP
    update patient_visit
    set  CAT_TEMPLATE1  = v_c.CAT_TEMPLATE
    where PATIENT_ID = v_c.PATIENT_ID
      and DEPEND_SNO = v_c.DEPEND_SNO
      and PATIENT_VISITNO = v_c.PATIENT_VISITNO;
    If v_count = 300 Then
      v_count := 0;
      --DBMS_OUTPUT.PUT_LINE('commit test');
    else
      v_count := v_count + 1;
    end if;
  end LOOP;
  commit;
end;
/
```

Cursor Program-1 (Solution)

Introduction to Database Administration

TOPIC 145

Programming using an Array

Programming using an Array

Programming using an Array

What is an Array?

Declaration of an Array

Basic Example of an Array

Associative Array

Associative Array - An Example

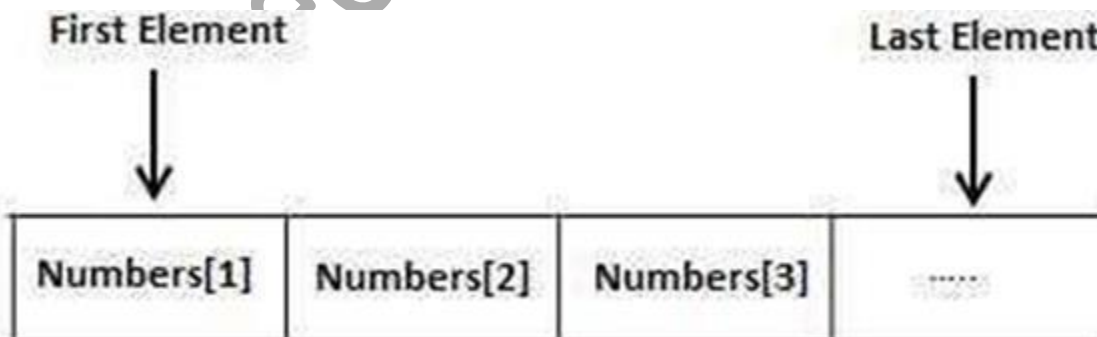
Use of Array with an Exception

Array Null and Empty Collection

Programming using an Array

What is an Array?

All **VARRAYS** consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



https://www.tutorialspoint.com/plsql/plsql_collections.htm

Programming using an Array

Declaration of an Array

A varray type is created with the CREATE TYPE statement. You must specify the maximum size and the type of elements stored in the varray.

```
CREATE Or REPLACE TYPE namearray AS  
VARRAY(3) OF VARCHAR2(10);
```

```
/
```

Programming using an Array

Basic Example of an Array

```
DECLARE  
    CURSOR c_emp is  
    SELECT  ename FROM EMP;  
    TYPE c_list is varray (20) of EMP.ename%type;  
    ename_list c_list := c_list();  
    counter integer :=0;  
BEGIN  
    FOR n IN c_emp LOOP  
        counter := counter + 1;  
        ename_list.extend;  
        ename_list(counter) := n.ename;  
        dbms_output.put_line('Employee(' || counter || '):'  
        || ename_list(counter));  
    END LOOP;  
END;  
/
```

Programming using an Array

Associative Array

An index-by table is a set of key-value pairs. Each key is unique and is used to locate the corresponding value. The key can be either an integer or a string.

```
TYPE type_name IS TABLE OF element_type [NOT NULL] INDEX BY subscript_type;
```

```
table_name type_name;
```

Programming using an Array

Associative Array – An Example

```

DECLARE
    TYPE salary IS TABLE OF NUMBER INDEX BY VARCHAR2(20);
    salary_list salary;
    name    VARCHAR2(20);
BEGIN
    -- adding elements to the table
    salary_list('Sarah') := 62000;
    salary_list('Jamil') := 75000;
    salary_list('Wisal') := 100000;
    salary_list('Umar') := 78000;

    -- printing the table
    name := salary_list.FIRST;
    WHILE name IS NOT null LOOP
        dbms_output.put_line
            ('Salary of ' || name || ' is ' || TO_CHAR(salary_list(name)));
        name := salary_list.NEXT(name);
    END LOOP;
END;
/

```

Programming using an Array

Use of Array with an Exception

```

SQL> DECLARE
2   CURSOR name_cur IS
3   SELECT ename
4   FROM EMP
5   WHERE rownum <= 10;
6   TYPE ename_type IS TABLE OF EMP.ename%TYPE
7   INDEX BY BINARY_INTEGER;
8   ename_tab ename_type;
9   v_counter INTEGER := 0;
10  BEGIN
11   FOR name_rec IN name_cur LOOP
12   v_counter := v_counter + 1;
13   ename_tab(v_counter) := name_rec.ename;
14   DBMS_OUTPUT.PUT_LINE ('ename('||v_counter||'): '||
15   ename_tab(v_counter));
16   END LOOP;
17   DBMS_OUTPUT.PUT_LINE ('ename(11): '|| ename_tab(11));
18  EXCEPTION
19   WHEN NO_DATA_FOUND THEN
20   DBMS_OUTPUT.PUT_LINE(SQLERRM);
21  END;
22  /
ename(1):
ename(2): ABC
ename(3): XYZ
ename(4): SMITH
ename(5): ALLEN
ename(6): WARD
ename(7): JONES
ename(8): MARTIN
ename(9): BLAKE
ename(10): CLARK
ORA-01403: no data found

```

Programming using an Array

Array Null and Empty Collection

```
DECLARE
    TYPE integer_type IS TABLE OF INTEGER;
    integer_tab integer_type;

    v_counter integer := 1;
BEGIN
    DBMS_OUTPUT.PUT_LINE (integer_tab(v_counter));
END;
```



```
DECLARE
*
ERROR at line 1:
ORA-06531: Reference to uninitialized collection
ORA-06512: at line 7
```

Introduction to Database Administration

```
DECLARE
  TYPE integer_type IS TABLE OF INTEGER;
  integer_tab integer_type := integer_type();

  v_counter integer := 1;
BEGIN
  DBMS_OUTPUT.PUT_LINE (integer_tab(v_counter));
END;
```

DECLARE

ERROR at line 1:

ORA-06533: Subscript beyond count

ORA-06512: at line 7

TOPIC 146

Examples with Advance Topics

Examples with Advance Topics

Examples with Advance Topics

Use of Records Arrays

Use of PRAGMA Exception

PL/SQL Dynamic SQL

PL/SQL Dynamic SQL - Example-1

PL/SQL Dynamic SQL - Example-2

Examples with Advance Topics

Use of Records Arrays

```

Declare
Cursor c1 is
    select sal
    from emp;
TYPE sal_type IS VARRAY(20) OF Integer;
v_col sal_TYPE := sal_type();
k number := 0;
Begin
    for i in C1 LOOP
        k := k+1;
        v_col.extend; --collection method
        v_col(k) := i.sal;
    end loop;
    for j in 1..4 LOOP
        DBMS_OUTPUT.PUT_LINE('sal is col('||j||') '
        ||v_col(j));
    end loop;
End;

```

Examples with Advance Topics

Use of PRAGMA Exception

```

CREATE OR REPLACE TRIGGER Employee_permit_changes
BEFORE INSERT OR DELETE OR UPDATE ON EMP
DECLARE
    Dummy          INTEGER;
    Not_on_weekends EXCEPTION;
    Nonworking_hours EXCEPTION;
    PRAGMA EXCEPTION_INIT (Not_on_weekends, -4097);
    PRAGMA EXCEPTION_INIT (Nonworking_hours, -4099);
BEGIN
    -- Check for weekends:
    IF (TO_CHAR(Sysdate, 'DAY') = 'SAT' OR
        TO_CHAR(Sysdate, 'DAY') = 'SUN') THEN
        RAISE Not_on_weekends;
    END IF;
    -- Check for work hours (8am to 6pm):
    IF (TO_CHAR(Sysdate, 'HH24') < 8 OR
        TO_CHAR(Sysdate, 'HH24') > 18) THEN
        RAISE Nonworking_hours;
    END IF;
EXCEPTION
    WHEN Not_on_weekends THEN
        Raise_application_error(-20324, 'Might not change '
            || 'employee table during the weekend');
    WHEN Nonworking_hours THEN
        Raise_application_error(-20326, 'Might not change '
            || 'emp table during Nonworking hours');
END;
/

```

Examples with Advance Topics

PL/SQL Dynamic SQL

Dynamic SQL is a programming methodology for generating and running SQL statements at run time.

It is useful when writing general- purpose and flexible programs like ad hoc query systems, when writing programs that must run database definition language (DDL) statements, or when you do not know at compilation time the full text of a SQL statement or the number or data types of its input and output variables.

Examples with Advance Topics

PL/SQL Dynamic SQL – Example-1

Creating a table during run-time.

```

BEGIN
    execute immediate 'create table MyTable('
                                || 'ID CHAR(3),'
                                || 'Name VARCHAR2(50))';
END;
/

```

Note: grant create table to <user> is required

Examples with Advance Topics

PL/SQL Dynamic SQL - Example-2

Set to Null a column at run time

```

CREATE TABLE EMP_temp AS SELECT * FROM EMP;
/

DECLARE
    -- Set to NULL automatically at run time
    a_null CHAR(1);
BEGIN
    EXECUTE IMMEDIATE 'UPDATE EMP_TEMP SET sal = :x'
        USING a_null;
END;
/

```

End

Introduction to Database Administration

TOPIC 147

Concepts of Oracle Instance-1

Concepts of Oracle Instance-1

Concepts of Oracle Instance-1

Introduction

Configuration Option

Instance Structure

Database Instance Framework

Concepts of Oracle Instance-1

Introduction

A database instance is a set of memory structures that manage database files.

A Database is a set of physical files on disk created by the CREATE DATABASE statement.

Introduction

The instance manages its associated data and serves the users of the database.

Every running Oracle database is associated with at least one Oracle database instance. Because an instance exists in memory and a database exists on disk, an instance can exist without a database and a database can exist without an instance.

Concepts of Oracle Instance-1

Configuration Option



During installation, oracle has an option to install instance, database or both

Concepts of Oracle Instance-1

Instance Structure

When an instance is started, Oracle Database allocates a memory area called the system global area (SGA) and starts one or more background processes. The SGA serves various purposes, including the following:

- Maintaining internal data structures that are accessed by many processes and threads concurrently

- Caching data blocks read from disk

- Buffering redo data before writing it to the online redo log files

Concepts of Oracle Instance-1

Instance Structure

Storing SQL execution plans

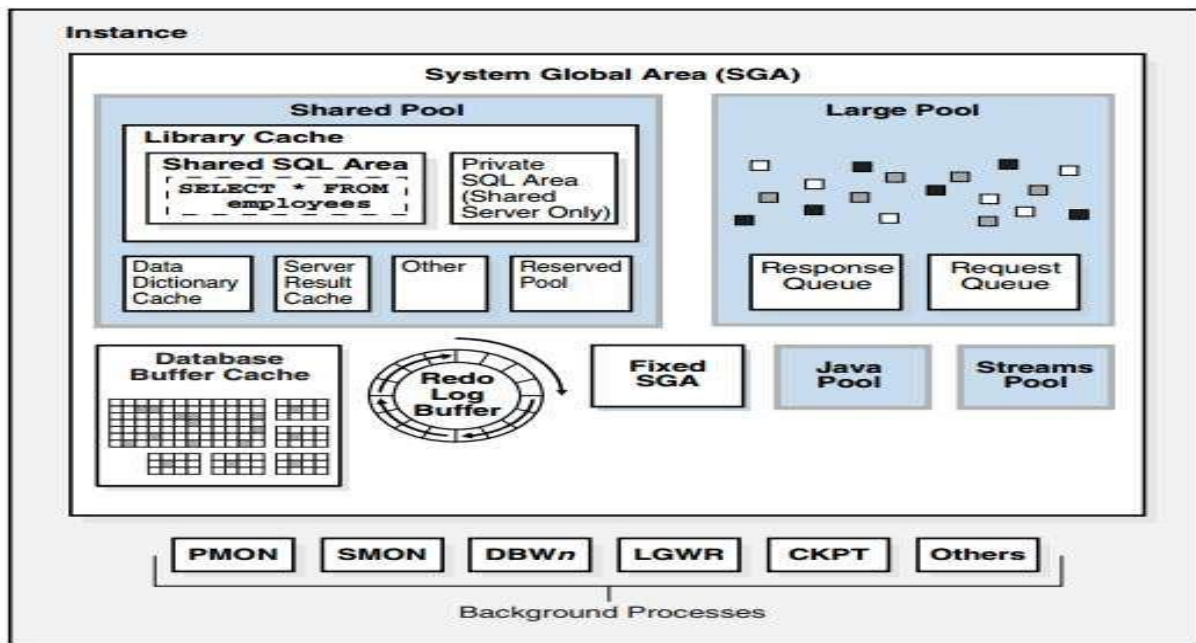
The SGA is shared by the Oracle processes, which include server processes and background processes, running on a single computer. The way in which Oracle processes are associated with the SGA varies according to operating system

Concepts of Oracle Instance-1

Instance Structure

A database instance includes background processes. Server processes, and the process memory allocated in these processes, also exist in the instance. The instance continues to function when server processes terminate.

Concepts of Oracle Instance-1



Database Instance Framework

Introduction to Database Administration

TOPIC 148

Concepts of Oracle Instance-2

Concepts of Oracle Instance-2

Concepts of Oracle Instance-2

Database Instance Configuration

Database Instance Configuration

Database Single & RAC Instance Configuration

Concepts of Oracle Instance-2

Database Instance Configuration

You can run Oracle Database in either of the following mutually exclusive configurations:

Single-instance configuration

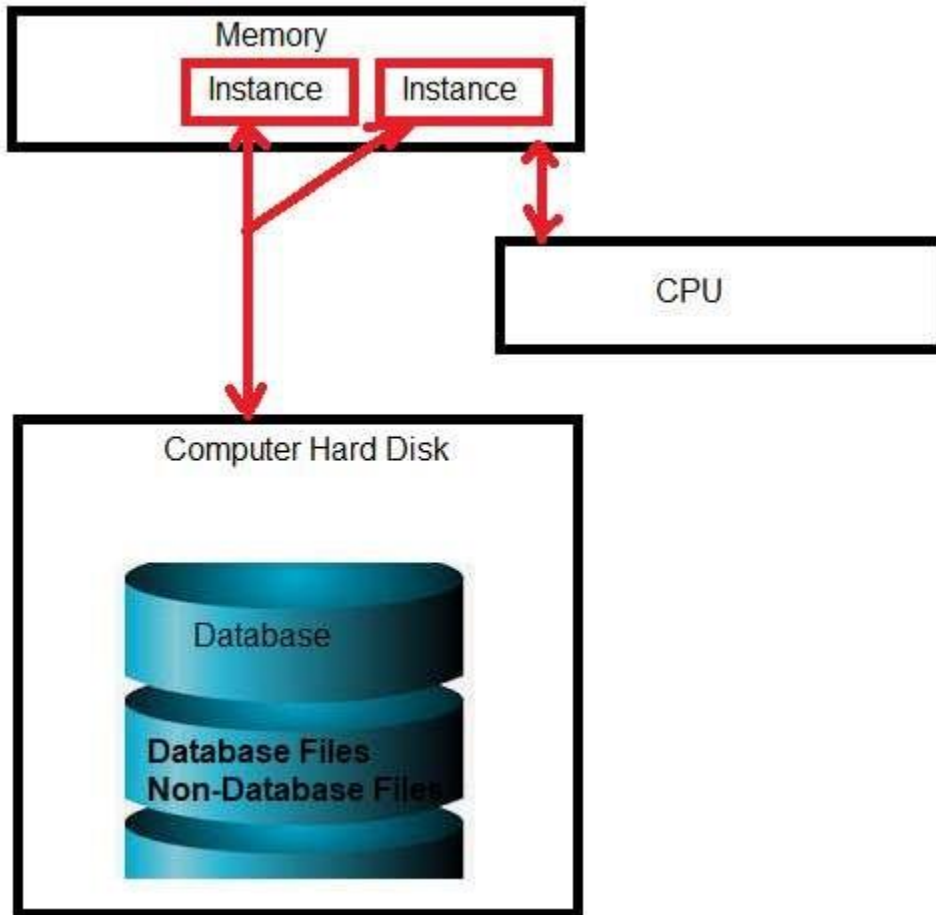
A one-to-one relationship exists between the database and an instance.

Oracle Real Application Clusters (Oracle RAC) configuration

A one-to-many relationship exists between the database and instances.

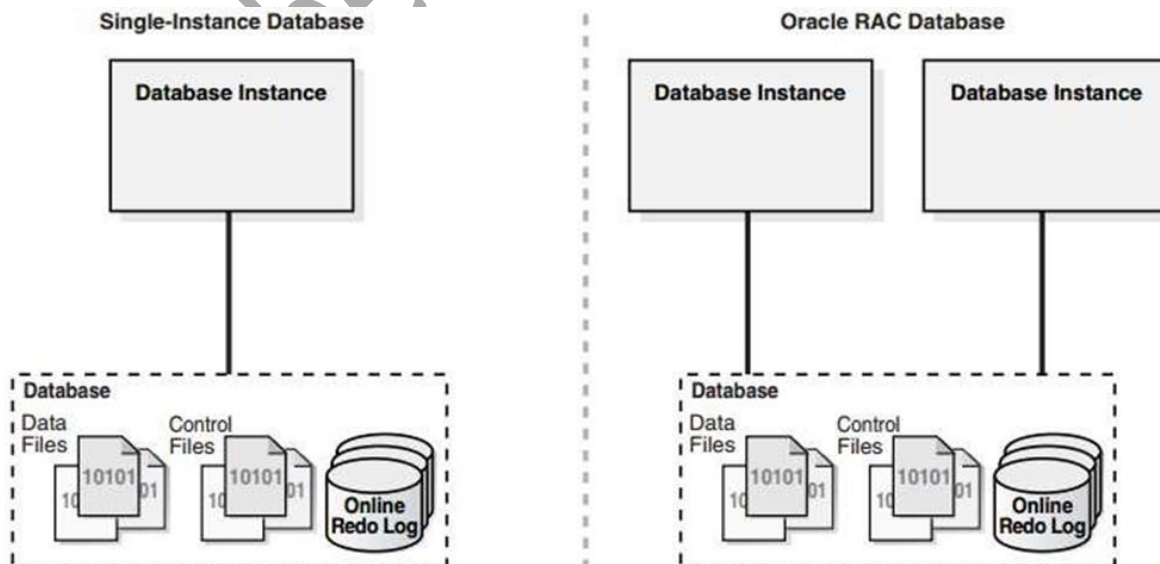
Concepts of Oracle Instance-2

Database Instance Configuration



Concepts of Oracle Instance-2

Database Single & RAC Instance Configuration



TOPIC 149

Concepts of Oracle Instance-3

Concepts of Oracle Instance-3

Concepts of Oracle Instance-3

Database Instance Configuration

Duration of an Instance

Instance Startup

Instance Shutdown & Startup

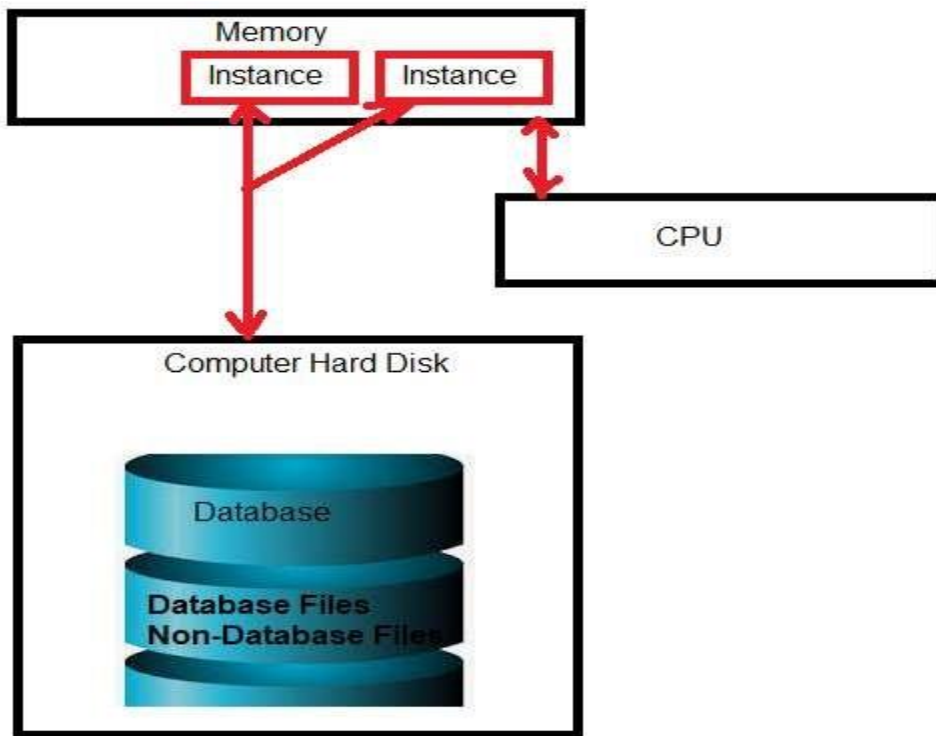
Instance & SID

Oracle Home & SID at Windows OS

Display Instance Name using Data Dictionary View

Concepts of Oracle Instance-3

Database Instance Configuration



Concepts of Oracle Instance-3

Duration of an Instance

An instance begins when it is created with the STARTUP command and ends when it is terminated. During this period, an instance can associate itself with one and only one database. Furthermore, the instance can mount a database only once, close it only once, and open it only once. After a database has been closed or shut down, you must start a different instance to mount and open this database.

Concepts of Oracle Instance-3

Instance Startup

Login with user sys / as sysdba

After shutdown database, run the following:

```
SQL> STARTUP
ORACLE instance started.
```

```
Total System Global Area  468729856 bytes
Fixed Size                  1333556 bytes
Variable Size               440403660 bytes
Database Buffers            16777216 bytes
Redo Buffers                 10215424 bytes
Database mounted.
Database opened.
```

Concepts of Oracle Instance-3

Instance Startup

This query shows the time that the current instance was started:

```
SQL> SELECT
  2  TO_CHAR(STARTUP_TIME, 'MON-DD-RR HH24:MI:SS')
  3  AS "Inst Start Time" FROM V$INSTANCE;

Inst Start Time
-----
SEP-30-20 20:56:54
```

Concepts of Oracle Instance-3

Instance Shutdown & Startup

This query shows the time that the current instance was started:

```

SQL>
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.

Total System Global Area 1071333376 bytes
Fixed Size                  1375792 bytes
Variable Size               696254928 bytes
Database Buffers            369098752 bytes
Redo Buffers                 4603904 bytes
Database mounted.
Database opened.
SQL> SELECT
  2  TO_CHAR(STARTUP_TIME, 'MON-DD-RR HH24:MI:SS')
  3  AS "Inst Start Time" FROM V$INSTANCE;

Inst Start Time
-----
OCT-01-20 08:48:52

```

Concepts of Oracle Instance-3

Instance & SID

The system identifier (SID) is a unique name for an Oracle database instance on a specific host. On UNIX and Linux, Oracle Database uses the SID and Oracle home values to create a key to shared memory.

Also, the SID is used by default to locate the parameter file, which is used to locate relevant files such as the database control files.

Concepts of Oracle Instance-3

Instance & SID

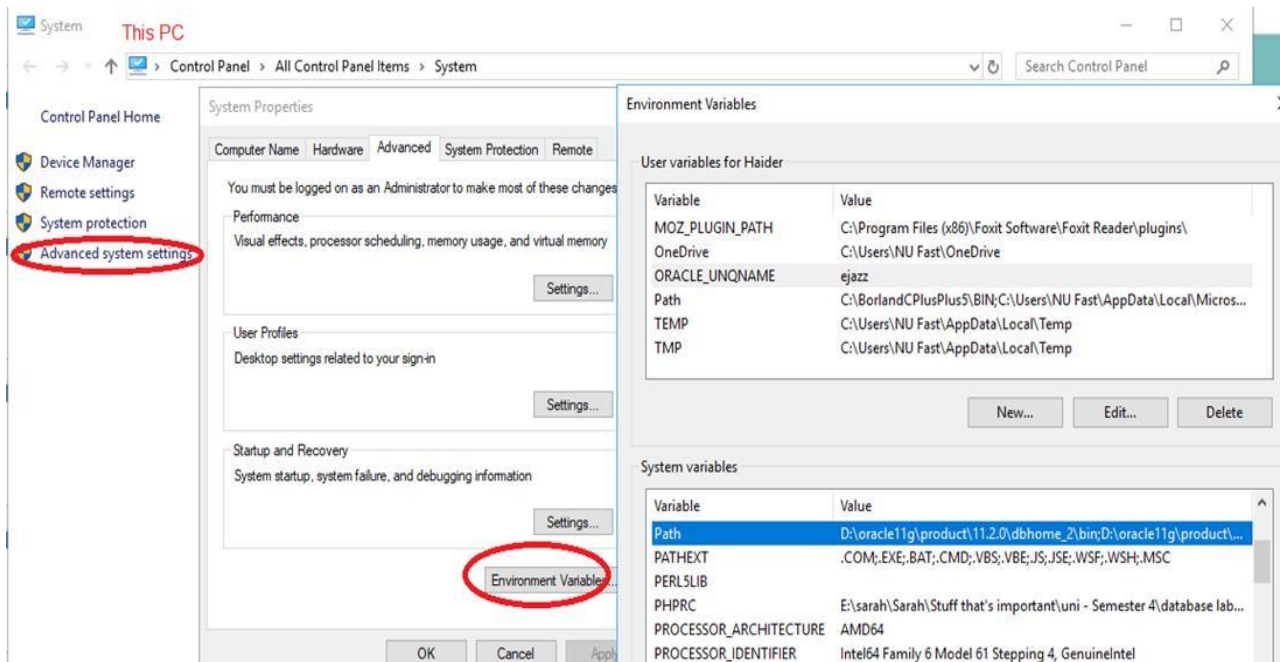
On most platforms, the ORACLE_SID environment variable sets the SID, whereas the ORACLE_HOME variable sets the Oracle home.

When connecting to an instance, clients can specify the SID in an Oracle Net connection or use a net service name.

Oracle Database converts a service name into an ORACLE_HOME and ORACLE_SID.

Concepts of Oracle Instance-3

Oracle Home & SID at Windows OS



Concepts of Oracle Instance-3

Display Instance Name using Data Dictionary View

```
SQL> SELECT TO_CHAR(STARTUP_TIME, 'MON-DD-RR HH24:MI:SS') AS "Inst Start Time",
2 Instance_name , substr(Host_name,1,20)
3 FROM V$INSTANCE;
```

```
Inst Start Time  INSTANCE_NAME  SUBSTR(HOST_NAME,1,2
-----
OCT-01-20 08:48:52  orcl          HAIDER
```

Introduction to Database Administration

TOPIC 150

Instance Startup & Shutdown-1

Instance Startup & Shutdown-1

Instance Startup & Shutdown-1

Overview of Instance and Database Startup

SVRCTL Utility Help

Example - SVRCTL command

Progress from Shutdown & Startup OPEN

Instance Startup

Instance Startup MOUNT

Instance Startup NOMOUNT

Instance Startup & Shutdown-1

Overview of Instance and Database Startup

A database instance provides user access to a database.

In a typical use case, you manually start an instance and then mount and open the database, making it available for users. You can use the SQL*Plus STARTUP command, Oracle Enterprise Manager (Enterprise Manager), or the **SVRCTL utility** to perform these steps.

Instance Startup & Shutdown-1

SVRCTL Utility Help

Command Prompt

```
Microsoft Windows [Version 10.0.17134.1726]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\NU Fast>srvctl -h
Usage: srvctl [-V]
Usage: srvctl add database -d <db_unique_name> -o <oracle_home> [-m <domain>] [-s <start_options>] [-t <stop_options>] [-a "<diskgroup_list>"]
Usage: srvctl config database [-d <db_unique_name> [-a]]
Usage: srvctl start database -d <db_unique_name> [-o <start_options>]
Usage: srvctl stop database -d <db_unique_name> [-o <stop_options>] [-f]
Usage: srvctl status database -d <db_unique_name> [-f] [-v]
Usage: srvctl enable database -d <db_unique_name>
Usage: srvctl disable database -d <db_unique_name>
```

Instance Startup & Shutdown-1

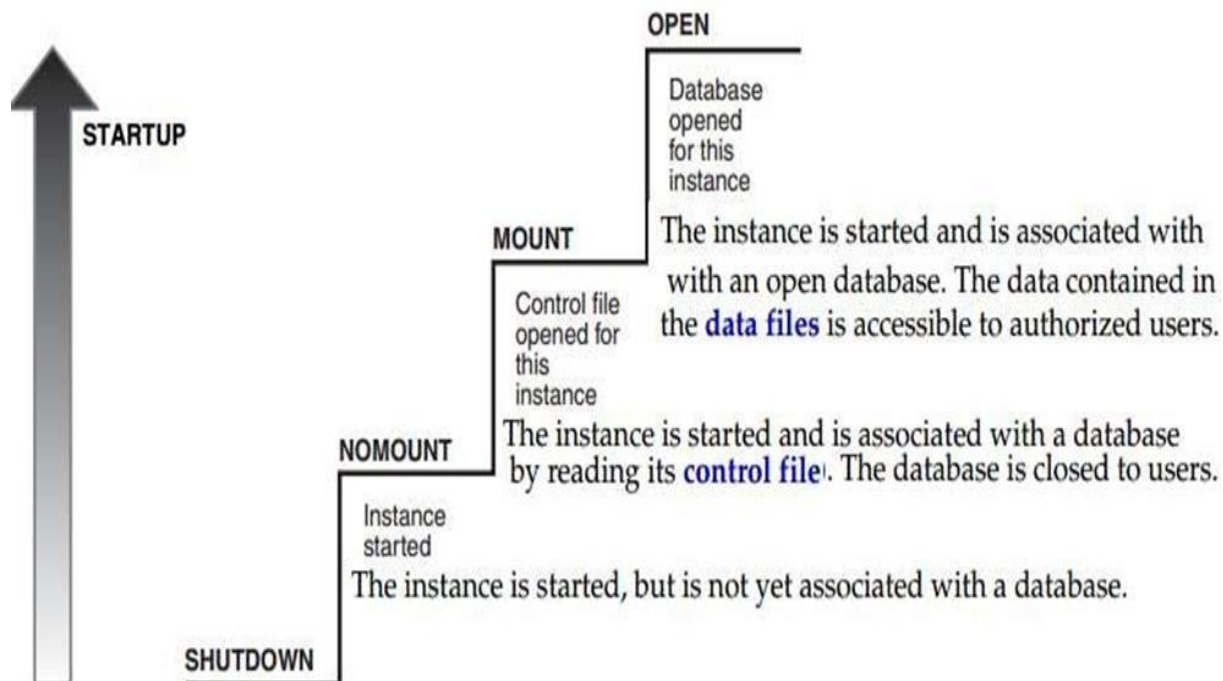
Example - SVRCTL command

Syntax to add a named service to a database with preferred instances in list one and available instances in list two, using basic failover for the available instances:

```
srvctl add service -d crm -s sales -r crm01,crm02 -a crm03
```

Instance Startup & Shutdown-1

Progress from Shutdown & Startup OPEN



Instance Startup & Shutdown-1

Instance Startup

Login with user sys / as sysdba

After shutdown database, run the following:

```
SQL> STARTUP
ORACLE instance started.
```

```
Total System Global Area  468729856 bytes
Fixed Size                  1333556 bytes
Variable Size              440403660 bytes
Database Buffers           16777216 bytes
Redo Buffers                10215424 bytes
Database mounted.
Database opened.
```

Instance Startup & Shutdown-1

Instance Startup

This query shows the time that the current instance was started:

```
SQL> SELECT
  2  TO_CHAR(STARTUP_TIME, 'MON-DD-RR HH24:MI:SS')
  3  AS "Inst Start Time" FROM V$INSTANCE;

Inst Start Time
-----
SEP-30-20 20:56:54
```

Instance Startup & Shutdown-1

Instance Startup MOUNT

First perform shutdown immediate then startup **mount**:

```

SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL>
SQL> startup mount
ORACLE instance started.

Total System Global Area 1071333376 bytes
Fixed Size                  1375792 bytes
Variable Size               696254928 bytes
Database Buffers            369098752 bytes
Redo Buffers                 4603904 bytes
Database mounted.

```

Instance Startup & Shutdown-1

Instance Startup NOMOUNT

First perform shutdown immediate then startup **nomount**:

```

SQL> shutdown immediate
ORA-01109: database not open

Database dismounted.
ORACLE instance shut down.
SQL>
SQL> startup nomount
ORACLE instance started.

Total System Global Area 1071333376 bytes
Fixed Size                  1375792 bytes
Variable Size               696254928 bytes
Database Buffers            369098752 bytes
Redo Buffers                 4603904 bytes
SQL> _

```

End

Database creation and certain backup & recovery operations are performed.

Introduction to Database Administration

TOPIC 151

Instance Startup & Shutdown-2

Instance Startup & Shutdown-2

Instance Startup & Shutdown-2

Connection with Administrator Privileges

Instance Started to perform steps

How Database is Mounted?

How Database is opened?

Read-only Opened Database

Progress from Shutdown & Startup OPEN

Instance Startup & Shutdown-2

Connection with Administrator Privileges

Database startup and shutdown are powerful administrative options that are restricted to users who connect to Oracle Database with administrator privileges.

Normal users do not have control over the current status of an Oracle database.

Instance Startup & Shutdown-2

Connection with Administrator Privileges

Depending on the operating system, one of the following conditions establishes administrator privileges for a user:

The operating system privileges of the user enable him or her to connect using administrator privileges.

Instance Startup & Shutdown-2

Connection with Administrator Privileges

The user is granted the SYSDBA or SYSOPER system privileges and the database uses password files to authenticate database administrators over the network.

When you connect with the SYSDBA system privilege, you are in the schema owned by SYS.

When you connect as SYSOPER, you are in the public schema.

Instance Startup & Shutdown-2

Instance Started to perform steps

Searches for a server parameter file in a platform-specific default location and, if not found, for a text initialization parameter file(specifying STARTUP with the SPFILE or PFILE parameters overrides the default behavior)

Reads the parameter file to determine the values of initialization parameters

Instance Startup & Shutdown-2

Instance Started to perform steps

Allocates the SGA based on the initialization parameter settings

Starts the Oracle background processes

Opens the alert log and trace files and writes all explicit parameter settings to the alert log in valid parameter syntax

Instance Startup & Shutdown-2

How Database is Mounted?

The instance mounts a database to associate the database with this instance. To mount the database, the instance obtains the names of the database control files specified in the CONTROL_FILES initialization parameter and opens the files.

Instance Startup & Shutdown-2

How Database is Mounted?

Oracle Database reads the control files to find the names of the data files and the online redo log files that it will attempt to access when opening the database.

In a mounted database, the database is closed and accessible only to database administrators.

Instance Startup & Shutdown-2

How Database is Mounted?

Administrators can keep the database closed while completing specific maintenance operations. However, the database is not available for normal operations

Instance Startup & Shutdown-2

How Database is opened?

Opening a mounted database makes it available for normal database operations

Opens the online data files in tablespaces other than undo tablespaces

Acquires an undo tablespace

Opens the online redo log files

By default, the database opens in read/write mode.
Instance Startup & Shutdown-2

Read-only Opened Database

Data files can be taken offline and online. However, you cannot take permanent tablespaces offline.

Offline data files and tablespaces can be recovered.

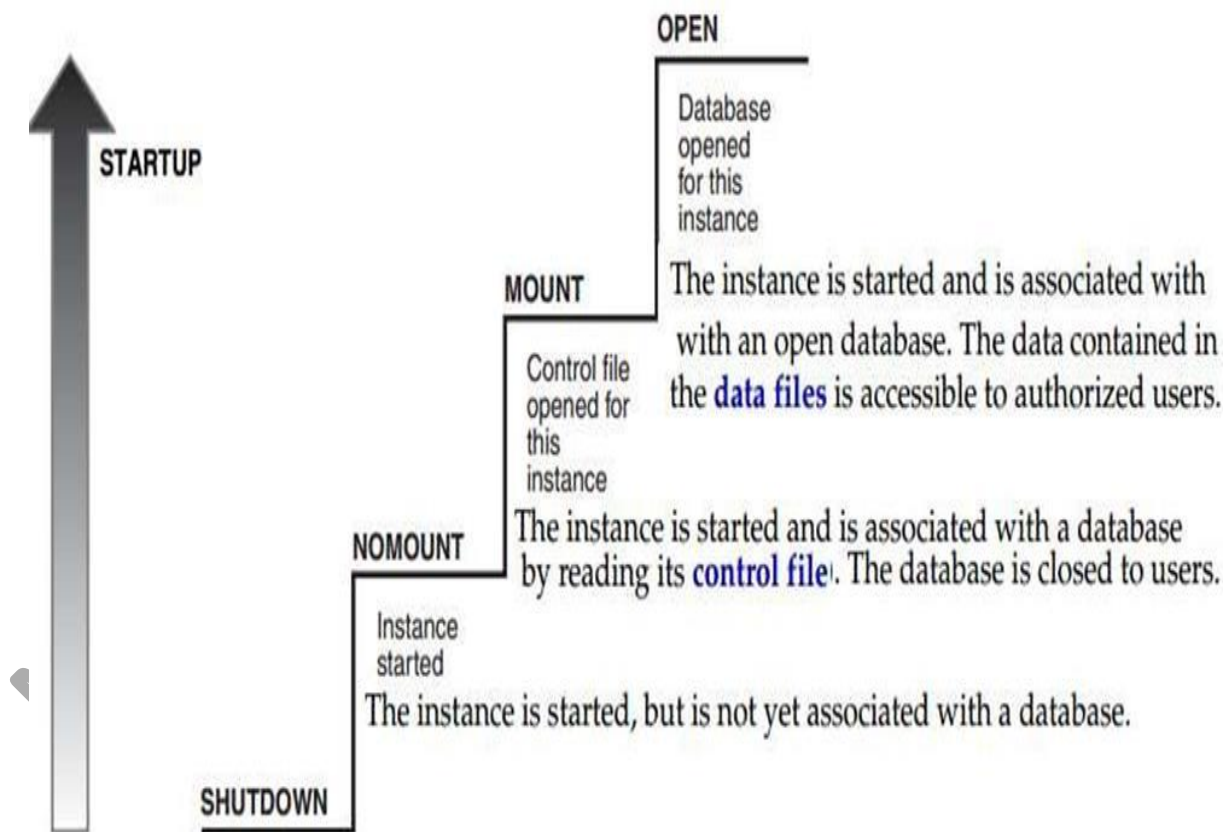
The control file remains available

Temporary tablespaces created with the CREATE TEMPORARY TABLESPACE statement are read/write.

Writes to operating system audit trails, trace files, and alert logs can continue.

Instance Startup & Shutdown-2

Progress from Shutdown & Startup OPEN



TOPIC 152

Instance Startup & Shutdown-3

Instance Startup & Shutdown-3

Instance Startup & Shutdown-3

Database with Shutdown Instance

Progress from OPEN to Shutdown

Steps to perform Database Shutdown

Shutdown ABORT

Shutdown IMMEDIATE

Shutdown TRANSACTIONAL

Shutdown NORMAL

Shutdown Modes Summary

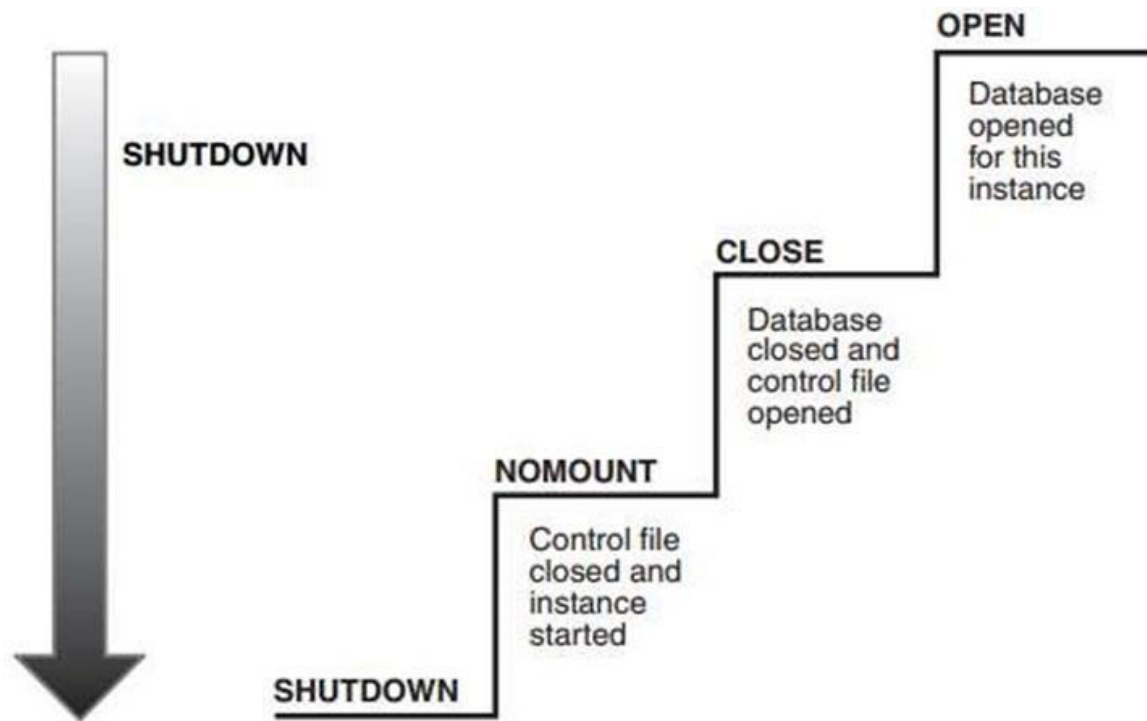
Instance Startup & Shutdown-3

Database with Shutdown Instance

In a typical use case, you manually shut down the database, making it unavailable for users while you perform maintenance or other administrative tasks.

You can use the SQL*Plus SHUTDOWN command or Enterprise Manager to perform these steps.

Instance Startup & Shutdown-3



Progress from OPEN to Shutdown

Instance Startup & Shutdown-3

Steps to perform Database Shutdown

Database closed

The database is mounted, but online data files and redo log files are closed.

Database unmounted

The instance is started, but is no longer associated with the control file of the database.

Database instance shut down

The database instance is no longer started.

Instance Startup & Shutdown-3

Steps to perform Database Shutdown

Oracle Database does not go through all of the preceding steps in an instance failure or SHUTDOWN ABORT, which immediately terminates the instance.

Instance Startup & Shutdown-3

Shutdown ABORT

Used in emergency situation. Shutting down instance may take substantially longer.

SHUTDOWN ABORT does not checkpoint the open data files

Instance Startup & Shutdown-3

Shutdown IMMEDIATE

Fastest way to manage. Active transactions are terminated and uncommitted changes are rolled back.

Instance Startup & Shutdown-3

Shutdown TRANSACTIONAL

This mode prevents users from starting new transactions, but waits for all current transactions to complete before shutting down.

Instance Startup & Shutdown-3

Shutdown NORMAL

This is the default mode of shutdown. The database waits for all connected users to disconnect before shutting down.

Instance Startup & Shutdown-3

Shutdown Modes Summary

A database administrator with SYSDBA or SYSOPER privileges can shut down the database using the SQL*Plus SHUTDOWN command or Enterprise Manager.

Database Behavior	ABORT	IMMEDIATE	TRANSACTIONAL	NORMAL
Permits new user connections	No	No	No	No
Waits until current sessions end	No	No	No	Yes
Waits until current transactions end	No	No	Yes	Yes
Performs a checkpoint and closes open files	No	Yes	Yes	Yes

TOPIC 153

Overview of Checkpoints

Overview of Checkpoints

Overview of Checkpoints

What is a Checkpoint?

Purpose of Checkpoint?

Recovery without Checkpoints

Recovery using Checkpoints

Overview of Checkpoints

What is a Checkpoint?

A checkpoint is a crucial mechanism in consistent **database shutdowns, instance recovery, and Oracle Database operation** generally.

A data structure that indicates the checkpoint position, which is the **SCN** in the redo stream where instance recovery must begin. The checkpoint position is determined by the oldest dirty buffer in the database buffer cache.

Overview of Checkpoints

What is Checkpoint?

The checkpoint position acts as a pointer to the **redo stream** and is stored in the **control file** and in each data file header.

The writing of modified database buffers in the **database buffer cache** to **disk**.

Whenever checkpoint happens all the dirty buffers of DB buffer cache (volatile storage) moves into data files (Non volatile storage)

Overview of Checkpoints

Purpose of Checkpoint?

Oracle Database uses checkpoints to achieve the following goals:

Reduce the time required for recovery in case of an instance or media failure

Ensure that dirty buffers in the buffer cache are written to disk regularly

Ensure that all committed data is written to disk during a consistent shutdown

Establish data consistency

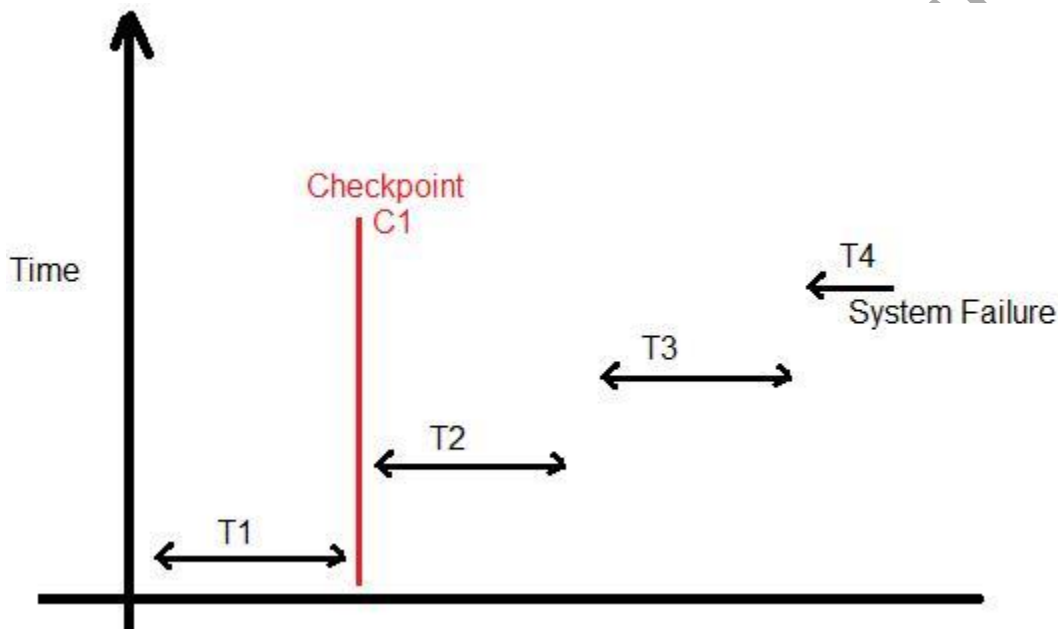
Overview of Checkpoints

Recovery without Checkpoints

Extra time is wasted in processing the entire log file Redoing of unnecessary transactions

Overview of Checkpoints

Recovery using Checkpoints



Checkpoint is a mechanism in which all the previous logs are removed from the system and are stored permanently in the storage disk.

Introduction to Database Administration

TOPIC 154

Initiate Checkpoints

Initiate Checkpoints

Initiate Checkpoints

When to initiate Checkpoint

Checkpoint Process

Thread Checkpoints Type

Tablespace and Data File Checkpoints

Type

Incremental Checkpoints Type

Checkpoints Processor and Database files

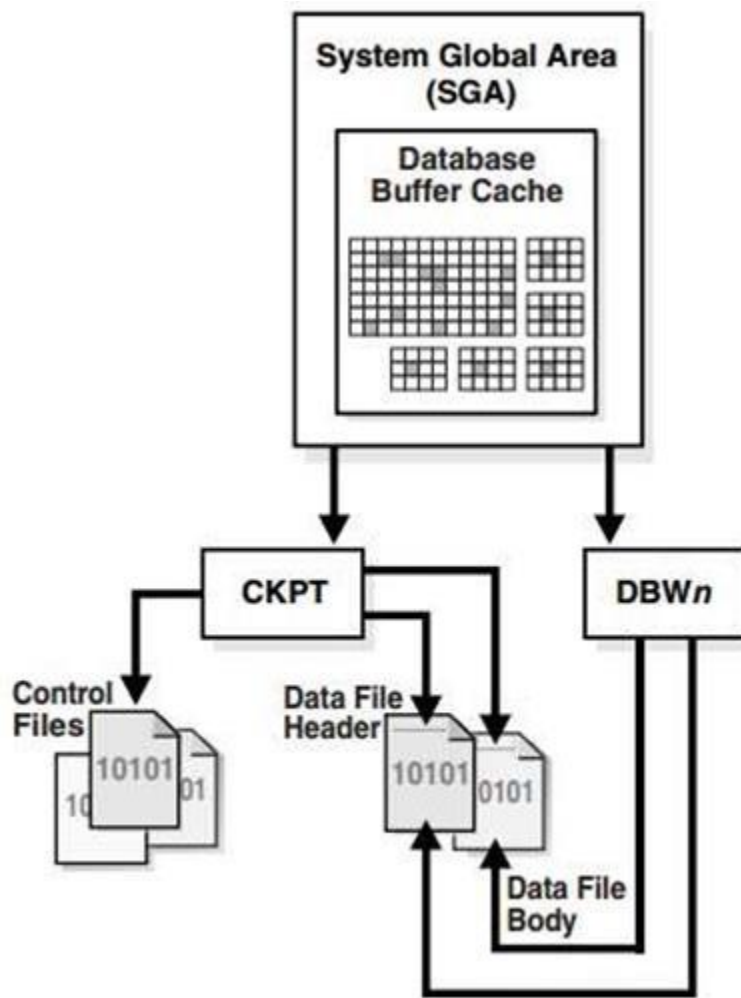
Initiate Checkpoints

When to initiate Checkpoint

The checkpoint process (CKPT) is responsible for writing checkpoints to the **data file headers** and **control file**.

Checkpoints occur in a variety of situations

Initiate Checkpoints



Checkpoint Process

The checkpoint process (CKPT) updates the control file and data file headers with checkpoint information and signals DBWn to write blocks to disk. Checkpoint information includes the checkpoint position, SCN, location in online redo log to begin recovery, and so on.

Initiate Checkpoints

Thread Checkpoints Type

The database writes to disk all buffers modified by redo in a specific thread before a certain target. The set of thread checkpoints on all instances in a database is a database checkpoint.

Thread checkpoints occur in the following situations:

Consistent database shutdown

ALTER SYSTEM CHECKPOINT statement

Online redo log switch

ALTER DATABASE BEGIN BACKUP statement

Initiate Checkpoints

Tablespace and Data File Checkpoints Type

The database writes to disk all buffers modified by redo before a specific target. A tablespace checkpoint is a set of data file checkpoints, one for each data file in the tablespace. These checkpoints occur in a variety of situations, including making a **tablespace read-only** or **taking it offline normal, shrinking a data file, or executing**

ALTER TABLESPACE BEGIN BACKUP;

Initiate Checkpoints

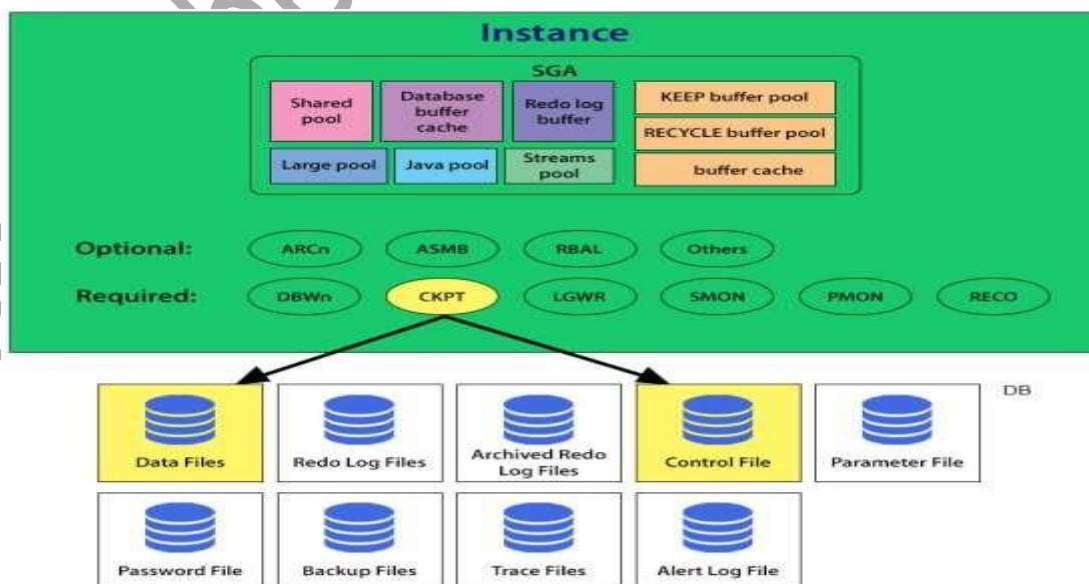
Incremental Checkpoints Type

An incremental checkpoint is a type of thread checkpoint partly intended to avoid writing large numbers of blocks at online redo log switches. DBWn checks at least every **three seconds** to determine whether it has work to do.

When DBWn writes dirty buffers, it advances the checkpoint position, causing CKPT to write the checkpoint position to the control file, but **not to the data file headers**.

Initiate Checkpoints

Checkpoints Processor and Database files



TOPIC 155

Checkpoints Commands Samples

Checkpoints Commands Samples

Checkpoints Commands Samples

Checkpoints Commands Samples

When will a Checkpoint occur

Checkpoint Occurring Command

Data Dictionary View for Checkpoint

SCN is Changed After another Checkpoint

SCN is not Changed After Switch Logfile

Checkpoints Commands Samples

When will a Checkpoint occur

At each switch of the redo log files

Once the number of seconds defined in the LOG_CHECKPOINT_TIMEOUT is reached Once the current redo log file reaches the size

LOG Checkpoint Interval x size of IO-OS blocks, Example $10000 \times 512 = 5\text{MB}$

Directly by the Alter System switch Logfile command

Directly by the Alter System Checkpoint command

Checkpoints Commands Samples

Checkpoint Occurring Command

Run the following command with sys user

Alter system checkpoint;

Checkpoints Commands Samples

Data Dictionary View for Checkpoint

```
SQL> select Checkpoint_change#, checkpoint_time, sum(Bytes)/1024/1024 DataSize ,
2 from v$datafile
3 group by Checkpoint_change#, checkpoint_time;
```

CHECKPOINT_CHANGE#	CHECKPOIN	DATASIZE
19352187	02-OCT-20	18749.75
19352187	02/10/2020 10:06:26 AM	305
19352187	02/10/2020 10:06:26 AM	8.75
19352187	02/10/2020 10:06:26 AM	1000
19352187	02/10/2020 10:06:26 AM	1000

Checkpoints Commands Samples

SCN is Changed After another Checkpoint

```
19352187 02-OCT-20 18749.75
```

```
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.
```

```
Total System Global Area 1071333376 bytes
Fixed Size 1375792 bytes
Variable Size 696254928 bytes
Database Buffers 369098752 bytes
Redo Buffers 4603904 bytes
Database mounted.
Database opened.
```

```
SQL> select Checkpoint_change#, TO_CHAR(checkpoint_time, 'DD/MM/YYYY HH:MI:SS AM') CHKP_Time,
2 Bytes/1024/1024
3 from v$datafile;
```

CHECKPOINT_CHANGE#	CHKP_TIME	BYTES/1024/1024
19352722	02/10/2020 10:19:42 AM	710
19352722	02/10/2020 10:19:42 AM	710
19352722	02/10/2020 10:19:42 AM	305
19352722	02/10/2020 10:19:42 AM	8.75
19352722	02/10/2020 10:19:42 AM	1000
19352722	02/10/2020 10:19:42 AM	1000
19352722	02/10/2020 10:19:42 AM	500

Checkpoints Commands Samples

SCN is not Changed After Switch Logfile

```

19353247 02/10/2020 10:25:15 AM      1024
19353247 02/10/2020 10:25:15 AM      800

22 rows selected.

SQL>
SQL> Alter System switch Logfile;

System altered.

SQL> select Checkpoint_change#, checkpoint_time, sum(Bytes)/1024/1024 DataSize
  2  from v$datafile
  3  group by Checkpoint_change#, checkpoint_time;

CHECKPOINT_CHANGE# CHECKPOIN  DATASIZE
-----
19353247 02-OCT-20   18749.75

```

Introduction to Database Administration

TOPIC 156

Instance Recovery

Instance Recovery

Instance Recovery

Overview of Instance Recovery

Purpose of Instance Recovery

How Instance Recovery is Performed?

Instance Recovery

Overview of Instance Recovery

Instance recovery is the process of applying records in the **online redo log** to **data files** to reconstruct changes made after the most recent checkpoint.

Instance recovery occurs automatically when an administrator attempts to open a database that was previously shut down inconsistently.

Instance Recovery

Overview of Instance Recovery

Instance recovery occurs automatically when an administrator attempts to open a database that was previously shut down inconsistently.

```
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.

Total System Global Area 1071333376 bytes
Fixed Size                  1375792 bytes
Variable Size               696254928 bytes
Database Buffers            369098752 bytes
Redo Buffers                 4603904 bytes
Database mounted.
Database opened.
```

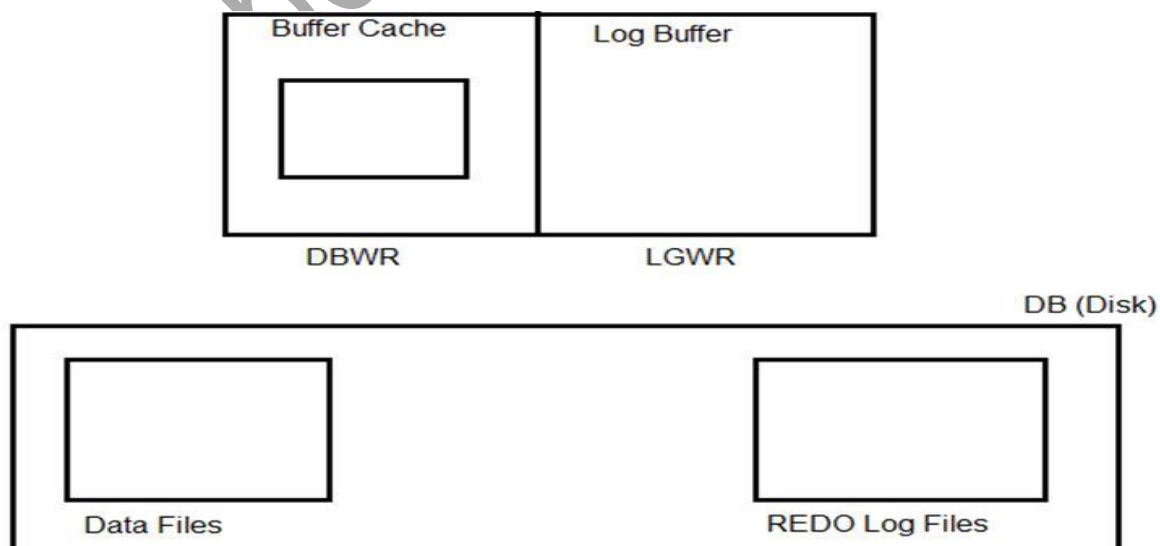
Instance Recovery

Purpose of Instance Recovery

Instance recovery ensures that the database is in a consistent state after an instance failure. The files of a database can be left in an inconsistent state because of how Oracle Database manages database changes.

Instance Recovery

How Instance Recovery is Performed?



TOPIC 157

Instance Recovery Protocol

Instance Recovery Protocol

Instance Recovery

Situations for Instance Recovery

Performing Instance Recovery

Importance of CKPT for Instance Recovery

Instance Recovery Protocol

Situations for Instance Recovery

If an instance of an open database fails, either because of a SHUTDOWN ABORT statement or abnormal termination, then the following situations can result:

Data blocks committed by a transaction are not written to the data files and appear only in the online redo log. These changes must be reapplied to the database.

Instance Recovery Protocol

Situations for Instance Recovery

The data files contains changes that had not been committed when the instance failed. These changes must be rolled back to ensure transactional consistency.

Instance recovery uses only online redo log files and current online data files to synchronize the data files and ensure that they are consistent

Instance Recovery Protocol

Performing Instance Recovery

Whether instance recovery is required depends on the state of the redo threads. A redo thread is marked open in the control file when a database instance opens in read/write mode, and is marked closed when the instance is shut down consistently.

If redo threads are marked open in the control file, but no live instances hold the thread enqueues corresponding to these threads, then the database requires instance recovery.

Instance Recovery Protocol

Performing Instance Recovery

Oracle Database performs instance recovery automatically in the following situations:
The database opens for the first time after the failure of a single-instance database or all instances of an Oracle RAC database. This form of instance recovery is also called crash recovery. Oracle Database recovers the online redo threads of the terminated instances together.
Instance Recovery Protocol

Performing Instance Recovery

Some but not all instances of an Oracle RAC database fail. Instance recovery is performed automatically by a surviving instance in the configuration.

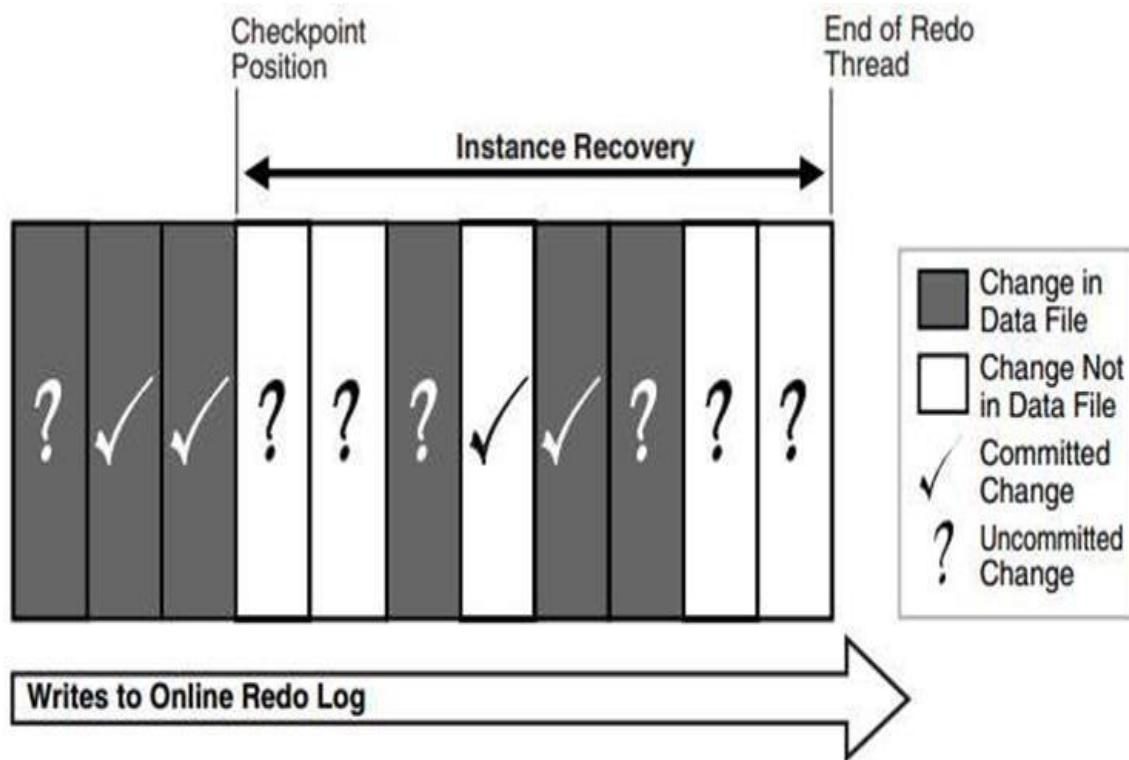
The SMON background process performs instance recovery, applying online redo automatically. No user intervention is required
Instance Recovery Protocol

Importance of CKPT for Instance Recovery

Instance recovery uses checkpoints to determine which changes must be applied to the data files. The checkpoint position guarantees that every committed change with an SCN lower than the checkpoint SCN is saved to the data files.
Instance Recovery Protocol

Importance of CKPT for Instance Recovery

During instance recovery, the database must apply the changes that occur



between the checkpoint position and the end of the redo thread.
Introduction to Database Administration

TOPIC 158

Managing Recovery using MS SQL Server
Managing Recovery using MS SQL Server

MS SQL Server's Recovery

Restore & Recovery Overview
Overview of Restore Scenarios
Steps to Restore a Database
Accelerated Database Recovery
Steps to Restore a Database
Decision to adopt ADR
To View & Change the Recovery Model
- Transact SQL
Without ADR, DB Recovery Follows ARIES
DB Recovery Follows ARIES using ADR

Managing Recovery using MS SQL Server

Restore & Recovery Overview

To recover a SQL Server database from a failure, a database administrator has to restore a set of SQL Server backups in a logically correct and meaningful restore sequence.:

The database (a complete database restore)

The data file (a file restore)

The data page (Bulk load, restore individual DB)

Managing Recovery using MS SQL Server

Overview of Restore Scenarios

A *restore scenario* in SQL Server is the process of restoring data from one or more backups and then recovering the database. The supported restore scenarios depend on the recovery model of the database and the edition of SQL Server.

Complete DB Restore

File Restore (individual files)

Managing Recovery using MS SQL Server

Steps to Restore a Database

To perform a database restore, the Database Engine executes three steps:

Creates the database and transaction log files if they do not already exist.

Copies all the data, log, and index pages from the backup media of a database to the database files.

Applies the transaction log in what is known as the [recovery process](#).

Managing Recovery using MS SQL Server

Accelerated Database Recovery

Accelerated Database Recovery (ADR) improves database availability, especially in the presence of long running transactions, by redesigning the SQL database engine recovery process. ADR is new in SQL Server 2019.

ADR is available for Azure SQL Database, Azure SQL Managed Instance and Azure Synapse SQL.

Managing Recovery using MS SQL Server

Steps to Restore a Database

To perform a file restore, the Database Engine executes two steps:

Creates any missing database file(s).
Copies the data from the backup devices to the database file(s).

If you restore a file, you cannot recover it and bring it online until it has been rolled far enough forward to be consistent with the database.

Managing Recovery using MS SQL Server

Decision to adopt ADR

The following types of customers should consider enabling ADR:

Customers that have workloads with long running transactions.

Customers that have seen cases where active transactions are causing the transaction log to grow significantly.

Customers that have experienced long periods of database unavailability due to SQL Server long running recovery (such as unexpected SQL Server restart or manual transaction rollback).

Managing Recovery using MS SQL Server

To View & Change the Recovery Model - Transact SQL

Connect to the Database Engine. From the Standard bar, click New Query.

SQL

```
SELECT name, recovery_model_desc
FROM sys.databases
WHERE name = 'model' ;

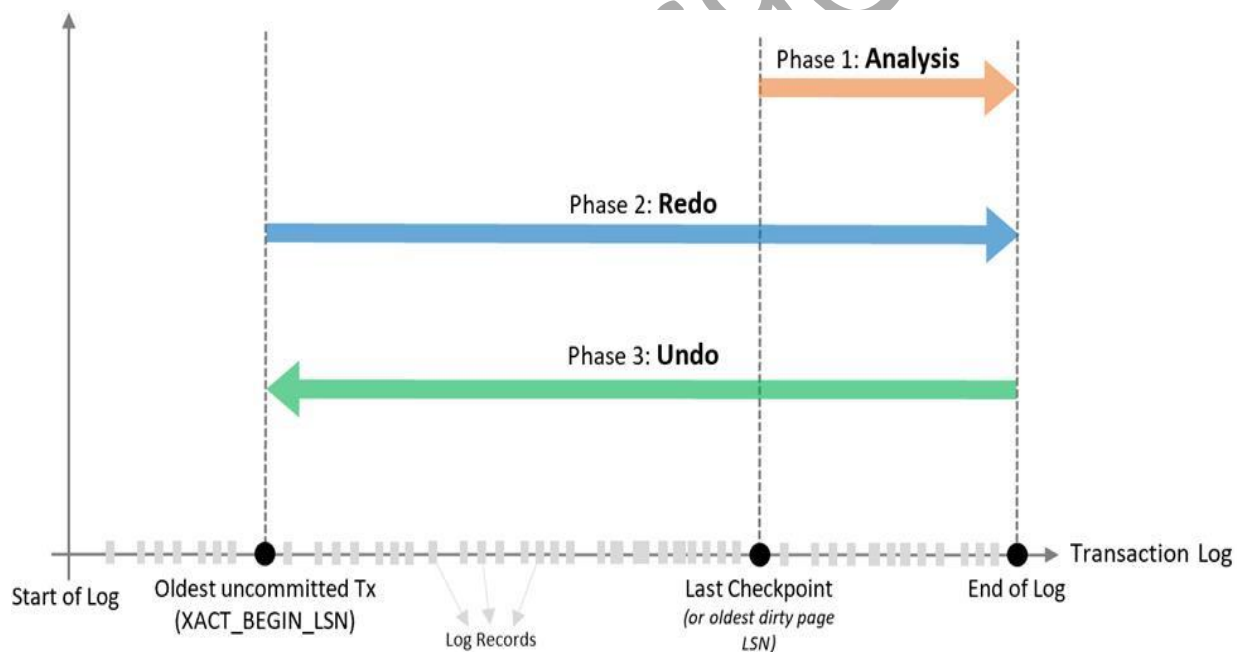
GO
```

SQL

```
USE [master] ;  
ALTER DATABASE [model] SET RECOVERY FULL ;
```

Copy and paste the following example into the query window and click Execute. This example shows how to query the [sys.databases](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-databases-catalog-view) catalog view to learn the recovery model of the model database.

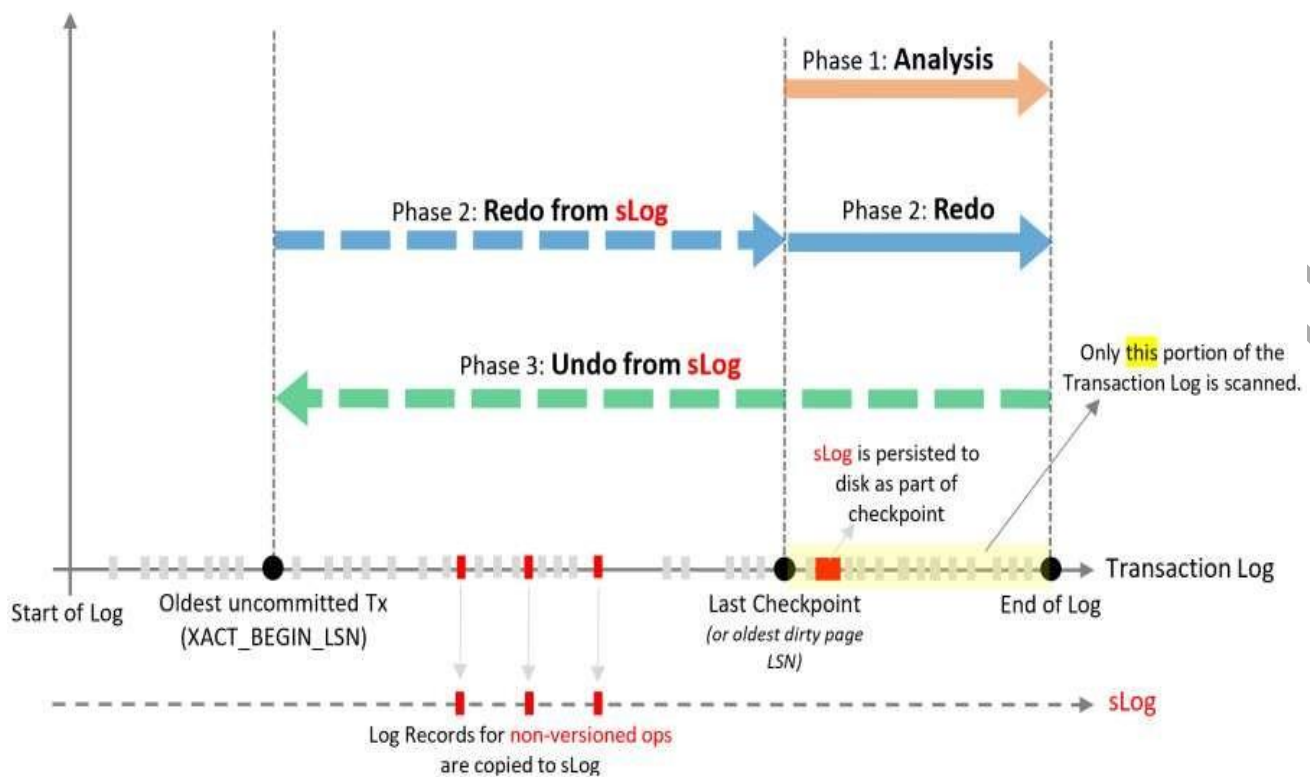
Managing Recovery using MS SQL Server



Without ADR, DB Recovery Follows ARIES

[docs.microsoft.com](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-databases-catalog-view)

Managing Recovery using MS SQL Server



DB Recovery Follows ARIES using ADR

docs.microsoft.com

Introduction to Database Administration

TOPIC 159

Phases of Instance Recovery using Oracle

Phases of Instance Recovery

Phases of Instance Recovery

Introduction

Rolling Forward

After Roll Forward

Rolling Forward & Rolling Backward

Phases of Instance Recovery

Introduction

The first phase of instance recovery is called cache recovery or rolling forward, and involves reapplying all of the changes recorded in the online redo log to the data files.

Because rollback data is recorded in the online redo log, rolling forward also regenerates the corresponding undo segments.

Phases of Instance Recovery

Rolling Forward

Rolling forward proceeds through as many online redo log files as necessary to bring the database forward in time.

After rolling forward, the data blocks contain all committed changes recorded in the online redo log files. These files could also contain uncommitted changes that were either saved to the data files before the failure, or were recorded in the online redo log and introduced during cache recovery.

Phases of Instance Recovery

After Roll Forward

After the roll forward, any changes that were not committed must be undone. Oracle Database uses the checkpoint position, which guarantees that every committed change with an SCN lower than the checkpoint SCN is saved on disk.

Oracle Database applies undo blocks to roll back uncommitted changes in data blocks that were written before the failure or introduced during cache recovery. This phase is called rolling back or transaction recovery.

Phases of Instance Recovery

Rolling Forward & Rolling Backward

