**PL / SQL Fundamentals - I**

* Declaration of variable maximum length is 30 characters, and first letter should be a letter not a number and not a reserved word (do not use &, - / space) (use $, #, \_).
* SET SERVEROUTPUT ON / OFF; (If not use this the output message will not display).

DBMS\_OUTPUT.PUT\_LINE (' output Message')

**PLSQL DATA TYPES**

**SIMPLE DATA TYPES**

* Integer Data Types (BINARY\_INTEGER / PLS\_INTEGER AND (SIMPLE\_INTEGER introduced in Oracle 11g)
* Floating Data Types point allows for decimals.

(BINARY\_FLOAT / BINARY\_DOUBLE and NUMBER (Precision, scale) Precision 1 to 38 and scale -84 to 127)

* Character Data Types
* (CHAR(n) - It is a fixed Length character data from 1 to 32.767
* VARCHAR2(n) It is varying length character data from 1 to 32,767)

**PREDIFINED DATA TYPES**

* ROWID - Identifies a physical row in a table.
* UROWID - Identifies a physical or logical row in a table.

(when compared ROWID and UROWID its better to use UROWID)

* BOOLEAN - BOOLEAN have True / False / Null (By default it is Null)
* DATE - Date / Time
* **DEFAULT** and **:=** assignment operator both are same.

(The variable value is default to NULL (when not mentioned DEFAULT / ASSIGENMENT or etc.,))

* **CONSTANT** - Must use the CONSTANT keyword before the type of declaration

ex - nMinSalary CONSTANT NUMBER(6) := 20000

**COMPLEX DATA TYPES**

* **%TYPE** - Declare vSalary SchemaName.TableName.Field%TYPE;

Ex - Declare vAvgSalary Salary%TYPE;

* **%ROWTYPE** - Declare vEmpRec SchemaName.TableName%ROWTYPE;

(Need not to declare separate variables for each filed while retrieving the data from the same table)

ex - Declare x\_emp vEmpRec SchemaName.TableName%ROWTYPE;

Select empName, gender, salary

into vEmpRec.empName, vEmpRec.gender, vEmpRec.salary

from Employee where empid = 112233;

**dbms\_output.ouput\_line**(vEmpRec.EmpName || ' salary is ' || vEmpRec.salary ||')

* **TYPE..TABLE** - Declare TYPE vEmpIDarray IS TABLE OF employee.empid%TYPE

INDEX BY SIMPLE\_INTEGER;

* **TYPE..RECORD** - It is a user defined data type allows more flexibility than the %ROWTYPE;

Declare TYPE EmpRecord IS RECORD (eid employee.EmpId%TYPE,

ename employee.EmpName%TYPE, bonus number(6));

* **||** two pipe symbol is concatenation operator.
* **q'!**....**!'** this syntax is used treat as a string (even if we use reserved words like single quote ', &, etc.,)
* EmpSalary := **&enter\_salary** (it will ask the user to enter the salary)
* sysdate + 10 (it will add 10 days to system date)
* Boolean (ex - Declare bCheck BOOLEAN :=FALSE; bCheck := (1000 > 100) it returns True / False.
* Sample functions UPPER / LOWER / MAX / ROUND

ex - ROUND(6750, -3) result = 7000

ROUND(6750, -2) result = 6800

ROUND(6750, 0) result = 6750

ROUND(6750.60, 1) result = 6751

* If **SELECT...INTO** statement does not return any rows, then the NO\_DATA\_FOUND exception is returned.

(before going to use **INTO** statement, first check with the table is data having or not.).

* Update statement has no restrictions and no exceptions are thrown (whether they apply to one or many rows or no rows at all)
* **Create** **Sequence** **EmpIDSequence**;

ex - EmpIDSequence.NEXTVAL or

SELECT EmpIDSequence.NEXTVAL INTO newEmpID FROM dual; or

newEmpID := EmpIDSequence.NEXTVAL;

**Note:** Before dropping the table, drop the Sequence also otherwise table will Drop but Sequence still lives

syntax - DROP SEQUENCE SequenceName)

**Note:** even if we delete the record, the sequence will not repeat the value, Sequence will always display a new number.

**Logic Control and Branching**

* GOTO - GOTO **ABC**; label declaration syntax to use GOTO Statement <<**ABC**>>

structures

LOOP - start point

EXIT / EXIT WHEN - to abort the loop

CONTINUE / CONTINUE WHEN - to exit the current iteration of the loop

END LOOP - end point.

ex - FOR I IN 1..1000 LOOP ------- END LOOP;

ex - FOR I IN REVERSE 1..1000 LOOP ------- END LOOP;

* **Indefinite Loop**

ex - LOOP -------- EXIT WHEN condition... ------ END LOOP:

* **Conditional Loop**

ex - WHILE condition.. LOOP ----- END LOOP;

* **Nested Loops** - Outer Loop and inner loop

IF THE ELSE structures

(ex - IF ----- THEN ELSIF ------ THEN -------- ELSE ------- END IF;)

* **CASE structures**

ex - CASE WHEN condition.... THEN -------;

WHEN condition.... THEN -------;

ELSE ------;

END CASE;

**EXCEPTIONS**

Trapping Exceptions

Identifying Exception Details

User\_Defined Exceptions

Exception Functions

Implicit Cursor Attributes

**Trapping Exceptions**

WHEN no\_data\_found THEN

WHEN no\_data\_found or two\_many\_rows THEN

WHEN OTHERS THEN

**Identify Exception Details (system defined exceptions)**

NO\_DATA\_FOUND, CURSOR\_ALREADY\_OPEN

INVALID\_CURSOR, CASE\_NOT\_FOUND

INVALID\_NUMBER, VALUE\_ERROR

ZERO\_DIVIDE, PROGRAM\_ERROR

STORAGE\_ERROR, TIMOUT\_ON\_RESOURCE

**EXCEPTION\_INIT**

SQLCODE / SQLERRM

SQL DML (implicit cursor)

**User Defined Exceptions**

ex - Declare not\_null\_constraint EXCEPTION

PRAGMA EXCEPTION\_INIT (not\_null\_constraint, -1407);

BEGIN

-------

EXCEPTION

WHEN not\_null\_constraint THEN

DBMS\_OUTPUT.PUT\_LINE('Message');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Message');

END;

**SQLCODE** : Returns the exception code.

**SQLERRM** : Returns the exception message.

ex – BEGIN

-------

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE ('Error Code: '|| **SQLCODE**); DBMS\_OUTPUT.PUT\_LINE ('Error Message: '|| **SQLERRM**);

END;

**Implicit Cursor**

* Each time a SQL DML statement is executed within PL/SQL, Oracle automatically creates an internal work area known as a SQL cursor.
* **Implicit Cursor / SQL Cursor Attributes -**

**SQL%FOUND** - (True/False) It Determines whether the DML statement affected any rows.

**SQL%NOTFOUND** - (True/False) Equals the number of rows affected by DML statement.

**SQL%ISOPEN** - (True/False) Indicates whether the cursor is currently open or not.

**Explicit Cursors** (General Cursor Syntax) -

Ex - Declare CURSOR CursorName IS SELECT \* FROM TblName;

vCursor TblName%ROWTYPE;

BEGIN

OPEN CursorName;

LOOP

FETCH NEXT CusrorName INTO vCursor

EXIT WHEN CursorName%NOTFOUND;

(CONTINUE WHEN Condition....! - It eliminates the current iteration and it goes to the next iteration)

----------

----------

END LOOP;

CLOSE CursorName;

END ;)

**FOR..LOOP Cursor**:

Syntax: BEGIN

FOR CursorName IS

(SELECT \* FROM TblName) LOOP

-----------

END LOOP;

END;

Note: Need not to follow the general cursor syntax, need not to open / close the cursor and need not to use FETCH)

* **Explicit Cursor Attributes:**

**CursorName%FOUND** (T/F)

**CursorName%NOTFOUND** (T/F)

**CursorName%ROWCOUNT** (INT) - Indicates which row the FETCH statement received.

**CursorName%ISOPEN** (T/F)

**Updatable Cursors:**

1. Once the UPDATE cursor is opened the undelaying data is LOCKED.

2. These rows can be locked until the cursor is closed.

3. syntax CREATE CursorName IS SELECT \* FROM TblName FOR UPDATE OF Fieldnames;

4. **WHERE CURRENT OF Clause** - This is used to do an action (update) on Current Record.

<https://www.techonthenet.com/oracle/cursors/current_of.php> (for update of field name in select)

(ex - UPDATE CursorName set filed Name = <’value’> **where CURRENT OF CursorName**;)

**Cursor Parameters:**

syntax: DECLARE CURSOR CursorName (pCursor datatype) IS

SELECT \* FROM TblName

where FieldName = pCursor;

pCursor = '123';

OPEN CursorName(pCursor);

**User Defined Event**

ex - Declare

eVariableName EXCEPTION

BEGIN

-------

IF THEN

RAISE eVariableName

END IF;

END;

(Note - once control has been passed to the EXCEPTION clause, control cannot be passed back to the BEGIN clause.)

**Nested Blocks:**

A Nested Block is a complete program PL/SQL program block enclosed with the BEGIN section of another block.

If an EXCEPTION occurs in Inner Block, the processing may continue in the Outer Block.

The boundaries of inner and outer blocks are identified by the labels and END statement.

syntax: <<**MainBlock**>>

DECLARE

BEGIN

<<**InnerBlock**>>

BEGIN

-----

EXCEPTION

WHEN OTHERS THEN

RAISE **vExceptionName;**

END **InnerBlock;**

EXCEPTION

WHEN **vExceptionName** THEN

--------

END **MainBlock;**

**PL/SQL FUNDAMENTALS – II**

1. Introducing Programing Units

2. Store Procedures & Functions

3. Creating & Maintaining a Package

4. Advanced Cursor Techniques.

5. Using System-Supplied Packages.

6. Database Triggers

7. Implementing Database Event Triggers.

**Store Procedure Creation**

CREATE OR REPLACE PROCEDURE sp\_Name

(parameter1 IN VARCHAR2,

parameter2 IN NUMBER DEFAULT 10,

parameter3 OUT CHAR

)

IS

vVariableName1 TblName.FieldName%TYPE;

vVariableName2 NUMBER;

vVariableName3 EXCEPTION;

BEGIN

----------------

EXCEPTION

--------------------

END sp\_Name ;

**CREATE OR REPLACE FUNCTION fn\_Name**

( pPmtr1 IN CHAR,

pPmtr2 IN NUMBER

)

RETURN BOOLEAN

IS

vVariableName1 NUMBER;

vVariableName2 NUMBER;

BEGIN

IF Condition.... THEN

RETURN (FALSE);

ELSE

RETURN (TRUE);

END IF;

EXCEPTION

-------

END fn\_Name;

**PARAMETER**

PARAMATER IN OUT BOTH CAN USE

If DEFAULT is not supplied and the calling does not supply a value, the parameter will be set to NULL.

CREATE PROCEDURE : Allows a database user to create a stored program units within own schema.

CREATE ANY PROCEDURE : Allows a database user to create a stored program units in any schema.

When using NAMED NOTATION you specify the name and value using the following syntax:

**parameter\_name=>parameter\_value**

FUNCTIONS can be used in

1. Within the column list of a SELECCT statement

2. Within the conditional expressions of WHERE, GROUP BY or HAVING clauses

3. Within the value list of INSERT or UPDATE commands.

Handling Compilation Errors

You can also generate a report for any stored procedure by appending the type and name of the object you wish to view.

The object types : FUNCTION, PROCEDURE, PACKAGE, PACKAGE BODY and TRIGGER.

---Using the show errors function.

syntax: SHOW ERRORS PROCEDURE Procedure\_Name

---Actually querying the user\_errors view.

syntax: select \* from USER\_ERRORS;

* **Recompiling & Dropping programs.**

To Recompile the PROCEDURE or FUNCTION

(if any structure has been changed in dependency table, the oracle automatically recompile it.)

(If you want to recompile manually then follow the syntax)

1. ALTER PROCEDURE ProcedureName COMPILE;

2. ALTER FUNCTION FuntionName COMPILE;

The SYS / SYSDBA super admin database account has access to run the UTL\_RECOMP() system-supplied package which will recompile all invalid objects within the database as part of single operation.

syntax: CONNECT SYS

BEGIN

UTL\_RECOMP.recomp\_parallel();

END;

* **INVALIDATION DURING EXECUTION**

Changes to dependent objects which occur during execution can certainly invalidate a procedure or function within a user session.

When encountered, Oracle will

\* Automatically force ROLLBACK

\* Nullify the SESSION STATE

Avoid Invalidation during Execution

Include an EXCEPTION HANDLER for the following error

ora-04068: existing state of package has been discarded

(Note: if you can change the dependency while executing a program, need to handle this error in program according to logic otherwise the program automatically ROLLBACK.)

**DATA DICTIONARY**

1. USER\_OBJECTS

2. USER\_SOURCE

3. USER\_ERRORS

4. USER\_OBJECT\_SIZE

5. USER\_DEPENDENCIES

syntax: DESCRIBE USER\_OBJECTS

it will display the columns in USER\_\*

(Note: SELECT text FROM user\_source WHERE NAME = 'ProcedureName' ORDER BY line;)

* **MANAGING DEPENDENCIES**

if you delete a dependency object and want to know the effected objects in db, need to check with the following query.

SELECT object\_name, status from USER\_OBJECTS WHERE object\_type in ('PROCEDURE', 'FUNCTION') ORDER BY object\_name;

* **TRAPPING DEPENDENCIES**

While the USER\_DEPENDENCIES table shows one level of dependency, the dependency tracking utility will recursively reflect all dependencies.

Dependency Tracking Utility Setup

It is a stored procedure. One that must be created within the schema before use.

It is a system supplied script.

it is named UtlDTree.SQL

Found the Database Administrator Directory (its dependent upon the version).

Execute the UtlDTree.SQL file then it creates Utility Setup.

(Note: it shows view IDEPTREE created.)

----Fire the Dependency Tracking Utility

Execute deptree\_fill('table','system','employee')

----Query the results of the utility

SELECT \* FROM ideptree;

CURSOR VARIABLES

A REF CUROSR type must be declared, followed by one or more instance of that type.

1. strong or restrictive definition - it is strictly limited to queries which are type-compatible.

2. weak or non-restrictive definition - It may be used for any query.

(See the video No. 75 for reference)

ex: for the strong or restrictive definition

ex: **CREATE OR REPLACE PACKAGE employee\_data**

AS

TYPE employee\_cv\_type IS REF CURSOR RETURN employee%ROWTYPE;

**PROCEDURE** **Open\_Employee\_data**(

employee\_cv IN OUT employee\_cv\_type,

x\_dno IN employee.dno%TYPE);

**PROCEDURE** **fetch\_employee\_data**(

employee\_cv IN employee\_cv\_type,

emplyee\_output OUT VARCHAR2);

END **employee\_data**;

/

ex: **CREATE OR REPLACE PACKAGE BODY employee\_data**

AS

PROCEDURE **open\_employee\_data**(

employee\_cv IN OUT employee\_cv\_type,

x\_dno IN employee.dno%TYPE);

IS

BEGIN

OPEN employee\_cv FOR

SELECT \* FROM employee where employee.dno = x\_dno;

END **open\_employee\_data**;

**PROCEDURE fetch\_employee\_data(**

employee\_cv IN employee\_cv\_type

employee\_output OUT VARCHAR2);

IS

employee\_row employee%ROWTYPE;

BEGIN

FETCH employee\_cv INTO employee\_ros

employee\_output := employee\_row.lname || ', ' || employee.salary;

END **fetch\_employee\_data**;

END employee\_data;

-------now the anonymus block starts here

DECLARE

department\_number employee.dno%TYPE := '&department';

employee\_cv employee\_data.employee\_cv\_type;

employee\_output VARCHAR2(1000)'

BEGIN

----Open the cursor

employee\_data.open\_employee\_data(employee\_cv, department\_no)

LOOP

employee\_data.fetch\_employee\_data(employee\_cv, employee\_output)

EXIT WHEN employee\_cv%NOTFOUND;

dbms\_output.putline (employee\_output);

END LOOP;

END;

-------now the anonymous block ends here

**TRIGGERS**

1. Statement Level Triggers: These fire once for the entire triggering statement.

2. Row-level Triggers: There fire once for each row affected by the triggering statement.

3. INSTEAD OF Triggers: Used to execute in place of any update operations on a particular complex view.

Syntax: CREATE OR REPLACE TRIGGER Trg\_Check

BEFORE DELETE OR UPDATE ON TblName

(BEFORE DELETE OR UPDATE OF FieldName ON TblName)

(FOR EACH ROW)--if it is used we call it as row level trigger

(WHEN Condition...)--if the condition satisfied then only the trigger fires.

DECLARE

-----

BEGIN

-----

EXECEPTION

RAISE\_APPLICATION\_ERROR(-20001, 'Message');

-----

END:

ex: **IF UPDATING**

AND :new.salary > old.salary THEN

------

ELSE

----

END IF;

(Here **new** indicating after updating, **old** indicating before updating values, **:** symbol have to use because it is within the body of the TRIGGER)

Note: 1. Additionally, you can qualify the UPDATE specification to list individual columns within the table.

2. Do not use COMMIT, ROLLBACK or SAVEPOINT statement in Trigger.

ex: **CREATE OR REPLACE TRIGGER Tr\_employee\_Journal**

AFTER INSERT OR UPDATE OF salary ON Employee

FOR EACH ROW

WHEN (NEW.Salary > 70000)

BEGIN

INSERT INTO audit\_entry(entry\_date, entry\_user, entry\_text, old\_value, new\_value)

VALUES (SYSDATE, USER, 'Salary > 70000' || :NEW.SSN, :OLD.Salary, :New.Salary)

END;

syntax: UPDATING - checking the transaction whether updating or not

DELETING - checking the transaction whether deleting or not

INSERTING - checking the transaction whether inserting or not

**INSTEAD OF TRIGGERS:** This type of triggers used on VIEWS.

syntax: **CREATE OR REPLACE TRIGGER Tr\_Name**

INSTEAD OF INSERT, UPDATE

ON View\_Name (not a table name)

BEGIN

-----

END;

Note : COMMIT / ROLLBACK / SAVEPOINT these commands are not used in trigger and if the trigger calling any Stored Procedure in that case in the Stored Procedure also doesn't have these words otherwise the trigger throws an error.

Note: you can call stored procedure in trigger with use of CALL statement

Note: SHOW ERRORS TRIGGER Trigger\_Name - it will show errors of the trigger.

Enable / disable the trigger

syntax: ALTER TRIGGER Trigger\_Name ENABLE | DISABLE:

You can also enable or disable all triggers for a given table

syntax: ALTER TABLE Table\_Name ENABLE ALL TRIGGERS;

ALTER TABLE Table\_Name DISABLE ALL TRIGGERS;

* Creating a **DISABLE TRIGGER**

When a trigger is created, it is automatically enabled, you may wish to initially disable the trigger and later enable it.

syntax: CREATE OR REPLACE TRIGGER TriggerName

BEFORE DELETE OR UPDATE ON employee

**DISABLE**

BEGIN

--------

END;

Multiple Triggers for the Single Table (FOLLOWS)

Syntax: **CREATE OR REPLACE TRIGGER Security\_Day\_Check**

BEFORE DELETE OR UPDATE ON Employee

**FOLLOWS** Security\_Hour\_Check

DECLARE

dy\_of\_week CHAR(3);

BEGIN

dy\_of\_week := TO\_CHAR(SYSDATE, 'DY')

IF dy\_of\_week IN ('SAT','SUN') THEN

RAIASE\_APPLICATION\_ERROR(-20601, 'Transaction rejected for security reasons - day of week');

END IF;

END;

syntax: **CREATE OR REPLACE TRIGGER Security\_Hour\_Check**

BEFORE DELETE OR UPDATE ON Employee

DECLARE

dy\_of\_hour number(2);

BEGIN

dy\_of\_hour := TO\_CHAR(SYSDATE ,'HH24')

IF dy\_of\_hour NOT BETWEEN 8 AND 17 THEN

RAISE\_APPLICATION\_ERROR(-26000, 'Transaction Rejected for Security Hour Check');

END IF;

END;

**IDENTIFYING SYSTEM EVENTS**

1. System Manager Events

BEFORE SHUTDOWN / AFTER STARTUP / BEFORE SERVERERROR / AFTER SERVERERROR

2. User Session Events

BEFORE LOGOFF / AFTER LOGON

3. User DDL Events

BEFORE/AFTER CREATE, ALTER, DROP, ANALYZE, AUDIT, NOAUDIT, RENAME, GRANT, REVOKE, TRUNCATE

(See then 96 No. video for Attributes)

\* Video No. 92,93 see again

**OTHER INFORMATION**

* **Private Temporary Tables –** It is added in oracle 18c version

CREATE PRIVATE TEMPORARY TABLE ORA$PTT\_tbl\_name as (select \* from employees)

on commit preserve definition;

Note: 1. Private temporary tables must be prefixed as per the database parameter (ex. ora$ptt).

2. Here when the session ends the data will be automatically deleted in temp table but temp table will not drop automatically.

3. Here once the data insertion was done in a particular session, that data will not appear in another session. (the data in a temporary table it is a session base.)

**CREATE A NEW SCHEMA**

CONN SYS as SYSDBA;

ALTER SESSION SET "\_ORACLE\_SCRIPT"=TRUE;

COMMIT;

CREATE USER EMP IDENTIFIED BY EMP1234;

COMMIT;

GRANT CREATE TABLE TO EMP;

GRANT CREATE PROCEDURE TO EMP;

GRANT CREATE TRIGGER TO EMP;

GRANT CREATE SESSION TO EMP;

GRANT CREATE SEQUENCE TO EMP;

GRANT unlimited tablespace to EMP;

ALTER USER <USER\_NAME> quota unlimited on <TABLE SPACE NAME>;

COMMIT;

CREATE TABLE EMP.EMPLOYEES (EMP\_ID NUMBER(6)

,EMP\_NAME VARCHAR2(50)

,GENDER CHAR(1)

,EMP\_DEPT VARCHAR2(50) );

COMMIT;

SELECT \* FROM EMP.EMPLOYEES;

**TO CREATE A NEW USER FOR EXISTING SCHEMA AND GRANTING SPECIFIC TABLE READ PRIVILIGES**

**--CONNECT TO AS DBA**

CONN SYS as SYSDBA;

ALTER SESSION SET "\_ORACLE\_SCRIPT"=TRUE;

**-- USER CREATION**

CREATE USER "EMP\_TEST\_USER" IDENTIFIED BY "EMP\_TEST\_USER"

DEFAULT TABLESPACE "USERS"

TEMPORARY TABLESPACE "TEMP";

**---GRANTING PRIVILIGES**

GRANT CREATE SESSION TO EMP\_TEST\_USER;

GRANT SELECT ON EMP.REV\_VILLAGE\_CODES TO EMP\_TEST\_USER;

GRANT ALTER ON EMP.REV\_VILLAGE\_CODES TO EMP\_TEST\_USER;

GRANT UPDATE ON EMP.REV\_VILLAGE\_CODES TO EMP\_TEST\_USER;

GRANT DELETE ON EMP.REV\_VILLAGE\_CODES TO EMP\_TEST\_USER;

**IDENTITY**

* **COLUMN SEQUENCE**
* Need not to pass a value while insertion, it automatically increments.
* While copying the table, the sequence will not copy.
* **GENERATED ALWAYS**
* Oracle always generates a value for the Identity Column. No Need not to insert a value to the Identity Column, if you attempt to insert a value to the identity column will cause an error.

ex: ALTER TABLE tbl\_name ADD (column\_name NUMBER GENERATED ALWAYS AS IDENTITY);

* **GENERATED DEFAULT**
* If you provide **NO VALUE** Oracle generates a value for the Identity column.
* If you provide **a VALUE** Oracle will insert that value for the identity column.
* If you provide **NULL** **value** Oracle will give an error.
* **GENERATED DEFAULT ON NULL**
* If you provide **a Null or NO VALUE** Oracle generates a value for the Identity column.

Note: Identity introduced In Oracle 12c version.

**FETCH CLAUSE (TOP ROWS like SQL)**

* SYNTAX: [ OFFSET offset ROWS] FETCH NEXT [ row\_count | percent PERCENT ] ROWS [ ONLY | WITH TIES ]

FETCH NEXT 1 ROWS.

FETCH FIRST 1 ROW.

ONLY | WITH TIES

The **ONLY** return exactly the number of rows or percentage of rows after FETCH NEXT (or FIRST).

The **WITH TIES** returns additional rows with the same sort keys as the last row fetched. Note that if you WITH TIES. You must specify the ORDER BY clause in the query, if you don’t the query will not return the additional values.

ex: 1. SELECT Product, Quantity FROM EMPLOYEES **FETCH FIRST 10 ROWS ONLY;**

2. SELECT Product, Quantity FROM EMPLOYEES ORDER BY QUANTITY **FETCH FIRST 10 ROWS WITH TIES;**

3. SELECT Product, Quantity FROM EMLOYEES ORDER BY QUANTITY **OFFEST 10 ROWS FETCH NEXT 10 ROWS ONLY.** (This query skips the first 10 rows and returns the next 10 ones.)

**CREATE NEW TABLE WITH DATA FROM SOUCE TABLE**

CREATE TABLE TBL\_NAME AS SELECT \* FROM SOURCE\_TBL\_NAME ;

**CREATE NEW DATABASE LINK WITH REMOTE SERVER**

CREATE DATABASE LINK PSQA connect to "**PRAJA\_SACHIVALAYAM**" identified by "**gws@epragati**"

using '**192.168.95.148:1521/xe'** ;

**JSON SELECTION**

Sample JSON:

'{

"department": "Finance and Accounting",

"employees": [

{

"name": "Higgins, Shelley",

"job": "Accounting Manager",

"hireDate": "2002-06-07T00:00:00"

},

{

"name": "Gietz, William",

"job": "Public Accountant",

"hireDate": "2002-06-07T00:00:00"

}

]

}'

ex: select M.id, J.\*

from json\_data M, json\_table (

M.json\_text, '$' columns (

department ,

nested employees[\*]

columns ( name , job , hire\_date date path '$.hireDate' )

) ) J

where M.id = 22 ;

ex: SELECT J.\* FROM JSON\_DATA M, JSON\_TABLE (

M.json\_text, '$' COLUMNS (J\_ID NUMBER PATH '$.id', j\_type varchar2 path '$.type', name, ppu

,nested batters.batter[\*] columns

(b\_id number path '$.id',b\_type varchar2 path '$.type' )

) ) J

WHERE M.ID = 41;

SELECT J.\* FROM JSON\_DATA M, JSON\_TABLE (

M.json\_text, '$' COLUMNS (J\_ID NUMBER PATH '$.id', j\_type varchar2 path '$.type', name, ppu

,nested topping[\*] columns

(t\_id number path '$.id',t\_type varchar2 path '$.type' )

) ) J

WHERE M.ID = 41;

* truncate table ORA$PTT\_tbl\_name drop storage
* **TO\_CHAR(**SYSDATE, 'MON')
* **MONTHS**\_BETWEEN(DATE1, DATE2)

Ex: Calculating Age

Months\_between(sysdate, DOB) ----- It will return the total months.

**DECODE**(DOB, NULL, 0, ROUND(MONTHS\_BETWEEN(sysdate, DOB)/12)

IF DOB is null then it returns “ 0 ” else No. of years will return.

* To **Re-order** the columns order in the table

ALTER TABLE TblName MODIFY (FieldName **INVISIBLE**);

ALTER TABLE TblName MODIFY (FieldName **VISIBLE**);

* SELECT \* FROM TEMP\_NAMES A, TEMP\_CHK\_IDENTITY B WHERE A.SNO = B.ID**(+)** ;

(Note: **(+)** means left join of a table)

* To Execute **.sql** file **syntax:** @path\filename.sql; ex: @D:\a.sql;

(If you have spaces in the path use double quotes ex: @”D:\path with spaces\a.sql”;)

* **Set define off;** if you know your script includes (or may include) data containing **‘ & ’** characters and you do not want to substitution behavior, then use **set define off;** to switch off the behavior while running the script.

ex: set define off;

insert into customers (customername) values (‘ Marks & Spencer ’).

* **DECODE:** The Oracle **DECODE ()** function allows you to add the procedural IF-THEN-ELSE logic to the query.

ex 1: SELECT DECODE (1, 1, ‘One’) FROM DUAL;

RETURN ‘One’

It works like IF 1 = 1 THEN

RETURN ‘One’

END IF;

ex 2: SELECT DECODE (3, 1, ‘One’, 2, ‘Two’, ‘None of One or Two’ ) FROM DUAL;

RETURN ‘One’

It works like IF 3 = 1 THEN RETURN ‘One’

ELSIF 3 = 2 THEN RETURN ‘Two’

ELSE RETURN ‘None of One or Two’

END IF;

You can use DECODE () function in GROUP BY, ORDER BY clause also.

Ex 3: SELECT DECODE(country\_id, 'US', 'United States', 'UK', 'United Kingdom', 'JP', 'Japan' , 'CA', 'Canada', 'CH', 'Switzerland', 'IT', 'Italy', country\_id) country , **COUNT**(\*) **FROM** locations **GROUP** **BY** country\_id **HAVING** **COUNT**(\*) > =2 **ORDER** **BY** country\_id;

* **NVL:** The Oracle **NVL()** function allows you to **replace a NULL** with a meaningful alternative in the results of a query.

Syntax: NVL(e1, e2);

The **NVL()** function accepts two arguments. If e1 evaluates to NULL, then NVL() function returns e2. If e1 evaluates NON-NULL, the NVL() function returns e1.

**NVL2:** The Oracle **NVL2()** function allow you to replace, if string not null it will return below 2nd, if it is null it will returnthe below 3rd.

SYNTAX: NVL2(string1, value\_if\_not\_null, value\_if\_null)

------TO KNOW THE **STORED PROCEDURE DEPENDENCIES**

**PROCEDURE** Z\_GET\_SPS\_DEPENDENCIES

(P\_DISP\_CUR OUT SYS\_REFCURSOR )

IS

CURSOR vCUR IS SELECT DISTINCT OBJECT\_NAME find\_obj\_name FROM USER\_OBJECTS WHERE OBJECT\_TYPE = 'PROCEDURE';

begin

DELETE FROM Z\_PROC\_DEPENDENCIES;

FOR vObj\_Name in vCUR

LOOP

EXIT WHEN vCUR%NOTFOUND;

INSERT INTO Z\_PROC\_DEPENDENCIES

(OBJ\_ID, OBJ\_TYPE, OBJ\_NAME, REF\_OBJ\_NAME, REF\_OBJ\_TYPE,REF\_OBJ\_ID, INSERTED\_DATE)

SELECT A.OBJECT\_ID, A.OBJECT\_TYPE, a.object\_name

, ao.object\_name ref\_object\_name, ao.object\_TYPE ref\_object\_TYPE

, b.referenced\_object\_id, SYSDATE

from SYS.all\_objects A, SYS.public\_dependency B

inner join sys.all\_objects AO on ao.object\_id = b.referenced\_object\_id

WHERE A.OBJECT\_TYPE = 'PROCEDURE' AND A.OBJECT\_NAME = vObj\_Name.find\_obj\_name

AND ao.object\_name NOT IN ('DUAL', 'DBMS\_OUTPUT' , 'DBMS\_UTILITY') and ao.owner NOT IN ('SYS', 'SYSTEM')

AND A.OBJECT\_ID = b.object\_id;

OPEN P\_DISP\_CUR FOR SELECT \* FROM Z\_PROC\_DEPENDENCIES ORDER BY OBJ\_NAME;

--dbms\_output.put\_line( vObj\_Name.find\_obj\_name );

END LOOP;

COMMIT;

END;

* ------ **TRUNC Function () and nearest Minute, start of week, start of month, start of year.**

ex:

WITH dates AS (

SELECT date'2015-01-01' d FROM dual union

SELECT date'2015-01-10' d FROM dual union

SELECT date'2015-02-01' d FROM dual union

SELECT timestamp'2015-03-03 23:44:32' d FROM dual union

SELECT timestamp'2015-04-11 12:34:56' d FROM dual

)

SELECT d "Original Date",

trunc(d) "Date, time removed",

to\_char(trunc(d, 'mi'), 'dd-mon-yyyy hh24:mi') "Nearest Minute",

trunc(d, 'iw') "Start of Week",

trunc(d, 'mm') "Start of Month",

trunc(d, 'year') "Start of Year"

FROM dates;

**To get the Scope\_Identity in oracle:**

ex:

set serveroutput on;

declare nr\_id number;

begin

INSERT INTO employees (ID, NAME) VALUES (6, 'NAME\_1')

returning ID into nr\_id;

COMMIT;

dbms\_output.put\_line ('nr\_id = ' || nr\_id);

end;

**USING:**

SELECT location\_id, street\_address, postal\_code, country\_name

FROM locations

JOIN countries

**USING (country\_id)** ORDER BY COUNTRY\_ID;

Note: When we use the USING clause in a join statement, the join column is **not qualified** with table aliases. (**table alias does not work here**.)

**Comma Separator (,) into table**

WITH temp AS (SELECT '0,1,2,3,4,5,6' str FROM dual)

SELECT trim (regexp\_substr(str, '[^,]+', 1, LEVEL)) str FROM temp

CONNECT BY instr (str, ',', 1, LEVEL - 1) > 0 ;

**Space Separator ( ) into table (it will return “TOI”)**

WITH temp AS (SELECT 'TIMES OF INDIA' str FROM dual)

SELECT trim (regexp\_substr(str, '[^ ]+', 1, LEVEL)) str FROM temp

CONNECT BY instr (str, ' ', 1, LEVEL - 1) > 0 ) ;

**To Find Lock tables:**

select sess.sid, lockobj.session\_id,  sess.serial#, sess.username, sql\_text  
from v$sqlarea sqlarea, v$session sess, v$locked\_object lockobj  
where sess.sql\_hash\_value = sqlarea.hash\_value  
and   sess.sql\_address    = sqlarea.address  
and   sess.username is not null;

(or)

select \* from v$locked\_object;

(or) to be cross check

SELECT STR4, P\_VENDOR\_CODE,p\_from\_date,p\_to\_date,p\_dept\_id,p\_season\_id, p\_suthali\_price, P\_BAG\_TYPE,p\_bag\_capacity ,

p\_normal\_price ,P\_INSERTED\_BY FROM(WITH input\_data AS

(SELECT P\_BAG\_TYPE AS STR1,p\_bag\_capacity AS STR2,p\_normal\_price AS STR3, p\_district\_id AS STR4 FROM dual

) ,

temp\_data1 AS

(SELECT rownum rn,

trim(COLUMN\_VALUE) str1

FROM input\_data,

xmltable(('"'

|| REPLACE(str1, ',', '","')

|| '"'))

) ,

temp\_data2 AS

(SELECT rownum rn,

trim(COLUMN\_VALUE) str2

FROM input\_data,

xmltable(('"'

|| REPLACE(str2, ',', '","')

|| '"'))

),

temp\_data4 AS

(SELECT rownum rn,

trim(COLUMN\_VALUE) str4

FROM input\_data,

xmltable(('"'

|| REPLACE(str4, ',', '","')

|| '"'))

),

temp\_data3 AS

(SELECT rownum rn,

trim(COLUMN\_VALUE) str3

FROM input\_data,

xmltable(('"'

|| REPLACE(str3, ',', '","')

|| '"'))

)

SELECT t1.str1, t2.str2, t3.str3, t4.str4

FROM temp\_data1 t1

Join temp\_data2 t2 ON t1.rn = t2.rn

Join temp\_data3 t3 ON t1.RN=t3.RN

Cross Join temp\_data4 t4 );

**To Kill the session:**

ALTER SYSTEM KILL SESSION ‘ SID, SERIAL# ’;

**MERGE INTO Statement Syntax:**

MERGE INTO member\_staging x

USING (SELECT member\_id, first\_name, last\_name, rank FROM members) y

ON (x.member\_id = y.member\_id)

WHEN MATCHED THEN

UPDATE SET x.first\_name = y.first\_name,

x.last\_name = y.last\_name,

x.rank = y.rank

WHERE x.first\_name <> y.first\_name OR

x.last\_name <> y.last\_name OR

x.rank <> y.rank

WHEN NOT MATCHED THEN

INSERT(x.member\_id, x.first\_name, x.last\_name, x.rank)

VALUES(y.member\_id, y.first\_name, y.last\_name, y.rank);

**DATABASE SIZE:**

select

"Reserved\_Space(MB)", "Reserved\_Space(MB)" - "Free\_Space(MB)" "Used\_Space(MB)","Free\_Space(MB)"

from(

select

(select sum(bytes/(1014\*1024)) from dba\_data\_files) "Reserved\_Space(MB)",

(select sum(bytes/(1024\*1024)) from dba\_free\_space) "Free\_Space(MB)"

from dual );

**DATABASE SIZE AS PER SCHEMA:**

SELECT OWNER, SUM (bytes) / 1024 / 1024 / 1024 AS GB FROM dba\_segments GROUP BY OWNER;

**HOW TO SLECT THE XML DATA FROM TABLE**

SELECT GSWS\_IND\_SUR\_XML\_DATA\_SEQ.NEXTVAL, SYSDATE, XT.TRANS\_ID, XT.EMP\_NAME, XT.ESC\_NUMB, XT.AADHAR\_NUM, DIST\_NAM

FROM GSWS\_IND\_SUR\_XML\_DATA x,

XMLTABLE('/MAINC/SUBJ'

PASSING x.xml\_data

COLUMNS

TRANS\_ID VARCHAR2(35) PATH 'TRANSID',

EMP\_NAME VARCHAR2(100) PATH 'EMPNAME',

ESC\_NUMB VARCHAR2(50) PATH 'ESCNUMBER',

AADHAR\_NUM VARCHAR2(15) PATH 'AADHAR',

DIST\_NAM VARCHAR2(50) PATH 'DIST'

) xt

where transaction\_id = '1109007920340617347990' ;

**Questions:**

1. [**Does foreign key constraint will create indexes?**](https://stackoverflow.com/questions/4127206/do-i-need-to-create-indexes-on-foreign-keys-on-oracle)

**No,** Foreign Key does not create Indexes on Oracle & SQL Serer.

Only UNIQUE and PRIMARY KEY constraints create indexes.

<https://stackoverflow.com/questions/4127206/do-i-need-to-create-indexes-on-foreign-keys-on-oracle>

## **LAG, LEAD, Dense\_rank, first\_value & last\_value (Analytical functions)**

* The **LAG** function is used to access data from a previous row.
* The **LEAD** function is used to return data from rows further down the result set.

Select empno, sal, LAG(sal, 1, 0) OVER (ORDER BY sal) AS sal\_prev FROM emp;

<https://oracle-base.com/articles/misc/lag-lead-analytic-functions>

1. **SET OPERATORS**
2. **Intersect (**only common rows returned)

The SQL **INTERSECT** clause/operator is used to combine two SELECT statements but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. **This means INTERSECT returns only common rows returned by the two SELECT statements.**

Just as with the UNION operator, the same rules apply when using the INTERSECT operator. **MySQL does not support the INTERSECT operator**.

1. **Minus (**only exclusive rows in first query)

**MINUS**gives **you the rows that are found in the first query and not in the second query** by removing from the results all the rows that are found only in the second query.

1. **Union** (It will combine both result sets including the duplicates)
2. **Union All** (It will combine both result sets and exclude the duplicates)

**SELECT expression1, expression2, ... FROM table1 WHERE conditions**

**MINUS**

**SELECT expression1, expression2, ... FROM table2 WHERE conditions;**

1. **Advantages of PLSQL**

PLSQL is a development tool, and it is an extended to standard SQL, we can also use conditional checking, branching and looping.

In PLSQL we can send a **BLOCK OF STATEMENTS** to the server which reduces network traffic. PLSQL Provides set of **Error Handling Mechanism.**

PLSQL have a portability that code developed in DOS version we can run on UNIX version.

1. **CURSORS**
2. **Implicit Cursors**.

Oracle opens a work area for each SQL operation for its internal processing in order to execute SQL statements. This area is private to SQL operations, and it’s called a implicit cursor. **If the Oracle engine opens a cursor for its internal processing, then it’s called as implicit cursor.**

1. **Explicit Cursors**
   1. **PARAMETERIZED CURSOR**
   2. **FOR CURSOR**
   3. **FOR UPDATE**

It will lock the records until the transaction is completes.

* 1. **WHERE CURRENT OF Cursor\_name**

Note: Without using For Update in select query, the WHERE CURRENT OF will not work.

[**https://www.youtube.com/watch?v=917dFN2Cc6M&list=PLb1qVSx1k1Vr0v4wVyvT3GEuA0J0M4xBm&index=48**](https://www.youtube.com/watch?v=917dFN2Cc6M&list=PLb1qVSx1k1Vr0v4wVyvT3GEuA0J0M4xBm&index=48)

1. **REF Cursors**

* **Using REF CURSORs is one of the most powerful, flexible, and scalable ways to return query results from an Oracle Database to a client application.**
* A REF CURSOR is a PL/SQL data type whose value is the memory address of a query work area on the database.
* A REF CURSOR is a pointer or a handle to a result set on the database. REF CURSORs are represented through the Oracle Ref Cursor ODP.NET class.

**REF CURSORs have the following characteristics:**

* **A REF CURSOR refers to a memory address on the database**. Therefore, the client must be connected to the database during the lifetime of the REF CURSOR in order to access it.
* A REF CURSOR involves an additional database round-trip. While the REF CURSOR is returned to the client, the actual data is not returned until the client opens the REF CURSOR and requests the data. Note that data is not be retrieved until the user attempts to read it.
* **A REF CURSOR is not updatable**. The result set represented by the REF CURSOR is read-only. You cannot update the database by using a REF CURSOR.
* **A REF CURSOR is not backward scrollable.** The data represented by the REF CURSOR is accessed in a forward-only, serial manner. You cannot position a record pointer inside the REF CURSOR to point to random records in the result set.
* **A REF CURSOR is a PL/SQL data type. You create and return a REF CURSOR inside a PL/SQL code block.**

1. **TRIGGERS**
   * **DML Triggers**

* **ROW\_LEVEL\_TRIGGERS**

Row level trigger will execute when the row level data is updated by the DML statements. In the case when row is not affected by the DML statements the trigger is not executing at all.

* **STATEMENT LEVEL TRIGGERS**

It fires when the statement affects rows in table, but the processing required is completely dependent on the number of rows effected.

* **INSTEAD OF TRIGGERS**

Oracle/SQL Server **INSTEAD OF trigger** to insert data into the base tables via a **non-updatable view / complex view.**

<https://www.oracletutorial.com/plsql-tutorial/oracle-instead-of-triggers/> (for oracle)

<https://www.sqlservertutorial.net/sql-server-triggers/sql-server-instead-of-trigger/#:~:text=An%20INSTEAD%20OF%20trigger%20is,does%20not%20occur%20at%20all> (for SQL Server)

<https://www.youtube.com/watch?v=YeW708IkrAE&list=PLdNnjIwbbBtZCZlOrg5q8Ti--iJBKnHWv&index=7>

* + **DDL Triggers**
  + You can write DDL Triggers on **Database level** or **Server level**
  + Based on DDL activity (create, alter, drop, rename)
  + You can also **prevent the DDL changes** by the user with the use of DDL Triggers.
  + You can disable the DDL Trigger and run the DDL script in the database.

<https://www.youtube.com/watch?v=N7XYyTy4lz8>

* + **Logon Triggers**

Logon triggers fire stored procedures in response to a LOGON event. This event is raised when a user session is established with an instance of SQL Server. Logon triggers fire after the authentication phase of logging in finishes, but before the user session is actually established.

1. How **many triggers can be created in one table**.?

You can create **12 DML triggers,** we can create UNLIMITED triggers in single table.

1. **JOINS AND TYPES**
2. INNER JOIN
3. OUTER JOIN (Left Outer, Right Outer and Full Outer)
4. CROSS JOIN (Table1, Table2)
5. SELF JOIN
6. NATURAL JOIN (MS SQL Does not support)
7. FULL JOIN

Joins understanding: <https://www.youtube.com/watch?v=xUsY2jWQa1w>

1. What it means of **using table1, table2 after from clause**?

SELECT \* FROM Table1, Table2

It means **CROSS JOIN (N x M) (4 X 3 = 12) Records will display**

It’s very old syntax, better use Join

SELECT \* FROM Table1 T1

CROSS JOIN Table2 T2 ON T1.ID = T2.ID

1. **Procedure**: A procedure does not have a return type and should not return any value but it can have a return statement that simply stops its execution and returns to the caller. A procedure is used to return multiple values otherwise it is generally similar to a function.
2. **Package**: A package is schema object which groups logically related PL/SQL types, items and subprograms. You can also say that it is a group of functions, procedure, variables and record statement. It provides modularity, due to this facility it aids application development. It is used to hide information from unauthorized users.
3. **Oracle Constraints:**

It is usually using to maintain the integrity of the data.

**Primary Key, Foreign Key, Check Null, Unique Constraint**

The **Check constraint** specifies a condition to the row.

**Primary key** constraint ensures that each row is identified via unique key.

**Unique key** constraint ensures that no column values has repeated the values except null.

1. **Deferred Constraint:**

DEFERRED constraints are **not checked until transaction commit/rollback**. Each constraint has its own IMMEDIATE or DEFERRED mode.

Upon creation, a constraint is given one of three characteristics:

**DEFERRABLE INITIALLY DEFERRED**, **DEFERRABLE INITIALLY IMMEDIATE**, or **NOT DEFERRABLE**

**DEFFERABLE NOVALIDATE**

**SQL Server** **does not support** **deferrable constraints.**

1. **Advantages of views and its types**

* Complex queries that need to be executed often can be saved in a view.
* Views is extremely useful when the query is used multiple times.
* It is virtual and occupies no space at all.
* View displays different types of data for different users.

1. **On Delete Cascade & On Update Cascade**

A foreign key with **cascade delete/update** means that if a record in parent table is deleted/updated, then the corresponding records in child table will automatically deleted/updated. This is called **cascade delete / cascade update** in Oracle.

It can be defined either a CREATE TABLE or ALTER TABLE statement.

<https://www.sqlshack.com/delete-cascade-and-update-cascade-in-sql-server-foreign-key/>

1. **LOB** Data types

The LOB datatypes such as BLOB, CLOB can store a large block of unstructured data i.e., like graphics, image, video clips etc., They provide random and piece of access to the data.

CLOB – Character based LOB (ex: paragraphs)

BLOB – Binay based LOB (ex: audio, video, images)

­­­

1. **Can we use “OUT” and “INOUT” parameter in a function.?**

**Yes**, we can use, but with a condition **such statement can’t use in a select** statement. We can use in a procedure with place holder variable that out parameter value will store in a place holder syntax

Kkk:= fn\_get\_number(1,abc) here abc is a place holder.

1. **Can function return more than one value.?**

**Yes**, it will return more than one value

We can use like collection / ref cursor in return.

1. Can We use **Commit in a trigger body**.?

**Yes**, we can use, but we need to use **PRAGMA\_AUTONOMUS** transaction.

**PRAGMA\_AUTONOMUS TRANSACTION** is an independent transaction which will be useful to commit the transaction alone in the triggers mostly.

<https://www.youtube.com/watch?v=6XOuhH21jOk>

1. What is **PRAGMA EXCEPTION** ?

**PRAGMA EXCEPTION\_INIT** allows a meaningful error description to the user when we get an predefined errors like as follows:

NO\_DATA\_FOUND, CURSOR\_ALREADY\_OPEN, INVALID\_CURSOR, CASE\_NOT\_FOUND, INVALID\_NUMBER

**DECLARE**

e\_no\_data\_found **EXCEPTION**;

PRAGMA EXCEPTION\_INIT(e\_no\_data\_found, -1403);

**BEGIN** -- PL/SQL block where an error might occur

**SELECT** employee\_name **INTO** v\_employee\_name

**FROM** employees **WHERE** employee\_id = 9999; -- if this ID does not exist

**EXCEPTION**

**WHEN e\_no\_data\_found THEN**

**DBMS\_OUTPUT.PUT\_LINE('No employee found with the given ID.');**

**END**;

1. What is **Rank** and **Dense\_Rank?**

The **rank** function will **skip** the rank in a logic,

but **dense\_rank** function **does not skip** the rank in a logic.

Ex: **Select** id, salary, **rank**() (**over** **order** **by** salary **desc** ) **from** **table**

**Output**: 1,2,4,7,8,11

**Select** id, salary, **dense\_rank**() (**over** **order** **by** salary **desc** ) **from** **table**

**Output**: 1,2,3,4,5,6

**Select** id, salary, **rank**() (**over** **Partition** **by** dept **order** **by** salary **desc**) **from** **table**

**Output**: 1,2,3,1,2,3

1. What is **Rowid, rownum, Row\_number()**.

* **ROWID** - It’s in built, oracle will generate rowed for each record in a table, its permanent unique identifier.
* **ROWNUM** - Oracle will give you just row number as serial number for your select query.
* **ROW\_NUMBER** – It’s an analytical function, it will generate a serial number for each row as mentioned in the query and even in the partition base also.

**select** rowid, rownum, **row\_number**() **over**(**partition** **by** dept (**order** **by** id) **from** table\_name;

**SELECT *rowid*,*rownum* fROM *EMP;***

**ROWID ROWNUM**

**----- ----------------------**

**AAAR4AAAFAAGzg7AAA, 1**

**AAAR4AAAFAAGzg7AAB, 2**

**AAAR4AAAFAAGzg7AAC, 3**

1. What are the **advantages and features are added in 12c**.

* The main advantage they have given **auto\_number feature.**
* Without using the sequence, we can directly use the **auto\_number.**
* Some **paging** kind of thing also upgraded in 12c.
* Mixing of Interval and Reference partitioning
* Global Index Maintenance (if you delete the partition and the Global index on the table will become unuseful, in 12c it will rebuild automatically, but not in 11g)
* Cascade Operation from parent to child in terms of partition.

1. What are the **New features that were added in 23c?**

* JSON Relational Duality Views
* Oracle Database 23c High Availability.
* Property Graphs
* SQL Firewall
* Enhanced Schema Level Privileges.
* SQL Transpiler.

1. How may maximum number of columns we can create in a table.?

1000

1. How may maximum number of columns we can use in Primary Key.?

In any table will be having only one Primary Key

But if it is composite primary key on the multiple columns then 32 max.

1. How many maximum numbers of indexes we can create in one table.?

UNLIMITED.

1. How many maximum numbers of columns we can use in one BITMAP index.?

30 columns

1. How many maximum numbers of columns **literals** we can use in IN clause.?

1000 constant literals can use.

Beyond that 1000 if you want use then go for “ select statement “ after IN operator.

1. How many **maximum numbers of constraints** we can create in one column.?

UNLIMITED

1. Maximum of nested/sub queries we can use in a SQL statement.

255

1. Can **DDL cause COMMIT automatically**.?

**Yes**.

1. How many maximum number of columns in a partition key.?

15 columns.

1. Maximum length of VARCHAR2.?

In SQL 4000 B

In PLSQL 32767 B

1. Maximum number of trigger CASCADE.?

32

1. Can we create package specs without a package body.?

**Yes**, we can

But vice-versa it is not possible.

1. Can we create a function with OUT parameter.?

Yes, we can, but we can’t call in a select statement.

1. Limit for **RAISE\_APPLICATION\_ERROR**.

That number would be -20999 to -20000.

Syntax: raise\_application\_error (error\_number, error\_message)

raise\_application\_error(-20111,'Credit Limit Exceeded');

1. Can we overload a standalone program.?

**No**, but we can overload, if we use the proc/function in the package.

1. Can we drop a procedure / function in a package.?

**No**, We can’t, but the way is simple, we can edit the proc/function and recompile it.

1. Can We update data dictionary.?

**No**, we can’t, it is only readable mode to the user.

1. Which Oracle function will give you the **first not null value** from all parameters?

**COALESCE**. (It will give you the values first not null column value)

1. **Convert NUMBER** into word.?

to\_char(to\_date(123, ‘j’),’jsp’) it’s convert to jullion value again its convert to the JSP.

1. **What happens to indexes when you rename a table in Oracle**?

* When you rename a table, Oracle **automatically transfers indexes, constraints, and grants on the old table to the new one**.
* In addition, it invalidates all objects that depend on the renamed table such as views, stored procedures, function, and synonyms. so, you need to validate those objects by manually.

1. How to **RENAME** **a table** in oracle?

**Alter** **Table** Old\_Table\_name **RENAME** **TO** New\_Table\_Name

1. How to **Add / RENAME / ALTER COLUMN** a column of table in oracle?

* **Add Column**

**ALTER** **TABLE** customers **ADD** (customer\_name varchar2(45),

city varchar2(40) **DEFAULT** **'Seattle'**)

* **Rename Column**

**ALTER** **TABLE** table\_name **RENAME** **COLUMN** old\_name **TO** new\_name;

* **Alter Column (Data Type)**
  + **Oracle 11g**

**ALTER** **TABLE** table\_name **MODIFY** (column\_name data\_type);

* + **SQL Server**

**ALTER** **TABLE** table\_name **ALTER** **COLUMN** (column\_name data\_type);

1. What is **SYNONYM**

It gives an alternative name to the OBJCECT (table / view).

We can use synonym names in select statement.

We can describe the synonym “desc synonymname”.

1. Can **SYNONYM** and table name can be same.?

**Yes**

(but table and view cannot use same name because it is using same table space)

(like table and index name can have the same name as synonym also can use the same name)

1. Difference between **REPLACE** and **TRANSLATE**?

**Replace:** It replaces one string with another string

Select replace (‘kiran’, ‘kumar’) from dual; Output: kumar

**Translate:** It replaces each character one by one.

Select (‘{kiran, kumar}’, ‘{ , }’, ‘( )’) from dual; Output: (kiran kumar)

1. Difference between **INSTR** and **SUBSTR**?

**SELECT** **instr**(**'hello'**,**'e'**) **FROM** dual; Output: it will return **2**

**SELECT** **substr**(**'hello'**,2,3) **FROM** ***dual***; Output: it will return **ell**

**select** **instr**(**'atchala ramana'**,**'a'**,1,2) **from** ***dual***; Output: it will return **5**

Note: it will return **second position** of 'a' character.

1. Difference between **Cluster Index** and **Non-Cluster Index**.?

<https://www.geeksforgeeks.org/difference-between-clustered-and-non-clustered-index/>

**Clustered index** is created only when both the following conditions satisfy –

The data or file, that you are moving into secondary memory should be in sequential or sorted order.

Whenever you apply clustered indexing in a table, it will perform sorting in that table only. You can create only one clustered index in a table like primary key.

Clustered index is as same as dictionary where the data is arranged by alphabetical order.

In clustered index, index contains pointer to block but not direct data.

**Non-Clustered Index** is similar to the index of a book. The index of a book consists of a chapter name and page number, if you want to read any topic or chapter then you can directly go to that page by using index of that book. No need to go through each and every page of a book.

The data is stored in one place, and index is stored in another place. Since, the data and non-clustered index is stored separately, then you can have multiple non-clustered index in a table.

In non-clustered index, index contains the pointer to data.

1. **Indexes in PostgreSQL BTree, Hash**, **Bitmap, GIN, GiST, SP-GiST** Indexes

<https://www.youtube.com/watch?v=fRbtzOFLFME> and <https://www.youtube.com/watch?v=812ALQWvIOc>

BTREE & BITMAP two indexes will use mostly in Data Ware Housing.

**BTREE Index:**

It is stored in **balanced tree format** sorted in the index column data along with the rowid.

BTREE can index <, >, <=, >=, <>

**HASH Index:**

It is same as BTREE, but HASH can index the “=“ equal operator only when we use in query,

**BITMAP Index:**

It stores the data into bitmap array only for the columns having the less number of distinct values. i. e., called in low cardinality columns.

**GIN & GiST Index**

There are two kinds of indexes that can be used to speed up full text searches. Note that indexes are not mandatory for full text searching, but in cases where a column is searched on a regular basis, an index is usually desirable

**GIN Index:**

It Stands for Generalized Inverted Index, you can use this index on JSON / JSONB columns to search for a particular key value in it.

**GiST Index:** It stands for Generalized Search Text

<https://www.youtube.com/watch?v=clrtT_4WBAw>

Need to complete the above link video for indexes in PostgreSQL

Does PostgreSQL **BTree** Index contain NULL values?

**PostgreSQL will not index NULL values,** This is an important point. Because an index will never include NULL values, It cannot be used to satisfy the ORDER BY clause of a query that returns all rows in a table.

1. Difference between **IDENTITY and INDEX**.?

<https://www.sqlshack.com/difference-between-identity-sequence-in-sql-server/>

1. *The* ***IDENTITY*** *property is tied to a particular table and* ***cannot*** *be shared among multiple tables since it is a table column property.*

*On the flip side the* ***SEQUENCE*** *object is defined by the user and* ***can*** *be shared by multiple tables since is it is not tied to any table.*

1. To generate the next **IDENTITY** value, a new row has to be inserted into the table. On the other hand, the next **VALUE** for a **SEQUENCE** object can simply be generated using the **NEXT VALUE** **FOR** clause with the sequence object.
2. The *value* for the IDENTITY property cannot be reset to its initial value. In contrast, the value for the SEQUENCE object can be reset.
3. Difference between **DELETE** and **TRUNCATE.**

Link: https://www.sqlshack.com/difference-between-sql-truncate-and-sql-delete-statements-in-sql-server/

|  |  |
| --- | --- |
| **SQL Delete** | **SQL Truncate** |
| Delete command is useful to delete all or specific rows from a table specified using a Where clause | The truncate command removes all rows of a table. We cannot use a Where clause in this. |
| It is a DML command | It is a DDL command. |
| SQL Delete command places lock on each row requires to delete from a table. | SQL Truncate command places a table and page lock to remove all records. |
| Delete command logs entry for each deleted row in the transaction log. | The truncate command does not log entries for each deleted row in the transaction log. |
| Delete command is slower than the Truncate command. | It is faster than the delete command. |
| It removes rows one at a time. | It removes all rows in a table by deallocating the pages that are used to store the table data |
| Identity does not reset it to the seed value. | Truncate command reset the identity to its seed value. |
| It requires more transaction log space | It requires less transaction log space |
| You require delete permission on a table to use this | You require Alter table permissions to truncate a table. |

1. What are the **operators** deals with the **NULL**.?

**NVL** and **NVL2**.

**NVL**: select nvl(name, ‘no-name’) from table\_name;

If value is null on name column, it will return ‘no-name’

**NVL2**: select nvl2(name, ‘name-exist’, ‘name-notexist’) from table\_name;

If value is null on name column, it will return ‘name-notexist’

If value is not null on name column it will return ‘name-exist'.

**NVL accepts only 2 arguments whereas COALESCE can take multiple arguments**

1. What **is Dual in Oracle**?
   * Dual is a built-in table in Oracle Database. That is automatically created along with data directory.
   * It’s one row and one column table with a column named DUMMY that has value of ‘X’. VARCHAR2(1) datatype.
   * Dual is in the schema of user **SYS** but it is accessible by the name DUAL to all users.
2. How **DECODE & COALESCE** function Will work?

**DECODE**

SELECT DECODE (3, 1, ‘One’, 2, ‘Two’, ‘None of One or Two’ ) FROM DUAL;

RETURN ‘None of One or Two’

It works like IF 3 = 1 THEN RETURN ‘One’

ELSIF 3 = 2 THEN RETURN ‘Two’

ELSE RETURN ‘None of One or Two’

END IF;

**COALESCE –** It will return **NON NULL** value.

SELECT COALESCE (address1, address2, addres2) result from table\_name;

It works like as follows

IF address1 is not null THEN result := address1;

ELSIF address2 is not null THEN result := address2;

ELSIF address3 is not null THEN result := address3;

ELSE result := null;

END IF;

1. Show two **PLSQL CURSOR EXCEPTIONS.**
2. CURSOR ALREADY OPEN
3. INVALID CURSOR.
4. If the **cursor is opened**, how we can find in PLSQL Block.?

Cursor\_Name**%ISOPEN**

1. **What packages are available to PL SQL developers?**

|  |  |  |
| --- | --- | --- |
| * **DBMS\_PIPE** | * **DBMS\_OUTPUT** | * **DBMS\_DDL** |
| * **DBMS\_JOB** | * **DBMS\_LOCK** | * **DBMS\_ALERT** |
| * **DBMS\_SQL** | * **DBMS\_TRANSACTION** | * **UTL\_FILE** |

1. **CURSOR ATTRIBUTES**

**Implicit cursor attributes**

* + **SQL%FOUND** - (True/False) It Determines whether the DML statement affected any rows.
  + **SQL%NOTFOUND** - (True/False) Equals the number of rows affected by DML statement.
  + **SQL%ISOPEN** - (True/False) Indicates whether the cursor is currently open or not.
  + **SQL%ROWCOUNT**

**Explicit cursor attributes**

* + **Cursor\_Name%FOUND** - (True/False) It Determines whether the DML statement affected any rows.
  + **Cursor\_Name%NOTFOUND** - (True/False) Equals the number of rows affected by DML statement.
  + **Cursor\_Name%ISOPEN** - (True/False) Indicates whether the cursor is currently open or not.
  + **Cursor\_Name%ROWCOUNT -**

1. What are **Database links** used for.?

The Database links used to communicate in order to form communication between various databases, or different environment like test, development and production. The **database links are read only to access** the information as well.

<https://www.guru99.com/pl-sql-interview-questions-answers.html>

1. What is the Difference between **DBMS** and **RDBMS**.?

|  |  |
| --- | --- |
| **DBMS** | **RDBMS** |
| DBMS stores data as file. | RDBMS stores data in tabular form. |
| No relationship between data. | Data is stored in the form of tables which are related to each other. |
| Normalization is not present. | Normalization is present. |
| It deals with small quantity of data. | It deals with large amount of data. |
| Data redundancy is common in this model. | Keys and indexes do not allow Data redundancy. |
| The data in a DBMS is subject to low security levels with regards to data manipulation. | There exist multiple levels of data security in a RDBMS. |
| Low software and hardware necessities. | Higher software and hardware necessities. |
| Examples: XML, Microsoft Access, etc. | Examples: MySQL, PostgreSQL, SQL Server, Oracle, etc. |

1. **Dynamic SQL**

Dynamic SQL provides you to write the program that reference SQL statements whose full text is **not known until runtime**.

You Should use Dynamic SQL in case where you do not know the exact SQL statements that must be executed dynamic SQL because the full text of SQL statements may be unknown at compilation.

Note: performance of Static SQL is generally better performance than Dynamic SQL

**If you want to create a table (or do any other DDL) in a stored procedure, you'd need to use dynamic SQL**

EXECUTE IMMEDIATE 'create table table\_2 as select col1, col2, col3, col4 from table\_1';

If you create the table dynamically, though, you'd also need to use dynamic SQL every time you wanted to subsequently query the table.

1. **Normalization Steps**

<https://www.youtube.com/watch?v=aAx_JoEDXQA>

**1st NF:** The data in each column should be atomic (No multiple values separated by comma)

No repeating column groups

Identify each record uniquely using primary key.

**2ND NF:** The table meets all conditions of 1NF

Move redundant data into separate table.

Create a relationship between these tables using foreign key.

**Partial Dependency**

**3rd NF:** The table meets all conditions of 1NF & 2NF

Does not contain columns (attributes) that are not fully dependent upon the Primary key.

**No Transitive Dependency.**

**BCNF (3.5 NF):** The table meets all conditions of 1NF, 2NF & 3NF

**Partial Dependency**

**4TH NF:**

**Multi Value Dependency**

**5TH NF:**

**Project Join Normal Form**

**Problems of Data Redundancy:**

Disk space wastage.

Data inconsistency

DML queries can become slow.

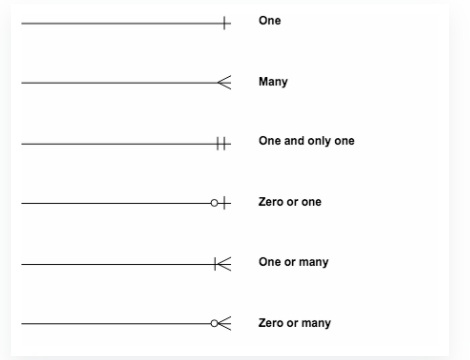
1. I have a Procedure it has 1000 lines of Code, I want to find the error line number?

DBMS\_UTILITY.FORMAT\_ERROR\_BACKTRACE() ;

This utility, if you mention in the exception block it will print the actual error line number as an output.

<https://livesql.oracle.com/apex/livesql/file/content_CSBE3ZJTSXBTUEI174J84WGO0.html>

1. ER Diagram symbols (Entity Relation)



1. What are schema types.?

a. Star Schema

b. Snowflake Schema

c. Galaxy Schema

d. Cluster Schema

<https://www.guru99.com/star-snowflake-data-warehousing.html>

1. What is the difference between a **mutating table** and a **constraining table**?

* A table that is being modified by the usage of the DML statement currently is known as a **mutating table**. It can also be a table that has triggers defined on it.
* A table used for reading for the purpose of referential integrity constraint is called a **constraining table**.

1. How to resolve the **Mutating error in trigger**

* The mutating trigger will occur whenever a row level trigger tries to modify or select data from the table that is already undergoing change.
* Mutating error get raised from row level trigger only, and not statement level trigger.

**Resolution:**

* You can use Package variable and you can use statement level trigger.

<https://www.youtube.com/watch?v=pyqNKQphRv8>

1. **PRAGMA Autonomous\_Transaction**

It’s an independent transaction, it will come from the parent transaction.

<https://www.youtube.com/watch?v=OK7KA99plXQ> (example)

<https://www.youtube.com/watch?v=R4dyvrVC9t0> (real time example)

1. **PRAGMA Exception Init**

Tells the complier to associate a particular error number with an identifier you have declared as an exception in your program.

1. **How to get a nof. Groups from a table (Group\_count)**

<https://www.youtube.com/watch?v=iiOKoMUE9B4>

1. **Maximum how many triggers is possible to apply on one table.?**

Twelve (12) Triggers only

1. **How can you debug your PL/SQL code?**

We can use DBMS\_OUTPUT and DBMS\_DEBUG statements for debugging our code:

* + DBMS\_OUTPUT prints the output to the standard console.
  + DBMS\_DEBUG prints the output to the log file.

1. **Is it possible to declare column which has the number data type and its scale larger than the precision? For example, defining columns like: column name NUMBER (10,100), column name NUMBER (10, -84)**

* **Yes**, these types of declarations are possible.
* Number (9, 12) indicates that there are 12 digits after decimal point. But since the maximum precision is 9, the rest are 0 padded like 0.000999999999.
* Number (9, -12) indicates there are 21 digits before the decimal point and out of that there are 9 possible digits, and the rest are 0 padded like 999999999000000000000.0

1. **In what cursor attributes the outcomes of DML statement execution are saved?**

The outcomes of the execution of the DML statement is saved in the following 4 cursor attributes:

* + SQL%FOUND: This returns TRUE if at least one row has been processed.
  + SQL%NOTFOUND: This returns TRUE if no rows were processed.
  + SQL%ISOPEN: This check whether the cursor is open or not and returns TRUE if open.
  + SQL%ROWCOUNT: This returns the number of rows processed by the DML statement.

1. **What are the advantages of packages in oracle?**
2. **Logical grouping** of subprograms

Logically group the related subprograms, objects, and variables

1. **Security**

Security of the code through implementation of private subprograms

1. Better Performance

Entire package is **loaded into memory** when the package is first referenced.

There is only one copy of the package for all users.

1. **Difference between PL/SQL and SQL:**

In PL/SQL there is **no buffer** to hold the values, so we pass into variables.

In SQL there is **buffer** to hold the data temporarily.

Ex: SQL> Select ename from emp where is empno = 7788;

PLSQL> Declare v\_name varchar2(10); Begin Select ename into v\_name from emp where empno = 7788;

1. What is the difference between **char, varchar and varchar2 in oracle?**

The major difference between a varchar and varchar2 in oracle database,

**Char** is a fixed length, it will add spaces in ending of the text, so it leads to a memory wastage.

**Varchar** is **ANSI** Standard

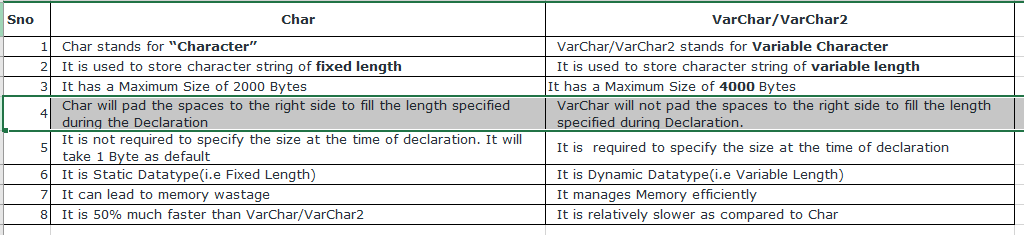
**Varchar2** is **Oracle** Standard, and it is dynamic data type, memory will not be wasted.

For every 1 character it will take 1 byte is stored in the memory. both are variable length data types, and it can take up to 4000 bytes only.

**Note:** There is no difference between VarChar and VarChar2 in Oracle. However, it is advised not to use VarChar for storing data as it is reserved for future use for storing some other type of variable. Hence, always use VarChar2 in place of VarChar.

Hence it is advised to use Char datatype when the length of the character string is fixed and will not change in the future. If the length of the character string is not fixed, then VarChar2 is preferred.

|  |  |
| --- | --- |
| **VarChar** | **nVarChar** |
| **Non-Unicode** Standard | **Unicode** Standard |
| Max length **8000** characters | Max length **4000** characters |
| Literals are enclosed with single quote  **Ex: ‘John’** | Literals are prefixed with **N** also,  **Ex: N’John’** |
| Each character takes **1 byte** | Each character takes **2 bytes** |



1. Key Differences Between **View and Materialized View.**

* The basic difference between View and Materialized View is that Views are **not stored** physically on the disk. On the other hands, Materialized Views are **stored** on the disc.
* **View** can be defined as a **virtual table** created as a result of the query expression. However, **Materialized** **View** is a **physical** **copy**, picture or snapshot of the base table.
* A view is always updated as the query creating View executes each time the View is used. On the other hands, Materialized View is updated manually or by applying triggers to it.
* Materialized View responds faster than View as the Materialized View is precomputed.
* Materialized View **utilizes** the **memory space** as it stored on the disk whereas, the View is just a **display** hence it does not require memory space.

1. How to **Describe a table** in **Oracle** & **SQL Server**?

In **Oracle** DESC [User\_Details]

In **SQL SERVER** EXEC SP\_COLUMNS [User\_Details]

1. **STRING TO ROWS**

**select** regexp\_substr(**'A1,A2,A4'**,**'[^,]+'**, 1, **level**) **from** dual

**connect** **BY** regexp\_substr(**'A1,A2,A4'**, **'[^,]+'**, 1, **level**) **is** **not** **null**;

1. **What are the Regular expression Types**
   * Regexp\_substr
   * Regexp\_like (Basic email format check)

**SELECT** id, **text** **FROM** example\_table

**WHERE** REGEXP\_LIKE(**text**, **'[A-Za-z0-9.\_%+-]**[**+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}**](mailto:+@[A-Za-z0-9.-]+\.%5bA-Za-z%5d%7b2,%7d)**'**)

* + Regexp\_replace
  + Regexp\_count

1. How can you decide when you create **BTree or BitMap Index**?

* You can create based on **Cardinality**.
* **Cardinality** means – distinct values or less (gender, married/unmarried).
* If **Low** Cardinality you can go for BitMap.
* If **High** Cardinality you can go for BTree.

1. **Collections:** **Varray** and **Nested Table** and **Associative Table**

* **Varray** Stored as per declaration. It can’t allow to add beyond the amount of declaration.

(The upper bound of limit has to be mentioned while declaration)

DECLARE TYPE t\_name\_type IS VARRAY(2) OF VARCHAR2(20) NOT NULL;

* **NESTED TABLE** has no limit (upper bound) as per declaration

The index will be maintained by the Oracle for VARRAY and NESTED TABLE.

We can delete any one of the row in NESTED TABLE but not VARRAY.

* **ASSOCIATIVE ARRAY**

It is also same a NESTED TABLE major difference is the INDEX portion.

The INDEX portion will be created and maintained by the user as well as datatype.

We can maintain the Index datatype and value datatype.

TYPE t\_capital\_type IS TABLE OF VARCHAR2(100) INDEX BY VARCHAR2(50);

<https://www.youtube.com/watch?v=T5Gu2eaq31k>

<https://www.youtube.com/watch?v=TgoHpkEjrYY&list=PLb1qVSx1k1VpAFTXopXvAh_D3PLcTntdm>

1. **Collection Types**

* **BINDING**

Assigning a value to PLSQL variables that appear in SQL statements is called **binding.**

**PLSQL variables** are RECORD, TABLE, VARRAY, REF CURSOR, BLOB, CLOB, NCLOB.

* **BULK BINDING**

That is, it binds an entire collection of values at once is called **Bulk Binding.**

* How to implement **BULK COLLECTION**

It reduces the **CONTEXT SWITCHING** from the **PLSQL** engine to **SQL** Engine and SQL engine back to PLSQL engine by using

FOR ALL STATEMENT / BULK COLLECT

* How “Bulk SQL” **Reduces** the performance

SAVE EXCEPTIONS

Bulk collect and For All examples

<https://www.youtube.com/watch?v=hBKiuNMHvNs>

1. **About BULK COLLECT**

To take advantage of bulk processing for queries, you simply put BULK COLLECT *before* the INTO keyword and then provide one or more collections *after* the INTO keyword. Here are some things to know about how BULK COLLECT works:

* **3 types of collections**: **VARRAYs, Nested Tables and Associative Arrays**.
* You can fetch into individual collections (one for each expression in the SELECT list or a single collection of records.
* The collection is always populated densely, starting from index value 1.
* If no rows are fetched, then the collection is emptied of all elements.

1. **Bulk Collect / Bulk Binding & Bulk Exception**

<https://www.oratable.com/forall-insert-exception-handling-bulk-dml/>

Ex: **Bulk DML: Script when run Script**

Declare

CURSOR c\_people IS select id, name from people;

TYPE t\_people IS TABLE OF c\_people%rowtype

index by binary\_integer;

l\_people t\_people;

l\_err\_msg varchar2(1000);

l\_err\_count number;

bulk\_errors exception;

pragma exception\_init(bulk\_errors, -24381);

begin

OPEN c\_people;

FETCH c\_people BULK COLLECT INTO l\_people;

CLOSE c\_people;

dbms\_output.put\_line ('Before FORALL');

dbms\_output.put\_line ('Total Count = '||l\_people.count);

begin

-- bulk insert and save exceptions

FORALL i IN 1..l\_people.count save exceptions

**insert into customer ( id, name ) values ( l\_people(i).id, l\_people(i).name );**

exception

when bulk\_errors then

l\_err\_count := sql%bulk\_exceptions.count;

dbms\_output.put\_line ('Error Count = '||l\_err\_count);

for i in 1 .. l\_err\_count loop

-- Print out details of each error during bulk insert

l\_err\_msg := 'Error#: ' || i

|| '; Array index: '

|| sql%bulk\_exceptions(i).error\_index || ': '

|| sqlerrm(-sql%bulk\_exceptions(i).error\_code);

dbms\_output.put\_line (l\_err\_msg);

end loop;

end;

dbms\_output.put\_line ('After FORALL');

end;

/

Before FORALL

Total Count = 7

Error Count = 2

Error#: 1; Array index: 3: ORA-00001: unique constraint (.) violated

Error#: 2; Array index: 5: ORA-00001: unique constraint (.) violated

After FORALL

PL/SQL procedure successfully completed.

1. **Explain Plan / functions / triggers / cursors / TKPROF / HINTS / PLSQL PACKAGES**
2. **SQL Trace** & **TKPROF**

* **SQL Trace** – SQL Trace facility records below information on executed SQL Statements.
* Parse, execute and fetch counts
* CPU and elapsed times.
* Physical reads and Logical reads
* Number of rows processed
* Misses on the library cache
* Optimizer mode
* Parsing User id
* Number of plan statistics captured

**Enable SQL TRACING**

For Database Level:

Dbms\_Monitor.Database\_Trace\_Enable;

For Session Level:

Execute Dbms\_Monitor.SESSION\_TRACE\_ENABLE;

Dbms\_Session.SESSION\_TRACE\_ENABLE.

Dbms\_Session\_set\_SQL\_TRACE;

Alter Session set SQL\_TRACE = TRUE;

* SQL Tracing records many vital info regarding actual query execution into a trace file. It writes query execution statistics like number of logical I/O, physical I/O, the CPU and elapsed timings, number of rows processed, query plans with row counts at each level, information on wait events & etc.,
* Its bit difficult to read the trace file.
* Using TKPROF utility file we can generate readable report. The most options in TKPROF report as follows:
* **TKPROF** also used some parameters like

**WAITS**: Specifies whether to record summary for any wait events found in the trace file. Values are YES or NO. The default is YES

**SORTS**: Sorts traced SQL statements in descending order of specified sort option before listing them into the output file.

**EXPLAIN**: Determines the execution plan for each SQL Statement in trace file and writes these execution plans to the out put file. TKPROF determines execution plans by issuing the EXPLAIN PLAN.

**You can generate TKPROF report using trace file using below command line in cmd.**

**Syntax** : CMD> tkprof <space> ora\_90\_ora\_1.trc <space> tk\_rep.prf

**Interpreting TKPROF Report**

1. **AWR Report (**Automatic Workload Repository**)?**

* Only, **sysdba** user can create AWR Report, lets connect with this user

$ **sqlplus / as sysdba**

* Then you should run the following script in your terminal / any GUI.

$ **@$ORACLE\_HOME/rdbms/admin/awrreport.sql**

(The above location will use oracle to hold all oracle scripts)

It will ask you in which format ‘html’ (defalt), ‘text’ or ‘active-html’

* Then It will ask you for format: hit enter (it will generate into html as it is default)
* Then It will ask you for num\_days: **3**

Then it will show you the list of snaps with Instance, DB Name, Snap Id, Snap Start, Snap Level.

Here you can take the Snap Ids **from** where **to** base on the date & time (ex: if you want to check the database performance between 1pm to 7pm). It has 7 hours; it has 7 files for each hour as per thumb rule.

* Oracle recommends to generate different AWR reports between these interval then you will get exact values in each report, if you are going to generate a single report for a particular snapshot intervals, you will average values only.
* Then It will ask you **begin\_**snap id: 1063
* Then It will ask you **End\_snap** id :1064
* Then it will ask you for report name: hit enter it will take default name and store it in where you started sql plus.
* Through **winsp** you can copy the report into local or else you can go and open the report (html) it will open in any one of the browser.
* Groups and elements will show in the report.
* Database Details (Name, Edition, Release, Role and etc.,)
* Database Host details (Server Name, OS, CPU, core, memory and etc.,)
* Snap Id details from begin to end
* Load Profile

Per Second metrics

* + Redo size (bytes)
  + Logical read (blocks)
  + Physical reads & Physical writes
  + Transactions
  + User calls
  + Roll backs
  + Parses & Hard Parses
* Wait Events
  + Log file switch (archiving needed)
  + Nof waits w.r.t its Total wait time (seconds)
  + Free buffer waits
* Wait classes
* SQL Statistics
  + SQL ordered by CPU Time.

<https://www.youtube.com/watch?v=QPJL1fswbO4>

1. **When do you need to rebuild an index?**

If your table gets effected by huge Deletes (at least 20%-30%) then you go for rebuilding the index.

(“ALL\_TAB\_MODIFICATION“ )

<https://www.youtube.com/watch?v=alvUF2V73Us> @20.00

1. **Partitioning ?**

**When Partition needed?**

* If your table size is greater than 2 GB
* If you are maintaining the rolling data (ex: last 3 months updateable and the remains readable)
* If your database is running on the clustered environment then HASH Partition will be useful.

**Types**

1. **Single Partitioning**
   * + - List Partition (partition by LIST)

Ex: Location, Region, countries, states

* + - * Hash Partition (partition by HASH)

Ex: Customer\_ID, Product\_ID

* + - * Range Partition (partition by RANGE)

Ex: Dates, scores (10 to 20 / 20 to 30)

1. **Composite Partitioning** (List / Hash / Referencing)

* The fundamental storage of unit in oracle in a BLOCK (each block consists of 8 KB only)

ex: Select \* from **user\_segments** where segment\_name = ‘Table\_name’

Blocks --🡪 extent (group of blocks)--🡪 segment (group of extents)

* If you want to analyze an index

ex: $ analyze index **Index\_name** validate structure;

ex: $ select \* from **index\_state**;

The BTree Index will be stored in Branch Block and Leaf Blocks (refer second link)

The Branch block holds the data pointer only (it holds the range of index only)

The Leaf block holds the Physical information pertaining to the data with the rowid.

**When you do a partition of table and index it has its own segment Separately**

**If you want to fetch a very less amount of data from a huge table. the Index is correct choice.**

**The Index is feasible to fetch 5% to 10% out of entire data.**

**If you want to fetch a considerable amount of data (25% out of entire data) the Index is not feasible, the Full Table Scan is perfect fit here.**

**Why because the index will take random I/O, Full Table Scan (FTS) will take Large Sequential I/O.**

**The Index will hit the same blocks multiple times to get the rows, but FTS will hit only one time.**

But Some time the FTS might be will slow, then partition of Index is the perfect solution.

* **“ User\_tab\_partition “** table -🡪 it will get partitions from the table

<https://docs.oracle.com/cd/B10500_01/server.920/a96524/c12parti.htm> (for partition syntax)

https://www.youtub e.com/watch?v=OQ8LXbxLI5g&list=PLUWdEQb\_2yqX9rJ9CtahlZvo0AD3A63JA&index=2

1. **Performance Tuning Steps through DBMS\_PROFILER.**

[**https://www.youtube.com/watch?v=G67gaP79htE&list=PLb1qVSx1k1VrPd8FT8WI6Btu754TZjqIs**](https://www.youtube.com/watch?v=G67gaP79htE&list=PLb1qVSx1k1VrPd8FT8WI6Btu754TZjqIs)

[**https://sivakacademy.blogspot.com/2019/06/profile-plsql-code-using-dbmsprofiler.html**](https://sivakacademy.blogspot.com/2019/06/profile-plsql-code-using-dbmsprofiler.html) (blog) for code

* To test performance tuning, we need to use DBMS\_PROFILER.
* Manually reading each line of code to identify performance issues in execution is very difficult.
* DBMS\_PROFILER helps in tuning by pinpointing where the lags occur.
* This tool/package provides functions and procedures for PL/SQL profiling.
* It makes it easier to identify performance bottlenecks in PL/SQL code.

**We must follow 4 step process:**

* 1. **Environment Setup**
  2. **Profiler Execution**
  3. **Analyze Profiler Data**
  4. **Optimize the PLSQL Code**

**Environment Setup**

**This is one time setup**; we have to run a file which we have during the oracle installation under the path “@c:\oraclee18c\WINDOWS.X64\_db\_home\_ORABLE18C\rdbms\admin\proftab.sql” **to create profiler tables** while running profiler on a block of code. You can check the following tables are created or not.

>> Select \* from user\_tables where table\_name like ‘%PROF%’;

**Profiler Execution**

Here you can explicitly call DBMS\_PROFILER to tune the code in the following way.

Example scenario: We have 3 procedures called “Proc\_a”, “Proc\_b” and “Proc\_c”.

* In Procedure A (proc\_a) We are calling Procedure B (Proc\_b)
* In Procedure B (proc\_b) We are calling Procedure C (Proc\_c)
* In Procedure C some custom logic is there for execution.

So, we can execute “proc\_a”, it’s automatically executing A, B and C.

* + - **START\_PROFILER** – It Starts PLSQL Profiling before execution of your PLSQL Program

>> exec DBMS\_PROFILER.START\_PROFILER (‘MY TEST RUN\_1’);

>> exec proc\_a:

**>>** execDBMS\_PROFILER.STOP\_PROFILER ();

* + - **STOP\_PROFILER** – It stops the profiler data collections in the user session & writes the data into profiler tables.
    - After completion execution of DBMS\_PROFILER, it will generate a data into following 3 Profiler Tables.

**Analyze Profiler Data**

**PROFILER TABLES: We can view Profiler data into these 3 tables**

1. **PLSQL\_PROFILER\_RUNS** – This contains information related to a profiling session things

run\_time, run\_id, run\_date, run\_owner, run\_duration

>> Select \* from plsql\_profiler\_runs: --Run Level (RUNID)

1. **PLSQL\_PROFILER\_UNITS** – This table holds details for each plsql (unit) that was executed during the profile run.

>> Select \* from plsql\_profiler\_units: --Unit / Block Level (RUNID, UNITNUMBER)

1. **PLSQL\_PROFILER\_ DATA** – This table contains the execution statistics for each line of code contained in the PLSQL code.

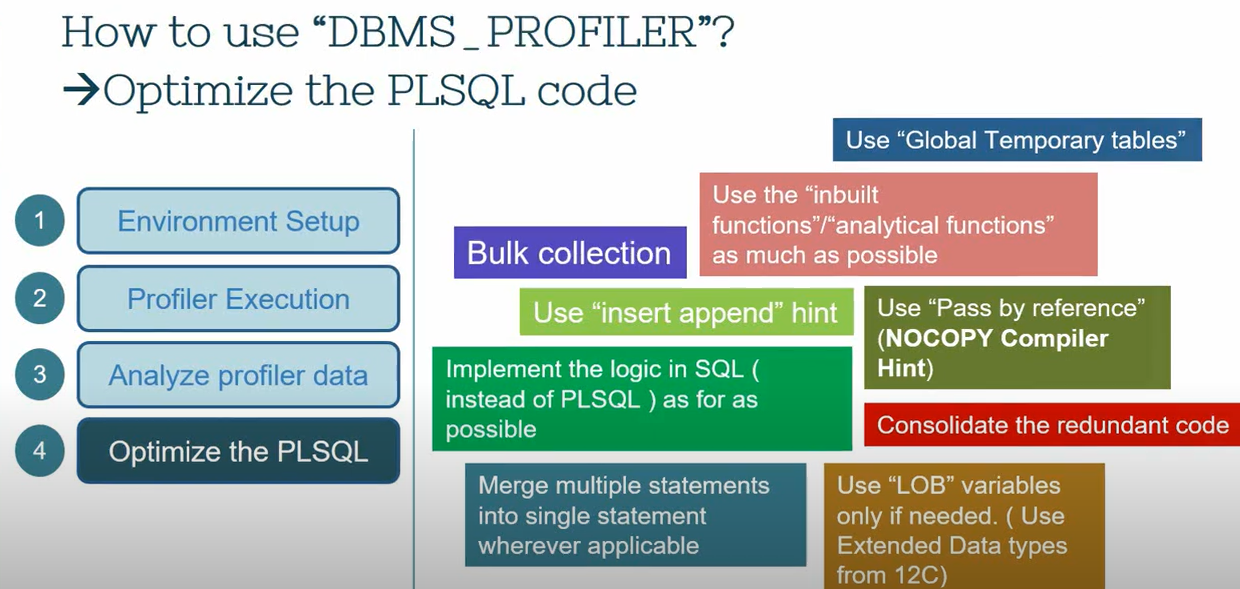
The columns are in the DATA table

Run\_id, Unit\_number, Line#, Total\_occur, Total\_Time, Min\_time, Max\_Time,

>> Select \* from plsql\_profiler\_data: --each line Level (RUNID, UNITNUMBER)

1. **Optimize the PLSQL Code**

Once we knew where the problem is occurring by referring the PLSQL\_PROFILER\_DATA table, then we have to take best optimization technique to improve the performance, the below are few examples only.



**Explain Plan**

1. **Join Methods**

Join Method is an algorithm used to perform the join by the optimizer.

* 1. **Nested Loop Join** (use\_nl) - The optimizer will choose when dataset is smaller.

It will take each record from the outer table (table\_a) and compare it with all records in the inner table (table\_b).

* 1. **Sort Merge join** (use\_merge) – The optimizer will choose when two tables are already sorted.

It will sort all records in both the outer table (table\_a) and the inner table (table\_b), and then search for matches.

* 1. **Hash Join** (use\_hash) – The optimizer will choose for larger data sets and equality operator used.

It will use a hash function to create a hash table from the inner table. Then, it will search for matches using this hash table

**Conclusion:**

* For Larger Datasets - Hash Join will be faster.
* For Smaller Datasets - Nested Loop join will be faster.
* For Sort Merge Join If Datasets are already sorted - Merge Join will be the faster.

<https://www.youtube.com/watch?v=pJWCwfv983Q>

(This video will be helpful more understanding on join methods).

2. **Table Access Methods**

* 1. **FTS (Full Table Scan)** (if you don’t create an index, it will cause FTS)
* **No indexes** are available on the table.
* There is a **parallel** execution (degree) specified on the table.
* Increasing the **MBRC** (Multi Block Read Count) parameter can encourage FTS (not recommended)
* a **FULL** hint was specified in the query statement.
  1. **Table Access by ROW-ID**
  2. **Index Unique Scan**
* It returns, at most, a **single rowid**
* It chosen by optimizer when there is an **equality predicate on a unique** (B-Tree) Index.
  1. **Index Range Scan**
* It is chosen if there is a **range predicate on unique index** (or)
* an **equality predicate on non-unique index**.
  1. **Index Skip Scan** (index\_ss)
* Not Index skip scans improve index scans by non-prefix columns. Often,   scanning index blocks is faster than scanning table data blocks.
  1. **Full Index Scan**
* It is chosen by optimizer, when index created on two columns in a table (id, name) and using the same order in the query.

Index ex: create index idx\_cust\_name\_id on Customers (**id, name**)

Query ex: select \* from Customers **order by id, name**

* **No sort operation is necessary** to satisfy the order by
  1. **Fast Full Index Scans** (index\_fss)
* It is chosen instead of FTS **when all of the columns needed are in the index**
* Fast Full Scan can **operate in Parallel** without having to be a partitioned index.
* Benefits from **Multi-block reads** to scan every block in the index.
  1. **Index Joins** (index\_join)
* Two or more indexes have all the columns needed for the query
  1. **Bit Map Indexes**
* Multiple bitmap indexes can be used to apply multiple where clause predicates or join predicates

\* Predicate means where clause

* Which Access method the optimizer will take?

Assume that there is a table with primary key for id

Query: select \* from Table\_name where id in (100,200,400)

Optimizer will choose **Index Unique Scan** and convert the query as (id = 100 or id = 200 or id =300

Query: select \* from Table\_name where id between (100 and 150)

Optimizer will choose **Index Range Scan**.

<https://www.youtube.com/watch?v=7rBf7IVOh3A&t=856s>

<https://www.youtube.com/watch?v=jYIjzmYCc0U> @8.40

Hash Access

Cluster Access

1. **Data Operations** like filter, sort or aggregation

**Explain Plan for Query**

**1st Way**

1. explain plan for select \* from customers; (The plan only will execute it and store it in the below table)

2. select \* from table(**dbms\_xplan.display**) ;

3. select \* from **plan\_table** where plan\_id = ''; (will get id from above query)

**2nd way**

* 1. Select \* from table (dbms\_xplan.display\_cursor())

It will show the latest query plan in the same session, and it shouldn’t show the stats.

You need to do two things a) add hint to query b) Pass an argument like below

* + - * 1. Select /\* + GATHER\_PLAN\_STATISTICS \*/ fields names from table ………………
        2. Select \* from table(dbms\_xplan.display\_cursor(format=>’ALLSTATS LAST/FULL’))

**3rd Way**

SQL AUTOTRACE

(But this one will give you the exaction plan & statistics after giving the result of a query)

(But it will give more information than 1st way)

1. **Scenario Based**

* Create a new table and insert million records
* Get count (\*) from the same table it will take time (ex: 30seconds)
* Delete the data and commit it and get count (\*) it will take same time as earlier
* But if you do truncate instead of delete and get count(\*) it will run very faster.
* **Conclusion: Truncate will reset the Watermark, this is the reason**

1. **Scenario Based**

Q: Let’s say I have duplicated data in my existing table, I don’t have right to delete to the duplicate data as it is in production, but I want to restrict further coming data it should not be duplicated?

A: We have to add an primary key constraint for Id Column, but if you run alter statement on a table to add a constraint it will throw an error. We have to add a special key that “**DEFFERABLE NOVALIDATE**”

**ALTER** **TABLE** TBL\_STUDENT **ADD** **CONSTRAINT** TBL\_STUDENT\_PK **PRIMARY** **KEY** (ID) **DEFERRABLE NOVALIDATE**;

**Sample Data: TBL\_STUDENT**

|  |  |
| --- | --- |
| **Id** | **Name** |
| 1 | Kiran |
| 2 | Kumar |
| 3 | Kota |
| 1 | Kiran |

1. **How to Print 1 to 10 numbers in the PLSQL BLOCK (and reverse also)?**

**BEGIN**

**FOR** I **in** (**REVERSE**) 1..10 ---If you mention reverse keyword it will print 10 to 1.

**LOOP**

DBMS\_OUTPUT.PUT\_LINE(I);

**END** **LOOP**;

**END**;

1. **Scenario Based**

Q: We have a table with 1 million records. Your task is to delete 10,000 records at each time until the table is completely empty.

A: We must write a block with loop, until no record found to delete.

**DECLARE**

l\_deleted\_count **number**;

**BEGIN**

**LOOP**

**DELETE** **FROM** TABLE\_1 **WHERE** ROWNUM <= 10000;

---- Get the count of deleted records

l\_deleted\_count := **SQL**%ROWCOUNT;

**COMMIT**;

**EXIT WHEN** l\_deleted\_count = 0;

**END** **LOOP**;

**END**;

1. **How SELECT Statement work internally in Oracle?**

**What is Parsing, Hard Parsing, Soft Parsing?**

A computer screen shot of a diagram

Description automatically generated

<https://www.youtube.com/watch?v=cWRwhwMkN6k&list=PLb1qVSx1k1Vpo2IyVvu7LYOU6C81DRISk&index=3>

1. **What is FORWARD REFERENCE?**

Forward reference means calling a procedure for execution which was defined later in the same block.

The compiler does not know, which object it is.?, that’s why it will throw an error?

This kind of calling is known as **“FORWARD REFERENCE”**.

To avoid this type of error we have to mention what object type it is, like Procedure proc2.

Detailed explanation:

DECLARE

If you don’t mention under declaration that PROCEDURE Proc\_2, it will throw an error. This is called “**forward declaration**”

If you mention, the compiler will understand that’s a procedure which is defined next to it.

**PROCEDURE Proc\_2;**

PROCEDURE Proc\_1 as

Begin

**Proc\_2** ------Here we are calling Procedure 2 which is defined next

Dbms\_output.put\_line(‘This is Procedure 1’) ;

End;

PROCEDURE Proc\_2 as

Begin

Dbms\_output.put\_line (‘This is Procedure 2’);

End;

Begin

Proc\_1;

End;

<https://www.youtube.com/watch?v=dm3RszE6St0> @19.00 time

**HINTS**

Hints for Optimization Approaches and Goals

ALL\_ROWS

CHOOSE

FIRST\_ROWS

RULE

Hints for Enabling Optimizer Features

Hints for Access Paths

Hints for Join Operators

Hints for Parallel Execution

Hints for Query Transformations

Hints for Online Application Upgrade

Additional Hints

**Hint Syntax**

select /\* + all\_rows + \*/ employe\_id, last\_name from hr.employees order by employee\_id

**INDEX\_COMBINE**

**INDEX\_JOIN** – It will combine the two indexes while execution

**USE\_CONCAT** – If we are using OR operator in statement if we use use\_concat as a hint It will improve the performance

Select /\* + use\_concat + \*/ from ht.employees where employee\_id > 50 or salary < 50000;

**PARLLEL\_INDEX** – If we are selecting huge data, it will be useful to distribute the load into parrlel sessions.

Select /\* + **parallel\_index**(e, indexname, 8) + \*/ from hr.employees e ;

**PARALLEL**

Select /\* + **PARALLEL FULL** (e) \* / \* from emp e ;

Select /\* + **PARALLEL USE\_MERGE** (Emp dept) \* / \* from emp dept where emp.deptno = dept.deptno ;

----SORT MERGE JOIN

Select /\* + **PARALLEL USE\_HASH** (Emp dept) \* / \* from emp dept where emp.deptno = dept.deptno ;

--- HASH JOIN

**LEADING**

Select /\* + LEADING (dept) + \*/ from emp, dept where emp.deptno = dept.deptno

**APPEND –** whenever you are doing DML operation the data will be inserted in archive log first then it will affect into our main table, if we use APPEND hint it will skip the Archive log activity and it will directly apply to our main table.

Insert /\* + APPEND \*/ into mytmp select /\* + cache (e) \*/ from emp e ;

Commit;

BINDING

CONTEXT SWITCHING

Binding types

In-Bind

Out\_Bind

Define

BULK COLLECT / FOR ALL

**Few more points to understand execution plan / performance tuning**

* **Statistics time I/O**
* **Spooling** (Duplicate Aggregation) – Only Lazy spool is bad
  + but, It is not bad all the time. You can’t avoid it in recursive CTE
* **Hash Match** (Unsorted ordered data)
  + It will happen in joins, if the joining fields doesn’t have an index, so optimizer will order it with the use of Hash Match. It will affect logical reads as well
* **Key Lookup** (missing column in index)
  + You are retrieving a field that doesn’t have an index, the optimizer will use key lookup to display the field. But it won’t effect on logical reads.
* **Bad Views**
* Sub Queries Minimization
* Partition Elimination
* Index Scan
* Index Seek
* Sargable query
* <https://www.youtube.com/watch?v=t2R0-xcKw44>

1. **How to load external .txt / .csv file into table in oracle?**

There are 3 methods to load external file data into a table.

* 1. **SQL Loader** (User for larger datasets, efficient and flexible data loading)
  2. **External Tables** (Use for direct querying of external files)
  3. **UTL\_FILE** (Custom file processing, for smaller datasets specific data manipulation in PLSQL)

1. **SQL Loader**

It is Oracle utility, which will be used to upload the data into Oracle database, you can easily export csv / txt files data into the oracle database.

**Input file:** This is the **file from which data** should be loaded. It can be a .txt file or .csv file

**Control file:** This will define the **configuration parameters** of the input files; this will tell how to load the data into the table.

**Log file**: This will record the **action of SQL Loader** and can be used for reference.

**Bad file**: This will load the records which are **failed** while insertion (due to any reason)

**Discard file**: Records which are **not satisfied** with the condition will be placed here.

**Important Loader Scenarios:**

* + 1. We can load multiple input data files into one table (Multiple files 🡪 1 table)

We have to mention that many files as **INFILE** in the control file

* + 1. We can load single input data file into multiple tables (1 file 🡪 Multiple Tables)

We must mention in control file by using **WHEN clause** after inset into table\_name

Ex: If we have multiple department related data in the input file we can mention in the WHEN clause WHEN dept\_number = 10

**Several clauses in SQL Loader:**

<https://docs.oracle.com/en/database/oracle/oracle-database/19/sutil/oracle-sql-loader-control-file-contents.html#GUID-4FE42E2C-8D84-4966-B150-A96CC83FA8B3>

<https://www.youtube.com/watch?v=PT0P_G5dcBY>

<https://www.youtube.com/watch?v=dn0o87N3FAE> FILLER AND SKIP IN CONTROLFILE

<https://www.youtube.com/watch?v=QSx_tuVJ2Vk> regexp\_replace functions in control file

<https://www.youtube.com/watch?v=qaEZtyHc5do> loading of CLOB BLOB from the other files which means loading of Character LOB and Binary LOB data into table.

**Clauses:**

**FILLER:** It will **skip / eliminate the data column** in the control file while loading

**SKIP:** SKIP(N)OPTIONSSKIP =1

It will **skip the record** (Generally it will skip the header column)

If SKIP = 3 means, it will skip the first 3 records in the data file during load.

**BADFILE:** It will create a explicitly if it is needed with the specified name.

**DISCARDFILE:** It will create a Discard File if it is needed with the specified name.

**DISCARDMAX:** It will move the mentioned max count records into Discard file.

**WHEN:** It will filter the rows in the data file while loading, with the use of this you can load into multiple target tables.

**CONSTANT:** It will give you the static value for the input data file during data load.

**LOBFILE:** LOBFILE (LOBFILE PATH), It will read the LOBfile w.r.t mentioned path in control file during load.

**APPEND | INSERT | TRUNCATE | REPLACE:** These will be helpful during the data load based on the table data scenario.

**SILENT:** ERRORS | FEEDBACK | HEADER | ALL

**FUNCTION:** Columnname “upper (:columnname)”

Syntax:

**Create sql loader control file with .ctl extension**

# vi **control**.ctl

--- Put below into the control file

**LOAD** **DATA**

INFILE ‘load\_file.**csv**’

**INSERT** **INTO** **TABLE** STUDENT

FILEDS TERMINATED **BY** ‘,’ OPTIONALLY ENCLOSED **BY** ‘”’

TAILING NULLCOLS

(ROLLNO, **NAME**)

SYNTAX: **To** **start** the **import**

**Goto** **to** the directory **in** command prompt

D:\DEMO> SQLLDR username/**password** **CONTROL**=CONTROL\_file.txt

Several INFILE mydat1.dat BADFILE mydat1.bad DISCARDFILE mydat1.dis

INFILE mydat2.dat

INFILE mydat3.dat DISCARDFILE mydat3.dis

INFILE mydat4.dat DISCARDMAX 10 0 Clauses we can use **in** **control** file **while** loading **data** thru. **SQL** Loader.

1. **Magic Tables in SQL Server**

These are the **temporary logical tables** that are created by the SQL **server whenever there are insertion or deletion or update (D.M.L) operations**. The recently performed operation on the rows gets stored in magic tables automatically. These are not physical table but they are just temporary internal tables. These **magic tables can’t be retrieved directly**, we need to **use triggers** to access these magic tables to get the deleted and inserted rows.

**When the following operations are done:**

* **INSERT –**   
  The recently inserted row gets added to the **INSERTED** magic table.
* **DELETE –**  
  The recently deleted row gets added to the **DELETED** magic table.
* **UPDATE –**  
  The updated row gets stored in **INSERTED** magic table and the old row or previous row gets stored in the **DELETED** magic table.

<https://www.geeksforgeeks.org/magic-tables-in-sql-server/>

1. **Unused Columns in Table for Oracle**

**Alter** **table** table\_name **set** unused (field5, field6);

**Select** \* **from** table\_name; the above two fields won’t come.

**Alter** **table** table\_name **drop** unused **columns;** (it will **drop** field5&6 cols)

* Compress and uncompress the partition from below url

<https://livesql.oracle.com/apex/livesql/docs/sqlrf/alter-table/drop-unused-cols.html>

1. **Difference between Global Temp Tables and Permanent Tables?**

Temporary Tables are very similar to the Permanent Tables, but

* **Permanent Tables get created in the database you specify**, and it will be there Permanently until and unless you (delete) and drop them.
* **Temporary Tables get created in Temp DB** only, and automatically deletes when no longer used or when the Server get restarted.

**Different ways to Create Temp Tables in SQL Server.**

1. **Table Variables** (DECLARE @customers TABLE) are visible only to the connection that creates it and are deleted when the batch or stored procedure ends.
2. **Local Temporary Tables** (CREATE TABLE #customers) are visible only to the connection that creates it and are deleted when the connection is closed.
3. **Global Temporary Tables** (CREATE TABLE ##customer) are visible to everyone and are deleted when all connections that have referenced them have closed.
4. **TempDB Permanent Tables** (USE tempdb CREATE TABLE customers) are visible to everyone and are deleted when the server is restarted.
5. **Difference between Local Temp Tables and Global Temporary Tables?**

**Local Temporary Tables:**

* It is available, **only for the current connection that has created the table.**

(It means the current query window)

* It is dropped, **automatically when the connection that has created is closed.**
* If it is **created in Stored Procedure, it will automatically drop when procedure execution done**.
* It is also **possible for different connections** (User1 and User2), to create a Local Temporary Table with the **same name**.
* **#** (Single Pound symbol) represent for Local Temporary Tables.
* Local Temporary Table Creation Syntax:

**In SQL**CREATE TABLE #ZZZ\_CUSTOMER\_TEMP\_TABLE (ID INT, CUSTOMER\_NAME VARCHAR(50));

**In Oracle**

CREATE PRIVATE TEMPORARY TABLE **ORA$PTT**\_TABLE\_NAME(ID INT, CUSTOMER\_NAME VARCHAR(50))

ON COMMIT DROP DEFINITION ( OR ) --- After commit TABLE will be dropped

ON COMMIT PRESERVE DEFINITION ; --- After commit TABLE will be presented.

**Global Temporary Tables (GTT):**

* It is available, **for all the connections and users.**

(It means all users and all query windows)

* It is dropped, **automatically when the last connection that is referencing that global temporary table is closed.**
* If it is **created in Stored Procedure, it will not be automatically dropped when procedure execution done**.
* It **wouldn’t be possible** **for different connections** (User1 and User2), to create a Local Temporary Table with the **same name**.
* **##** (Double Pound symbols) represents for Global Temporary Tables.
* Global Temporary Table Creation Syntax:

**In SQL** CREATE TABLE ##ZZZ\_CUSTOMER\_TEMP\_TABLE (ID INT, CUSTOMER\_NAME VARCHAR(50));

**In Oracle**

CREATE GLOBAL TEMPORARY TABLE TABLE\_NAME(ID INT, CUSTOMER\_NAME VARCHAR(50))

ON COMMIT DELETE ROWS ( OR ) --- After commit ROWS will be deleted

ON COMMIT PRESERVE ROWS ; --- After commit ROWS will be presented

<https://www.youtube.com/watch?v=oGuS1rdfaMI> (For sql)

<https://stackoverflow.com/questions/2920836/local-and-global-temporary-tables-in-sql-server> (For sql)

<https://www.ktexperts.com/oracle-18c-new-feature-private-temporary-tables/> (For Oracle)

1. **Tips to Improve SQL Query Performance**

* Use **EXISTS** instead of IN to check existence of data.
* Avoid **\*** in SELECT statement. Give the name of columns which you require.
* Choose appropriate Data Type. E.g. To store strings use **varchar** in place of text data type. Use text data type, whenever you need to store large data (more than 8000 characters).
* Avoid **nchar** and **nvarchar** if possible since both the data types takes just double memory as **char and varchar**.
* Avoid **NULL** in fixed-length field. In case of requirement of NULL, use variable-length (varchar) field that takes less space for NULL.
* Avoid Having Clause. Having clause is required if you further wish to filter the result of an aggregations.
* Create Clustered and Non-Clustered Indexes.
* Keep clustered index small since the fields are used in clustered index may also used in non-clustered index.
* Most selective columns should be placed leftmost in the key of a non-clustered index.
* Drop unused Indexes.
* Better to create indexes on columns that have integer values instead of characters. Integer values use less overhead than character values.
* Use joins instead of sub-queries.
* Use WHERE expressions to limit the size of result tables that are created with joins.
* Use TABLOCKX while inserting into a table and TABLOCK while merging.
* Use WITH (NOLOCK) while querying the data from any table.
* Use SET NOCOUNT ON and use TRY- CATCH to avoid deadlock condition.
* Avoid Cursors since cursor are very slow in performance.
* Use Table variable in place of Temp table. The use of Temp tables required interaction with TempDb database which is a time taking task.
* Use UNION ALL in place of UNION if possible.
* Use Schema name before SQL objects name.
* Use Stored Procedure for frequently used data and more complex queries.
* Keep transaction as small as possible since transaction lock the processing tables data and may results into deadlocks.
* Avoid prefix “sp\_” for user defined stored procedure name because SQL server **first** search the user defined procedure **in the master database** and after that in the current session database.
* Avoid use of Non-correlated Scalar Sub Query. Use this query as a separate query instead of part of the main query and store the output in a variable, which can be referred to in the main query or later part of the batch.
* Avoid Multi-statement Table Valued Functions (TVFs). Multi-statement TVFs are more costly than inline TVFs.

1. **Query was running fine yesterday but it’s very slow today | Tuning**

* A lot of DMLs would have happened recently. Therefore, Statistics might not be updated.
* Might be Database Parameters modified.
* Might be the Index was dropped.
* Any scheduled jobs are running at the same time
* Any other background process might be consuming the more memory
* May be the concurrent users are very high now accessing the same database.
* May be the any database patch applied.
* We need to check the query is changed recently by any reason.
* May be any Network Issue.
* May be the buffer cache / Temp Space got increased to 100%.
* We should verify the Database Server is good / Not.
* Finally, again we may check the execution plan, maybe we didn’t notify any bug which was already there and missed at that time.

<https://www.youtube.com/watch?v=nHL1YgtTapA>

1. **Query was running fine yesterday but it’s very slow today, No Data changes were happened.**

* **Memory Issue** in the Database Server.
* Parallelly **any Scheduled jobs** are running.
* **Index** may be **Invalid**.
* **CPU Utilization** may be reached too **High**.
* **Statistics** mightnot be updated.
* **AWR** Report
* **Query Plan** review

1. **Database Performance Tunning Steps**

* E – Business Logic
* D – Data Design
* E – Application Design
* D – Change DB Structure (like indexes)
* D – Tuning SQL
* D – Access Path / Access Method
* E – Memory Allocation – Instance Tuning (RAM)
* E/D – Tune the I/O – Physical Storage
* D – Tune the resources (dead locks)
* E – Tune the Server

Note: E for External, D for Databases

1. How to **FIND & DELETE duplicate records** in a table ?

----Find duplicates WITH GROUP BY

**Select** ***sno***, ***sname***, **count**(\*) **from** ***students***

**Group** **by** sno, sname

**Having** **count**(\*) > 1

--- Find duplicates WITHOUT GROUP BY

**Select** ***rowid***, *s1*.\* **from** ***students*** *s1* **where** ***rowid*** <

(**select** **max**(***rowid***) **from** ***students*** *s2* **where** *s1*.***sno*** = *s2*.***sno***);

**----DELETE DUPLICATES**

--- **DELETE ONLY ONE DUPLICATE RECORD**

**Delete** **from** ***students*** *s1* **where** ***rowid*** **not** **in**

(**select** **max**(***rowid***) **from** ***students*** **group** **by** ***sno***) ;

**---- DELETE ALL DUPLICATE RECORDS (have to check again)**

**Delete** **From** Zzz\_Test\_A **Where** Rowid **Not** **In** (

**select** **MIN**(rowid) **from** zzz\_test\_a **group** **by** name, salary ) ;

(**OR**)

**Delete** **From** Zzz\_Test\_A **Where** Rowid **Not** **In** (

**select** **MAX**(rowid) **from** zzz\_test\_a **group** **by** **name**, salary ) ;

1. Find a **Nth Highest salary**?

**--Way-1 with CTE**

**select** \* **from** (

**select** **name**, ***salary***, **dense\_rank**() **over** (**order** **by** ***salary*** **DESC**) *rank*

**from** ***zzz\_test\_a*** ) *cte*

**where** *rank* = 2;

**---Way-2 with CORELATED SUB QUERY**

**select** \* **from** ***zzz\_test\_a*** *a* **where** **2**-1 = (

**select** **count**(**distinct** ***salary***) **from** ***zzz\_test\_a*** *b* **where** *b*.***salaray*** > *a*.***salaray***

);

1. Difference between **Natural Join** and **Using Clause**?
   * NATURAL JOIN and USING Clause are **mutually** **exclusive**.
   * It should **not have a qualifier**(table name or Alias) in the referenced columns.
   * **NATURAL JOIN** uses **all the columns** with matching names and datatypes to join the tables. The **USING** Clause can be used to **specify only those columns** that should be used for an **EQUIJOIN**.
2. **FOR LOOP (example)**

<https://www.oracletutorial.com/plsql-tutorial/plsql-for-loop/>

1. **FOR ALL Use case**

The FORALL statement is specifically designed for bulk operations in PL/SQL. It allows you to perform DML operations on a collection of values in a single context switch, significantly improving performance.

* **Bulk Binding:** Executes a single DML statement for all elements in a collection, minimizing context switches.
* **Performance:** Enhanced performance for large datasets due to reduced context switching.
* **DML Operations Only:** Primarily used for INSERT, UPDATE, and DELETE operations.

1. **Difference between FOR and FOR ALL**

### Key Differences:

1. **Execution Method:**
   * **FOR:** Executes SQL statements individually for each iteration.
   * **FORALL:** Executes a **single SQL statement for all elements** in the collection.
2. **Performance:**
   * **FOR:** Involves **multiple context switches** between PL/SQL and SQL engines, which can be slow for large datasets.
   * **FORALL:** **Minimizes context switches** by executing the SQL statement in bulk, resulting in better performance for large datasets.
3. **Usage:**
   * **FOR:** General-purpose looping construct suitable for various operations, including those that require complex logic.
   * **FORALL:** Specifically designed **for bulk DML operations**, such as INSERT, UPDATE, and DELETE.
4. **Error Handling:**
   * **FOR:** Handles errors individually for each iteration.
   * **FORALL:** Can use the **SAVE EXCEPTIONS clause** to handle errors and continue processing the remaining elements.

**FOR ALL Example:**

Case: Write PLSQL block to load all employees info from collection into **EMP\_**BKP table using FOR ALL.

**DECLARE**

**TYPE** t\_emp\_list **IS** **TABLE** **OF** EMP%ROWTYPE;

lv\_emp\_list t\_emp\_list := t\_emp\_list();

**BEGIN**

**SELECT** \*

BULK **COLLECT** **INTO** lv\_emp\_list

**FROM** emp

**ORDER** **BY** empno;

**--Using For Loop (it will degrade the performance due to context switching**

**FOR** I **in** lv\_emp\_list..**first**..lv\_emp\_list.**last**

**LOOP**

**INSERT** **INTO** emp\_bkp **values** lv\_emp\_list(i);

**END** **LOOP**;

**--- Using FOR ALL, it will increase the performance nearly 10times better with FOR LOOP (when we use FOR ALL we need not use LOOP and END LOOP explicitly)**

**FOR** ALL I **in** lv\_emp\_list..**first**..lv\_emp\_list.**last**

**INSERT** **INTO** emp\_bkp **values** lv\_emp\_list(i);

**END;**

Database Documents

1. Conceptual

2. Logical

3. Physical

OLTP / OLAP (Applications)

(DML) / (Non-DML)

Tran / Analy

Change Data Capture (Config.)

Oracle Golden Gate (ETL Tool) (EE)

Oracle SQL \* Lite (free)

AWS DMS (Data Migration Service)

Snowflake

NDS / DDS / Data Mart

**PostgreSQL**

1. Random id generation
2. Store Procedures introduced in PostgreSQL 11 Version, (as we know that upto PostgreSQL 10 version we are all using functions as Procedures, the 11th version introduced in October 2018).
3. The default port is **5432.**
4. If you want to migrate ORACLE to PostgreSQL then PostgreSQL 11th version is suitable why because in this version the store procedure technology introduced.
5. POSTGRESQL ARCHITECTURE

<https://www.youtube.com/watch?v=Ko0ldsIgJ7o>

1. Post Master (when the new connection / database cluster starts this will start from the PostMaster)
2. Shared memory (The complete share Memory itself is a catche)

Shared Buffers (like a catche)

WAL Buffers

(like a log book)

Any DML statement happens it will update into shared memory and an entry In WAL Buffers. It should be containing the OLD and NEW, Every Commit occurs the data will be pushed into the WAL Segments.

We can’t take a backup every minute, so it will store every log into this. It is kind of a logbook)

Process Array

1. Data Files
2. WAL Segments
3. Archived WAL (When WAL segments got full it will push into the archived WAL, it will help us like we have a Database backup on 5th, but the database crashed on the 8th, in this case we will apply while restoring the backup file i.e., on 5th in addition to this will apply archived WAL till 8th, so the latest data will come.)
4. Error Log files. (which is containing the log messages, Warning messages and information messages that major thing happening to the database, not the DML and DDL changes
5. What is **the latest version of PostgreSQL** running in the marker.?

**14.2 version it was release in the year Sep-2021**

1. How to check the PostgreSQL version.?

**Select version() ;**

1. How to get the first ‘N’ records in PostgreSQL.?

**We can use LIMIT.**

1. What is difference between RDBMS and ORDBMS.?

<https://www.geeksforgeeks.org/difference-between-rdbms-and-ordbms/>

1. UPSERT function will work like MERGE INTO Statement.
2. What is **maximum size of a table** in PostgreSQL?

PostgreSQL has maximum size of a table is **32 TB**

1. **What are the indices of PostgreSQL ?**

**Indices of PostgreSQL** are inbuilt functions or methods like**GIST Indices**, **hash table** and **B-tree (Binary tree)** which can be used by the user to scan the index in a backward manner. Users can also define their indices of PostgreSQL.

1. **How to change a filed / column data type in a table in PostgreSQL?**

ALTER TABLE table\_name ALTER COLUMN column\_name [SET DATA] TYPE new\_data\_type;

**ALTER** **TABLE** epatasala.gsws\_mandals\_master **ALTER** **COLUMN** lgd\_dist\_code **TYPE** **int4** **USING** lgd\_dist\_code::**int4**;

1. **What are the disadvantages of PostgreSQL?**[**↑**](https://www.adaface.com/blog/postgresql-interview-questions/#questionsindex)

* Postgres **is not owned by one organization**. So, it has had trouble getting its name out there despite being fully featured and comparable to other DBMS systems
* Changes made for **speed improvement** requires more work than MySQL as PostgreSQL focuses on compatibility
* **Many open source apps support MySQL, but may not support PostgreSQL**
* On **performance metrics**, it is slower than MySQL.

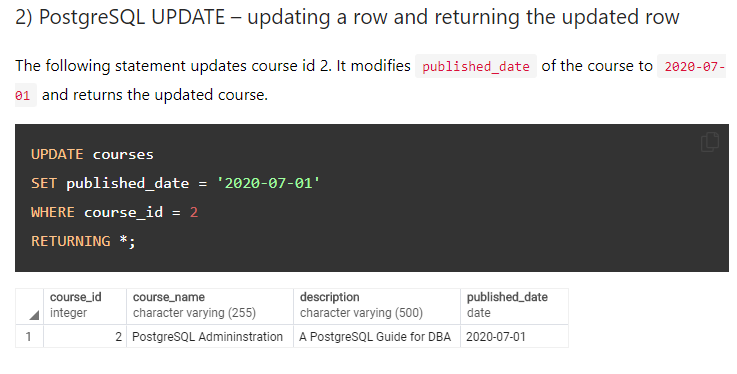
1. **What are the advantages of PostgreSQL?**[**↑**](https://www.adaface.com/blog/postgresql-interview-questions/#questionsindex)

* PostgreSQL can run dynamic websites and web apps as a LAMP stack option.
* PostgreSQL's **write-ahead logging** makes it a **highly fault-tolerant** database.
* PostgreSQL **source code is freely available** under an **open-source** license. This allows you the freedom to use, modify, and implement it as per your business needs.
* PostgreSQL supports **geographic objects** so you can use it for **location-based services** and geographic information systems
* PostgreSQL supports geographic objects so it can be used as a geospatial data store for location-based services and geographic information systems
* To learn Postgres, you don't need much training as it’s easy to use
* **Low maintenance** administration for both embedded and enterprise use

1. **How Many Columns can a postgres table have?**

**1600 columns**

1. **Returning in PostgreSQL**



1. **What is Constant?**

There are three kinds of *implicitly-typed constants* in PostgreSQL: **strings, bit strings**, and **numbers**. Constants can also be specified with explicit types,

* A **String constant** in SQL is an arbitrary sequence of characters bounded by single quotes ('), for example 'This is a string'. To include a single-quote character within a string constant, write two adjacent single quotes, e.g., 'Dianne''s horse’.

(String constants have a sequence of characters, usually bound by single quotes.)

* **Bit constant**
* **Number constant**

1. **Sequence Manipulation Functions?**

| **Function** | **Return Type** | **Description** |
| --- | --- | --- |
| currval(regclass) | bigint | Return value most recently obtained with nextval for specified sequence |
| lastval() | bigint | Return value most recently obtained with nextval for any sequence |
| nextval(regclass) | bigint | Advance sequence and return new value |
| setval(regclass, bigint) | bigint | Set sequence's current value |
| setval(regclass, bigint, boolean) | bigint | Set sequence's current value and is called flag |

1. **What is UUID, SERIAL, IDENTITY COLUMNS, AUTO GENERATION PRIMARY KEYS?**

**UUID = “uuid-ossp” function**

**Uuid\_generate\_v4() , It’s a 128 bit**

<https://www.cybertec-postgresql.com/en/uuid-serial-or-identity-columns-for-postgresql-auto-generated-primary-keys/>

<https://www.youtube.com/watch?v=lX23m3hNV9A>

1. **Difference between uuid and guid?**

UUID is a term that stands for **Universal Unique Identifier**. Similarly, GUID stands for **Globally Unique Identifier**.

1. Difference between **serial and bigserial**?

**serial** 4 bytes autoincrementing integer 1 to **2147483647**

**bigserial** 8 bytes large autoincrementing integer 1 to **9223372036854775807**

1. **ROWID** in **Oracle** and **ROW\_NUBMER()** in **Postgres/SQL Server?**

The PostgreSQL row\_number() window function can be used for most purposes where you would use rowid. Whereas in Oracle the rowid is an intrinsic numbering of the result data rows, in Postgres row\_number()

**Select row\_number() over() from table\_name;**

**(or)**

**Select row\_number() over(order by field\_name) from table\_name;**

1. **Regular expression in postgresql**

To get the records started with Alphabet / Digit.

<https://www.sqlshack.com/working-with-regular-expressions-in-postgresql/>

1. How to **RESTRICT SPACES** (like CHECK constraint) & CUSTOME DATA TYPE into a field in postgres?

PostgreSQL gives us the flexibility to create user defined custom datatypes besides using the built in datatypes.

We can create a custom datatype using either **CREATE DOMAIN**or **CREATE TYPE**. CREATE DOMAIN creates the user defined datatype with support to use constraints such as NOT NULL, CHECK, etc. CREATE TYPE creates a composite user defined datatype, which is used in a stored procedure as the data type of the returned value.

<https://www.sqlservercentral.com/articles/array-and-custom-data-types-in-postgresql#:~:text=We%20can%20create%20a%20custom,type%20of%20the%20returned%20value>.

1. What is **MVCC?**

The promise is simple: **reads never block writes and vice versa**. Postgres achieves this via a mechanism called **Multi Version Concurrency Control**. This technique is not unique to Postgres: there are several databases that implement some form of MVCC including Oracle, Berkeley DB, CouchDB and many more.

1. **Difference between JSON and JSONB?**

In most cases JSONB is likely what you want when looking for a NoSQL, schema-less, datatype. Hstore and JSON can have their place as well but it’s less common. More broadly, JSONB isn’t always a fit in every data model. Where you can normalize there are benefits, but if you do have a schema that has a large number of optional columns (such as with event data) or the schema differs based on tenant id then JSONB can be a great fit. In general, you want:

**JSONB** - In most cases

**JSON** - If you’re just processing logs, don’t often need to query, and use as more of an audit trail

**HStore** - Can work fine for text-based key-value looks, but in general JSONB can still work great here.

[**https://stackoverflow.com/questions/39637370/difference-between-json-and-jsonb-in-postgres**](https://stackoverflow.com/questions/39637370/difference-between-json-and-jsonb-in-postgres)

1. **Delete duplicates in PostgreSQL?**

**CREATE** **TABLE** ipr\_hms.basket(

id SERIAL **PRIMARY** **KEY**,

fruit **VARCHAR**(50) **NOT** **NULL** );

**INSERT** **INTO** ipr\_hms.basket(fruit) **values**('apple');

**INSERT** **INTO** ipr\_hms.basket(fruit) **values**('apple');

**INSERT** **INTO** ipr\_hms.basket(fruit) **values**('orange');

**INSERT** **INTO** ipr\_hms.basket(fruit) **values**('orange');

**INSERT** **INTO** ipr\_hms.basket(fruit) **values**('orange');

**INSERT** **INTO** ipr\_hms.basket(fruit) **values**('banana');

* **Delete Duplicate rows with using**

With keeping lowest id

**Delete** **from** basket a **USING** basket b **where** a.id **>** b.id **AND** a.fruit = b.fruit;

(or)

With keeping highest id

**Delete** **from** basket a **USING** basket b **where** a.id **<** b.id **AND** a.fruit = b.fruit;

* **Delete duplicate rows with using immediate table**

**SELECT** **DISTINCT** **ON** (fruit) fruit, id **FROM** ipr\_hms.basket; (Distinct on clause fyi)

-- step 1

**CREATE** **TABLE** ipr\_hms.basket\_temp (**LIKE** basket);

-- step 2

**INSERT** **INTO** ipr\_hms.basket\_temp(fruit, id)

**SELECT**

**DISTINCT** **ON** (fruit) fruit, id **FROM** ipr\_hms.basket;

-- step 3

**DROP** **TABLE** ipr\_hms.basket;

-- step 4

**ALTER** **TABLE** ipr\_hms.basket\_temp **RENAME** **TO** ipr\_hms.basket;

* **Delete duplicate rows without id/unique identifier**

**Delete FROM ipr\_hms.basket T1**

**using ipr\_hms.basket T2**

**WHERE T1.ctid < T2.ctid** -- select the "older" ones

**AND T1.fruit = T2.fruit** -- list columns that define duplicates

;

Link: <https://www.postgresqltutorial.com/how-to-delete-duplicate-rows-in-postgresql/>

<https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-select-distinct/>

1. **What is CTID?**

It is a block, and consists of physical rows by their blocks.

These are usually used by index entries to identify the physical rows.

the ctid can be used to locate the row version very quickly, a row's ctid will change **if it is updated or moved by VACUUM FULL**. Therefore CTID is useless as a long-term row identifier. A primary key should be used to identify logical rows.

1. **Can we use set operators for collections as well?**

**Yes**, we can use.

<https://oracle-base.com/articles/8i/collections-8i#:~:text=The%20MULTISET%20UNION%20operator%20joins,ALL%20operators%20are%20functionally%20equivalent.&text=The%20DISTINCT%20keyword%20can%20be,operations%20to%20removes%20the%20duplicates>.

1. **Some of the functionalities in PostgreSQL**

There are mainly five functionalities in PostgreSQL

There Relational database objects

* 1. Client-server and WAL
  2. DB validation
  3. Extensibility and SQL Support
  4. Application program interface
  5. Procedural Language and Multi version concurrency control

1. How can one perform **queries using multiple databases**?

The cross-database queries can be made possible by using **dblink** or **contrib** with the help of function calls.

1. How do you identify the time/date gap between **two timestamps** in postrgres?

**Age function**

ex: age(arrivaltime, departuretime)

[**https://learnsql.com/cookbook/how-to-calculate-the-difference-between-two-timestamps-in-postgresql/**](https://learnsql.com/cookbook/how-to-calculate-the-difference-between-two-timestamps-in-postgresql/)

1. Difference between datatypes of  **timestamp and timestamptz**?

**Timestamptz** includes timezone offset

**Timestamp** doesn’t include it.

<https://kb.objectrocket.com/postgresql/postgresql-timestamp-vs-timestamptz-616#:~:text=time%20related%20tasks.-,Timestamp%20vs%20Timestamptz%20%E2%80%93%20What's%20the%20Difference%3F,important%20to%20keep%20or%20not>.

1. How can you update **stats in PostgreSQL?**

You must make an **explicit vacuum cell** to update statistics in PostgreSQL. This is arrived at through a definite method, and **only vacuum** with the option is analyzed.

Its syntax is **VACUUM ANALYZE**

1. **How can we use variables which are declared at Package level?**
2. **Clob, Blob**
3. **Compound Trigger. (intro. In 11g)**
4. **To know the executed program name (from where the statement is executing like (sql developer tool / dbeaver / frontend(.exe) you need to pass ‘MODULE’ as parameter For “ sys\_context “ program.**

SELECT SYS\_CONTEXT ('USERENV', ‘MODULE’) FROM DUAL;

1. How to create Directory and how to use it?
2. Sequence question?

(created new sequence with 100 and selecting currval of syntax from dual what is the output)

Ans: It will give error

1. What is Single Sub Query & Multiple Sub Query?
2. View Types (Force view and inline views)?

Inline view :

**The subquery specified in the FROM clause of a query is called an inline view**. Because an inline view can replace a table in a query, it is also called a derived table. Sometimes, you may hear the term subselect, which is the same meaning as the inline view.

<https://www.oracletutorial.com/oracle-view/inline-view-in-oracle/>

1. Difference between inline view and subquery?

The first difference is that inline views can contain multiple columns, while subqueries (in the Oracle meaning) should return only one. The reason is simple – an inline view works like a table and tables can contain more than one column. Subqueries, on the other hand, generally work as a single value

1. PLSQL Data Types?

PLS\_INTEGER, BINARY\_INTEGER, SIMPLE\_INTEGER

1. Syntax to create varray / nested table / associated table?

Create or replace TYPE my\_nested\_table IS TABLE OF VARCHAR2(10);

Create table (

Sub\_id number,

Sub\_name varchar(20),

Sub\_schedule\_day my\_nested\_table

) NEASTED TABLE sub\_schedule\_day STORE AS nested\_tab\_space ;

Granting Procedure execution to a user

CREATE ROLE db\_execproc;

GO

EXEC sp\_addrolemember N'db\_execproc', N'Dvv\_Prod';

GO

GRANT EXECUTE ON SCHEMA::dbo TO db\_execproc;

GO

**DevOps**

* + 1. Difference between Deploy and Release.

Release: It’s visible to public.

Deploy: Changes / code is ready in the production.

You have to use some feature flags from Deploy to Release.