What is PL/SQL?

- Procedural Language/ Standard Query Language.
- It is a procedural server side programming language
- Used to bridge the gap of SQL being non-procedural.
- Case-insensitive programming language.

```
DECLARE
  --Declaration statements
BEGIN
  --Executable statements
EXCEPTION
  --Exception handling statements
END;
```

Advantages of PL\SQL

- Tight integration with SQL
- Business logic can be directly implemented at database level
- High performance, High productivity
- Support object oriented programming

Tip:
Format SQL : Ctrl+F7
Open worksheet: ALT+F10

PL/SQL Blocks

Blocks are basic programming units in PL/SQL programming language

Anonymous Block

As this block is created without a name, this block does not create any object in the database. Thus, the scope for reusability is zero. It compiles every time you execute.

Named Block

It creates a database object. Complied for one time and stored for reuse.

Early vs. Late Binding

Late binding means code is compiled at execution. Early binding means code is compiled prior to execution.

PL/SQL Data types

Scalar

Single values with no internal components.

Numeric: `BINARY_DOUBLE, BINARY_FLOAT, BINARY_INTEGER, DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, NATURAL, NATURALN, NUMBER, NUMERIC, PLS_INTEGER, POSITIVE, POSITIVEN, REAL, SIGNTYPE, SMALLINT`

Character: `CHAR, CHARACTER, LONG, LONG RAW, NCHAR, NVARCHAR2, RAW, ROWID, STRING, UROWID, VARCHAR, VARCHAR2`

Boolean: `BOOLEAN`

Datetime: `DATE, TIMESTAMP, TIMESTAMP WITH TIMEZONE, TIMESTAMP WITH LOCAL TIMEZONE, INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND`

Large Object (LOB)

Pointers to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.

`BFILE (Binary File - Points to File Outside DB), BLOB (Binary Large Object), CLOB (Character Large Object), NCLOB (National Character Large Object)`

Composite

Data items that have internal components that can be accessed individually.
RECORDS, COLLECTIONS

Reference

Pointers to other data items.

PLS_INTEGER and BINARY_INTEGER

PLS_INTEGER

- PLS_INTEGER gives better performance
- Requires less storage space than INTEGER and NUMBER types
- Use when more calculations are there
- Uses machine arithmetic; while NUMBER requires additional calls to library routines.
- INTEGER require extra runtime checks

BINARY_INTEGER

- Used for declaring signed integer variables
- Stored in binary format , which takes less space
- Calculations can run slightly faster because the values are already stored in binary format
- Uses native math libraries which does not allocate memory to store the variable until a value is assigned.

Note:
- **NULL** and **Boolean** cannot be printed
- To turn on output: `SET SERVEROUTPUT ON;`
- Only one LONG column can be used per table.
- The minimum column width that can be specified for a varchar2 data type column is one.
- The value for a CHAR data type column is blank-padded to the maximum defined column width.

RAW Datatype

In Oracle PL/SQL, RAW is a data type used to store binary data, or data which is byte oriented (for example, graphics or audio files). One of the most important things to note about RAW data is that it can only be queried or inserted; RAW data cannot be manipulated. RAW data is always returned as a hexadecimal character value

CONSTANT, DEFAULT, NOT NULL

```
SET SERVEROUTPUT ON;
DECLARE
  V_PI CONSTANT NUMBER(7,6):=3.14; --Assigning is mandatory
  V_NAME VARCHAR2(20) DEFAULT 'Unknown'; --Assigning is mandatory
  V_AGE NUMBER NOT NULL :=50; --Assigning is mandatory
BEGIN
  DBMS_OUTPUT.PUT_LINE('v_pi:' ||V_PI);
  DBMS_OUTPUT.PUT_LINE('v_name:'||V_NAME);
  DBMS_OUTPUT.PUT_LINE('v_age:'||V_AGE);
END;
```

Host/Bind/Session Variable

It is a variable of the interface. This variable can be bonded with SQL or PL\SQL anonymous block. The scope of these variables is till the end of the session. These variables always preceded with a colon (:).

```
VARIABLE v_bind1 VARCHAR2(25); --Not PL/SQL statement
DECLARE
BEGIN
  :v_bind1 := 'Binding 1';
  DBMS_OUTPUT.PUT_LINE(:v_bind1);
END;
--Not PL/SQL statements--
VARIABLE v_bind2 VARCHAR2(25);
VARIABLE v_bind3 VARCHAR2(25);
EXECUTE :v_bind2 := 'Binding 2'; -- SQL*Plus command
EXECUTE :v_bind3 := 'Binding 3';
PRINT :v_bind2;
PRINT; -- Displays all bind variable values in the session
----
SET AUTOPRINT ON -- To turn on automatic printing of bind variable while assigning
```

Anchored Data type/ Inheriting data type

It is used to pick up data type and size from a previously declared object into a new variable. Advantage of this is, when you change the data type or size of the field in the table, it will also affect this variable. So there is less maintenance.

Ex: VNAME `EMP.ENAME %TYPE;`	`%ROWTYPE` : Record datatype variable

Composite Variable

It is a variable created by combining two or more individual variables, called indicators, into a single variable.

Records and Collections

- Records and Collections are composite variable types.
- A **record** is a group of related data items which can have different data types called fields. Each element is addressed by VARIABLE_NAME.FIELD_NAME.
- A **collection** is group of elements, which have the same data type called elements. Each element is addressed by a unique subscript. Ex: VARIABLE_NAME(INDEX).

Table-based Record

```
DECLARE
  emp_rec emp %ROWTYPE;
BEGIN
  SELECT * INTO emp_rec
  FROM emp
  WHERE empno=7369;
END;
```

Cursor-based Record

```
DECLARE
  CURSOR emp_cur IS
    SELECT empno, ename
    FROM emp;
BEGIN
  FOR i IN emp_cur LOOP
  DBMS_OUTPUT.PUT_LINE(i.ename);
  END LOOP;
END;
```

User-defined Record

```
DECLARE
  TYPE mytype IS RECORD(
    vemp_name VARCHAR2(20),
    vemp_salary emp.sal %TYPE);
  vdata mytype;
BEGIN
  SELECT ename,sal INTO vdata
  FROM emp WHERE empno=7369;
DBMS_OUTPUT.PUT_LINE(vdata.vemp_name);
END;
```

Nested Tables

- Persistent collection - Stores data and structure physically into the database as database object
- No upper limits on rows (Unbounded)
- Need external table for its storage (**STORE AS** clause `--while creating table`)
- Initialization needed before assigning values to elements

Nested Table type as block member

```
DECLARE
  TYPE names_table IS TABLE OF VARCHAR2(10);
  TYPE grades IS TABLE OF INTEGER(2);
  names names_table;
  marks grades;
  total INTEGER(3);
BEGIN
  names := names_table('Kavita','Pritam','Ayan','Rishav','Aziz'); -- Initialization
  names(1) := 'Gaurav'; --Assigning
  marks := grades(98,97,78,87,92);
  total := names.COUNT;
  FOR i IN 1 .. total LOOP
    DBMS_OUTPUT.PUT_LINE('Student:' || names(i) || ' Marks:' || marks(i));
  END LOOP;
END;
```

Nested table type as Database Object

```
CREATE OR REPLACE TYPE my_nested_table IS TABLE OF VARCHAR2(10);
----
CREATE TABLE my_subject(
  sub_id NUMBER(3),
  sub_name VARCHAR2(20),
  sub_schedule_day my_nested_table --nested table type
) NESTED TABLE sub_schedule_day --name of the column you want to use as nested table column
STORE AS nested_tab_space; -- storage table for your nested table type (user-defined name)
----
INSERT INTO my_subject VALUES(101,'Maths',my_nested_table('Monday','Friday'));
----
SELECT sub.sub_id, sub.sub_name,ss_day.COLUMN_VALUE
FROM my_subject sub,
TABLE(sub.sub_schedule_day) ss_day -- Table expression
```

Nested table using user defined datatype

```
CREATE OR REPLACE TYPE object_type AS OBJECT( --type object_type now can be used as any other
built-in datatype like VARCHAR or NUMBER
  obj_id NUMBER,
  obj_name VARCHAR2(10)
);
----
CREATE OR REPLACE TYPE my_nesd_tbl IS TABLE OF object_type; --It is not possible to add size
limit to user defined datatype like object_type(5) as we do with VARCHAR2(5)
----
CREATE TABLE base_table(
  tab_id NUMBER,
  tab_ele my_nesd_tbl
) NESTED TABLE tab_ele STORE AS store_tab_1;
```

VARRAYs

- Persistent collection - Stores data and structure physically into the database as database object
- Can hold fixed number of elements(Bounded)
- Modified version of Nested tables
- Stored in-line with their parent record as raw value in the parent table (No need of STORE AS clause)
- Initialization needed before assigning values to elements

VARRAYs as block member

```
DECLARE
  TYPE team_four IS VARRAY(4) OF VARCHAR2(15);
  team team_four;
BEGIN
  team := team_four('John','Mary','Alberto','Juanita'); -- Initialization
  team(3) := 'Pierre'; --Assigning
  team(4) := 'Yvonne';
  FOR i IN 1..team.LIMIT LOOP
    DBMS_OUTPUT.PUT_LINE(i || team(i));
  END LOOP;
END;
```

Interview (1) Questions and Answers

? Get the first day of the month

```
SELECT TRUNC (SYSDATE, 'MONTH') "First day of current month" FROM DUAL;
```

? Get the first day of the Year

```
SELECT TRUNC (SYSDATE, 'YEAR') "Year First Day" FROM DUAL;
```

? Get the last day of the month

```
SELECT TRUNC (LAST_DAY (SYSDATE)) "Last day of current month" FROM DUAL;
```

? Get the last day of the year

```
SELECT ADD_MONTHS (TRUNC (SYSDATE, 'YEAR'), 12) - 1 "Year Last Day" FROM DUAL
--(TRUNC is used to get extract first day of the year from SYSDATE + 12 months = 1st
Jan next year) - 1 day
```

? Get number of days in current month

```
SELECT CAST (TO_CHAR (LAST_DAY (SYSDATE), 'dd') AS INT) number_of_days FROM DUAL;
```

? Get number of days left in current month

```
SELECT SYSDATE,
       LAST_DAY (SYSDATE) "Last",
       LAST_DAY (SYSDATE) - SYSDATE "Days left"
  FROM DUAL;
```

? Get number of days between two dates

```
SELECT ROUND ( (MONTHS_BETWEEN ('01-Feb-2014', '01-Mar-2012') * 30), 0) num_of_days
FROM DUAL;
(OR)
SELECT TRUNC(sysdate) - TRUNC(e.hire_date) FROM employees e;
```

? Display each months start and end date upto last month of the year

```
SELECT ADD_MONTHS (TRUNC (SYSDATE, 'MONTH'), i) start_date,
       TRUNC (LAST_DAY (ADD_MONTHS (SYSDATE, i))) end_date
  FROM XMLTABLE (
          'for $i in 0 to xs:int(D) return $i'
          PASSING XMLELEMENT (
            d,
            FLOOR (MONTHS_BETWEEN (
              ADD_MONTHS (TRUNC (SYSDATE, 'YEAR') - 1, 12),
              SYSDATE)))
  COLUMNS i INTEGER PATH '.');
```

START_DATE	END_DATE
01-06-2019	30-06-2019
01-07-2019	31-07-2019
01-08-2019	31-08-2019
01-09-2019	30-09-2019
01-10-2019	31-10-2019
01-11-2019	30-11-2019
01-12-2019	31-12-2019

? Get number of seconds passed since today (since 00:00 hr)

```
SELECT (SYSDATE - TRUNC (SYSDATE)) * 24 * 60 * 60 num_of_sec_since_morning FROM
DUAL;
```

? Get number of seconds left today (till 23:59:59 hr)

```
SELECT (TRUNC (SYSDATE+1) - SYSDATE) * 24 * 60 * 60 num_of_sec_left FROM DUAL;
```

? Check if a table exists in the current database schema

```
SELECT table_name
  FROM user_tables
 WHERE table_name = 'TABLE_NAME';
```

? Check if a column exists in a table

```
SELECT column_name AS FOUND
  FROM user_tab_cols
 WHERE table_name = 'TABLE_NAME' AND column_name = 'COLUMN_NAME';
```

? Showing the table structure

```
SELECT DBMS_METADATA.GET_DDL ('TABLE', 'TABLE_NAME', 'USER_NAME') FROM DUAL;
-- to get DDL for a view just replace first argument with ‘VIEW’ and second with
your view name and so.
```

? Getting current schema

```
SELECT SYS_CONTEXT ('userenv', 'current_schema') FROM DUAL;
```

? Changing current schema

```
ALTER SESSION SET CURRENT_SCHEMA = new_schema;
```

? Database version information

```
SELECT * FROM v$version;
```

? Database default information

```
SELECT username,
       profile,
       default_tablespace,
       temporary_tablespace
  FROM dba_users;
```

? Database Character Set information

```
SELECT * FROM nls_database_parameters;
Get Oracle version
SELECT VALUE
  FROM v$system_parameter
 WHERE name = 'compatible';
```

? Store data case sensitive but to index it case insensitive

```
CREATE TABLE tab (col1 VARCHAR2 (10));

CREATE INDEX idx1
   ON tab (UPPER (col1));

ANALYZE TABLE tab COMPUTE STATISTICS;
 --In your query you might do UPPER(..) = UPPER(..) on both sides to make it case
insensitive. Now in such cases, you might want to make your index case insensitive
so that they don’t occupy more space.
```

? Checking autoextend on/off for Tablespaces

```
SELECT SUBSTR (file_name, 1, 50), AUTOEXTENSIBLE FROM dba_data_files;
(OR)
SELECT tablespace_name, AUTOEXTENSIBLE FROM dba_data_files;
```

? Adding datafile to a tablespace

```
ALTER TABLESPACE data01 ADD DATAFILE '/work/oradata/STARTST/data01.dbf'SIZE 1000M
AUTOEXTEND OFF;
```

To modify VARRAY size limit

```
ALTER TYPE type_name MODIFY LIMIT new_size_limit [INVALIDATE | CASCADE]
```

--**INVALIDATE**: Marks all dependent TYPES and TABLES as INVALID
--**CASCADE**: Cascades(propagate) the change to all dependent TYPES and TABLES

VARRAY as Database Object

```
CREATE OR REPLACE TYPE dbObj_vry IS VARRAY(5) OF NUMBER;
----
CREATE TABLE calendar(
  day_name VARCHAR2(25),
  day_date dbObj_vry); -- No STORE AS clause is needed
----
INSERT INTO calendar VALUES('Sunday',dbObj_vry(7,14,21,28));
----
SELECT tab1.day_name, tab1.day_date
FROM calendar tab1; -- Without Table expression
----
SELECT tab1.day_name, vry.COLUMN_VALUE
FROM calendar tab1,
TABLE (tab1.day_date) vry; -- Table expression
```

--without table expression

DAY_NAME	COLUMN_VALUE
Sunday	HR.DBOBJ_VRY(7,14,21,28)

--with table expression

DAY_NAME	COLUMN_VALUE
Sunday	7
Sunday	14
Sunday	21
Sunday	28

Associative Arrays/ Index by table

- Non-Persistent collection - Stores data and structure just for one session. No database object can be created.
- Unbounded collection
- Hold similar data type in key-value pair
- Can access elements using numbers and strings as subscript values.
- Similar to hash table in other languages.
- Not need of initialization before assigning values to elements

```
DECLARE
  TYPE salary IS TABLE OF NUMBER(5) INDEX BY VARCHAR2(20);
  salary_list salary;
  name VARCHAR2(20);
BEGIN
  salary_list('Ranjish') := 6200; --No initialization needed before assigning
  salary_list('Minakshi') := 75000;
  salary_list('Martin') := 10000;
  name := salary_list.FIRST;
  WHILE name IS NOT NULL LOOP
    DBMS_OUTPUT.PUT_LINE('Salary of ' || name || 'is ' || salary_list(name));
    name := salary_list.NEXT(name);
  END LOOP;
END;
```

NLS_SORT and NLS_COMP

- In associative arrays, string index's sort order is determined by the initialization parameters NLS_SORT and NLS_COMP parameter
- If you change the value of either parameter after populating an associative array indexed by string, then the collection methods FIRST, LAST, NEXT, and PRIOR might return unexpected values or raise exceptions. If you have changed these parameter values during your session, restore their original values before operating on associative arrays indexed by string.
- Default value for both parameter is BINARY

Collection Methods (3 Procedures + 7 Functions)

DELETE- Deletes elements from collection using index.

```
    names.DELETE; --delete all
    names.DELETE(1); --delete index 1
    names.DELETE(3,6) --delete index from 3 to 6
```

TRIM - Deletes elements from end of varray or nested table.

```
    names.TRIM; --removes one element from the end of the collection
    names.TRIM(5); --removes 5 elements from the end of the collection
```

EXTEND - Memory for storing data has to be allocated before assigning value to the individual elements in the collection. It adds elements to end of varray or nested table. Cannot be used with Associative array.

```
    names.EXTEND; -- occupy one element with NULL
    names.EXTEND(5); -- occupy 5 elements with NULL
    names.EXTEND(5,1); --5 elements in the collection will be initialized with the value in
    the index 1 that is 28.
```

EXISTS - Returns TRUE if and only if specified element (index) of varray or nested table exists.

```
    IF names.EXISTS(1) THEN
        DBMS_OUTPUT.PUT_LINE(names.COUNT);
    END IF;
```

FIRST,**LAST** - Returns first and last index (subscript) in collection.

```
    DBMS_OUTPUT.PUT_LINE (names.FIRST); -- prints the index of first element
    DBMS_OUTPUT.PUT_LINE (names(names.LAST)); -- prints the value of last element
```

COUNT - Returns number of elements in collection. No empty indexes are counted.

```
    DBMS_OUTPUT.PUT_LINE(names.COUNT);
```

LIMIT - Returns maximum number of elements that collection (varray only) can have whether it is empty or not. For nested tables and associative arrays, which have no limit in size, LIMIT will return NULL.

```
    DBMS_OUTPUT.PUT_LINE(names.LIMIT);
```

PRIOR, **NEXT** - Returns index that precedes and succeeds specified index.

```
    DBMS_OUTPUT.PUT_LINE (names.PRIOR(3)); -- prints the index of previous element
    DBMS_OUTPUT.PUT_LINE (names(names.NEXT(3))); to print the value of next element
```

EXTEND Procedure with 1 argument

```
DECLARE
  TYPE team_four IS VARRAY(4) OF VARCHAR2(15);
  team team_four := team_four();
BEGIN
--if we do not want to initialize like
team := team_four('John','Mary','Al','Ju');
EXTEND will help to initialize those memory
with NULL values
  team.EXTEND(4); --will occupy 4 elements
  team(3) := 'Pierre';
  team(4) := 'Yvonne';
  FOR i IN 1..team.LIMIT LOOP
    DBMS_OUTPUT.PUT_LINE(i || team(i));
  END LOOP;
END;
```

EXTEND Procedure without argument

```
DECLARE
  TYPE team_four IS VARRAY(4) OF VARCHAR2(15);
  team team_four := team_four();--have to
initialize without any values though. Cannot
keep it as - team team_four;
BEGIN
  FOR i IN 1..team.LIMIT LOOP
    team.EXTEND;--Only one element will be
occupied with NULL. If we try to assign next
element, we will get error; because the memory
for the next element is not initialized.
    team(i) := 'Pierre' || i;
    DBMS_OUTPUT.PUT_LINE(i || team(i));
  END LOOP;
END;
```

Control Statements

Simple CASE

```
SELECT product_id, company_name,
  (CASE product_id
    WHEN 'p1' THEN 'Cameras'
    WHEN 'p2' THEN 'Mobiles'
    ELSE 'Not available'
  END) AS product
FROM product;
```

Searched CASE

```
SELECT product_id, company_name,
  (CASE
    WHEN product_id = 'p1' THEN 'Cameras'
    WHEN product_id = 'p2' THEN 'Mobiles'
    WHEN product_id = 'p3'
        AND company_name = 'Samsung' THEN 'TV'
    ELSE 'Not available'
  END) AS product
FROM product;
```

IF statement

```
IF condition THEN
        --Statement1;
ELSIF condition THEN
        --Statement2;
ELSE
        --Statement3;
END IF;
```

Simple Loop

```
LOOP
  --statement1;
  EXIT WHEN condition;
END LOOP;
----
LOOP
  --statement1;
  IF condition THEN
    EXIT;
  END IF;
END LOOP;
```

Cursor For Loop

```
FOR loop_counter IN cursor_name LOOP
  --statement1;
END LOOP;
```

Numeric For Loop

```
FOR loop_counter IN [REVERSE]
lower_limit .. Upper_limit
LOOP
  --statement1;
END LOOP;
```

GOTO

A GOTO statement in PL/SQL programming language provides an unconditional jump from the GOTO to a labeled statement in the same subprogram.

```
DECLARE
   a NUMBER(2) := 10;
BEGIN
   <<LOOPSTART>>
   WHILE a < 20 LOOP
   DBMS_OUTPUT.PUT_LINE (a);
      a := a + 1;
      IF a = 15 THEN
         a := a + 1;
         GOTO LOOPSTART;
      END IF;
   END LOOP;
END;
```

RETURN

The RETURN statement immediately completes the execution of a subprogram and returns control to the caller. In procedures, a RETURN statement cannot contain an expression. The statement just returns control to the caller before the normal end of the procedure is reached. Do not confuse the RETURN statement with the RETURN clause in a function spec, which specifies the datatype of the return value.

While Loop

- Best usable when number of iterations to be performed are unknown
- Executes till the condition become FALSE

```
WHILE condition
LOOP
  --statement1;
END LOOP;
```

EXIT

When the EXIT statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
If you are using nested loops (i.e., one loop inside another loop), the EXIT statement will stop the execution of the innermost loop and start executing the next line of code after the block.

EXIT WHEN

```
EXIT WHEN condition;
```

The EXIT WHEN statement allows the condition in the WHEN clause to be evaluated. If the condition is true, the loop completes and control passes to the statement immediately after the END LOOP.

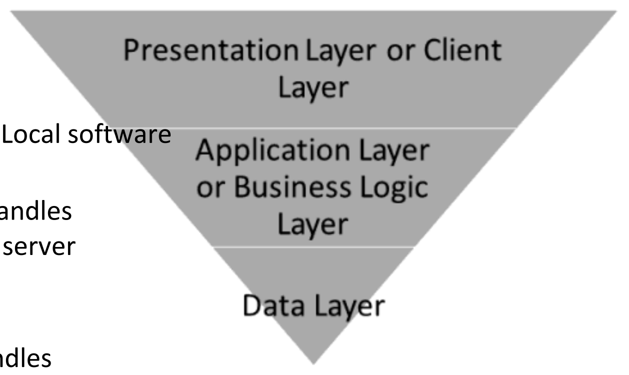
CONTINUE

The CONTINUE statement causes the loop to skip the remainder of its body and it forces the next iteration of the loop to take place.

DBMS vs. RDBMS

DBMS	RDBMS
Stores data as a file	Data is stored in the form of tables
Stores data in either navigational or hierarchical form	Uses a tabular structure where the headers are the column names, and the rows contain corresponding values
Supports single user only	Supports multiple users
Does not support Normalization	Can be Normalized
Data redundancy is common in this model	Keys and indexes do not allow Data redundancy
No relationship between data	Data is stored in the form of tables which are related to each other with the help of foreign keys
Examples of DBMS are a file system, XML, Windows Registry, etc.	Example of RDBMS is MySQL, Oracle, SQL Server, etc.

Software Architecture


Presentation Layer or Client Layer
Application Layer or Business Logic Layer
Data Layer

One-Tier Architecture

Contains all the layers in a single software package. Ex: Local software

Two-Tier Architecture

Also known as client-server application. Client system handles both presentation layer and application layer, database server handles data layer. Ex: Email client software

Three-Tier Architecture

Also known as web based application. Client system handles presentation layer, application server handles application layer and database server handles data layer. Ex: Gmail

Data dictionary

A read-only collection of metadata (data about data) about database tables and views containing reference information about the database, its structures, and its users.

Instance vs. Database

Instance is the combination of the system global area (SGA) and background processes (PMON,SMON etc.). An instance is associated with one and only one database. In an Oracle Real Application Clusters configuration, multiple instances can access a single database simultaneously. A **database** is a set of physical files on disk, that store data. An instance can exist without a database and a database can exist without an instance. To switch instance in a database system, one needs to change the value of **ORACLE_SID**. `SET ORACLE_SID=orcl`
Database Configuration Assistant(DBCA) can be used to create single instance databases, or it can be used to create or add instances in an Oracle Real Application Clusters environment.

Increasing datafile size

```
ALTER DATABASE DATAFILE '/u01/app/Test_data_01.dbf' RESIZE 2G;
```

Find the Actual size of a Database

```
SELECT SUM (bytes) / 1024 / 1024 / 1024 AS GB FROM dba_data_files;
```

Find the size occupied by Data in a Database or Database usage details

```
SELECT SUM (bytes) / 1024 / 1024 / 1024 AS GB FROM dba_segments;
```

Find the size of the SCHEMA/USER

```
SELECT SUM (bytes / 1024 / 1024) " MB size"
  FROM dba_segments
 WHERE owner = '&owner';
```

CPU usage of the USER

```
SELECT ss.username, se.SID, VALUE / 100 cpu_usage_seconds
    FROM v$session ss, v$sesstat se, v$statname sn
   WHERE     se.STATISTIC# = sn.STATISTIC#
         AND NAME LIKE '%CPU used by this session%'
         AND se.SID = ss.SID
         AND ss.status = 'ACTIVE'
         AND ss.username IS NOT NULL
ORDER BY VALUE DESC;
```

Last SQL fired by the User on Database

```
SELECT S.USERNAME || '(' || s.sid || ')-' || s.osuser UNAME,
         s.program || '-' || s.terminal || '(' || s.machine || ')' PROG,
         s.sid || '/' || s.serial# sid,
         s.status "Status",
         p.spid,
         sql_text sqltext
    FROM v$sqltext_with_newlines t, V$SESSION s, v$process p
   WHERE     t.address = s.sql_address
         AND p.addr = s.paddr(+)
         AND t.hash_value = s.sql_hash_value
ORDER BY s.sid, t.piece;
```

Show Query progress in database

```
SELECT a.sid,
         a.serial#,
         b.username,
         opname OPERATION,
         target OBJECT,
         TRUNC (elapsed_seconds, 5) "ET (s)",
         TO_CHAR (start_time, 'HH24:MI:SS') start_time,
         ROUND ( (sofar / totalwork) * 100, 2) "COMPLETE (%)"
    FROM v$session_longops a, v$session b
   WHERE     a.sid = b.sid
         AND b.username NOT IN ('SYS', 'SYSTEM')
         AND totalwork > 0
ORDER BY elapsed_seconds;
```

Get current session id, process id, client process id?

```
SELECT b.sid,
       b.serial#,
       a.spid processid,
       b.process clientpid
  FROM v$process a, v$session b
 WHERE a.addr = b.paddr AND b.audsid = USERENV ('sessionid');
```

V$SESSION.SID AND V$SESSION.SERIAL# is database process id
V$PROCESS.SPID is shadow process id on this database server
V$SESSION.PROCESS is client PROCESS ID

Last SQL Fired from particular Schema or Table:

```
SELECT CREATED, TIMESTAMP, last_ddl_time
  FROM all_objects
 WHERE     OWNER = 'MYSCHEMA'
       AND OBJECT_TYPE = 'TABLE'
       AND OBJECT_NAME = 'EMPLOYEE_TABLE';
```

Find Top 10 SQL by reads per execution

```
SELECT *
  FROM (  SELECT ROWNUM,
                 SUBSTR (a.sql_text, 1, 200) sql_text,
                 TRUNC (
                    a.disk_reads / DECODE (a.executions, 0, 1,
a.executions))
                    reads_per_execution,
                 a.buffer_gets,
                 a.disk_reads,
                 a.executions,
                 a.sorts,
                 a.address
            FROM v$sqlarea a
        ORDER BY 3 DESC)
 WHERE ROWNUM < 10;
```

Oracle SQL query over the view that shows actual Oracle connections.

```
SELECT osuser,
         username,
         machine,
         program
    FROM v$session
ORDER BY osuser;
```

Oracle SQL query that show the opened connections group by the program that opens the connection.

```
SELECT program application, COUNT (program) Numero_Sesiones
    FROM v$session
GROUP BY program
ORDER BY Numero_Sesiones DESC;
```

Oracle SQL query that shows Oracle users connected and the sessions number for user

```
SELECT username Usuario_Oracle, COUNT (username) Numero_Sesiones
    FROM v$session
GROUP BY username
ORDER BY Numero_Sesiones DESC;
```

Environment Variables

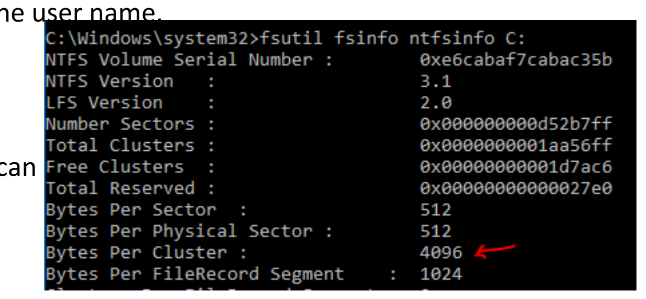
On the Windows platform, Oracle Universal Installer (OUI) automatically assigns values to ORACLE_BASE, ORACLE_HOME and ORACLE_SID in the Windows registry.
ORACLE_HOME - C:\oraclexe\app\oracle\product\11.2.0\server
ORACLE_BASE - C:\oraclexe\app\oracle

Schema

A schema is the set of database objects (tables, indexes, views, etc) that belong to a user. A database user owns a database schema, which has the same name as the user name.

Logical Storage Structures

Data blocks



```
C:\Windows\system32>fsutil fsinfo ntfsinfo C:
NTFS Volume Serial Number :       0xe6cabaf7cabac35b
NTFS Version   :                  3.1
LFS Version    :                  2.0
Number Sectors :                  0x000000000d52b7ff
Total Clusters :                  0x0000000001aa56ff
Free Clusters  :                  0x00000000001d7ac6
Total Reserved :                  0x00000000000027e0
Bytes Per Sector  :               512
Bytes Per Physical Sector :       512
Bytes Per Cluster :               4096
Bytes Per FileRecord Segment    : 1024
```

Data blocks are the smallest units of storage that oracle can use or allocate. One logical data block corresponds to a specific number of bytes of physical disk space. Each operating system has what is called a block size. Oracle requests data in multiples of Oracle blocks, not operating system blocks. Therefore, you should set the Oracle block size to a multiple of the operating system block size to avoid unnecessary I/O. Usually 8 kb

Extents

Extents are the logical unit of database which is made of contiguous multiple numbers of the oracle data blocks. Default is 1 MB

Segments

A segment is a set of extents which has been allocated for a specific data structure and all of which are stored in the same tablespace. For example, each table's data is stored in its own data segment, while each index's data is stored in its own index segment. If the table or index is partitioned, each partition is stored in its own segment. Whenever the existing space in a segment is completely used or full, oracle allocates a new extent for the segment. So the extents of a segment may or may not be contiguous on disk. The segments also can span datafiles, but the individual extents cannot.

Tablespaces

Tablespaces are the bridge between physical and logical components of the Oracle database. A tablespace is made up of one or more database datafiles. The datafiles are created automatically when the tablespace is defined. When you create a tablespace, you define the initial size of the associated datafile.

Types of Tablespaces

SYSTEM and **SYSAUX** tablespaces are always created when the database is created. The SYSTEM tablespace always contains the data dictionary tables for the entire database. The SYSAUX tablespace is an auxiliary tablespace to the SYSTEM tablespace.

Permanent Tablespace

Contains persistent schema object. Data persist beyond the duration of a session or transaction. Objects in permanent tablespaces are stored in data files.

Temporary Tablespace

Temporary tablespaces are used for special operations, particularly for sorting data results on disk and for hash joins in SQL. For SQL with millions of rows returned, the sort operation is too large for the RAM area and must occur on disk. The temporary tablespace is where this takes place.

Undo Tablespace

- Oracle Database keeps records of actions of transactions, before they are committed. These information are used to rollback or undo the changes to the database. These records are called rollback or undo records.
- When the instance starts up, the database automatically selects for use the first available undo tablespace. If there is no undo tablespace available, the instance starts, but uses the SYSTEM rollback segment for undo. This is not recommended, and an alert message is written to the alert log file to warn that the system is running without an undo tablespace.
- Committed undo information normally is lost when its undo space is overwritten by a newer transaction.
- The default value for the UNDO_RETENTION parameter is 900 seconds. The system retains undo for at least the time specified in this parameter.
- You can set the UNDO_RETENTION in the parameter file: `UNDO_RETENTION = 1800`
  `SQL> ALTER SYSTEM SET UNDO_RETENTION = 2400;`
- You can set `RETENTION` clause to either `GUARANTEE` or `NOGUARANTEE`. It specifies the database should preserve the unexpired undo data. This setting is useful if you need to issue an **Oracle Flashback Query** to correct a problem with the data. `RETENTION NOGUARANTEE` returns the undo behavior to normal. Space occupied by unexpired undo data in undo segments can be consumed if necessary by ongoing transactions. This is the default.
- You can create more than one undo tablespace but only one of them can be active at any given time.

Small-file Tablespace

Default type of tablespace in Oracle database. Can have multiple data files. Maximum of 1022 data files are allowed.

Big-file Tablespace

Suited for storing large amount of data. Allows maximum 1 data file

```sql
SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES
WHERE PROPERTY_NAME = 'DEFAULT_PERMANENT_TABLESPACE';
----
SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES
WHERE PROPERTY_NAME = 'DEFAULT_TEMP_TABLESPACE';
----
ALTER DATABASE DEFAULT TABLESPACE tbs_perm_01;
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tbs_temp_01;
----
CREATE TABLE tbl_tblspace (value1 NUMBER(2))
TABLESPACE SYSTEM;
```

```sql
CREATE SMALLFILE TEMPORARY TABLESPACE tblspce
    TEMPFILE 'C:\oraclexe\app\oracle\oradata\XE\dtlfle1.dbf'SIZE 100M,
            'C:\oraclexe\app\oracle\oradata\XE\dtlfle2.dbf' SIZE 100M
            AUTOEXTEND ON NEXT 500M
            MAXSIZE UNLIMITED
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 25M;
----
SELECT file_name, tablespace_name FROM dba_data_files WHERE tablespace_name
='tblspce'; --To check number od datafile associated with tablespace
DROP TABLESPACE tblspce INCLUDING CONTENTS AND DATAFILES;
ALTER DATABASE DATAFILE 'C:\oraclexe\app\oracle\oradata\XE\dtlfle1.dbf' OFFLINE DROP;
ALTER DATABASE DATAFILE 'C:\oraclexe\app\oracle\oradata\XE\dtlfle1.dbf' RESIZE 8M;
```

```sql
CREATE SMALLFILE TABLESPACE tblspce -- TEMPORARY TABLESPACE for temporary tablespace;
UNDO TABLESPACE for undo tablespace
    DATAFILE 'C:\app\oradata\dtlfle1.dbf' --TEMPFILE for temporary tablespace. DATAFILE
for permenant tablespace and undo tablespace
                                               SIZE 100M  -- M for MB; G for GB; T for TB
             'C:\app\oradata\dtlfle2.dbf' SIZE 100M
             AUTOEXTEND ON NEXT 500M --Extend it by 500 MB after the file gets filled
             MAXSIZE UNLIMITED
    RETENTION NOGUARANTEE --Only for undo tablespace
    LOGGING --LOGGING = Create logs for creation of tables, indexes, inserts etc.. Not
available for undo and temporary tablespace
             --NO LOGGING = Does not create logs
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 25M--UNIFORM = Allocate extents in the
tablespace with same size. Should be less than datafile size. Default for TEMPORARY
tablespace; Not available for undo tablespace.
    --AUTOALLOCATE = System will decide the size of extent allocation
    SEGMENT SPACE MANAGEMENT AUTO;--MANUAL/AUTO; Not available for temporary tablespace
and undo tablespace
----
CREATE SMALLFILE TABLESPACE tblspce
    DATAFILE 'C:\oraclexe\app\oracle\oradata\XE\dtlfle1.dbf'SIZE 100M,
             'C:\oraclexe\app\oracle\oradata\XE\dtlfle2.dbf' SIZE 200M
             AUTOEXTEND ON NEXT 500M
             MAXSIZE UNLIMITED
    LOGGING
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 25M
    SEGMENT SPACE MANAGEMENT AUTO;
 ----
CREATE SMALLFILE UNDO TABLESPACE tblspce
    DATAFILE 'C:\oraclexe\app\oracle\oradata\XE\dtlfle1.dbf'SIZE 100M,
             'C:\oraclexe\app\oracle\oradata\XE\dtlfle2.dbf' SIZE 100M
             AUTOEXTEND ON NEXT 500M
             MAXSIZE UNLIMITED
             RETENTION GUARANTEE;
```

Physical Database Limits

Item	Type of Limit	Limit Value
Database Block Size	Minimum	2048 bytes; must be a multiple of operating system physical block size
	Maximum	Operating system dependent; never more than 32 KB
Database Blocks	Minimum in initial extent of a segment.	2 blocks
	Maximum per datafile	Platform dependent; typically 222 - 1 blocks
Controlfiles	Number of control files	1 minimum; 2 or more (on separate devices) strongly recommended
	Size of a control file	Dependent on operating system and database creation options; maximum of 20,000 x (database block size)
Database files	Maximum per tablespace	Operating system dependent; usually 1022
	Maximum per database	65533 May be less on some operating systems Limited also by size of database blocks and by the DB_FILES initialization parameter for a particular instance
Database extents	Maximum per dictionary managed tablespace	4 GB * physical block size (with K/M modifier); 4 GB (without K/M modifier)
	Maximum per locally managed (uniform) tablespace	2 GB * physical block size (with K/M modifier); 2 GB (without K/M modifier)
Database file size	Maximum	Operating system dependent. Limited by maximum operating system file size; typically 222 or 4 MB blocks
MAXEXTENTS	Default value	Derived from tablespace default storage or DB_BLOCK_SIZE initialization parameter
	Maximum	Unlimited
Redo Log Files	Maximum number of logfiles	Limited by value of MAXLOGFILES parameter in the CREATE DATABASE statement Control file can be resized to allow more entries; ultimately an operating system limit
	Maximum number of logfiles per group	Unlimited
Redo Log File Size	Minimum size	50 KB
	Maximum size	Operating system limit; typically 2 GB
Tablespaces	Maximum number per database	64 K Number of tablespaces cannot exceed the number of database files because each tablespace must include at least one file
Bigfile Tablespaces	Number of blocks	A bigfile tablespace contains only one datafile or tempfile, which can contain up to approximately 4 billion (232) blocks. The maximum size of the single datafile or tempfile is 128 terabytes (TB) for a tablespace with 32K blocks and 32TB for a tablespace with 8K blocks.
Smallfile (traditional) Tablespaces	Number of blocks	A smallfile tablespace is a traditional Oracle tablespace, which can contain 1022 datafiles or tempfiles, each of which can contain up to approximately 4 million (222) blocks.
External Tables file	Maximum size	Dependent on the operating system. An external table can be composed of multiple files.

Physical Storage Structures

Data files

Every Oracle database has one or more physical data files, which contain all the database data. The data of logical database structures, such as tables and indexes, is physically stored in the data files.

Control files

Control file is a binary file which contains metadata specifying the physical structure of the database, including the database name and the names and locations of the database files.

Online redo log files

It is a set of two(minimum) or more log files. Oracle will write every change made in the database into the first log file, and when the first log file is full, Oracle will switch to the second log file and write. We can have multiple group of redo log files to keep mirrored copies.

Archived redo log files

An Oracle database can run in one of two modes. By default, the database is created in NOARCHIVELOG mode. In this mode, it will overwrite the redo log file once they are filled. In ARCHIVELOG mode, database archive all redo log files once they are filled instead of overwriting them.

```sql
SQL> ARCHIVE LOG LIST; -- To check whether Archive Log Mode is enabled or not
SQL> SELECT log_mode FROM v$database
SQL> SHUTDOWN IMMEDIATE; --To enable Archive Log Mode
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER DATABASE OPEN;
```

? Get number of objects per owner

```sql
SELECT owner, COUNT (owner) number_of_objects
    FROM dba_objects
GROUP BY owner
ORDER BY number_of_objects DESC;
```

? Convert number to words

```sql
SELECT TO_CHAR (TO_DATE (1526, 'j'), 'JSP') FROM DUAL;
--j= julian calendar format
--jsp = convert into words lower character
--JSP = upper
```

Output:
one thousand five hundred twenty-six

? Find string in package source code

```sql
SELECT *
  FROM dba_source
 WHERE UPPER (text) LIKE '%FOO_SOMETHING%'
AND owner = 'USER_NAME';
```

? Convert Comma Separated Values into Table

```sql
WITH csvs
     AS (SELECT 'AA,BB,CC,DD,EE,FF'
                   AS csvdata
           FROM DUAL)
    SELECT REGEXP_SUBSTR (csvs.csvdata, '[^,]+', 1, LEVEL) pivot_char
      FROM DUAL, csvs
CONNECT BY REGEXP_SUBSTR (csvs.csvdata,'[^,]+', 1, LEVEL) IS NOT NULL;
--The query can come quite handy when you have comma separated data string that you need
to convert into table so that you can use other SQL queries like IN or NOT IN
```

? Find the last record from a table

```sql
SELECT *
  FROM employees
 WHERE ROWID IN (SELECT MAX (ROWID) FROM employees);
```

(OR)

```sql
SELECT * FROM employees
MINUS
SELECT *
  FROM employees
 WHERE ROWNUM < (SELECT COUNT (*) FROM employees);
--Use this when your table does not have primary key or you cannot be sure if record
having max primary key is the latest one.
```

? Check if table contains any data

```sql
SELECT 1 FROM TABLE_NAME WHERE ROWNUM = 1;
```

Interview (2) Questions and Answers

? Row Data Multiplication in Oracle

```sql
WITH tbl
     AS (SELECT -2 num FROM DUAL
         UNION
         SELECT -3 num FROM DUAL
         UNION
         SELECT -4 num FROM DUAL),
     sign_val
     AS (SELECT CASE MOD (COUNT (*), 2) WHEN 0 THEN 1 ELSE -1 END val
           FROM tbl
          WHERE num < 0)
  SELECT EXP (SUM (LN (ABS (num)))) * val
    FROM tbl, sign_val
GROUP BY val;
```

? Generating Random Data In Oracle

```sql
SELECT LEVEL empl_id,
           MOD (ROWNUM, 50000) dept_id,
           TRUNC (DBMS_RANDOM.VALUE (1000, 500000), 2) salary,
           DECODE (ROUND (DBMS_RANDOM.VALUE (1, 2)),  1, 'M',  2, 'F') gender,
           TO_DATE (
                 ROUND (DBMS_RANDOM.VALUE (1, 28))
              || '-'
              || ROUND (DBMS_RANDOM.VALUE (1, 12))
              || '-'
              || ROUND (DBMS_RANDOM.VALUE (1900, 2010)),
              'DD-MM-YYYY')
              dob,
           DBMS_RANDOM.STRING ('x', DBMS_RANDOM.VALUE (20, 50)) address
      FROM DUAL
CONNECT BY LEVEL < 10000;
```

? Random number generator in Oracle

```sql
SELECT ROUND (DBMS_RANDOM.VALUE () * 100) + 1 AS random_num FROM DUAL;
--Change the multiplier to number that you want to set limit for.
```

? To fetch ALTERNATE records from a table. (EVEN NUMBERED)

```sql
SELECT * FROM emp WHERE ROWID IN (SELECT DECODE(MOD(ROWNUM,2),0,ROWID, NULL) FROM emp);
```

? To select ALTERNATE records from a table. (ODD NUMBERED)

```sql
SELECT * FROM emp WHERE ROWID IN (SELECT DECODE(MOD(ROWNUM,2),0,NULL ,ROWID) FROM emp);
```

? Find the 3rd MAX salary in the emp table.

```sql
SELECT DISTINCT sal FROM emp e1 WHERE 3 = (SELECT COUNT(DISTINCT sal) FROM emp e2
WHERE e1.sal <= e2.sal);
```

? Find the 3rd MIN salary in the emp table.

```sql
SELECT DISTINCT sal FROM emp e1 WHERE 3 = (SELECT COUNT(DISTINCT sal) FROM emp e2
WHERE e1.sal >= e2.sal);
```

? Select LAST n records from a table

```sql
SELECT * FROM emp MINUS SELECT * FROM emp WHERE ROWNUM <= (SELECT COUNT(*) - &n FROM
emp);
```

? List dept no., Dept name for all the departments in which there are no employees in the department.

```sql
SELECT * FROM dept WHERE deptno NOT IN (SELECT deptno FROM EMP);
```

(OR)

```sql
SELECT * FROM dept a WHERE NOT EXISTS (SELECT * FROM emp b WHERE a.deptno = b.deptno);
```

(OR)

```sql
SELECT empno,ename,b.deptno,dname FROM emp a, dept b WHERE a.deptno(+) = b.deptno AND
empno IS NULL;
```

Parameter files
To start a database instance, Oracle Database must read either a server parameter file(**SPFILE** - %ORACLE_HOME%\dbs\spfile%ORACLE_SID%.ora), which is recommended, or a text initialization parameter file (**PFILE** - %ORACLE_HOME%\database\init%ORACLE_SID%.ora). These files contain a list of configuration parameters like SGA size, name of database, name and location of database control files for that instance and database. SPFILE is binary file and only Oracle database can read or write into that file. You can modify the parameter's values with the `ALTER SYSTEM SET` command.
MAXDATAFILES specifies the maximum number of datafiles that can be open in the database.
MAXINSTANCES specifies that only one instance can have this database mounted and open.
```
      SQL> STARTUP PFILE = 'C:\ora\pfile\init.ora'
      SQL> CREATE SPFILE FROM PFILE = 'C:\ora\pfile\init.ora'
      SQL> CREATE PFILE = 'C:\ora\pfile\init.ora' FROM SPFILE
      SQL> CREATE SPFILE FROM MEMORY
```
Password file
Stores passwords for users with administrative privileges.
Location: %ORACLE_HOME%\database\PWD%ORACLE_SID%.ora
Networking files
These files are used to configure the different network components of the Oracle database. These include files such as tnsnames.ora and listener.ora. The "listener.ora" file contains server side network configuration parameters. The "tnsnames.ora" file contains client side network configuration parameters.
Location: %ORACLE_HOME%\network\ADMIN
IO Error : The Network Adapter could not establish connection
```
     CMD> lsnrctl status -- to check the status of listener
     CMD> lsnrctl start -- to start listener
```
Trace file (.trc)
Trace File are trace (or dump) file that Oracle Database creates to help you diagnose and resolve operating problems. Each server and background process writes to a trace file. When a process detects an internal error, it writes information about the error to its trace file.
Alert log
The alert log file is a chronological log of messages and errors written out by an Oracle Database. Typical messages found in this file is: database startup, shutdown, log switches, space errors, etc. This file should constantly be monitored to detect unexpected messages and corruptions.
Location:%ORACLE_BASE%\diag\rdbms\%ORACLE_SID%\%ORACLE_SID%\trace
```
      SQL> SHOW PARAMETER background
```
System Change Number (SCN)
SCN (System Change Number) is a primary mechanism to maintain data consistency in Oracle database. Every time a user commits a transaction, Oracle records a new SCN. The SCN_TO_TIMESTAMP function can be used to map between the SCN to a point in time. ORA_ROWSCN pseudo column is useful for determining approximately when a row was last updated.
```
      SQL> SELECT CURRENT_SCN FROM V$database;
      SQL> SELECT SCN_TO_TIMESTAMP(ORA_ROWSCN) FROM employees;
      SQL> SELECT ORA_ROWSCN, last_name FROM employees WHERE employee_id = 188;
```

Memory Architecture
System global area (SGA)
System Global Area forms the part of the system memory shared by all the processes belonging to a single Oracle database instance. Each database instance has its own SGA. To start an instance: `SQL> STARTUP`
SGA contain:
- Database buffer cache: Any data coming in or going out of the database will pass through the buffer cache. When Oracle receives a request to retrieve data, it will first check if the data is already in the buffer. This practice allows server to avoid unnecessary I/O.
- Shared pool: The shared pool is like a buffer for SQL statements. It contains Oracle's library cache, which is responsible for collecting, parsing, interpreting, and executing all of the SQL statements. In library cache we have Shared SQL Areas and Private SQL Area (Shared Server Mode Only). Oracle represents each SQL statement it executes with a shared SQL area and a private SQL area. Oracle recognizes when two users are executing the same SQL statement and reuses the same shared part for those users. However, each user must have a separate copy of the statement's private SQL area. A shared SQL area is a memory area that contains the parse tree and execution plan for a single SQL statement. A private SQL area is a memory area that contains data such as bind information and runtime buffers. Each session that issues a SQL statement has a private SQL area. Many private SQL areas can be associated with the same shared SQL area.
- Redo log buffer: The server processes generate redo data into the log buffer as they make changes to the data blocks in the buffer. LGWR subsequently writes entries from the redo log buffer to the online redo log. LGWR initiates a flush of this area in one of the following scenarios: Every three seconds, Whenever someone commits, When LGWR is asked to switch log files, When the redo buffer gets one-third full or contains 1MB of cached redo log data

Program global area (PGA)
A PGA is a non-shared memory region which is private to each server and background process; that contains a private SQL area (Dedicated Server Mode Only) and a session memory area (in UGA). There is one PGA for each process. Do not confuse a private SQL area, which is in the PGA, with the shared SQL area, which stores execution plans in the SGA. Multiple private SQL areas in the same or different sessions can point to a single execution plan in the SGA. Session memory is the memory allocated to hold a session's variables (logon information) and other information related to the session.
User global area (UGA)
The UGA is memory area associated with a user session which allocates memory for session variables, such as logon information, session state, and other information required by a database session. For dedicated sessions, the UGA is a part of PGA and for shared sessions, the UGA is inside the SGA.
Dedicated vs. Shared Server Process
In Dedicated server process, each client process connects to a dedicated server process. If you have 100 dedicated server connections, there will be 100 processes (PGA) executing on their behalf.
In Shared server process, the database uses a pool of shared server processes for multiple sessions. A client process communicates with a dispatcher, which is a process that enables many clients to connect to the same database instance without the need for a dedicated server process for each client.
Background Processes
SMON: System Monitor recovers after instance failure and monitors temporary segments and extents.
PMON: Process Monitor recovers failed process resources.
DBWR: Database Writer or Dirty Buffer Writer process is responsible for writing dirty buffers from the database buffer cache to the database data files. Generally, DBWR only writes blocks back to the data files on commit, or when the cache is full and space has to be made for more blocks.
LGWR: Log Writer process is responsible for writing the log buffers out to the redo logs.
ARCn: The optional Archive process writes filled redo logs to the archive log location(s).

Oracle Versions Milestones
Version 6
PL/SQL language introduced
Version 7
PL/SQL stored program units introduced
Version 8
Support for table partitioning
Version 8i
'i' stands for internet computing. Provided native support for internet protocols and server-side support for Java enabling the database to be deployed in a multitier environment.
Version 9i
Introduced **Oracle RAC** enabling multiple instances to access a single database simultaneously.
The benefits of Real Application Clusters are:
- Ability to spread CPU load across multiple servers
- Continuous Availability / High Availability (HA)
  - Protection from single instance failures
  - Protection from single server failures
- Scalability
Version10g
- 'g' stands for grid computing (A computing architecture that coordinates large numbers of servers and storage to act as a single large computer).
- This release enabled organizations to virtualize computing resources by building a grid infrastructure based on low-cost commodity servers. A key goal was to make the database self-managing and self-tuning.
- Oracle Automatic Storage Management (**Oracle ASM**) helped achieve this goal by virtualizing and simplifying database storage management.
Version 12c
- 'c' stands for cloud computing.
- Introduced Multitenant architecture. The multitenant architecture enables an Oracle database to function as a multitenant container database (**CDB**). A CDB includes zero, one, or many pluggable databases (**PDB**s).
- A PDB is a portable collection of schemas, schema objects, and non-schema objects. You can unplug a PDB from a CDB and plug it into a different CDB.
- All Oracle databases before Oracle Database 12c were non-CDBs.

Connect using CMD
```
CMD> SET ORACLE_SID=orcl
CMD> sqlplus hr/hr
SQL> SHOW user;
----
CMD> sqlplus /nolog --not connected to any user
----
CMD> sqlplus sys/oracle as sysdba@192.168.11.1:1521/orcl --login as DBA to different
machine
----
CMD> sqlplus / as sysdba --login as DBA using system credentials
----
SQL> SELECT name from v$database; --Shows the database which is currently connected
```
Save to file
```
SPOOL filename
SET TIME ON TIMING ON -- to show the time
SPOOL OFF
```
DROP a user
```
DROP USER username;
----
DROP USER username CASCADE; -- delete all the objects created by user
```
DROP User if the user is connected to database
```
ALTER SYSTEM ENABLE RESTRICTED SESSION; --By default, database is in OPEN STARTUP
mode
--Now we changed from OPEN to RESTRICTED
-- In RESTRICTED mode, the users who are already connected can continue their work
without any interruptions unless they are somehow disconnected
--If they are disconnected, they need RESTRICTED privilage to connect to database
again in RESTRICTED mode
-- No new user can connect to the database
SELECT sid, serial# FROM v$session WHERE username ='username';--Get the serial
number of user which we want to drop.
-- We got SID = 67 and SERIAL # = 638
ALTER SYSTEM KILL SESSION '67,638';-- disconnect the user from database
DROP USER username CASCADE;-- drop the user\
ALTER SYSTEM DISABLE RESTRICTED SESSION; -- back to OPEN mode
```
UNLOCK a User
```
ALTER USER username IDENTIFIED BY password ACCOUNT UNLOCK;
```
Create an EXTERNAL User(Operating system user)
```
CMD> hostname -- to get hostname of the system
CMD> echo %username% -- get username
SQL> SHOW PARAMETER OS; -- take the value of 'os_authent_prefix'
--Name of external user should be prefixed with value of os_authent_prefix +
hostname + \ + username
SQL> CREATE USER "OPS$HOSTNAME\USERNAME" IDENTIFIED EXTERNALLY;
SQL> GRANT CREATE SESSION TO "OPS$HOSTNAME\USERNAME";
CMD> sqlplus / -- to login to sqlplus as external user
```

Synonyms and Alias
Synonyms are a type of database objects. They refer to other objects in the database. A public synonym and a private synonym can exist with the same name for the same table.
`CREATE [PUBLIC] SYNONYM synonym_name FOR object_name;`
Alias is just another name for view, table or column inside a query. It does not create an object.
`SELECT ename AS name FROM emp;`

How to get 3 Max salaries ?
```
SELECT DISTINCT sal FROM emp a WHERE 3 >= (SELECT COUNT(DISTINCT sal) FROM emp b
WHERE a.sal <= b.sal) ORDER BY a.sal DESC;
```
How to get 3 Min salaries ?
```
SELECT DISTINCT sal FROM emp a WHERE 3 >= (SELECT COUNT(DISTINCT sal) FROM emp b
WHERE a.sal >= b.sal);
```
How to get nth max salaries ?
```
SELECT DISTINCT hiredate FROM emp a WHERE &n = (SELECT COUNT(DISTINCT sal) FROM emp b
WHERE a.sal >= b.sal);
```
Select DISTINCT RECORDS from emp table.
```
SELECT * FROM emp a WHERE ROWID = (SELECT MAX(ROWID) FROM emp b WHERE a.empno=b.empno);
```
How to delete duplicate rows in a table?
```
DELETE FROM emp a WHERE ROWID != (SELECT MAX(ROWID) FROM emp b WHERE a.empno=b.empno);
```
Count of number of employees in department wise.
```
SELECT COUNT(empno), b.deptno, dname FROM emp a, dept b
WHERE a.deptno(+)=b.deptno GROUP BY b.deptno,dname;
```
Count the total salary deptno wise where more than 2 employees exist.
```
SELECT  deptno, sum(sal) As totalsal
FROM emp
GROUP BY deptno
HAVING COUNT(empno) > 2;
```
Difference between varchar and varchar2 data types?
VARCHAR is used to support distinction between NULL and empty string, as ANSI standard prescribes. VARCHAR occupies space for the NULL values.VARCHAR2 does not distinguish between a NULL and empty string. VARCHAR2 will not occupy space for the NULL values. VARCHAR2 is an oracle standard

In which language Oracle has been developed?
Oracle has been developed using C Language.

What is RAW datatype?
RAW datatype is used to store values in binary data format. The maximum size for a raw in a table in 32767 bytes.

What is USING Clause and give example?
The USING clause is used to specify with the column to test for equality when two tables are joined.

What is WITH CHECK OPTION?
WITH CHECK OPTION is used for a view to prevent the invisible rows from being updated.
```
CREATE
    VIEW cars AS SELECT
    car_id, car_name, brand_id
    FROM cars
    WHERE brand_id = 3 WITH CHECK OPTION
```
What is a sub query and what are the different types of subqueries?
Sub Query is also called as Nested Query or Inner Query which is used to get data from multiple tables. A sub query is added in the where clause of the main query.

There are two different types of subqueries:
Correlated sub query
A Correlated sub query cannot be as independent query but can reference column in a table listed in the from list of the outer query.

Non-Correlated subquery
This can be evaluated as if it were an independent query. Results of the sub query are submitted to the main query or parent query.

Select FIRST n records from a table.
```
SELECT * FROM emp WHERE ROWNUM <= &n;
```
What is key preserved table?
A table is set to be key preserved table if every key of the table can also be the key of the result of the join. It guarantees to return only one copy of each row from the base table. A table may be key preserved in one join view and may not be key preserved in another view.
Ex: `SELECT e.*,d.* FROM emp,dept WHERE emp.dno = dept.dno;`
In this join, empno is the key in its own table and also in the result of this join. So key is preserved.
In this, empno is unique but dno is not unique. So emp table is key preserved but dept is not.

How can we delete duplicate rows in a table?
Duplicate rows in the table can be deleted by using ROWID.

What is the fastest query method to fetch data from the table?
Row can be fetched from table by using ROWID. Using ROW ID is the fastest query method to fetch data from the table.

What is the maximum number of triggers that can be applied to a single table?
12 is the maximum number of triggers that can be applied to a single table.

How can we view last record added to a table?
```
SELECT * FROM (SELECT * FROM employees ORDER BY ROWNUM DESC) WHERE ROWNUM<2;
```
What is the data type of DUAL table?
The DUAL table is a one-column table present in oracle database. The table has a single VARCHAR2(1) column called DUMMY which has a value of 'X'.

What is a Literal?
A Literal is a string that can contain a character, a number, or a date that is included in the SELECT list and that is not a column name or a column alias. Date and character literals must be enclosed within single quotation marks (' '), number literals need not.

What is a DATA DICTIONARY?
DICTIONARY: Is a collection of tables created and maintained by the Oracle Server. It contains database information. Q.

What is a SET UNUSED option?
SET UNUSED option marks one or more columns as unused so that they can be dropped when the demand on system resources is lower. Unused columns are treated as if they were dropped, even though their column data remains in the table's rows. After a column has been marked as unused, you have no access to that column. A select * query will not retrieve data from unused columns. In addition, the names and types of columns marked unused will not be displayed during a DESCRIBE, and you can add to the table a new column with the same name as an unused column. The SET UNUSED information is stored in the USER_UNUSED_COL_TABS dictionary view.

SQL Commands

DDL (Data Definition Language)

Used to define the structure of the database.

`CREATE DROP ALTER TRUNCATE COMMENT PURGE RENAME`

DML (Data Manipulation Language)

Used to manage the data in the database

`INSERT UPDATE DELETE`

DCL (Data Control Language)

Used to control the access to database and various permissions.
It is also considered as a DDL command.

`GRANT REVOKE`

TCL (Transaction Control Language)

Used to manage the transactions made by the DML commands

`COMMIT ROLLBACK SAVEPOINT`

DQL (Data Query Language)

Used to retrieve data from the database

`SELECT`

SQL Constraints

```
NOT NULL
UNIQUE
PRIMARY KEY
FOREIGN KEY
CHECK
```

Data Dictionary (Constraints)

USER_CONSTRAINTS - brief
USER_CONS_COLUMNS - detailed

```
SAVEPOINT sp1;
ROLLBACK TO sp1;
```

```
SQL>SET FEED 1 LINES 120 PAGES 500
NUMWIDTH 5;
```

Tip:

Copy structure and data from a table:

```
CREATE TABLE t2_name AS SELECT * FROM t1_name; --only the NOT NULL constraint defined on
the specified columns would be passed to the new table.
```

Copy only structure of a table:

```
CREATE TABLE t2_name AS SELECT * FROM t1_name WHERE 1=2;
```

Create a table with selected column from another table

```
CREATE TABLE t2_name AS SELECT empno,ename,sal FROM emp WHERE 2 = 1 ;
```

Insert into table from another table

```
INSERT INTO temp_emp SELECT * FROM employees;
----
INSERT INTO temp_emp (employee_id,first_name) (SELECT employee_id,first_name FROM employees);
```

```
CREATE TABLE tbl_student(
  s_id NUMBER(2) CONSTRAINT pk_sid PRIMARY KEY,
  s_fname VARCHAR2(20) CONSTRAINT uk_sfname UNIQUE,
  s_lname VARCHAR2(20) CONSTRAINT nn_slname NOT NULL,
  s_join VARCHAR2(20) DEFAULT ('New Joinee'),
  s_gender CHAR(1) CONSTRAINT chk_sgender CHECK(s_gender IN('m','f')),
  s_marks NUMBER(3),
  CONSTRAINT fk_smarks
    FOREIGN KEY(s_marks) REFERENCES tbl_player(pid) --should be a primary key
);
----
ALTER TABLE DROP CONSTRAINT pk_prdmstr;  --drop constraint
----
ALTER TABLE product_master ADD CONSTRAINT pk_prdmstr PRIMARY KEY (product_id,product_name); --
Composite PK
----
ALTER TABLE books ADD CONSTRAINT fk_bk FOREIGN KEY (book_author_id) REFERENCES authors
(author_id)
[ON DELETE NO ACTION] --Default; Won't allow to delete parent row
[ON DELETE SET NULL] --Sets the foreign key row as NULL if parent row is deleted;
[ON DELETE CASCADE] --Deletes the foreign key row also if parent row is deleted
----
ALTER TABLE product_master DISABLE CONSTRAINT pk_prdmstr; --disable constraint
----
ALTER TABLE product_master ENABLE CONSTRAINT pk_prdmstr;  --enable constraint
----
ALTER TABLE tbl_player ADD (full_name VARCHAR2(20)); --add column
----
ALTER TABLE tbl_player MODIFY (full_name VARCHAR2(40));  --modify column
----
ALTER TABLE emp RENAME COLUMN gender TO sex; --rename column
----
ALTER TABLE tbl_player DROP COLUMN full_name; --drop column
----
ALTER TABLE test RENAME TO example; --rename table
RENAME TEMP_TEST TO TEMMP_TEST; --rename table
----
ALTER TABLE table_name READ ONLY; -- Read only for other users as well as owner. If you
want to restrict other users only, then grant SELECT privilege only
ALTER TABLE table_name READ WRITE;
```

WITH Clause

- The SQL WITH clause allows you to give a sub-query block a name (also called sub-query refactoring) (cannot use table name), which can be referenced in several places within the main SQL query.
- For debugging/development purposes it's sometimes useful to store results in Global Temporary Tables; however, that's not a best practice for production code.

```
WITH emp AS(
    SELECT * FROM employees WHERE department_id = 10)
SELECT * FROM emp;
----
WITH employee (id, firstname, lastname, hobbies) AS
 (SELECT 1, 'a', 'b', '1' FROM DUAL UNION
  SELECT 2, 'a', 'b', '2' FROM DUAL UNION
  SELECT 3, 'a', 'b', '3' FROM DUAL UNION
  SELECT 5, 'e', 'f', '2' FROM DUAL)
SELECT *
FROM employee;
```

Global Temporary Table

- Used to store session specific data. After session gets disconnected, these tables will be flushed.
- It is faster than normal table as it is temporary.

```
CREATE GLOBAL TEMPORARY TABLE table_name
(column_definition [,column_definition])
[ON COMMIT {DELETE | PRESERVE} ROWS]

CREATE GLOBAL TEMPORARY TABLE students
(stud_id NUMERIC(10) NOT NULL ,
name VARCHAR2(50) NOT NULL)
ON COMMIT PRESERVE ROWS;
--PRESERVE ROWS - table is bound to a session. After session disconnects, it will be deleted
--DELETE ROWS - table is bound to a transaction. After a transaction ends, it will be deleted.
Default.
```

Group by and Having

GROUP BY is used to group rows that have same values. Optionally it is used in conjunction with aggregate functions to generate report.
HAVING condition is used to restrict the rows affected by the GROUP BY clause. It is similar to WHERE clause.

Privileges

A privilege is a right to execute a particular type of SQL statement or to access another user's object. There are two categories of privileges:

System Privileges

- A system privilege is the right to perform a particular action
- Only users who have been granted a specific system privilege with the ADMIN OPTION or users with the system privileges such as GRANT ANY PRIVILEGE or GRANT ANY OBJECT PRIVILEGE can grant or revoke system privileges to other users
- Cannot grant system privileges and object privileges in single statement
- ADMIN OPTION is available while creating SYSTEM privileges only.

```
GRANT CREATE ANY TABLE TO username;
----
GRANT CREATE SESSION , CREATE VIEW , CREATE SEQUENCE TO username;
----
GRANT CREATE PROCEDURE TO username1, username2;
----
GRANT CREATE TRIGGER TO username WITH ADMIN OPTION;
```

Object Privileges

A object privilege is the right to perform a particular action on a specific schema object.

```
GRANT SELECT ON username.object_name TO other_username;
----
GRANT UPDATE , INSERT , INDEX , DELETE ON username.object_name TO other_username;
----
GRANT UPDATE(column_name) ON username.object_name TO other_username; -- for column
only
----
GRANT UPDATE ON username.object_name TO other_username WITH GRANT OPTION;
```

Syntax for tables

```
GRANT privilege-type ON [TABLE] { table-Name | view-Name } TO grantees
```

privilege-types

```
  ALL PRIVILEGES |  -- privilege type to grant all of the privileges to the
user or role for the specified table. You can also grant one or more table
privileges by specifying a privilege-list.
  privilege-list
```

privilege-list

```
 table-privilege {, table-privilege }*
```

table-privilege

```
  DELETE |
  INSERT |
  REFERENCES [column list] |  -- privilege type to grant permission to create
a foreign key reference to the specified table. If a column list is specified
with the REFERENCES privilege, the permission is valid on only the foreign
key reference to the specified columns.
  SELECT [column list] |  --  If a column list is specified with the SELECT
privilege, the permission is valid on only those columns. If no column list
is specified, then the privilege is valid on all of the columns in the table.
  TRIGGER |
  UPDATE [column list] -- If a column list is specified, the permission
applies only to the specified columns. To update a row using a statement that
includes a WHERE clause, you must have the SELECT privilege on the columns in
the row that you want to update.
```

column list

```
  ( column_name {, column_name}* )
```

grantees

```
{ user_name | roleName | PUBLIC } [, { user_name| roleName | PUBLIC } ] *

Ex: GRANT SELECT ON TABLE schema_name.table_name to PUBLIC
```

Syntax for routines

```
GRANT EXECUTE ON { FUNCTION | PROCEDURE } routine-designator TO grantees
```

routine-designator

```
    function-name | procedure-name

Ex: GRANT EXECUTE ON PROCEDURE procedure_name TO george;
Ex: GRANT EXECUTE ON package_name TO george; -- Package
```

Syntax for sequence generators

```
GRANT USAGE ON SEQUENCE [ schemaName. ] user_name TO grantees -- If a schemaName
is not provided, the current schema is the default schema.
Ex: GRANT USAGE ON SEQUENCE order_id TO sales_role;
```

Syntax for user-defined types

```
GRANT USAGE ON TYPE [ schemaName. ] user_name TO grantees
Ex: GRANT USAGE ON TYPE price TO finance_role;
```

Syntax for roles

```
GRANT roleName [ {, roleName }* ] TO grantees
```

? What is a difference between ON DELETE CASCADE and ON DELETE SET NULL?

ON DELETE CASCADE Indicates that when the row in the parent table is deleted, the dependent rows in the child table will also be deleted. ON DELETE SET NULL Coverts foreign key values to null when the parent value is removed. Without the ON DELETE CASCADE or the ON DELETE SET NULL options, the row in the parent table cannot be deleted if it is referenced in the child table.

? What is a Candidate Key?

The columns in a table that can act as a Primary Key are called Candidate Key.

? Write a PL/SQL Program which raise a user defined exception on thursday?

```
DECLARE
a EXCEPTION
BEGIN
    IF TO_CHAR(SYSDATE, 'DY')='THU'
    THEN
    RAISE a;
    END IF;
    EXCEPTION
    WHEN a THEN
    DBMS_OUTPUT.PUT_LINE('my exception raised on thursday');
END;
```

? What is Flashback Query?

- Flashback query are handled by Database Administrator only. Flashback queries along allows content of the table to be retrieved with reference to specific point of time by using as of clause.
- Flashback queries generally uses undo file that is flashback queries retrieve old data before committing the transaction oracle provide two method for flashback queries
  Method1: using timestamp
  Method2: using scn number

? What will be the result of the query below? Explain your answer and provide a version that behaves correctly.

```
SELECT CASE WHEN NULL = NULL THEN 'YUP' ELSE 'NOPE' END AS RESULT FROM DUAL;
```

This query will actually yield "Nope", seeming to imply that null is not equal to itself! The reason for this is that the proper way to compare a value to null in SQL is with the is operator, not with =.
Accordingly, the correct version of the above query that yields the expected result (i.e., "Yup") would be as follows:

```
SELECT CASE WHEN NULL IS NULL THEN 'YUP' ELSE 'NOPE' END AS RESULT FROM DUAL;
```

This is because null represents an unknown value. If you don't know the value, you can't know whether it equals another value, so = null is always assumed to be false.

? What will be the result of the query below?

```
SQL> SELECT * FROM runners;
```

id	name
1	John Doe
2	Jane Doe
3	Alice Jones
4	Bobby Louis
5	Lisa Romero

```
SQL> SELECT * FROM races;
```

id	event	winner_id
1	100 meter dash	2
2	500 meter dash	3
3	cross-country	2
4	triathalon	NULL

```
SELECT * FROM runners WHERE id NOT IN (SELECT winner_id FROM races)
```

Surprisingly, given the sample data provided, the result of this query will be an empty set. The reason for this is as follows: If the set being evaluated by the SQL NOT IN condition contains any values that are null, then the outer query here will return an empty set, even if there are many runner ids that match winner_ids in the races table.
Knowing this, a query that avoids this issue would be as follows:

```
SELECT * FROM runners WHERE id NOT IN (SELECT winner_id FROM races WHERE winner_id
IS NOT null)
```

? Can we use having without aggregate function

```
SELECT department_id FROM employees GROUP BY department_id HAVING department_id IN
(60,50)
```

? Write a query to fetch values in table test_a that are and not in test_b without using the NOT keyword.

```
CREATE TABLE test_a(id NUMERIC);

INSERT INTO test_a(id) VALUES (10);
INSERT INTO test_a(id) VALUES (20);
INSERT INTO test_a(id) VALUES (30);
INSERT INTO test_a(id) VALUES (40);
INSERT INTO test_a(id) VALUES (50);
INSERT INTO test_b(id) VALUES (10);
INSERT INTO test_b(id) VALUES (30);
INSERT INTO test_b(id) VALUES (50);

SELECT id FROM test_a MINUS SELECT id FROM test_b
```

? Given a table TBL with a field Nmbr that has rows with the following values; Write a query to add 2 where Nmbr is 0 and add 3 where Nmbr is 1.

1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1

```
UPDATE tbl SET nmbr = CASE WHEN nmbr = 0 THEN nmbr+2 ELSE nmbr+3 END;
```

Roles

Role is a group of privileges

```
CREATE ROLE role_name NOT IDENTIFIED; -- Not identified= no password is required to
enable this role
----
CREATE ROLE role_name IDENTIFIED BY password; -- By default, the role granted will
be disabled if it has password
----
SET ROLE role_name IDENTIFIED BY password;-- to enable role
----
GRANT CREATE TABLE TO role_name WITH ADMIN OPTION;
----
GRANT SELECT ON username.object_name TO role_name; --Grant privileges to role
----
GRANT role_name1 TO role_name2; -- copy all privileges from role_name1 to
role_name2
----
GRANT role_name TO username; --Grant a role to user
```

WITH ADMIN OPTION vs. WITH GRANT OPTION

WITH GRANT OPTION	WITH ADMIN OPTION
Only for object privileges, not system privileges.	Only for system privileges, not object privileges.
Revoking any grant will cascade and revoke any privileges assigned by the privileged user	Revoking any grant will only revoke the privileges of that user only (will not cascade), leaving all other users granted.

Database link

It is an object that enables you to access object on another database. The other databases need not to be an Oracle Database system.

```
CREATE [PUBLIC] DATABASE LINK link_name
CONNECT TO remote_username IDENTIFIED BY remote_password
USING 'hostname:port_number/SID' or 'TNS_Name'

CREATE DATABASE LINK ACCDB CONNECT TO scott IDENTIFIED BY tiger USING 'DB1WORLD';
```

EXISTS - NOT EXISTS vs. IN - NOT IN

- In most cases the Oracle cost-based optimizer will create an identical execution plan for IN vs EXISTS, so there is no difference in query performance.
- If you are using the IN operator, the SQL engine will scan all records fetched from the inner query. On the other hand, if we are using EXISTS, the SQL engine will stop the scanning process as soon as it found a match.
- The EXISTS clause is much faster than IN when the subquery results is very large. Conversely, the IN clause is faster than EXISTS when the subquery results is very small.
- IN clause won't return anything if there are NULL values, but the EXISTS will return.

```
SELECT * FROM departments WHERE department_id NOT IN (SELECT department_id FROM
employees);-- AS there is one NULL value, query will not return any value
----
SELECT * FROM departments WHERE department_id NOT IN (SELECT NVL(department_id,0)
FROM employees); --NVL to change the NULL
----
SELECT * FROM departments d WHERE NOT EXISTS (SELECT 1 FROM employees e WHERE
e.department_id=d.department_id); --SELECT 1 because, EXISTS will only checks
whether it returns any rows. We are giving the condition in co-related sub query
```

CASE vs. DECODE

- DECODE performs an equality check only. CASE is capable of other logical comparisons such as < > etc.
- It is recommended to use CASE instead of DECODE
- DECODE is oracle proprietary function; CASE is ANSI standard
- CASE can be used in both SQL and PLSQL . But DECODE can be used only in SQL.
- CASE can be used in WHERE clause But you cant use DECODE in WHERE clause.
- CASE is Faster when compared to DECODE since DECODE is a function which takes time to load and run but the cost difference of DECODE and CASE is very very minimal.

Hierarchical Queries

`START WITH` -> Define the root node(s) of the hierarchy
`CONNECT BY` -> Defines how the current row (child) relates to a prior row(parent)
`PRIOR` -> Previous(parent) level
`LEVEL` -> The position (indentation) in the hierarchy of the current row in relation to the root node
`ORDER SIBLINGS BY` -> Order of the rows which all shares the same parent row
`CONNECT_BY_ROOT` -> To get the root node associated with the current row
`SYS_CONNECT_BY_PATH(column_name,'delimiter')` -> Delimited break from the root node to the current row
`CONNECT_BY_ISLEAF` -> 1 for leaf nodes and 0 for non-leaf nodes
`CONNECT BY NOCYCLE` -> No to raise errors if loop exist. (Ex: KING is manager of BLAKE, BLAKE is manager of KING)

```
SELECT empno, ename, job, PRIOR ename --PRIOR ename = mgr name from previous
level
FROM emp
START WITH mgr IS NULL
CONNECT BY mgr = PRIOR empno;
```

```
SELECT empno, ename, job, PRIOR ename
FROM emp
START WITH empno = 7566
CONNECT BY mgr = PRIOR empno;
```

```
SELECT LPAD('', LEVEL *2, '') || family_member AS who,
mother,
father
FROM your_family
START WITH family_member in ('Aliyamma','Thomas') --level 1 of hierarchy
CONNECT BY NOCYCLE PRIOR family_member IN (mother,father) --PRIOR gives access to level
above
```

```
CREATE TABLE your_family (family_member VARCHAR2(15),mother VARCHAR2(15),father
VARCHAR2(15), birth_date DATE);
----
INSERT INTO your_family VALUES('Aliyamma',NULL,NULL,to_date('01/01/1980','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Thomas',NULL,NULL,to_date('01/01/1980','mm/dd/yyyy'));
INSERT INTO your_family
VALUES('Jose','Aliyamma','Thomas',to_date('01/01/1981','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Sheeja',NULL,NULL,to_date('01/01/1981','mm/dd/yyyy'));
INSERT INTO your_family
VALUES('Mercy','Aliyamma','Thomas',to_date('01/01/1983','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Winny',NULL,NULL,to_date('01/01/1983','mm/dd/yyyy'));
INSERT INTO your_family
VALUES('Annie','Aliyamma','Thomas',to_date('01/01/1982','mm/dd/yyyy'));
INSERT INTO your_family VALUES('James',NULL,NULL,to_date('01/01/1982','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Teni','Sheeja','Jose',to_date('01/01/1987','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Tom','Sheeja','Jose',to_date('01/01/1984','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Sini','Mercy','Winny',to_date('01/01/1988','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Nano','Mercy','Winny',to_date('01/01/1989','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Betty','Annie','James',to_date('01/01/1986','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Merly','Annie','James',to_date('01/01/1985','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Prince',NULL,NULL,to_date('01/01/1985','mm/dd/yyyy'));
INSERT INTO your_family VALUES('Nonu','Merly','Prince',to_date('01/01/1990','mm/dd/yyyy'));
ORDER SIBLINGS BY birth_date;
```

Insert random data to table

```
INSERT INTO table_name
SELECT ROWNUM,
FLOOR(DBMS_RANDOM.VALUE(90,9000)),
TRUNC(SYSDATE) - FLOOR(DBMS_RANDOM.VALUE(90,9000)) FROM DUAL
CONNECT BY LEVEL <= 1000000
```

View

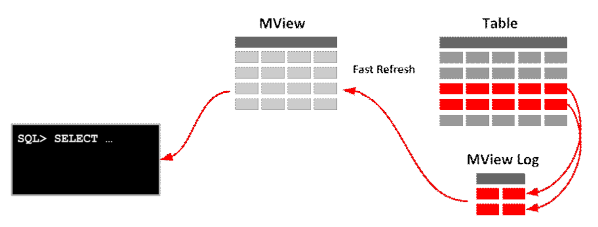
View is a logical table stored in a database. The WITH CHECK OPTION clause is an optional part of the CREATE VIEW statement. The WITH CHECK OPTION clause prevents you from updating or inserting rows that are not visible through the view. Rows cannot be deleted through a view if the view definition contains the DISTINCT keyword.

```
CREATE OR REPLACE VIEW v_test_table AS
SELECT ids, name FROM test_table WHERE ids in (1,2) WITH CHECK OPTION;

INSERT INTO v_test_table VALUES(5,'HAMUN'); --[Error] Execution (4: 13):
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Materialized Views in Oracle

- A materialized view, or snapshot as they were previously known, is a table segment whose contents are periodically refreshed based on a query, either against a local or remote table.
- For data warehousing purposes, the materialized views commonly created are aggregate views, single-table aggregate views, and join views.
- A normal view uses a query to pull data from the underlying tables.
- In replication environments, the materialized views commonly created are primary key, rowid, and subquery materialized views.
- Flashback Table operation is not supported on materialized views.
- Snapshot is an old term for materialized view.

```
-- Normal
CREATE MATERIALIZED VIEW view-name
[BUILD [IMMEDIATE | DEFERRED]]
REFRESH [FAST | COMPLETE | FORCE ]
WITH [ROWID | PRIMARY KEY]
ON [COMMIT | DEMAND ]
[[ENABLE | DISABLE] QUERY REWRITE]
AS
SELECT ...;
```



```
-- Pre-Built
CREATE MATERIALIZED VIEW existing-table-name
ON PREBUILT TABLE
REFRESH [FAST | COMPLETE | FORCE ]
WITH [ROWID | PRIMARY KEY] -- If Primary Key is not available in base table,
use ROWID
ON [COMMIT | DEMAND ]
[[ENABLE | DISABLE] QUERY REWRITE]
AS
SELECT ...;
```

IMMEDIATE : The materialized view is populated immediately.
DEFERRED : The materialized view is populated on the first requested refresh.
FAST : A fast refresh is attempted. If materialized view logs are not present against the source tables in advance, the creation fails. `-- Have to create MVIEW LOG before creating MVIEW`
COMPLETE : The table segment supporting the materialized view is truncated and repopulated completely using the associated query.
FORCE : A fast refresh is attempted. If one is not possible a complete refresh is performed.
ON COMMIT : The refresh is triggered by a committed data change in one of the dependent tables.
ON DEMAND : The refresh is initiated by a manual request or a scheduled task.
QUERY REWRITE: tells the optimizer if the materialized view should be consider for query rewrite operations.
ON PREBUILT TABLE: tells the database to use an existing table segment, which must have the same name as the materialized view and support the same column structure as the query.
Remember to gather stats after building the materialized view.

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS(
    OWNNAME => 'SCOTT', -- Owner Name
    TABNAME => 'EMP_MV'); -- Materialized view name
END;
/
```

Interview (3) Questions and Answers

? What will the result be from the following query:

```
CREATE TABLE dbo.envelope(id int, user_id int);
CREATE TABLE dbo.docs(idnum int, pageseq int, doctext varchar(100));

INSERT INTO dbo.envelope VALUES
  (1,1),
  (2,2),
  (3,3);

INSERT INTO dbo.docs(idnum,pageseq) VALUES
  (1,5),
  (2,6),
  (null,0);

UPDATE docs SET doctext=pageseq FROM docs INNER JOIN envelope ON envelope.id=docs.idnum
WHERE EXISTS (
  SELECT 1 FROM dbo.docs
  WHERE id=envelope.id);
```

The result of the query will be as follows:

Idnum	pageseq	doctext
1	5	5
2	6	6
NULL	0	NULL

The EXISTS clause in the above query is a red herring. It will always be true since ID is not a member of dbo.docs. As such, it will refer to the envelope table comparing itself to itself!
The idnum value of NULL will not be set since the join of NULL will not return a result when attempting a match with any value of envelope.

? What is wrong with this SQL query? Correct it so it executes properly.

```
SELECT Id, YEAR(BillingDate) AS BillingYear
FROM Invoices
WHERE BillingYear >= 2010;
```

The expression BillingYear in the WHERE clause is invalid. Even though it is defined as an alias in the SELECT phrase, which appears before the WHERE phrase, the logical processing order of the phrases of the statement is different from the written order. Most programmers are accustomed to code statements being processed generally top-to-bottom or left-to-right, but T-SQL processes phrases in a different order.

The correct query should be:

```
SELECT Id, YEAR(BillingDate) AS BillingYear
FROM Invoices
WHERE YEAR(BillingDate) >= 2010;
```

? How to find a duplicate record? duplicate records with one field

```
SELECT name, COUNT(email)
 FROM users
 GROUP BY email
 HAVING COUNT(email) > 1
```

? How to find a duplicate record? duplicate records with more than one field

```
SELECT name, email, COUNT(*)
 FROM users
 GROUP BY name, email
 HAVING COUNT(*) > 1
```

? What will be the result of the query? Why? What would be a better way to write it?

Id	Name	ReferredBy
1	John Doe	NULL
2	Jane Smith	NULL
3	Anne Jenkins	2
4	Eric Branford	NULL
5	Pat Richards	1
6	Alice Barnes	2

Here is a query written to return the list of customers not referred by Jane Smith:

```
SELECT Name FROM Customers WHERE ReferredBy <> 2;
```

Although there are 4 customers not referred by Jane Smith (including Jane Smith herself), the query will only return one: Pat Richards. All the customers who were referred by nobody at all (and therefore have NULL in their ReferredBy column) don't show up. But certainly those customers weren't referred by Jane Smith, and certainly NULL is not equal to 2, so why didn't they show up?

SQL Server uses three-valued logic, which can be troublesome for programmers accustomed to the more satisfying two-valued logic (TRUE or FALSE) most programming languages use. In most languages, if you were presented with two predicates: ReferredBy = 2 and ReferredBy <> 2, you would expect one of them to be true and one of them to be false, given the same value of ReferredBy. In SQL Server, however, if ReferredBy is NULL, neither of them are true and neither of them are false. Anything compared to NULL evaluates to the third value in three-valued logic: UNKNOWN.

The query should be written in one of two ways:

```
SELECT Name FROM Customers WHERE ReferredBy IS NULL OR ReferredBy <> 2
```

? Given a table SALARIES, such as the one below, that has m = male and f = female values. Swap all f and m values (i.e., change all f values to m and vice versa) with a single update query and no intermediate temp table.

Id	Name	Sex	Salary
1	A	m	2500
2	B	f	1500
3	C	m	5500
4	D	f	500

```
UPDATE SALARIES SET sex = CASE sex WHEN 'm' THEN 'f' ELSE 'm' END
```

Create Materialized View Logs

Since a complete refresh involves truncating the materialized view segment and re-populating it using the related query, it can be quite time consuming and involve a considerable amount of network traffic when performed against a remote table. To reduce the replication costs, materialized view logs can be created to capture all changes to the base table since the last refresh. This information allows a fast refresh, which only needs to apply the changes rather than a complete refresh of the materialized view.
To take advantage of the fast refresh, connect to the master instance and create the materialized view log.

```
CREATE MATERIALIZED VIEW LOG ON scott.emp
WITH [ROWID | ,PRIMARY KEY | ,SEQUENCE]
[[INCLUDING | EXCLUDING] NEW VALUES];
```

The **NEW VALUES** clause lets you determine whether Oracle Database saves both old and new values for update DML operations in the materialized view log.

INCLUDING

Specify INCLUDING to save both new and old values in the log. If this log is for a table on which you have a single-table materialized aggregate view, and if you want the materialized view to be eligible for fast refresh, then you must specify INCLUDING.

EXCLUDING

Specify EXCLUDING to disable the recording of new values in the log. This is the default. You can use this clause to avoid the overhead of recording new values. Do not use this clause if you have a fast-refreshable single-table materialized aggregate view defined on the master table.

When MVIEW LOG is created, MLOG$ table is automatically created. Also, if you add a Primary Key into the MVIEW, additional table RUPD$ (Global Temporary Table) is created.

MLOG$	
Primary Key	
SNAPTIME$$	DATE
DMLTYPE$$	VARCHAR2(1)
OLD_NEW$$	VARCHAR2(1)
CHANGE_VECTOR$$	RAW(255)
XID$$	NUMBER

RUPD$	
Primary Key	
DMLTYPE$$	VARCHAR2(1)
SNAPID	INTEGER
CHANGE_VECTOR$$	RAW(255)

MLOG$					
Primary Key	SNAPTIME$$	DMLTYPE$$	OLD_NEW$$	CHANGE_VECTOR$$	XID$$
280	01-01-2000	I	N	FE	1688935759611360

Refresh Materialized Views

If a materialized view is configured to refresh on commit, you should never need to manually refresh it, unless a rebuild is necessary. Remember, refreshing on commit is a very intensive operation for volatile base tables. It makes sense to use fast refreshes where possible.
For on demand refreshes, you can choose to manually refresh the materialized view or refresh it as part of a refresh group.
The following code creates a refresh group defined to refresh every minute and assigns a materialized view to it.

```
BEGIN
   DBMS_REFRESH.MAKE(
     NAME                 => 'SCOTT.MINUTE_REFRESH',
     LIST                 => '',
     NEXT_DATE           => SYSDATE,
     INTERVAL            => '/*1:MINS*/ SYSDATE + 1/(60*24)',
     IMPLICIT_DESTROY   => FALSE,
     LAX                  => FALSE,
     JOB                  => 0,
     ROLLBACK_SEG        => NULL,
     PUSH_DEFERRED_RPC  => TRUE,
     REFRESH_AFTER_ERRORS => TRUE,
     PURGE_OPTION        => NULL,
     PARALLELISM         => NULL,
     HEAP_SIZE           => NULL);
END;
------
BEGIN
   DBMS_REFRESH.ADD(    -- Adds materialized views to a refresh group.
     NAME => 'SCOTT.MINUTE_REFRESH',
     LIST => 'SCOTT.EMP_MV',
     LAX  => TRUE);
END;
```

Note:
A materialized view can be manually refreshed using the DBMS_MVIEW package.
`EXEC DBMS_MVIEW.REFRESH('EMP_MV');`

Cleaning Up

To clean up we must remove all objects.

```
CONNECT scott/tiger@db2
DROP MATERIALIZED VIEW emp_mv;
DROP DATABASE LINK ACCDB;
------
BEGIN
  DBMS_REFRESH.DESTROY(name => 'SCOTT.MINUTE_REFRESH');
END;
------
CONNECT scott/tiger@db1
DROP MATERIALIZED VIEW LOG ON scott.emp;
```

Note:
Rather than using a refresh group, you can schedule DBMS_MVIEW.REFRESH called using the Oracle Scheduler

Read-Only, Updatable, and Writeable Materialized Views

A materialized view can be either read-only, updatable, or writeable. Users cannot perform data manipulation language (DML) statements on read-only materialized views, but they can perform DML on updatable and writeable materialized views.

Read-Only Materialized Views

You can make a materialized view read-only during creation by omitting the FOR UPDATE clause.

```
CREATE MATERIALIZED VIEW hr.employees AS SELECT * FROM hr.employees@orc1.world;
```

Updatable Materialized Views

- You can make a materialized view updatable during creation by including the FOR UPDATE clause or enabling the equivalent option in the Replication Management tool. For changes made to an updatable materialized view to be pushed back to the master during refresh, the updatable materialized view must belong to a materialized view group.
- Updatable materialized views enable you to decrease the load on master sites because users can make changes to the data at the materialized view site. The following is an example of an updatable materialized view:

```
CREATE MATERIALIZED VIEW hr.departments FOR UPDATE AS SELECT * FROM
hr.departments@orc1.world;
```

The following statement creates a materialized view group:

```
BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPGROUP (
      gname => 'hr_repg',
      master => 'orc1.world',
      propagation_mode => 'ASYNCHRONOUS');
END;
```

The following statement adds the hr.departments materialized view to the materialized view group, making the materialized view updatable:

```
BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'departments',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
```

Writeable Materialized Views

A writeable materialized view is one that is created using the FOR UPDATE clause but is not part of a materialized view group. Users can perform DML operations on a writeable materialized view, but if you refresh the materialized view, then these changes are not pushed back to the master and the changes are lost in the materialized view itself. Writeable materialized views are rarely used.

Primary Key Materialized Views

- Primary key materialized views are the default type of materialized view. They are updatable if the materialized view was created as part of a materialized view group and FOR UPDATE was specified when defining the materialized view. An updatable materialized view must belong to a materialized view group that has the same name as the replication group at its master site or master materialized view site. In addition, an updatable materialized view must reside in a different database than the master replication group.
- Changes are propagated according to the row-level changes that have occurred, as identified by the primary key value of the row (not the ROWID). The following is an example of a SQL statement for creating an updatable, primary key materialized view:

```
CREATE MATERIALIZED VIEW oe.customers FOR UPDATE AS SELECT * FROM
oe.customers@orc1.world;
```

Object Materialized Views

If a materialized view is based on an object table and is created using the OF type clause, then the materialized view is called an object materialized view. An object materialized view is structured in the same way as an object table. That is, an object materialized view is composed of row objects, and each row object is identified by an object identifier (OID) column.

ROWID Materialized Views

For backward compatibility, Oracle supports ROWID materialized views in addition to the default primary key materialized views. A ROWID materialized view is based on the physical row identifiers (rowids) of the rows in a master.
The following is an example of a CREATE MATERIALIZED VIEW statement that creates a ROWID materialized view:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH WITH ROWID AS SELECT * FROM
oe.orders@orc1.world;
```

Complex Materialized Views

To be fast refreshed, the defining query for a materialized view must observe certain restrictions. If you require a materialized view whose defining query is more general and cannot observe the restrictions, then the materialized view is complex and cannot be fast refreshed.

Specifically, a materialized view is considered complex when the defining query of the materialized view contains:

- A CONNECT BY clause
- An INTERSECT, MINUS, or UNION ALL set operation
- In some cases, the DISTINCT or UNIQUE keyword, although it is possible to have the DISTINCT or UNIQUE keyword in the defining query and still have a simple materialized view
- An aggregate function
- Joins other than those in a subquery
- In some cases, a UNION operation. Specifically, a materialized view with a UNION operation is complex if any one of these conditions is true:
    - Any query within the UNION is complex. The previous bullet items specify when a query makes a materialized view complex.
    - The outermost SELECT list columns do not match for the queries in the UNION. In the following example, the first query only has order_total in the outermost SELECT list while the second query has customer_id in the outermost SELECT list. Therefore, the materialized view is complex.

Dynamic performance views

There are several V$ views that Oracle maintains which can provide performance statistics about the SQL that has run, how often a statement has been executed and many other performance metrics. Ex: VSQL, VSQLAREA, V$SQLTEXT

? Write a SQL query using UNION ALL (not UNION) that uses the WHERE clause to eliminate duplicates. Why might you want to do this?
You can avoid duplicates using UNION ALL and still run much faster than UNION DISTINCT (which is actually same as UNION) by running a query like this:

```
SELECT * FROM mytable WHERE a=X UNION ALL SELECT * FROM mytable WHERE b=Y AND a!=X
```

The key is the AND a!=X part. This gives you the benefits of the UNION (a.k.a., UNION DISTINCT) command, while avoiding much of its performance hit.

? What is the difference between the WHERE and HAVING clauses?
The WHERE clause is used to filter records from a result. The filtering occurs before any groupings are made.
The HAVING clause is used to filter values from a group (i.e., to check conditions after aggregation into groups has been performed).

? What is the difference between single-row functions and multiple-row functions?
Single-row functions work with single row at a time. Ex: UPPER, LOWER, LENGTH.
Multiple-row functions work with data of multiple rows at a time. Ex: SUM, AVG, COUNT

? What the group by clause used for?
The group by clause combines all those records that have identical values in a particular field or any group of fields.

? Write an SQL query to display the text CAPONE as:

```
C
A
P
O
N
E
SELECT SUBSTR('CAPONE', LEVEL, 1)
FROM DUAL CONNECT BY LEVEL <= LENGTH('CAPONE');
```

? What will be the output of below snippet?
Select * FROM "Test"."EMP";

```
 ID
----
  1
  2
  3
  4
  5
(5 rows)
SELECT SUM(1) FROM "Test"."EMP";  -- total row count
SELECT SUM(2) FROM "Test"."EMP";  --total row count * 2
SELECT SUM(3) FROM "Test"."EMP"; -- total row count * 3
```

? Write a query to to get the list of users who took the a training lesson more than once in the same day, grouped by user and training lesson, each ordered from the most recent lesson date to oldest date.

```
SELECT * FROM users;
```

user_id	username
1	John Doe
2	Jane Don
3	Alice Jones
4	Lisa Romero

```
SELECT * FROM training_details;
```

user_training_id	user_id	training_id	training_date
1	1	1	"2015-08-02"
2	2	1	"2015-08-03"
3	3	2	"2015-08-02"
4	4	2	"2015-08-04"
5	2	2	"2015-08-03"
6	1	1	"2015-08-02"
7	3	2	"2015-08-04"
8	4	3	"2015-08-03"
9	1	4	"2015-08-03"
10	3	1	"2015-08-02"
11	4	2	"2015-08-04"
12	3	2	"2015-08-02"
13	1	1	"2015-08-02"
14	4	3	"2015-08-03"

```
SELECT
      u.user_id,
      username,
      training_id,
      training_date,
      count( user_training_id ) AS count
  FROM users u JOIN training_details t ON t.user_id = u.user_id
  GROUP BY u.user_id,
           username,
           training_id,
           training_date
  HAVING count( user_training_id ) > 1
  ORDER BY training_date DESC;
```

Aggregations and Transformations

Materialized views can be used to improve the performance of a variety of queries, including those performing aggregations and transformations of the data. This allows the work to be done once and used repeatedly by multiple sessions, reducing the total load on the server.
Create a materialized view to perform the aggregation in advance (pre-aggregated data), making sure you specify the ENABLE QUERY REWRITE clause.

```
CREATE MATERIALIZED VIEW emp_aggr_mv
BUILD IMMEDIATE
REFRESH FORCE
ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT deptno, SUM(sal) AS sal_by_dept
FROM   emp
GROUP BY deptno;

EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'EMP_AGGR_MV');
```

When using materialized views to improve performance of transformations and aggregations, the QUERY_REWRITE_INTEGRITY and QUERY_REWRITE_ENABLED parameters must be set or the server will not be able to automatically take advantages of query rewrites. These parameters may be set in the pfile or spfile file if they are needed permanently. Later releases have them enabled by default.

```
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = TRUSTED;
ALTER SESSION SET QUERY_REWRITE_ENABLED = TRUE;
```

Normalization

It is a series of steps to obtain a database design that allows for efficient access and eliminate redundancy.

1st Normal Form

- Each cell of a table must contain atomic value.

2nd Normal Form

- Qualify 1st NF
- Every non-prime attribute must be fully functional dependent on key; not part of key(composite key). (No partial dependency)

Prime attribute = part of key; **Non-Prime attribute** = which are not part of key

```
student_id+subject_id -- primary key
mark -- dependent on student and subject_id
teacher -- only depend on subject; not student. Hence it is a partial dependent
column
```

Solution: Move teacher name to subject table and use subject_id only
Solution: Move teacher name to different table and use teacher_id

3rd Normal Form

- Qualify 2nd NF
- all the attributes in a table are determined only by the candidate keys of that relation and not by any non-prime attributes (No transitive dependency)
- 3NF data modeling was ideal for online transaction processing (OLTP) applications with heavy order entry type of needs.

```
student_id+subject_id -- primary key
mark -- dependent on student and subject_id
exam_name -- dependent on student and subject. Ex. IT student will have Java exam
total_marks -- dependent on exam name. Total marks changes based on exam. Ex:
Maths Total: 80, Biology: 40
--But exam_name is not a part of Primary Key(non-prime). This is called transitive
dependency.
```

Solution: Move exam_name and total_marks to a different table and use exam_id

BCNF (Boyce Codd Normal Form)

- Qualify 3rd NF
- It is a slightly stronger version of the third normal form (3NF)
- For any dependency A derives B, A should be a super key(If A is non-prime attribute, it cannot derive B- a prime attribute)
- Only in rare cases does a 3NF table not meet the requirements of BCNF.
- A 3NF table that does not have multiple overlapping candidate keys is guaranteed to be in BCNF.

```
student_id+subject -- primary key
professor -- non-prime attribute. professor can derive the subject which is a
prime attribute. Professor is not a super key.
```

Solution: Move professor and subject to a different table and use professor_id

4th Normal Form

- Qualify BCNF
- It should not have Multi-Valued dependency
- Conditions for Multi-Valued dependency:
  - A table should have at least 3 columns to have Multi-Valued dependency
  - For A derives B, for a single value of A, more than one value of B exist
  - For this table with A, B, C columns, B and C should be independent

--sid is deriving course and hobby (more than 1 value)

s_id	Course	Hobby
1	Science	Cricket
1	Maths	Hockey

-- this two rows of data will give rise to two more rows of data

s_id	Course	Hobby
1	Science	Hockey
1	Maths	Cricket

--no relationship between course and hobby

Solution: Move two columns (Course and Hobby) into two independent tables and use course_id and hobby_id

ACID property

It stands for

- **Atomicity** - "All or nothing" transactions
- **Consistency** -Changes are consistent across the systems and objects
- **Isolation**-Locking the row for concurrent write happens
- **Durability**-Ability to recover lost data

Composite key and Candidate key

- A **composite key** is a primary key composed of multiple columns used to identify a row.
- A **candidate key** can be any column or combination of column that can qualify as unique key. There can be multiple candidate key in a table. Each candidate key can qualify as Primary key.

SQL Joins

Natural Join

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.

```
SELECT ename,dname FROM emp NATURAL JOIN dept;
----
SELECT ename,dname FROM emp JOIN dept
ON (emp.manager_id = dept.manager_id AND emp.department_id = dept.department_id);
----
SELECT ename,dname FROM emp JOIN dept USING(manager_id,department_id);
```

EQUI JOIN (Same result as INNER Join)

Join which contain '=' operator in the join condition.

```
SELECT ename,dname FROM emp,dept WHERE emp.dept_no = dept.dept_no;
```

NON-EQUI JOIN

Join which contain an operator other than '=' in the join condition. <>, >, <, !=, BETWEEN etc.

```
SELECT ename,dname FROM emp,dept WHERE salary > 1000;
```

SELF JOIN

Joining a table by itself.

```
SELECT e1.ename AS employee, e2.ename AS manager FROM emp e1, emp e2 WHERE e1.mgr=
e2.empno;
```

LEFT OUTER JOIN

Displays all matching and non-matching records from left and only matching records from right.

```
SELECT ename,dname FROM emp LEFT OUTER JOIN dept ON (emp.dept_no = dept.dept_no);
----
SELECT ename,dname FROM emp LEFT OUTER JOIN dept USING (dept_no);
----
SELECT ename,dname FROM emp,dept WHERE emp.dept_no = dept.dept_no(+);
```

CROSS JOIN

Gives Cartesian product.

```
SELECT ename,dname FROM emp CROSS JOIN dept;
```

RIGHT OUTER JOIN

Displays all matching and non-matching records from right and only matching records from left.

```
SELECT ename,dname FROM emp RIGHT OUTER JOIN dept ON (emp.dept_no = dept.dept_no);
----
SELECT ename,dname FROM emp RIGHT OUTER JOIN dept USING (dept_no);
----
SELECT ename,dname FROM emp,dept WHERE emp.dept_no(+) = dept.dept_no;
```

FULL OUTER JOIN

Displays all matching and non-matching records from both sides.

```
SELECT ename,dname FROM emp FULL OUTER JOIN dept ON (emp.dept_no = dept.dept_no);
----
SELECT ename,dname FROM emp FULL OUTER JOIN dept USING (dept_no);
```

INNER JOIN (Simple Join)

Display all records which have a match.

```
SELECT ename,dname FROM emp INNER JOIN dept USING (dept_no);
----
SELECT ename,dname FROM emp INNER JOIN dept ON (emp.dept_no = dept.dept_no);
```

INSERT ALL, INSERT FIRST

Unconditional INSERT ALL

```
INSERT ALL
  INTO mytable (column1, column2, column_n) VALUES (value1, value2, value3)
  INTO mytable2 (column1, column2, column_n) VALUES (value1, value2, value3)
  INTO mytable (column1, column2, column_n) VALUES (value1, value2, value3);

INSERT ALL
  INTO dest_tab1 VALUES(id, description)
  INTO dest_tab2 VALUES(id, description)
  INTO dest_tab3 VALUES(id, description)
SELECT id, description
FROM   source_tab;
```

An unconditional INSERT ALL statement can be used to pivot or split data. In the following example we convert each single row representing a week of data into separate rows for each day.

```
INSERT ALL
  INTO pivot_dest VALUES (id, 'mon', mon_val)
  INTO pivot_dest VALUES (id, 'tue', tue_val)
  INTO pivot_dest VALUES (id, 'wed', wed_val)
  INTO pivot_dest VALUES (id, 'thu', thu_val)
  INTO pivot_dest VALUES (id, 'fri', fri_val)
SELECT *
FROM   pivot_source;
```

? Print the rows which have 'Yellow' in one of the columns C1, C2, or C3, but without using OR.

Table is as follows:

ID	C1	C2	C3
1	Red	Yellow	Blue
2	NULL	Red	Green
3	Yellow	NULL	Violet

```
SELECT * FROM table_name WHERE 'Yellow' IN (C1, C2, C3)
```

? Imagine a single column in a table that is populated with either a single digit (0-9) or a single character (a-z, A-Z). Write a SQL query to print 'Fizz' for a numeric value or 'Buzz' for alphabetical value for all values in that column.
Example:
['d', 'x', 'T', 8, 'a', 9, 6, 2, 'V']

...should output:
['Buzz', 'Buzz', 'Buzz', 'Fizz', 'Buzz','Fizz', 'Fizz', 'Fizz', 'Buzz']

```
SELECT col, CASE WHEN UPPER(col) = LOWER(col) THEN 'Fizz' ELSE 'Buzz' END AS FizzBuzz FROM TABLE;
```

Interview (4) Questions and Answers

? Write a query to insert/update Col2's values to look exactly opposite to Col1's values.

Col1	Col2
1	0
0	1
0	1
0	1
1	0
0	1
1	0
1	0

```
UPDATE table_name SET col2 = CASE WHEN col1 = 1 THEN 0 ELSE 1 END
```

? What is the difference between IN and EXISTS?
IN:
Works on List result set
Doesn't work on subqueries resulting in Virtual tables with multiple columns
Compares every value in the result list
Performance is comparatively SLOW for larger resultset of subquery
EXISTS:
Works on Virtual tables
Is used with co-related queries
Exits comparison when match is found
Performance is comparatively FAST for larger resultset of subquery

? Write a single query to calculate the sum of all positive values of x and he sum of all negative values of x.

```
 x
------
 2
-2
 4
-4
-3
 0
 2
```

```
SELECT SUM(CASE WHEN x>0 THEN x ELSE 0 END)sum_pos,SUM(CASE WHEN x<0 THEN x ELSE 0 END)sum_neg FROM
a;
```

? Write a query that produces the output:
Given the table mass_table:

weight
5.67
34.567
365.253
34

Write a query that produces the output:

weight	kg	gms
5.67	5	67
34.567	34	567
365.253	365	253
34	34	0

```
SELECT weight, TRUNC(weight) AS kg, NVL(SUBSTR(weight - TRUNC(weight), 2), 0) AS gms
FROM mass_table;
```

Conditional INSERT ALL

```
INSERT ALL
  WHEN id <= 3 THEN
    INTO dest_tab1 VALUES(id,
description)
  WHEN id BETWEEN 4 AND 7 THEN
    INTO dest_tab2 VALUES(id,
description)
  WHEN id >= 8 THEN
    INTO dest_tab3 VALUES(id,
description)
  ELSE
     INTO dest_tab3 VALUES(id,
     description)
SELECT id, description
FROM   source_tab;
```

A single condition can be used for multiple INTO clauses.

```
INSERT ALL
  WHEN id <= 3 THEN
    INTO dest_tab1 VALUES(id,
description)
  WHEN id BETWEEN 4 AND 7 THEN
    INTO dest_tab2 VALUES(id,
description)
    INTO dest_tab3 VALUES(id,
description)
SELECT id, description
FROM   source_tab;
```

You can use a condition of "1=1" to force all rows into a table.

```
INSERT ALL
  WHEN id <= 3 THEN
    INTO dest_tab1 VALUES(id, description)
  WHEN id BETWEEN 4 AND 7 THEN
    INTO dest_tab2 VALUES(id, description)
  WHEN 1=1 THEN
    INTO dest_tab3 VALUES(id, description)
SELECT id, description
FROM   source_tab;
```

INSERT FIRST

Using INSERT FIRST makes the multi-table insert work like a CASE expression, so the conditions are tested until the first match is found, and no further conditions are tested. We can also include an optional ELSE clause to catch any rows not already cause by a previous condition.

```
INSERT FIRST
  WHEN id <= 3 THEN
    INTO dest_tab1 VALUES(id,
description)
  WHEN id <= 5 THEN
    INTO dest_tab2 VALUES(id,
description)
  ELSE
    INTO dest_tab3 VALUES(id,
description)
SELECT id, description
FROM   source_tab;
```

```
INSERT FIRST
  WHEN id <= 3 THEN
    INTO dest_tab1 VALUES(id,
description)
  ELSE
    INTO dest_tab2 VALUES(id,
description)
    INTO dest_tab3 VALUES(id,
description)
SELECT id, description
FROM   source_tab;
```

- Multi-table inserts can only be performed on tables, not on views or materialized views.
- You cannot perform a multi-table insert via a DB link.
- You cannot perform multi-table inserts into nested tables.
- The sum of all the INTO columns cannot exceed 999.
- Sequences cannot be used in the multi-table insert statement. It is considered a single statement, so only one sequence value will be generated and used for all rows.
- Multi-table inserts can't be used with plan stability.

Pivoting Insert

In Datawarehouse we come across situations where non-relational data has to be stored in a relational format. Pivoting is an operation in which one has to build a transformation such that each record from any input stream, such as a non-relational database table, must be converted into multiple records for a more relational database format.

```
CREATE TABLE  SALES_SOURCE (EMPNO NUMBER(5), WEEKID NUMBER(2),SALES_M
NUMBER(8,2),SALES_TU NUMBER(8,2),SALES_W NUMBER(8,2),SALES_TH NUMBER(8,2),
SALES_F NUMBER(8,2));
------
CREATE TABLE SALES_INFO (EMPID NUMBER(6), WEEK NUMBER(2),SALES NUMBER(8,2));
------
INSERT INTO SALES_SOURCE VALUES(176,6,2000,3000,1000,5000,6000);
```

EMPNO	WEEKID	SALES_M	SALES_TU	SALES_W	SALES_TH	SALES_F
176	6	2000	3000	1000	5000	6000

```
INSERT ALL INTO SALES_INFO
VALUES(EMPLOYID,WEEKID,SALES_MON)
    INTO SALES_INFO
VALUES(EMPLOYID,WEEKID,SALES_TU)
    INTO SALES_INFO
VALUES(EMPLOYID,WEEKID,SALES_WED)
    INTO SALES_INFO
VALUES(EMPLOYID,WEEKID,SALES_TH)
    INTO SALES_INFO VALUES(EMPLOYID,WEEKID,SALES_F)
    SELECT EMPNO EMPLOYID,WEEKID WEEKID,SALES_M
SALES_MON,
    SALES_TU SALES_TU,SALES_W SALES_WED,SALES_TH
SALES_TH,SALES_F SALES_F FROM  SALES_SOURCE;
```

EMPID	WEEK	SALES
176	6	2000
176	6	3000
176	6	1000
176	6	5000
176	6	6000

Calling External Procedures

We can call C program or Java method from PLSQL. With external procedures, you can make "callouts" and "callbacks". Callout means a call to an external procedure. Callback means when the external procedure calls the database to perform SQL operations.

```
CREATE OR REPLACE PROCEDURE s_pr_upload(
connectString IN VARCHAR2, dt_in_trd_date IN VARCHAR2, user IN VARCHAR2,
numout_inserted OUT BINARY_INTEGER, numout_updated OUT BINARY_INTEGER,
retCode OUT BINARY_INTEGER, prc_id IN VARCHAR2)
AS
    EXTERNAL
    LANGUAGE C
    NAME "program_name"
    LIBRARY "DLL_name or shared_library_name"

PARAMETERS(connectString,dt_in_trd_date,user,numout_inserted,numout_updated,retCod
e,prc_id);
```

SQL Injection

SQL Injection is one of the techniques uses by hackers to hack a website by injecting SQL commands in data fields.

Ex1: In the username field in a webpage if a user types, Raj OR 1=1, the SQL statement for accepting this field will look like this.

`SELECT * FROM users WHERE username = ‘Raj’ OR 1 = 1;`

This will return all the rows in that table because 1 = 1 is always true.

Ex2:

Username: RAJ; `DROP TABLE users`

SQL Statement: `SELECT * FROM users WHERE username = ‘Raj’; DROP TABLE users;`

Ex3:

l_query := '`SELECT claim_id, claim_amount FROM claims WHERE ssn =`' || l_ssn;

The user passes the following for the value l_ssn

'123456789' `UNION ALL SELECT claim_id, claim_amount FROM claims`

The value of the l_query variable is transformed to:

SQL Statement: `SELECT claim_id, claim_amount FROM claims WHERE ssn = '123456789' UNION ALL SELECT claim_id, claim_amount FROM claims`

Use the following approaches to avoid SQL injection vulnerabilities:

- Never accept inputs from end user that you glue into a sql statement. They should be bound (bind variables) into the query not concatenated.
- Validate user input
- Use AUTHID CURRENT_USER while creating objects which will prevent the execution of object with privileges of owner

SET operators

UNION – Combines multiple outputs and eliminates duplicates
UNION ALL – Combines multiple outputs and keeps the duplicates
INTERSECT – Gives common rows from the both tables
MINUS – Eliminate the rows which are present in the second table from the first table

SQL operators

ARITHMETIC +, -, /, *, %

LOGICAL

ALL - TRUE if all of the subquery values meet the condition

```
SELECT * FROM emp WHERE sal > ALL(800,2900,5000,6000);
```

ANY - TRUE if any of the subquery values meet the condition

```
SELECT * FROM emp WHERE sal > ANY(800,2900,5000,6000);
```

EXISTS - TRUE if the subquery returns one or more records. It is possible to do with the JOIN to get the same result. But EXISTS is faster. If there are 10 categories and 20 billion products, EXISTS search for the 1st category and finds it in 3rd row. The search is complete. Next search for 2nd category – finds in 5th row – search is complete. If we are using JOIN instead of EXISTS, JOIN will join all the categories with all the 20 billion products and shows the result after that.

```
SELECT * FROM dept WHERE EXISTS (SELECT * FROM emp WHERE deptno = 30);
```

BETWEEN - TRUE if the operand is within the range of comparisons
IN - TRUE if the operand is equal to one of a list of expressions
LIKE - TRUE if the operand matches a pattern
AND - TRUE if all the conditions separated by AND is TRUE
OR - TRUE if any of the conditions separated by OR is TRUE
NOT - Displays a record if the condition(s) is NOT TRUE

COMPARISION

<>, <, >, <=, >=, =

Aggregate functions

Aggregate functions are used to compute a column that is returned from a SELECT query.
Ex: MIN, MAX, AVG, SUM, COUNT, COUNT (*)

INTERVAL Data Type

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';

SELECT INTERVAL '21-2' YEAR TO MONTH FROM DUAL; -- +21-02 -- 21 years 2 Months
SELECT INTERVAL '100-2' YEAR(3) TO MONTH FROM DUAL; -- +100-02 -- 100 years 2 Months
SELECT INTERVAL '5' YEAR FROM DUAL; -- +05-00 -- 5 years 0 Months
SELECT INTERVAL '500' MONTH(3) FROM DUAL; -- +41-08 -- 41 years 8 Months
SELECT INTERVAL '1-0' YEAR TO MONTH FROM DUAL; -- +01-00 -- 1 year 0 Months
SELECT INTERVAL '1' YEAR - INTERVAL '3' MONTH FROM DUAL; -- +00-09  -- 0 Year 9
Months
SELECT INTERVAL '2 3:04:11.333' DAY TO SECOND(3) FROM DUAL; -- +02 03:04:11.333000 --
2 Days 3 Hours 4 Minute 11.333 seconds
SELECT INTERVAL '2 3:04' DAY TO MINUTE FROM DUAL; -- +02 03:04:00.000000 -- 2 Days 3
Hours 4 Minute
SELECT INTERVAL '2 3' DAY TO HOUR FROM DUAL; -- +02 03:00:00.000000 -- 2 Days 3 Hours
SELECT INTERVAL '2' DAY FROM DUAL; -- +02 00:00:00.000000 -- 2 Days
SELECT INTERVAL '3:04:11' HOUR TO SECOND(3) FROM DUAL; -- +00 03:04:11.000000 -- 0
Days 3 Hours 4 Minute 11 seconds
SELECT INTERVAL '4:11' MINUTE TO SECOND(3) FROM DUAL; -- +00 00:04:11.000000 -- 0
Days 0 Hours 4 Minute 11 seconds
SELECT INTERVAL '3:04' HOUR TO MINUTE FROM DUAL; -- +00 03:04:00.000000 -- 0 Days 3
Hours 4 Minute 0 seconds
SELECT INTERVAL '500' MINUTE FROM DUAL; -- +00 08:20:00.000000 -- 0 Days 8 Hours 20
Minute 0 seconds
SELECT INTERVAL '1' DAY - INTERVAL '20' HOUR FROM DUAL; -- +00 04:00:00.000000 -- 0
Days 4 Hours 0 Minute 0 seconds
SELECT NUMTOYMINTERVAL(28,'MONTH') FROM DUAL; -- +02-04 -- 2 Years 4 Months
SELECT NUMTODSINTERVAL(28,'HOUR') FROM DUAL; -- +01 04:00:00.000000 -- 1 Day 4 Hours
SELECT TO_YMINTERVAL('29-11') FROM DUAL; -- +29-11 -- 29 Years 11 Months
SELECT TO_DSINTERVAL('2 10:33:45') FROM DUAL; -- +02 10:33:45.000000 -- 2 Days 10
Hours 33 Minute 45 Seconds
SELECT EXTRACT(HOUR FROM NUMTODSINTERVAL(28,'HOUR')) FROM DUAL; -- 4 -- 4 Hours
SELECT EXTRACT(HOUR FROM NUMTODSINTERVAL(500,'MINUTE')) FROM DUAL; -- 8 -- 8 Hours
SELECT EXTRACT(MINUTE FROM NUMTODSINTERVAL(500,'MINUTE')) FROM DUAL; -- 20 -- 20
Minute
SELECT CAST(SYSDATE AS TIMESTAMP) FROM DUAL; -- 15-JUN-19 1:26:23.000000 PM
SELECT SESSIONTIMEZONE FROM DUAL; -- +05:30
```

? Write a query to generate below output:
Consider the Employee table below.

Emp_Id	Emp_name	Salary	Manager_Id
10	Anil	50000	18
11	Vikas	75000	16
12	Nisha	40000	18
13	Nidhi	60000	17
14	Priya	80000	18
15	Mohit	45000	18
16	Rajesh	90000	–
17	Raman	55000	16
18	Santosh	65000	17

Write a query to generate below output:

Manager_Id	Manager	Average_Salary_Under_Manager
16	Rajesh	65000
17	Raman	62500
18	Santosh	53750

```
SELECT b.emp_id AS "manager_id",
          b.emp_name AS "manager",
          AVG(a.salary) AS "average_salary_under_manager"
FROM employee a,
     employee b
WHERE a.manager_id = b.emp_id
GROUP BY b.emp_id, b.emp_name
ORDER BY b.emp_id;
```

? Write a query to print the class obtained by each student. First class above 60, second class between 50 and 60 and pass between 49 and 35 and score below that is fail.

```
SELECT * FROM tbl_students;

SELECT stu.*, (CASE
      WHEN marks > 60 THEN
        'First Class'
      WHEN marks BETWEEN 50 AND 60 THEN
        'Second Class'
      WHEN marks BETWEEN 49 AND 35 THEN
        'Pass'
      WHEN marks <= 35 THEN
        'Fail'
      END) AS class
      FROM tbl_students stu;
```

Marks	Name
100	Prashant
22	Roshan
35	Amar

? Write a query to fetch the second highest score and the name of the student who scored it.

```
SELECT * FROM (SELECT stu.*, DENSE_RANK() OVER (ORDER BY marks DESC) AS rank
  FROM tbl_students stu) WHERE rank =2;
```

? Write a procedure to pass the marks as parameters and fetch a result set which would give a list of all students scoring above these marks.

```
CREATE OR REPLACE PROCEDURE prg_stumarks(p_marks NUMBER) IS
  CURSOR stu_cur IS
    SELECT * FROM tbl_students WHERE marks > p_marks;
  v_stutable tbl_students %ROWTYPE;
BEGIN
  OPEN stu_cur;
  LOOP
    FETCH stu_cur INTO v_stutable;
    EXIT WHEN stu_cur %NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Name:' || v_stutable.name || ' Marks:' || v_stutable.marks);
  END LOOP;
  CLOSE stu_cur;
END;

EXEC prg_stumarks(30);
```

? Query to find duplicate column names.

```
SELECT COLUMN_NAME, COUNT(COLUMN_NAME)
    FROM USER_TAB_COLUMNS
    GROUP BY COLUMN_NAME
    HAVING COUNT (COLUMN_NAME) > 1;
```

? While marking data entry into above table operator made a mistake and has entries a few names multiple times. Write a query to extract those rows that have repeating names.

```
SELECT * FROM tbl_players;

SELECT * FROM(
  SELECT plyr.*,
  COUNT(*) OVER (PARTITION BY player) cnt
  FROM tbl_players plyr) WHERE cnt >1;
```

Id	Player
1	Ram
2	Rahul
3	Sachin

Id	Player
4	Rahul
5	Ram
6	Amrish

? Now after pulling all the records having the same name, delete all those rows which are duplicates using a single delete statement

```
DELETE FROM tbl_players plyr_a
  WHERE ROWID > (    --ROWID = will delete all duplicate records.
                     --ROWID < will not delete anything as it is taking MIN(ROWID)
                     --ROWID > will keep the row with minimum ROWID and deletes all above that
    SELECT MIN(ROWID) FROM tbl_players plyr_b -- taking minimum ROWID for keeping
      WHERE plyr_a.player = plyr_b.player); -- Duplicate data matching
```

? What will be the result if we add NULL with other values?

```
SELECT 2+3+NULL FROM DUAL; -- NULL; a number + null = null
SELECT SUM(number1) FROM tbl_join2; -- The SUM function will return the correct value of the sum
of the column even if the column contains nulls
```

```sql
SELECT DBTIMEZONE FROM DUAL; -- +00:00
SELECT TO_TIMESTAMP('10-07-2014','DD-MM-YYYY') FROM DUAL; -- 10-JUL-14
12:00:00.000000000 AM
SELECT CURRENT_TIMESTAMP FROM DUAL; -- 15-JUN-19 1:28:44.017031 PM +05:30
SELECT CURRENT_TIMESTAMP(3) FROM DUAL; -- 15-JUN-19 1:28:53.094 PM +05:30
SELECT SYSTIMESTAMP(4) FROM DUAL; -- 15-JUN-19 1:30:33.1550 PM +05:30

DECLARE
time1 INTERVAL YEAR TO MONTH; --Default number of digit is 2
time2 INTERVAL DAY TO SECOND;
time3 INTERVAL YEAR(3) TO MONTH; --3 digit number of years
time4 INTERVAL DAY(4) TO SECOND(9);
BEGIN
time1 := '21-2';
time2 := '2 3:04:11.33';
time3 := '100-2';
END;
```

```sql
SELECT TO_CHAR(SYSDATE + 7/24,'DD-MON-YYYY HH24:MI:SS') FROM DUAL; -- SYSDATE +
7 hours -- 16-JUN-2019 23:40:31
SELECT TO_CHAR(SYSDATE - 7/24,'DD-MON-YYYY HH24:MI:SS') FROM DUAL; -- past 7
hours -- 16-JUN-2019 09:40:46
SELECT TO_CHAR(SYSDATE - 7/1400,'DD-MON-YYYY HH24:MI:SS') FROM DUAL; -- past 7
minutes   60minutes*24hours = 1400 -- 16-JUN-2019 16:33:43
SELECT TO_CHAR(SYSDATE - 7/86400,'DD-MON-YYYY HH24:MI:SS') FROM DUAL; -- past 7
seconds  60minutes*60seconds*24hours = 86400 -- 16-JUN-2019 16:41:01
SELECT TO_CHAR(SYSDATE - 7/24/60/60,'DD-MON-YYYY HH24:MI:SS') FROM DUAL; -- past
7 seconds
SELECT TO_CHAR(SYSDATE - 7/24/60,'DD-MON-YYYY HH24:MI:SS') FROM DUAL; -- past 7
minutes
SELECT TO_CHAR(TRUNC(SYSDATE+1/24,'HH'),'DD-MON-YYYY HH24:MI:SS') FROM DUAL; --
Every one hour starting with the next hour -- 16-JUN-2019 17:00:00
SELECT TO_CHAR(SYSDATE + 1,'DD-MON-YYYY HH24:MI:SS') FROM DUAL; -- 1 Day
SELECT TO_CHAR(SYSDATE + 1/24,'DD-MON-YYYY HH24:MI:SS') FROM DUAL; -- 1 Hour
SELECT TO_CHAR(SYSDATE + 1/48,'DD-MON-YYYY HH24:MI:SS') FROM DUAL; -- 1/2 Hour
SELECT TO_CHAR(SYSDATE + 1/96,'DD-MON-YYYY HH24:MI:SS') FROM DUAL; -- 15 Minutes
SELECT TO_CHAR(SYSDATE, 'HH24') FROM DUAL; -- Hour in 24 hour format -- 16
SELECT TO_CHAR(SYSDATE, 'DY') FROM DUAL; -- Current Day -- SUN
```

Oracle SALT

Oracle Service Architecture Leveraging Tuxedo (SALT) is an add-on product option for Oracle Tuxedo. Oracle SALT allows external Web services applications to invoke Tuxedo services as Web services, and Tuxedo applications to invoke external Web services. Oracle SALT does not require any coding to achieve this. Oracle SALT complies with standard Web service specifications (SOAP 1.1, SOAP 1.2, and WSDL 1.1), allowing Oracle SALT to interoperate with other Web service products.
Web services are a set of functions packaged into a single entity made available to other systems on a network. They can be shared and used as a component of distributed Web-based applications. The network can be a corporate intranet or the Internet.

Comment on columns

```sql
COMMENT ON COLUMN employees.employee_id IS 'EMPLOYEE ID';
SELECT * FROM USER_COL_COMMENTS WHERE TABLE_NAME = 'EMPLOYEES';
COMMENT ON COLUMN employees.employee_id IS ''; -- To drop the comment
```

Monitoring Sessions

- In SQL Developer, click Tools, then Monitor Sessions.
- Right-click in any row in the display, and explore the options available as shown in the context menu commands, which include Trace Session, Kill Session, and Find/Highlight

To monitor the top SQL statements:

- In SQL Developer, click the Reports navigator tab, and expand the hierarchy as follows: All Reports, then Data Dictionary Reports, then Database Administration, then Top SQL.

To monitor long operations:

- In SQL Developer, click the Reports navigator tab, and expand the hierarchy as follows: All Reports, then Data Dictionary Reports, then Database Administration, then Sessions.
- Under Sessions, select Active Sessions.
- If you are asked to select a connection, select one for SYS AS SYSDBA.
- Check the UP_TIME value for each listed session, and note any that you consider to be longer than desired or expected.

To find invalid schema objects:

- If the Reports navigator does not appear in SQL Developer, choose the Reports option from the View menu to display the Reports navigator.
- The Reports navigator appears.
- In the Reports navigator, expand the All Reports node, then expand the Data Dictionary Reports node, then expand the All Objects node, and then click Invalid Objects.
- The Select Connection dialog box appears.
- In the Select Connection dialog box, select the connection to use, or create a new connection.
- If you want the invalid objects report to include information about only the invalid objects in your own schema, use a connection for your own schema.
- If you want the invalid objects report to include information about invalid objects throughout the database, use a connection for a privileged user, such as SYS. In this example, the connection chosen is for the SYS user.
- Click OK.
- The Enter Bind Values dialog box appears.
- Click Apply.
- The Invalid Objects tab appears in the object pane. This tab lists the invalid objects in your schema or in the database (depending on the connection you specified in the Select Connection dialog box).
- When you right-click the row for a particular invalid object on the Invalid Objects tab, the Compile option appears. Select that option to recompile the invalid object.

DBMS_STATS

Oracle uses a cost-based optimizer to determine the appropriate query plan for the given SQL statement. DBMS_STATS gathers the statistics that allow Oracle to make this determination.

```sql
BEGIN
 DBMS_STATS.GATHER_TABLE_STATS('HR','EMPLOYEES');
END;
----
EXEC DBMS_STATS.GATHER_INDEX_STATS(OWNNAME => 'SCOTT', INDNAME => 'EMP_BITMAP_IDX');
```

```sql
SELECT OWNER, OBJECT_TYPE, OBJECT_NAME, STATUS FROM DBA_OBJECTS WHERE STATUS =
'INVALID' AND OWNER IN('TRPROD','BMVPROD','TRARCHVL','TRADT')
UNION
SELECT OWNER, OBJECT_TYPE, OBJECT_NAME, STATUS FROM DBA_OBJECTS WHERE STATUS =
'VALID' AND OWNER IN('TRPROD','BMVPROD','TRARCHVL','TRADT') ORDER BY
OBJECT_TYPE;

SELECT OWNER, OBJECT_TYPE,
COUNT(CASE STATUS WHEN 'VALID' THEN STATUS END) "VALID",
COUNT(CASE STATUS WHEN 'INVALID' THEN STATUS END) "INVALID",
COUNT(1) "TOTAL"
FROM DBA_OBJECTS WHERE OWNER IN('TRPROD','BMVPROD','TRARCHVL','TRADT')
GROUP BY OWNER, OBJECT_TYPE ORDER BY OBJECT_TYPE;

SELECT OWNER, TABLE_NAME, NUM_ROWS FROM DBA_TABLES WHERE TABLE_NAME LIKE 'DRVTS%'
AND OWNER IN ('TRPROD','BMVPROD','TRARCHVL','TRADT');

SELECT TABLE_OWNER, TABLE_NAME, INSERTS, UPDATES, DELETES FROM ALL_TAB_MODIFICATIONS
WHERE TABLE_NAME LIKE 'DRVTS%' AND TABLE_OWNER IN
('TRPROD','BMVPROD','TRARCHVL','TRADT');

SELECT * FROM ALL_TAB_COLUMNS WHERE TABLE_NAME LIKE 'DRVTS%' AND OWNER IN
('TRPROD','BMVPROD','TRARCHVL','TRADT');;

SELECT MAX(ORA_ROWSCN),SCN_TO_TIMESTAMP(MAX(ORA_ROWSCN)) FROM ACTVTY_LOG;

SELECT CHECKSUM_AGG(BINARY_CHECKSUM(TABLE_NAME, COLUMN_NAME,
CHARACTER_MAXIMUM_LENGTH)) FROM INFORMATION_SCHEMA.COLUMNS;

SELECT STANDARD_HASH('sdsd'||'sds') FROM DUAL;

SELECT DBMS_CRYPTO.HASH(DBMS_METADATA.GET_DDL('sdsdsd', 'sdsdaaa', '111'),
DBMS_CRYPTO.HASH_SH1) FROM DUAL;

SELECT OWA_OPT_LOCK.CHECKSUM(ACL_TIME) FROM ACTVTY_LOG;

SELECT ORA_HASH(ACL_TIME) FROM ACTVTY_LOG;

SELECT CHECKSUM_AGG(BINARY_CHECKSUM(*)) FROM ACTVTY_LOG;

DECLARE
HASH_VALUE VARCHAR2(100);
BEGIN
HASH_VALUE := DBMS_SQLHASH.gethash('SELECT * FROM
ACTVTY_LOG',DBMS_CRYPTO.HASH_MD5);
DBMS_OUTPUT.PUT_LINE(HASH_VALUE);
END;

SELECT * FROM DBA_OBJECTS WHERE TRUNC(CREATED)=TRUNC(SYSDATE) AND OWNER='PROD'
ORDER BY CREATED DESC;

SELECT * FROM DBA_OBJECTS WHERE TRUNC(LAST_DDL_TIME)=TRUNC(SYSDATE) AND
OWNER='PROD' ORDER BY LAST_DDL_TIME DESC;
```

Types of ORDER BY

Example using a correlation name

```sql
SELECT CITY_NAME, COUNTRY AS NATION
     FROM CITIES
     ORDER BY NATION
```

Example using a numeric expression

```sql
SELECT name, salary, bonus FROM employee
    ORDER BY salary+bonus
```

Example using a function

```sql
SELECT i, len FROM measures
   ORDER BY sin(i)
```

Example specifying null ordering

```sql
SELECT * FROM t1 ORDER BY c1 DESC NULLS LAST
```

```sql
SELECT EMPLOYEE_ID FROM EMPLOYEES ORDER BY SALARY; -- sort using the column which
is not in select statement
```

- If SELECT DISTINCT is specified or if the SELECT statement contains a GROUP BY clause, the ORDER BY columns must be in the SELECT list.
- An ORDER BY clause prevents a SELECT statement from being an updatable cursor.
- If the null ordering is not specified then the handling of the null values is:
  - NULLS LAST if the sort is ASC
  - NULLS FIRST if the sort is DESC

Indexes

- Indexes are used to search the rows in the table quickly. If the index is not present, the select query has to read the whole table (full table scan) and returns the rows. With index, the rows can be retrieved quickly.
- Indexes are logically and physically independent of data. So you can drop or create indexes without affecting the data.
- They decrease performance on inserts, updates, and deletes.
- Indexes take additional disk space.

B-Tree Index

- B-Tree or balanced tree indexes are most common type of index.
- It stores the ROWID and the index key value in the tree structure
- B-Tree indexes are most useful on columns that appear on the WHERE clause
- When creating an index, a ROOT block is created, then BRANCH blocks are created and finally LEAF blocks
- B-Tree indexes automatically stay balanced by growing BRANCH and LEAF
  `CREATE [UNIQUE] INDEX index_name ON tabel_name(column_name1[,column_name2]);`

Reverse Key Index

- It is a type of b-tree index that can physically store the index in reverses the bytes of each index key while keeping the column order
- A reverse key Index can be useful if the primary key (or other Index) of an column that is filled by a sequence.
  When generating new records in the base table, because of the used sequence you'll get a high contention on the same Index branches and thus database blocks; It can have a high impact on Insert on that table.
- When using a reverse key index, this behavior - the usage of a sequence - can and will be spread over many more branches and database blocks, thus relieving the Index contention
  `CREATE INDEX t1_id_idx ON t1 (id) REVERSE;`

Write a query to create the following table ID (Primary Key and auto increment as 1 not null), playername (not null), iscaptain (true or false value),writeup(no restriction on the length), after creating, add a new column 'full_name'.

```sql
CREATE TABLE tbl_player (pid NUMBER (2) CONSTRAINT pk_plyrid PRIMARY KEY,
player_name VARCHAR2(20) CONSTRAINT nn_plyrname NOT NULL,
iscaptain VARCHAR2(5) CONSTRAINT chk_plyrcap CHECK(iscaptain IN ('true','false')),
writeup VARCHAR2(1000));

CREATE SEQUENCE plyr_id
START WITH 1
INCREMENT BY 1;

CREATE OR REPLACE TRIGGER plyr_insrt
BEFORE INSERT ON tbl_player
FOR EACH ROW
BEGIN
  SELECT plyr_id.NEXTVAL
  INTO :NEW.pid
  FROM DUAL;
END;

INSERT INTO tbl_player VALUES(NULL,'Ashok','true','Description');

ALTER TABLE tbl_player ADD (full_name VARCHAR2(20));
```

Create a function which will take a parameter as the total balls and return its equivalent overs. Ex; if we pass 8, output should be 1.3 (keeping in mind each over is 6 balls).

```sql
CREATE OR REPLACE FUNCTION func_overs(p_balls NUMBER)
RETURN NUMBER IS
BEGIN
  RETURN(p_balls/6);
END;
----
DECLARE
    overs NUMBER(3,1);
BEGIN
    overs := func_overs(8);
    DBMS_OUTPUT.PUT_LINE(overs);
END;
```

We have a table named tbl_authors which is very important. Hence we should restrict any delete or update statements on this. Create a trigger to print a message which says "The content of this table cannot be deleted"

```sql
ALTER TABLE tbl_author READ ONLY;
----
CREATE OR REPLACE TRIGGER trg_auth
BEFORE DELETE OR UPDATE ON tbl_author
BEGIN
  IF DELETING THEN
    RAISE_APPLICATION_ERROR(-20000, 'The content of this table cannot be deleted');
  ELSIF UPDATING THEN
    RAISE_APPLICATION_ERROR(-20000, 'The content of this table cannot be updated');
  END IF;
END;
```

Write a query to fetch the team id and team names from Cricket_Team_Detail table shown above. Sort the query results by team name with the first team name being 'India' and then the remaining results are sorted asc by team name.

```sql
SELECT ct1.* FROM Cricket_Team_Detail ct1 WHERE team_name = 'India'
UNION ALL
  SELECT * FROM
    (SELECT ct2.* FROM Cricket_Team_Detail ct2
         WHERE team_name != 'India'
         ORDER BY team_name ASC);

-- In a SET operation, ORDER BY is allowed only at the end of statement
```

Team Id	Team Name
1	Australlia
2	England
3	India
4	South Africa
5	Sri Lanka

Sort the above table by team name with the first team name being 'India', second team name as 'Mumbai' and then the remaining results are sorted asc by team name.

```sql
SELECT * FROM tbl_team
ORDER BY (CASE
    WHEN team = 'India' THEN 1
    WHEN team = 'Mumbai' THEN 2
    ELSE 3
    END);
```

Cricket_Bat table stores 11 players each for the two teams playing a cricket match. The table stores data for all the international cricket matches played till date. Write a query to fetch all the Match_Id, Team_Id combination having less than 11 batsman associated to that combination.

```sql
SELECT MATCH_ID,TEAM_ID,COUNT(*) FROM CRICKET_BAT
GROUP BY MATCH_ID,TEAM_ID
HAVING COUNT(*) < 11;
```

Find the number of 'c' in the string 'abccccd'

```sql
SELECT LENGTH('abccccd')-LENGTH(REPLACE('abccccd','c',NULL)) FROM DUAL;
```

NULL

What will be the result of different joins in this table

T1
1
1
1
1
1

T2
1
1
1

INNER JOIN/ NATURAL JOIN/ LEFT OUTER JOIN/ RIGHT OUTER JOIN/ FULL OUTER JOIN/ CROSS JOIN
15 row

Display the salary of previous employee for each row

```sql
SELECT first_name,salary,LAG(salary) OVER (ORDER BY salary) FROM employees a;
```

What are the limitations of ROWNUM?

```sql
SELECT * FROM employees WHERE ROWNUM > 5; -- no result
SELECT * FROM employees WHERE ROWNUM < 5; --first four rows
SELECT * FROM employees WHERE ROWNUM = 5; -- no result
```

Row numbers are only useful for = 1, < something or <= something. Conditions testing for ROWNUM values greater than a positive integer are always false. The first row fetched is assigned a ROWNUM of 1 and makes the condition false. The second row to be fetched is now the first row and is also assigned a ROWNUM of 1 and makes the condition false. All rows subsequently fail to satisfy the condition, so no rows are returned.

Function-based Index

- Function-based indexes are efficient for evaluating statements that contain functions in their WHERE clauses.
- The database only uses the function-based index when the function is included in a query.

```
SELECT emp_id,emp_name FROM emp WHERE emp_name = UPPER('Raj');
CREATE INDEX idx_fbi_emp ON emp(UPPER(emp_name));
```

Bitmap Index

- In a bitmap index, the database stores a bitmap for each index key.
- In a conventional B-tree index, one index entry points to a single row. In a bitmap index, each index key stores pointers to multiple rows.
- Bitmap indexes are primarily designed for data warehousing or environments in which queries reference many columns in an ad hoc fashion.

Situations that may call for a bitmap index include:

- The indexed columns have low cardinality, that is, the number of distinct values are small (more duplicate rows) compared to the number of table rows.
- The indexed table is either read-only or not subject to significant modification by DML statements.

```
CREATE BITMAP INDEX emp_bitmap_idx ON index_demo(gender)
```

Composite Indexes

A composite index, also called a concatenated index, is an index on multiple columns in a table.

Unique and Non-unique Indexes

Indexes can be unique or non-unique. Oracle creates unique index for Primary key and unique key constraints.

USING INDEX

If you require more explicit control over the indexes associated with UNIQUE and PRIMARY KEY constraints, the database lets you:

- Specify an existing index that the database is to use to enforce the constraint
- Specify a CREATE INDEX statement that the database is to use to create the index and enforce the constraint

```
CREATE TABLE a (
     a1 INT PRIMARY KEY USING INDEX (CREATE INDEX ai ON a (a1)));
---------
CREATE TABLE b(
     b1 INT,
     b2 INT,
     CONSTRAINT bu1 UNIQUE (b1, b2)
                    USING INDEX (CREATE UNIQUE INDEX bi ON b(b1, b2)),
     CONSTRAINT bu2 UNIQUE (b2, b1) USING INDEX bi);
---------
CREATE TABLE c(c1 INT, c2 INT);
CREATE INDEX ci ON c (c1, c2);
ALTER TABLE c ADD CONSTRAINT cpk PRIMARY KEY (c1) USING INDEX ci;
```

High Cardinality & Low Cardinality

High Cardinality: Number of total records (unique values. Ex: employee id)
Normal Cardinality: Number of total records(less duplicates. Ex: employee name, address)
Low Cardinality: Number of total records (more duplicates. Ex: Yes or No, Male or Female)

SYS_CONTEXT

Used for getting information about the environment

```
SYS_CONTEXT (namespace,parameter[,length])
SELECT SYS_CONTEXT ('USERENV','SESSION_USER') session_user FROM DUAL; -- Current
DB user
SELECT SYS_CONTEXT ('USERENV','ISDBA') isdba FROM DUAL; -- DB user is DBA or not
(Boolean)
SELECT SYS_CONTEXT ('USERENV','HOST') host FROM DUAL; -- Computer name
SELECT SYS_CONTEXT ('USERENV','INSTANCE') instance FROM DUAL; -- DB Instance
(Number)
SELECT SYS_CONTEXT ('USERENV','IP_ADDRESS') ip_address FROM DUAL; -- Local IP
address
SELECT SYS_CONTEXT ('USERENV','DB_NAME') db_name FROM DUAL; -- SID
SELECT SYS_CONTEXT ('USERENV','CURRENT_SCHEMA') current_schema FROM DUAL; --
Schema Name
SELECT SYS_CONTEXT ('USERENV','OS_USER') os_user FROM DUAL; -- OS UserName
SELECT SYS_CONTEXT ('USERENV','SID') sessionnum FROM DUAL; -- Session Number
SELECT SYS_CONTEXT ('USERENV','LANGUAGE') language FROM DUAL; -- OS Language
```

VSIZE

To know the number of bytes occupied by the column

```
SELECT VSIZE('This is VSIZE') FROM DUAL;
SELECT VSIZE(first_name) FROM employees; --Gives size occupied by each row
```

Analytical Functions

- Analytical functions are similar to aggregate function. Aggregate functions are applied on entire selected data. Analytical function offers additional stuffs.
- It helps to avoid self join
- Built on Aggregate Functions
- Row-wise Group Data
- Complex Operations (Statistical, Reporting, etc.)
- NOT a replacement for Aggregate Functions

Syntax

```
analytical_fucntion(column1,column2...)
OVER(
[query_partition_clause]
[order_by_clause]
[windowing_clause]
)
```

RANK

Will rank the rows with gaps in ranking sequence if there are ties.

```
--Ranking the salary inside each department
SELECT department_id,salary, RANK() OVER (PARTITION BY department_id ORDER BY
salary DESC)
AS ranking FROM employees;
--Ranking the salary inside each department without any duplicate (department and
salary); No same ranking in same department
SELECT department_id,salary, RANK() OVER (PARTITION BY department_id ORDER BY
salary DESC)
AS ranking FROM employees
GROUP BY department_id, salary;
```

DENSE_RANK

Will rank the rows with no gaps in ranking sequence if there are ties.

```
--To find 4th highest paid employee in the organization
SELECT * FROM(
    SELECT ename,sal, DENSE_RANK() OVER (ORDER BY sal DESC) AS ranking FROM emp
) WHERE ranking = 4;
```

ROW_NUMBER

ROWNUM is a pseudo-column which assigns the number to the result set prior to the ORDER BY being evaluated. So ORDER BY ROWNUM does not work. But ROW_NUMBER is physical number which assigns the number to the result set after the ORDER BY is evaluated.

Conditions testing for **ROWNUM** values greater than a positive integer are always false. For example, this query returns no rows:

```
SELECT * FROM employees WHERE ROWNUM > 1;
```

The first row fetched is assigned a ROWNUM of 1 and makes the condition false. The second row to be fetched is now the first row and is also assigned a ROWNUM of 1 and makes the condition false.

```
SELECT * FROM (SELECT ROW_NUMBER() OVER (ORDER BY salary) AS row_numb,a.* FROM employees
a)
WHERE row_numb=3;
----
SELECT * FROM (SELECT ROWNUM AS row_numb,a.* FROM employees a ORDER BY salary)
WHERE row_numb=3;
```

COUNT

Total number of elements in a group. Same as aggregate function but better performance by using analytical function.

```
SELECT COUNT(*) OVER (PARTITION BY salary),a.* FROM employees a;
```

LEAD

Next value; Immediate next value will be the default. But can choose which position we want to get.

LAG

Previous value; Immediate previous value will be the default. But can choose which position we want to get.

```
SELECT LAG(salary,2,2) OVER (PARTITION BY department_id ORDER BY salary ) AS lag, --
LAG(column_name,value_from_which_position,if_null); Ex:2nd previous value,instead_of_NULL,show
2
 LEAD(salary,2,1) OVER (PARTITION BY department_id ORDER BY salary ) AS lead,a.* FROM
employees a;
 --LEAD(column_name,value_from_which_position,if_null); Ex:2nd next value,instead_of_NULL,show
1
```

FIRST_VALUE

Gives first value of group

LAST_VALUE

Gives last value of group

```
SELECT FIRST_VALUE(salary) OVER (PARTITION BY department_id ORDER BY salary ) AS FIRST_VALUE,
 LAST_VALUE(salary) OVER (PARTITION BY department_id ORDER BY salary ) AS LAST_VALUE,a.* FROM
employees a; --LAST_VALUE till now; Not for the entire fetch
```

FIRST

Lowest value from a ordered list. Can use only with DENSE_RANK.

LAST

Highest value from a ordered list. Can use only with DENSE_RANK.

DENSE_RANK will arrange the list of data in the table in sorted manner.
FIRST will return lowest rank value.
LAST will return highest rank value.
Aggregate function will not have any role as FIRST and LAST have priority.

```
SELECT employee_id,department_id,salary,
MIN(salary) KEEP(DENSE_RANK FIRST ORDER BY salary) OVER (PARTITION BY department_id) AS
lowest, --Not allowed to use ORDER BY with PARTITION BY in FIRST and LAST
MAX(salary) KEEP(DENSE_RANK LAST ORDER BY salary) OVER (PARTITION BY department_id) AS highest
FROM employees;
----
SELECT MIN(first_name) KEEP(DENSE_RANK FIRST ORDER BY salary) AS min_ename,MIN(salary) AS
min_sal,
MAX(first_name) KEEP(DENSE_RANK LAST ORDER BY salary) AS max_ename,MAX(salary) AS max_sal
FROM employees;
```

WINDOWING CLAUSE

The windowing_clause controls the window or group of rows within the partition. The windowing_clause has two basic forms.

```
RANGE BETWEEN start_point AND end_point
ROWS BETWEEN start_point AND end_point
```

When using ROWS BETWEEN, you are indicating a specific number of rows relative to the current row.
When you use RANGE BETWEEN, you are referring to a range of values in a specific column relative to the value in the current row. As a result, oracle doesn't know how many rows are included in the range until the ordered set is created.

Start point and End point

UNBOUNDED PRECEDING	= all rows before the current row -> fixed --cumulative
UNBOUNDED FOLLOWING	= all rows after the current row -> fixed --reverse cumulative
x PRECEDING	= x rows before the current row -> relative
x FOLLOWING	= x rows after the current row -> relative
CURRENT ROW	= The window starts or ends at the current row. Can be used as start or end point

```
SELECT SUM(salary) OVER(PARTITION BY department_id ORDER BY salary)
AS sum_sal,department_id,salary FROM employees; -- cumulative sum of rows dept wise
----
SELECT SUM(salary) OVER(PARTITION BY department_id ORDER BY salary
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
AS sum_sal,department_id,salary FROM employees;--sum of previous and next upcoming rows in the
respective dept + the current one
----
SELECT SUM(salary) OVER(PARTITION BY department_id ORDER BY salary
ROWS UNBOUNDED PRECEDING)
AS sum_sal,department_id,salary FROM employees;--sum of previous rows in the respective dept +
the current one; excludes next one
```

```
--Find average salary department-wise with all the details of employees
SELECT AVG(salary) OVER (PARTITION BY dept_id),a.* FROM emp a; -- instead of GROUP BY -
PARTITION BY

--Find employee details who joined recently department-wise
SELECT * FROM (SELECT LEAD(hire_date) OVER (PARTITION BY dept_id ORDER BY hire_date) as
recent_joinee,a.*
FROM emp a)
WHERE recent_joinee IS NULL;
--PARTITION BY = First it will execute for a particular department and the next after that. All
the execution will be independent of other departments.
--LEAD = allows to access next row based on order clause. Ex: query took dept_id '90' and
ordered by hire_date. So in each row, we will get hire_date of next employee. If there is no
hire_date for next employee, then that employee will be the recent one.

--cumilative salary department-wise
SELECT SUM(salary) OVER (PARTITION BY dept_id ORDER BY emp_id) AS cum_salary,a.* FROM emp a;

--cumilative salary for entire organization
SELECT SUM(salary) OVER (ORDER BY emp_id) AS cum_salary,a.* FROM emp a;

--sum of salary with employee details department-wise
SELECT SUM(salary) OVER (PARTITION BY dept_id) AS cum_salary,a.* FROM emp a;

--sum of salary for entire organization
SELECT SUM(salary) OVER () AS cum_salary,a.* FROM emp a;

--To get the average of two employess who are joined before that employee and in same
department
SELECT AVG(salary) OVER (PARTITION BY dept_id ORDER BY hire_date
ROWS 2 PRECEDING),a.* FROM emp a;
--Windowing will restrict the number rows taken by partition by
--ROWS 2 means it will take 3 consecutive rows and calculate the average. Next time it will
take next 3 consecutive rows and calculate the average. If ROWS 3, then it will take 4
consecutive rows and take the average
```

Do these programs

```
--select the details of employees with minimum salary on their designation
--select the oldest employees from each department. Add an additional column and give them a
10% bonus
--display all employee details along with the count of reportees their manager has. Ex: neena
has manager steven and steven has 14 reportees. show neen's details along with 14
--details of all employees along with an additional column which shows average of salary
partitioned by designation but only for employees that were hired before 12 years
```

```
----
SELECT SUM(salary) OVER(PARTITION BY department_id ORDER BY salary
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
AS sum_sal,department_id,salary FROM employees;--sum of next upcoming rows in the
respective dept + the current one; excludes previous one
----
SELECT SUM(salary) OVER(PARTITION BY department_id ORDER BY salary
ROWS BETWEEN CURRENT ROW AND 3 PRECEDING)
AS sum_sal,department_id,salary FROM employees; --sum of previous 3 rows in the respective
dept + the current one; excludes next one
----
SELECT SUM(salary) OVER(PARTITION BY department_id ORDER BY salary
ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING)
AS sum_sal,department_id,salary FROM employees;--sum of next 3 rows in the respective dept
+ the current one; excludes previous one
```

Oracle MERGE statement

The Oracle MERGE statement selects data from one or more source tables and updates or inserts it into a target table. The MERGE statement becomes convenient when you want to combine multiple INSERT , UPDATE ,and DELETE statements in a single operation. This statement is a convenient way to combine multiple operations. It lets you avoid multiple INSERT, UPDATE, and DELETE DML statements. **MERGE is a deterministic statement. That is, you cannot update the same row of the target table multiple times in the same MERGE statement. The source can be a table, view, or the result of a subquery.**

```
MERGE INTO target_table
USING source_table
ON search_condition
WHEN MATCHED THEN
UPDATE SET col1 = value1, col2 = value2,...
WHERE <update_condition>
[DELETE WHERE <delete_condition>]
WHEN NOT MATCHED THEN
INSERT (col1,col2,...)
values(value1,value2,...)
WHERE <insert_condition>;
```

If the result is true, then Oracle updates the row with the corresponding data from the source table. In case the result is false for any rows, then Oracle inserts the corresponding row from the source table into the target table.

```
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);
```

Dictionary

DICTIONARY is a view that contains the names of all the data dictionary views that the user can access.

```
SELECT * FROM DICTIONARY
```

TABLE_NAME	COMMENTS
CDB_EXPORT_OBJECTS	
CDB_MINING_MODEL_PARTITIONS	
CDB_SSCR_RESTORE	
CDB_XSTREAM_STMTS	

USER_CONSTRAINTS

```
SELECT constraint_name, constraint_type, search_condition, r_constraint_name,
delete_rule, status
FROM user_constraints
WHERE table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
EMP_JOB_NN	C	"JOB_ID" IS NOT NULL			ENABLED
EMP_SALARY_MIN	C	salary > 0			ENABLED
EMP_EMAIL_UK	U				ENABLED
EMP_EMP_ID_PK	P				ENABLED
EMP_DEPT_FK	R		DEPT_ID_PK	NO ACTION	ENABLED
EMP_JOB_FK	R		JOB_ID_PK	NO ACTION	ENABLED

Pseudo columns XMLTABLE, XMLTYPE, COLUMN_VALUE

```
SELECT * FROM XMLTABLE('<a>123</a>');

SELECT COLUMN_VALUE FROM (XMLTable('<a>123</a>'));
```

COLUMN_VALUE
<a>123</a>

```
CREATE TYPE phone AS TABLE OF NUMBER;
CREATE TYPE phone_list AS TABLE OF phone;

SELECT t.COLUMN_VALUE from table(phone(1,2,3)) t;
```

COLUMN_VALUE
1
2
3

```
CREATE TABLE xwarehouses OF XMLTYPE;

INSERT INTO xwarehouses VALUES
  (xmltype('<?xml version="1.0"?>
  <Warehouse>
    <WarehouseId>1</WarehouseId>
    <WarehouseName>Southlake, Texas</WarehouseName>
    <Building>Owned</Building>
    <Area>25000</Area>
    <Docks>2</Docks>
    <DockType>Rear load</DockType>
    <WaterAccess>true</WaterAccess>
    <RailAccess>N</RailAccess>
    <Parking>Street</Parking>
    <VClearance>10</VClearance>
  </Warehouse>'));

SELECT e.getClobVal() FROM xwarehouses e; --Stored as CLOB
```

```
E.GETCLOBVAL()
<?xml version="1.0"?>
<Warehouse>
  <WarehouseId>1</WarehouseId>
  <WarehouseName>Southlake, Texas</WarehouseName>
  <Building>Owned</Building>
  <Area>25000</Area>
  <Docks>2</Docks>
  <DockType>Rear load</DockType>
  <WaterAccess>true</WaterAccess>
  <RailAccess>N</RailAccess>
  <Parking>Street</Parking>
  <VClearance>10</VClearance>
</Warehouse>
```

Logical Delete

On large tables the process of physically removing a column can be very time and resource consuming. For this reason you may decide to logically delete it.

```
ALTER TABLE table_name SET UNUSED (column_name);
ALTER TABLE table_name SET UNUSED (column_name1, column_name2);
```

Any constrains defined on the column would be removed by the above command, Views become invalid and Indexes on the column will be deleted. Once this is done the columns will no longer be visible to the user. During the off-peak hours, you can drop the unused columns physically using the following statement:

```
ALTER TABLE table_name DROP UNUSED COLUMNS;
```

On large tables you can reduce the amount of undo logs accumulated by using the CHECKPOINT option which forces a checkpoint after the specified number of rows have been processed.

```
ALTER TABLE table_name DROP UNUSED COLUMNS CHECKPOINT 250;
```

The DBA_UNUSED_COL_TABS view can be used to view the number of unused columns per table.

Wrapped

- Wrapping is the process of hiding PL/SQL source code.
- Wrapping helps to protect your source code from business competitors and others who might misuse it.
- Wrapped source files can be moved, backed up, and processed by SQL*Plus and the Import and Export utilities, but they are not visible through the static data dictionary views *_SOURCE.
- You cannot edit PL/SQL source code inside wrapped files. Either wrap your code after it is ready to ship to users or include the wrapping operation as part of your build environment.
- To change wrapped PL/SQL code, edit the original source file and then wrap it again.
- Wrapping is not a secure method for hiding passwords or table names.
- To hide the workings of a trigger, write a one-line trigger that invokes a wrapped subprogram.
- Wrapping does not detect syntax or semantic errors.
- Wrapped PL/SQL units are not downward-compatible.

```
CMD> wrap iname="wrap_test.sql" --wrap_test.sql is the file name of a procedure (no
space between the equal sign)
PL/SQL Wrapper: Release 18.0.0.0.0 - Production on Thu Aug 1 16:26:09 2019
Version 18.3.0.0.0
Copyright (c) 1982, 2018, Oracle and/or its affiliates.  All rights reserved.
Processing wrap_test.sql to wrap_test.plb --output file
SQL> @wrap_test.plb
SQL> call wraptest();
----
DECLARE
    PACKAGE_TEXT VARCHAR2(32767) := 'CREATE PACKAGE emp_actions AS
      PROCEDURE raise_salary (emp_id NUMBER, amount NUMBER);
      PROCEDURE fire_employee (emp_id NUMBER);
      END emp_actions;';
BEGIN
DBMS_DDL.CREATE_WRAPPED(PACKAGE_TEXT);
END;
```

High Water Mark

- This is a term used with table segments stored in the database.
- If you envision a table, for example, as a 'flat' structure or as a series of blocks laid one after the other in a line from left to right, the high-water mark (HWM) would be the rightmost block that ever contained data.
- HWM starts at the first block of a newly created table. As data is placed into the table over time and more blocks get used, the HWM rises. If we delete some (or even all) of the rows in the table, we might have many blocks that no longer contain data, but they are still under the HWM, and they will remain under the HWM until the object is rebuilt, truncated, or shrunk.

Interview (5) Questions and Answers

? What will be the result of different joins in this table

tbl_join1
1
1
2
3
2
NULL
5
5

tbl_join2
2
2
1
1
1
1
3
4
5
NULL
NULL

INNER JOIN / NATURAL JOIN
2
2
2
2
1
1
1
1
1
1
1
1
3
5
5

LEFT OUTER JOIN
2
2
2
2
1
1
1
1
1
1
1
1
3
5
5
NULL

RIGHT OUTER JOIN
1
1
1
1
1
1
1
1
2
2
3
2
2
5
5
NULL
NULL
4

FULL OUTER JOIN
2
2
2
2
1
1
1
1
1
1
1
1
3
4
5
5
NULL
NULL

CROSS JOIN	
1	2
1	2
1	1
1	1
1	1
1	1
1	3
1	4
1	5
1	NULL
1	NULL
1	2
1	2
1	1
1	1
1	1
1	1
1	3
1	4
1	5
1	NULL
1	NULL
2	2
2	2
2	1
2	1
2	1
2	1
2	3
2	4
2	5
2	NULL
2	NULL
3	2
3	2
3	1
3	1
3	1
3	1
3	3
3	4
3	5
3	NULL
3	NULL
2	2
2	2
2	1
2	1
2	1
2	1
2	3
2	4
2	5
2	NULL
2	NULL
NULL	2
NULL	2
NULL	1
NULL	1
NULL	1
NULL	1
NULL	3
NULL	4
NULL	5
NULL	NULL
NULL	NULL
5	2
5	2
5	1
5	1
5	1
5	1
5	3
5	4
5	5
5	NULL
5	NULL
5	2
5	2
5	1
5	1
5	1
5	1
5	3
5	4
5	5
5	NULL
5	NULL

? What will be the result of this query?

```
SELECT 1 FROM DUAL
UNION ALL
SELECT 'A' FROM DUAL; --
ORA-01790: expression must have same datatype as corresponding expression
```

1 and 'A' are different datatypes. Hence cannot be in same column

? Can we call procedure inside a function in PL/SQL?

Yes

? Join two tables and fetch the sum of values group by id in two columns

Flowers	
id	amount
1	40
2	110
3	110
4	20
1	140
2	100
3	30
4	130

Pots	
id	amount
1	70
2	140
3	170
1	140
2	110
3	50

Result		
id	amount1	amount2
1	180	210
2	210	250
4	150	0
3	140	220

```
SELECT
  ff.id,
  NVL((SELECT SUM(f.amount) FROM flowers f WHERE f.id = ff.id),0)
amount1, NVL((SELECT SUM(p.amount) FROM pots p WHERE p.id = ff.id),0)
Amount2 FROM flowers ff FULL OUTER JOIN pots pp ON (ff.id =pp.id)
GROUP BY ff.id
```

? Update a table with data from another table

```
UPDATE temp_emp tmp
   SET (first_name,salary) = (SELECT emp.first_name, emp.salary
FROM employees emp WHERE tmp.employee_id = emp.employee_id)
----
MERGE INTO temp_emp tmp
USING
(SELECT * FROM employees) emp
ON(tmp.employee_id = emp.employee_id)
WHEN MATCHED THEN UPDATE SET
tmp.first_name = emp.first_name,
tmp.salary = emp.salary;
```

? Insert into a table from another table except its first row

```
INSERT INTO temp_emp (SELECT * FROM employees WHERE ROWID !=
    (SELECT ROWID FROM employees WHERE ROWNUM = 1));
----
INSERT INTO temp_emp (SELECT employee_id, first_name, last_name, email,
phone_number, hire_date, job_id, salary, commission_pct, manager_id,
department_id FROM (SELECT e.*,ROW_NUMBER() OVER (ORDER BY ROWID)rn
FROM employees e)
    WHERE rn!=1);
----
INSERT INTO temp_emp (SELECT * FROM (SELECT * FROM
    (SELECT e.*,ROW_NUMBER() OVER (ORDER BY ROWID)rn FROM employees e)
    WHERE rn!=1)
PIVOT(
  MAX(1) -- fake
  FOR (rn) -- put the undesired columns here
  IN () -- no values here...
));
```

? What is Partition Pruning?

Partition pruning is an essential performance feature for data warehouses. In partition pruning, the optimizer analyzes FROM and WHERE clauses in SQL statements to eliminate unneeded partitions when building the partition access list. This functionality enables Oracle Database to perform operations only on those partitions that are relevant to the SQL statement.

? Write a SQL query to display table name, count of rows and columns, in the current schema?

```
SELECT A.*, (SELECT NUM_ROWS FROM ALL_TABLES WHERE TABLE_NAME = A.TABLE_NAME)
"ROWS" FROM
(SELECT ALL_TAB.TABLE_NAME, COUNT(1) "COLUMNS"
FROM ALL_TABLES ALL_TAB, ALL_TAB_COLUMNS ALL_COL
WHERE ALL_TAB.TABLE_NAME = ALL_COL.TABLE_NAME
AND ALL_TAB.OWNER = 'HR'
GROUP BY ALL_TAB.TABLE_NAME) A
```

Deferred Constraint

- During large transactions involving multiple dependencies it is often difficult to process data efficiently due to the restrictions imposed by the constraints. An example of this would be the update of a primary key (PK) which is referenced by foreign keys (FK). The primary key columns cannot be updated as this would orphan the dependant tables, and the dependant tables cannot be updated prior to the parent table as this would also make them orphans. Traditionally this problem was solved by disabling the foreign key constraints or deleting the original records and recreating them. Since neither of these solutions is particularly satisfactory Oracle 8i includes support for deferred constraints. A deferred constraint is only checked at the point the transaction is committed.
- By default constraints are created as NON DEFERRABLE but this can be overridden using the DEFERRABLE keyword.
- A deferred constraint is one that is enforced when a transaction is committed.
- A deferrable constraint is specified by using DEFERRABLE clause.
- Once you've added a constraint, you cannot change it to DEFERRABLE. You must drop and recreate the constraint.
- When you add a DEFERRABLE constraint, you can mark it as INITIALLY IMMEDIATE or INITIALLY DEFERRED.
- INITIALLY IMMEDIATE means that the constraint is checked whenever you add, update, or delete rows from a table.
- INITIALLY DEFERRED means that the constraint is only checked when a transaction is committed.

```
ALTER TABLE cust
ADD CONSTRAINT cust_id_pk
PRIMARY KEY(cust_id) DEFERRABLE INITIALLY DEFERRED;

ALTER SESSION SET CONSTRAINTS = DEFERRED;
ALTER SESSION SET CONSTRAINTS = IMMEDIATE;
```

- The ALTER SESSION... statements show how the state of the constraint can be changed. These ALTER SESSION... statements will not work for constraints that are created as NOT DEFERRABLE.

Constraint States

- Table constraints can be enabled and disabled using the CREATE TABLE or ALTER TABLE statement. In addition the VALIDATE or NOVALIDATE keywords can be used to alter the action of the state.
- ENABLE VALIDATE is the same as ENABLE. The constraint is checked and is guaranteed to hold for all rows.
- ENABLE NOVALIDATE means the constraint is checked for new or modified rows, but existing data may violate the constraint.
- DISABLE NOVALIDATE is the same as DISABLE. The constraint is not checked so data may violate the constraint.
- DISABLE VALIDATE means the constraint is not checked but disallows any modification of the constrained columns.

```
ALTER TABLE tab1 ADD CONSTRAINT fk_tab1_tab2
  FOREIGN KEY (tab2_id)
  REFERENCES tab2 (id)
  ENABLE NOVALIDATE;

ALTER TABLE tab1 MODIFY CONSTRAINTS fk_tab1_tab2 ENABLE VALIDATE;
```

Issues

- Exception handling has to be coded carefully as statements will not trigger exceptions directly. Often exceptions will only be picked up by the outermost exception handler which encloses the commit statement.
- Converting a NOVALIDATE constraint to VALIDATE may take a long time depending on the amount of data to be validated, although conversion in the other direction is not an issue.
- Enabling a unique or primary key constraint when no index is present causes the creation of a unique index. Likewise, disabling a unique or primary key will drop a unique index that it used to enforce it.

ROLLUP, CUBE, GROUPING Functions and GROUPING SETS

ROLLUP

The ROLLUP is an extension of the GROUP BY clause. The ROLLUP extension calculates multiple levels of subtotals across a group of columns along with the grand total.

- First, calculates the standard aggregate values in the GROUP BY clause.
- Then, progressively creates higher-level subtotals of the grouping columns, which are col2 and col1 columns, from right to left.
- Finally, creates a grand total.

If you have n columns listed in the ROLLUP, you will get n+ 1 level of subtotals with ROLLUP.

```
SELECT DEPARTMENT_ID,SUM(SALARY) FROM EMPLOYEES GROUP BY ROLLUP (DEPARTMENT_ID); --
n=1
```

DEPARTMENT_ID	SUM(SALARY)
10	4400 --level 1
20	19000 --level 1
	23400 --level 2

```
SELECT DEPARTMENT_ID, JOB_ID, SUM(SALARY) FROM EMPLOYEES GROUP BY ROLLUP
(DEPARTMENT_ID, JOB_ID); --n=2
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400 --level 1
10		4400 --level 2
20	MK_MAN	13000 --level 1
20	MK_REP	6000 --level 1
20		19000 --level 2
		23400 --level 3

```
SELECT DEPARTMENT_ID, JOB_ID, SUM(SALARY) FROM EMPLOYEES GROUP BY DEPARTMENT_ID,
ROLLUP(JOB_ID); --partial rollup
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
10		4400
20	MK_MAN	13000
20	MK_REP	6000
20		19000

CUBE

Cube gives 2 to the power n levels of subtotals.

```
SELECT DEPARTMENT_ID,SUM(SALARY) FROM EMPLOYEES WHERE DEPARTMENT_ID IN (10,20) GROUP
BY CUBE (DEPARTMENT_ID); --n=1; 2^1 = 2
```

DEPARTMENT_ID	SUM(SALARY)
	23400 --level 2
10	4400 --level 1
20	19000 --level 1

```
SELECT DEPARTMENT_ID, JOB_ID, SUM(SALARY) FROM EMPLOYEES WHERE DEPARTMENT_ID IN
(10,20) GROUP BY CUBE (DEPARTMENT_ID, JOB_ID); --2^2 = 4
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
		23400 --level 4
	MK_MAN	13000 --level 3
	MK_REP	6000 --level 3
	AD_ASST	4400 --level 3
10		4400 --level 2
10	AD_ASST	4400 --level 1
20		19000 --level 2
20	MK_MAN	13000 --level 1
20	MK_REP	6000 --level 1

```
SELECT DEPARTMENT_ID, JOB_ID, SUM(SALARY) FROM EMPLOYEES WHERE DEPARTMENT_ID IN
(10,20) GROUP BY DEPARTMENT_ID, CUBE(JOB_ID);
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10		4400
10	AD_ASST	4400
20		19000
20	MK_MAN	13000
20	MK_REP	6000

GROUPING

The Grouping function tells us if NULLs are caused by CUBE or ROLLUP

```
SELECT DEPARTMENT_ID, JOB_ID, SUM(SALARY), GROUPING(DEPARTMENT_ID) as DEPT,
GROUPING(JOB_ID) as JOBID FROM EMPLOYEES WHERE DEPARTMENT_ID IN (10,20) GROUP BY
CUBE (DEPARTMENT_ID, JOB_ID);
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)	DEPT	JOBID
		23400	1 --subtotal by dept	1 --subtotal by jobid
	MK_MAN	13000	1 --subtotal by dept	0
	MK_REP	6000	1 --subtotal by dept	0
	AD_ASST	4400	1 --subtotal by dept	0
10		4400	0	1 --subtotal by jobid
10	AD_ASST	4400	0	0
20		19000	0	1 --subtotal by jobid
20	MK_MAN	13000	0	0
20	MK_REP	6000	0	0

1 means column contains NULL because of the subtotal.
0 means regular value

Plan Stability

- Plan stability relies on preserving execution plans at a point in time when performance is satisfactory. In many environments, however, attributes for data types such as dates or order numbers can change rapidly. In this cases, permanent use of an execution plan can result in performance degradation over time as the data characteristics change.
- Plan stability preserves execution plans in stored outlines. An outline is implemented as a set of optimizer hints that are associated with the SQL statement. If the use of the outline is enabled for the statement, then Oracle Database automatically considers the stored hints and tries to generate an execution plan in accordance with those hints.
- Oracle Database can create a public or private stored outline for one or all SQL statements. The optimizer then generates equivalent execution plans from the outlines when you enable the use of stored outlines.

Modes of Database Shutdown

SHUTDOWN IMMEDIATE: Terminates any executing SQL, uncommitted changes are rolled back and disconnects the users; performs a check point then close the online datafiles
SHUTDOWN TRANSACTIONAL: Prevents users from starting new transactions but wait till all current transactions to complete before shutting down; performs a check point then close the online datafiles
SHUTDOWN NORMAL: Waits for all connected users to disconnect before shutting down; performs a check point then close the online datafiles
SHUTDOWN ABORT: Closes the datafiles without checkpoint, Instance recovery is required in the next startup.

? There are 3 switches in one room. One of the switch is for the bulb in next room. Find the correct switch with minimal travel to the next room.

Turn the 1st switch and wait for 5 minutes. Turn it off. Turn on the next switch. Go to next room. If the light is on, then the second switch is the correct one. If not, touch the bulb and see whether it is hot. If it is hot, then 1st switch is the correct one. If it is not hot, then 3rd switch is the correct one.

? There is a well with height of 30 feet. A snail inside the well is trying to get out of it. It travels 3 feet up and 2 feet down in 1 hour. In how many hours, the snail can reach at the top?

28 hours. In 27 hours, it reach 27 feet as the snail is moving only 1 feet per hour. In the next hour, it travels the next 3 feet up and it reached the top. No need to come back 2 feet.

? Fetch multiple phone numbers of same customer in same row

```
SELECT employee_id,
    RTRIM (XMLAGG (XMLELEMENT (e, phone_number || ',')).EXTRACT ('//text()'), ',') phone_number
    FROM temp_emp GROUP BY employee_id;
----
SELECT employee_id,
   LISTAGG (phone_number, ',')
WITHIN GROUP (ORDER BY phone_number) phone_number
FROM temp_emp GROUP BY employee_id;
```

? Using translate or replace, carriage return with space

```
SQL> SELECT '1st Line'||CHR(10)||CHR(13)||'2nd Line' FROM DUAL;
'1STLINE'||CHR(10)  -CHR(13) Carriage return; CHR(10) Next line = /r/n
------------------
1st Line
 2nd Line

SQL> SELECT TRANSLATE ('1st Line'||CHR(10)||CHR(13)||'2nd Line', CHR(10)||CHR(13), ' ')
FROM DUAL;
TRANSLATE('1STLIN
-----------------
1st Line 2nd Line

SQL> SELECT REPLACE( '1st Line'||CHR(10)||CHR(13)||'2nd Line', CHR(10) || CHR(13) ) FROM
DUAL;
REPLACE('1STLINE
----------------
1st Line2nd Line
```

? Update salary by 1.5 times and delete details of those employees who are in department 30 and 50 using an efficient way.

```
BEGIN
    UPDATE temp_test SET salary = salary * 1.5 WHERE department_id in (30,50);
    DELETE FROM temp_test WHERE department_id in (30,50);
END;
```

? Write a SQL query to swap two columns

Create a temporary column to swap data.

? Find number of NULL values in each column

a	b	c
1	NULL	2
1	1	3
NULL	1	NULL
0	2	NULL
NULL	3	NULL

```
SELECT COUNT(1)-COUNT(A),COUNT(1)-COUNT(B),COUNT(1)-COUNT(C) FROM NULL_ABC;
--Count(column_name) doesn't count NULL values
```

? What in local and global Index Partitioning?

A partitioned index is simply an index broken into multiple pieces. Partitioning can work several different ways. The tables can be partitioned and the indexes are not partitioned; the table is not partitioned but the index is; or both the table and index are partitioned. There are two types of partitioned indexes: local and global.
Local: Local indexes are indexes that are partitioned using the same partition key and same range boundaries as the partitioned table. Each partition of a local index will only contain keys and ROWIDs from its corresponding table partition. Local indexes can be b-tree or bitmap indexes.
Global: Global partitioned indexes contain keys from multiple table partitions in a single index partition. The partitioning key of a global partitioned index is different or specifies a different range of values from the partitioned table. The creator of the global partitioned index is responsible for defining the ranges and values for the partitioning key. Global indexes can only be b-tree indexes and only be range partitioned.
If the application is an OLTP one and users need quick response times, use a global index. If the application is a DSS one and users are more interested in throughput, use a local index.

```
CREATE INDEX year_idx ON all_fact (order_date) LOCAL
    (PARTITION name_idx1),
    (PARTITION name_idx2),
    (PARTITION name_idx3);
----
CREATE INDEX item_idx ON all_fact (item_nbr) GLOBAL
    (PARTITION city_idx1 VALUES LESS THAN (100)),
    (PARTITION city_idx1 VALUES LESS THAN (200)),
    (PARTITION city_idx1 VALUES LESS THAN (300));
```

? What is OLTP, DSS and OLAP?

Online Transaction Processing	Decision Support System
It is designed to support operational process	It is designed to support decision making process
Data is volatile	Data is non-volatile.
Data is in inconsistency form.	It is in consistent form.
It stores recent data for approximately 4 to 6 months data.	It stores One year data.
It follows normalized schema.	It follows star schema.
Online Transaction Processing	**Online Analytical Processing**
It is dynamic.	It is static [unchanged].
It follows normalization.	It follows denormalization.
It contains current data.	It contains historical data.
It is designed to support transactional process	It is designed to support decision making process
It contains detailed data.	It contains summarized information.

DBMS Packages

DBMS_SQL

The DBMS_SQL package provides an interface to use dynamic SQL to parse any data manipulation language (DML) or data definition language (DDL) statement using PL/SQL. Native Dynamic SQL is an alternative to DBMS_SQL that lets you place dynamic SQL statements directly into PL/SQL blocks. In most situations, Native Dynamic SQL is easier to use and performs better than DBMS_SQL. However, Native Dynamic SQL itself has certain limitations

```
DBMS_SQL.NATIVE --Specifies normal behaviour for the database to which the
program is connected. Can define behaviour as in Oracle version 6 and 7
DBMS_SQL.OPEN_CURSOR
DBMS_SQL.PARSE --Parsing the statement checks the statement's syntax and
associates it with the cursor in your program. You can parse any DML or DDL
statement. DDL statements are run on the parse, which performs the implied
commit.
DBMS_SQL.BIND_VARIABLE
DBMS_SQL.DEFINE_COLUMN --The columns of the row being selected in a SELECT
statement are identified by their relative positions as they appear in the
select list, from left to right
DBMS_SQL.EXECUTE --Call the EXECUTE function to run your SQL statement.
DBMS_SQL.FETCH_ROWS --The FETCH_ROWS function retrieves the rows that satisfy
the query.
DBMS_SQL.COLUMN_VALUE --call COLUMN_VALUE after fetching rows to actually
retrieve the values of the columns in the rows into your program
DBMS_SQL.VARIABLE_VALUE -- call VARIABLE_VALUE to retrieve the value of an OUT
parameter for an anonymous block
DBMS_SQL.CLOSE_CURSOR
```

DBMS_JOB

The DBMS_JOB package schedules and manages jobs in the job queue. COMMIT is needed after the statement.

```
DBMS_JOB.REMOVE(job_id) --Removes specified job from the job queue
DBMS_JOB.SUBMIT(job_num, procedure_name, next_date, interval) --job_num is out
parameter; Submits a new job to the job queue
```

Run Statspack Snapshot Every 1 Hour

```
VARIABLE jobno NUMBER;
VARIABLE instno NUMBER;
BEGIN
SELECT instance_number INTO :instno FROM v$instance;
DBMS_JOB.SUBMIT(:jobno, 'statspack.snap;', TRUNC(SYSDATE+1/24,'HH'),
'TRUNC(SYSDATE+1/24,''HH'')', TRUE, :instno);
COMMIT;
END;
/
```

```
DBMS_JOB.RUN(job_id) --Forces a specified job to run
```

DBMS_RANDOM

The DBMS_RANDOM package provides a built-in random number generator. DBMS_RANDOM is not intended for cryptography.

```
DBMS_RANDOM.VALUE(low_value,high_value) --gets a random number with 38 digit
decimal
DBMS_RANDOM.STRING(single_character,length) --This function gets a random string.

DBMS_LOCK.SLEEP(120) --This procedure suspends the session for a given period
of time (seconds).

DBMS_UTILITY.FORMAT_ERROR_BACKTRACE --This procedure displays the call stack at
the point where an exception was raised
DBMS_UTILITY.GET_TIME --This function determines the current time in 100th's of
a second. This subprogram is primarily used for determining elapsed time. The
subprogram is called twice – at the beginning and end of some process – and
then the first (earlier) number is subtracted from the second (later) number
to determine the time elapsed.
DBMS_UTILITY.GET_CPU_TIME -- CPU time

DBMS_OUTPUT.PUT_LINE
DBMS_OUTPUT.PUT_LINE($$PLSQL_LINE); -- Displays line number
DBMS_OUTPUT.DISABLE

DBMS_STATS.GATHER_TABLE_STATS --This procedure gathers table and column (and
index) statistics.
EXEC DBMS_STATS.GATHER_SCHEMA_STATS(USER, CASCADE => TRUE);

DBMS_XPLAN.DISPLAY -- to format and display the contents of a plan table.
DBMS_XPLAN.DISPLAY_AWR -- to format and display the contents of the execution
plan of a stored SQL statement in the AWR.
DBMS_XPLAN.DISPLAY_CURSOR -- to format and display the contents of the execution
plan of any loaded cursor.

DBMS_METADATA.GET_DDL
SELECT DBMS_METADATA.GET_DDL ('TABLE', 'EMPLOYEES', 'HR') FROM DUAL;
-- to get DDL for a view just replace first argument with 'VIEW' and second with your
view name and so.

DBMS_REFRESH.ADD -- Adds materialized views to a refresh group.
DBMS_REFRESH.MAKE -- To make materialized view refresh group
DBMS_REFRESH.CHANGE -- Changes the refresh interval for a refresh group.
DBMS_REFRESH.DESTROY -- Removes all of the materialized views from a refresh group
and deletes the refresh group.
DBMS_REFRESH.REFRESH -- Manually refreshes a refresh group.
DBMS_REFRESH.SUBTRACT -- Removes materialized views from a refresh group.

DBMS_MVIEW.REFRESH -- Refreshes one or more materialized views that are not members
of the same refresh group
DBMS_MVIEW.REFRESH_ALL_MVIEWS --Refreshes all materialized views
```

Table Clusters

- A table cluster is a group of tables that share common columns and store related data in the same blocks.
- When tables are clustered, a single data block can contain rows from multiple tables. For example, a block can store rows from both the employees and departments tables rather than from only a single table.
- The cluster key is the column or columns that the clustered tables have in common. For example, the employees and departments tables share the department_id column. You specify the cluster key when creating the table cluster and when creating every table added to the table cluster.
- The cluster key value is the value of the cluster key columns for a particular set of rows. All data that contains the same cluster key value, such as department_id=20, is physically stored together.
- Consider clustering tables when they are primarily queried (but not modified) and records from the tables are frequently queried together or joined. This benefits reduced Disk I/O for joins, improves access time for joins.
- Flashback Table operation is not supported on clustered tables

```
/*For example to create a cluster of EMP and DEPT tables in which the DEPTNO will be
cluster key, first create the cluster by typing the following command.*/
CREATE CLUSTER emp_dept (deptno NUMBER(2));
/*Then create index on it.*/
CREATE INDEX idx_empdept ON CLUSTER emp_dept;
/*Now create table in the cluster like this*/
CREATE TABLE dept (deptno NUMBER(2),
                   name VARCHAR2(20),
                   loc VARCHAR2(20))
                   CLUSTER emp_dept (deptno);
CREATE TABLE emp (empno NUMBER(5),
             name VARCHAR2(20),
             sal NUMBER(10,2),
             deptno NUMBER(2))
             CLUSTER emp_dept (deptno);
```

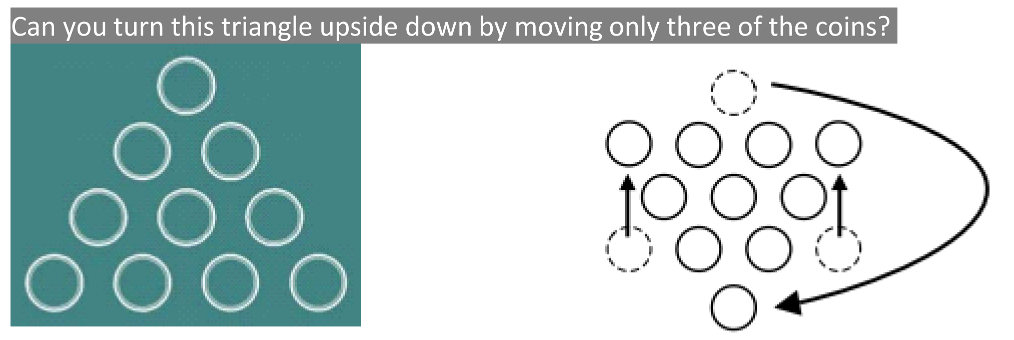
SQL Performance Tuning & Tips

- Ensure the OS has enough I/O bandwidth, CPU power and swap space (Swap space is the portion of virtual memory that is on the hard disk, used when RAM is full)
- Operating systems, provide data caches which will consume memory while offering little or no performance benefit for the database. By default, all database I/O goes through the file system cache. On some Linux and UNIX systems, direct I/O is available which will allow the database files to be accessed by bypassing the file system cache. So it can save CPU resources and memory and allows file system cache to be dedicated to non-database activity.
- But in some cases, database does not use the database buffer cache then the direct I/O may yield worst performance than using OS cache.
- Use **SQL_TRACE, AWR** to find the performance problem.
- Look for wait events.
- Find columns that should be indexed.
- Verify statistics are current (DBMS_STATS)
- Determine whether the SQL queries can be improved
- Use TRUNCATE instead of DELETE if you need to delete whole content of the table
- Use ROWID while deleting duplicate rows
- Use frequent COMMIT while performing batch transactions; so the temporary space will be released
- Use COUNT(column_name) instead of COUNT(*) or COUNT(1)
- Use WHERE clause instead of HAVING clause (Except GROUP function)
- Minimize the lookups in WHERE clause queries
- Use multi column UPDATE while doing updates (`UPDATE employees SET salary = 40000, email='skings' WHERE employee_id = 100;`)
- Use EXISTS instead of IN
- Use NOT EXISTS instead of NOT IN
- Use IN in place of OR in WHERE clause filter
- Avoid IS NULL or IS NOT NULL on indexed column
- Use Oracle hints
- Avoid number-to-character conversions because numbers and characters compare differently and lead to performance downgrade.
- Never use SELECT * in production code. At some point, someone will come and modify the table or view you're querying from. You might be fetching more columns than you actually need and it will create unnecessary overhead in the database.
- Use meaningful aliases, use AS for defining alias for easy reading
- Create your indexes carefully on all the tables where you have frequent search operations. Avoid index on the tables where you have less number of search operations and more number of insert and update operations.
- A full-table scan occurs when the columns in the WHERE clause do not have an index associated with them. You can avoid a full-table scan by creating an index on columns that are used as conditions in the WHERE clause of an SQL statement.
- Use pattern matching judiciously. 'LIKE' queries cause full-table scans
- For queries that are executed on a regular basis, try to use procedures.
- You can optimize bulk data loads by dropping indexes.
- Oracle has many tools for managing SQL statement performance but among them two are very popular. These two tools are –
  - **EXPLAIN PLAN** – tool identifies the access path that will be taken when the SQL statement is executed.
  - **TKPROF** —SQL Trace generates a low level trace file that has a complete chronological record of everything a session is doing and waiting for when it "talks" to the database. TKPROF on the other hand takes that trace file and aggregates all of the low level details in an easy to read report. This report can then be quickly analyzed to find the root cause of the slow performance.
    Ex: `TKPROF input.trc output.prd [options]`

    ```
    ALTER SYSTEM SET TIMED_STATISTICS = TRUE;
    ALTER SESSION SET SQL_TRACE = TRUE;
    SELECT COUNT(*) FROM  DUAL;
    ALTER SESSION SET SQL_TRACE = FALSE;
    SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Default Trace File';
    CMD>TKPROF C:\APP\diag\rdbms\orcl\orcl\trace\orcl_ora_11512.trc E:\orcl_ora_
    11512.txt;
    ```

    - Inefficient statements are mostly associated with a high number of block visits.
    - Both, soft parse and hard parse are counted as parse in tkprof. More specifically, the parse count is incremented when the statement is hashed.
- COUNT (*) and COUNT(1) - No much difference in performance (just fraction of seconds)

? Can you turn this triangle upside down by moving only three of the coins?



Interview (6) Questions and Answers

? Display the result as "Fail" for students who has less than 35 marks in a single subject or overall % are less than 40. For rest, result as "Pass"

Roll No	Subject	Marks
1	Chemistry	98
1	Mathematics	45
1	Physics	55
2	Chemistry	32
2	Mathematics	48
2	Physics	94
3	Chemistry	38
3	Mathematics	35
3	Physics	39

Roll No	Result
1	Pass
2	Fail
3	Fail

```
WITH FAILS AS
(SELECT ROLLNO, 'FAIL' FROM STUDENTS WHERE MARKS <= 35 GROUP BY ROLLNO
UNION
SELECT ROLLNO, 'FAIL' FROM STUDENTS WHERE (MARKS*300/100) <= 40 GROUP BY ROLLNO)

SELECT DISTINCT ROLLNO,'PASS' FROM STUDENTS WHERE ROLLNO NOT IN (SELECT ROLLNO FROM FAILS)
UNION SELECT * FROM FAILS;
```

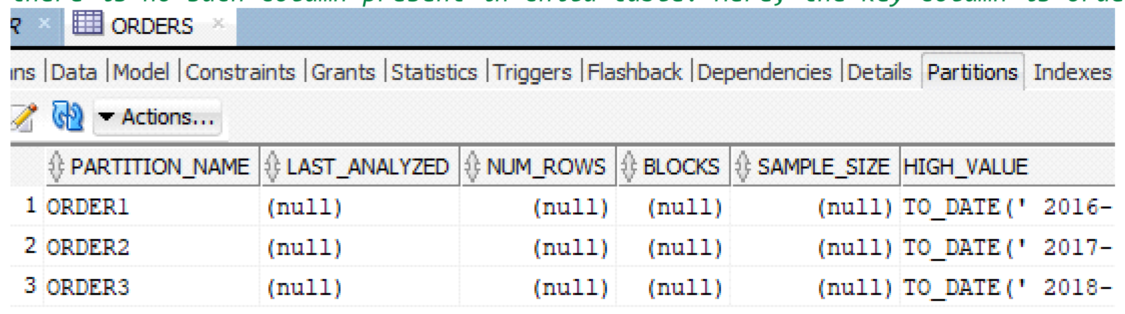
? Can we have index on VIEWS or MVIEWS or Global temporary table?

Creating index in VIEWS is not allowed but can create index on MVIEWS and Global temporary tables

? What is Reference Partitioning?

Reference partitioning enables the partitioning of two tables that are related to one another by referential constraints. The benefit of this extension is that tables with a parent-child relationship can be logically equipartitioned by inheriting the partitioning key from the parent table without duplicating the key columns.

```
CREATE TABLE orders(order_id NUMBER PRIMARY KEY,
    order_date DATE NOT NULL, customer_id NUMBER NOT NULL)
    PARTITION BY RANGE(order_date)
    (PARTITION y1 VALUES LESS THAN(TO_DATE('01-JAN-2016','DD-MON-YYYY')),
     PARTITION y1 VALUES LESS THAN(TO_DATE('01-JAN-2017','DD-MON-YYYY')),
     PARTITION y1 VALUES LESS THAN(TO_DATE('01-JAN-2018','DD-MON-YYYY')));
----
CREATE TABLE order_details(order_id NUMBER NOT NULL,
    product_id NUMBER NOT NULL, quantity NUMBER,
    CONSTRAINT fk_order_details FOREIGN KEY(order_id) REFERENCES orders(order_id))
    PARTITION BY REFERENCE(fk_order_details);
--This will allocate 3 partitions for this child table as same as for the parent table even though
there is no such column present in child table. Here, the key column is order_date
```



ORDERS
ns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes
Actions...

```
   PARTITION_NAME  LAST_ANALYZED  NUM_ROWS  BLOCKS  SAMPLE_SIZE  HIGH_VALUE
 1 ORDER1          (null)           (null)  (null)       (null)  TO_DATE(' 2016-
 2 ORDER2          (null)           (null)  (null)       (null)  TO_DATE(' 2017-
 3 ORDER3          (null)           (null)  (null)       (null)  TO_DATE(' 2018-
```

? What is INTERVAL Partitioning?

If we try to insert a record which is beyond the range specified in range partition, it will throw an error:
`ORA-14400: inserted partition key does not map to any partition`
Interval partitioning is a partitioning method introduced in Oracle 11g. This is a helpful addition to range partitioning where Oracle automatically creates a partition when the inserted value exceeds all other partition ranges.

```
SQL> CREATE TABLE test(sno NUMBER(6), last_name VARCHAR2(30), salary NUMBER(6))
        PARTITION BY RANGE(salary)
        INTERVAL  (5000)
         (PARTITION p1 VALUES LESS THAN (5000),
          PARTITION p2 VALUES LESS THAN (10000),
          PARTITION p3 VALUES LESS THAN (15000),
          PARTITION p4 VALUES LESS THAN (20000));
```

? What are the drawbacks of BULK COLLECT? Why we are using LIMIT clause?

Memory overhead can happen. LIMIT clause is used to limit that memory usage. LIMIT 100 is reasonable.

```
DECLARE
CURSOR EMP_CUR IS SELECT EMPLOYEE_ID FROM EMPLOYEES;
TYPE TAB_EMP IS TABLE OF EMPLOYEES.EMPLOYEE_ID%TYPE;
L_EMPNO TAB_EMP;
BEGIN
OPEN EMP_CUR;
LOOP
    FETCH EMP_CUR
    BULK COLLECT INTO L_EMPNO LIMIT 25;
    DBMS_OUTPUT.PUT_LINE('NEXT ROUND');
    FOR I IN 1..L_EMPNO.COUNT LOOP
      DBMS_OUTPUT.PUT_LINE(L_EMPNO(I));
    END LOOP;
    EXIT WHEN L_EMPNO.COUNT = 0;
END LOOP;

END;
```

PL/SQL Functions

- Functions are a standalone block that is mainly used for calculation purpose.
- Function must return a value which has the data type mentioned in the specification of function.
- Function with no DML statements can be directly called in SELECT query whereas the function with DML operation can only be called from other PL/SQL blocks.
- Stored procedures and functions (subprograms) can be compiled and stored, which will be ready to be executed
- In general, stored procedures and functions should not commit. Those sorts of transaction control decisions should be left to higher-level code that knows when a logical transaction is actually complete. If you commit inside of a stored procedure, you are limiting its reusability because a caller that wants the changes the procedure makes to be part of a larger transaction cannot simply call the procedure directly.

```
CREATE [OR REPLACE] FUNCTION function_name (parameter1 datatype, parameter2
datatype,..)  --size [VARCHAR2(10)] is not defined;
RETURN datatype IS  --size is not defined
    --declare variables;
BEGIN
    --executable statements;
    RETURN (return_value);
END;
```

```
CREATE OR REPLACE FUNCTION totalEmployees(dept_no IN NUMBER)
RETURN NUMBER IS
    total NUMBER(2) := 0;
BEGIN
    SELECT COUNT(*) INTO total FROM employees WHERE DEPARTMENT_ID = dept_no;
    RETURN total;
END;
/*
Calling Program
*/
DECLARE
    c NUMBER(2);
BEGIN
    c := totalEmployees(60);
    DBMS_OUTPUT.PUT_LINE(c);
END;
```

Actual vs. Formal Parameter

Formal - Parameter defined in the procedure or function.
Actual - Parameter provided by the calling statement to a procedure or function.

PL/SQL Stored Procedures

- Procedures are standalone blocks of a program that can be stored in the database.
- Procedure can have a RETURN statement to return the control to the calling block, but it cannot return any values through the RETURN statement.
- Procedures cannot be called directly from SELECT statements. They can be called from another block or through EXEC keyword.

```
CREATE [OR REPLACE] PROCEDURE procedure_name (parameter1 [IN|OUT|IN OUT] datatype,
parameter2..) IS
    --declare variables;
BEGIN
    --executable statements;
END [procedure_name];
```

```
CREATE OR REPLACE PROCEDURE greetings(gname IN VARCHAR2, greet OUT VARCHAR2, gloc IN OUT
VARCHAR2) IS
BEGIN
    greet := 'Hello ' || gname;
    gloc := ' from ' || gloc;
END greetings;
/*
Calling Program
*/
DECLARE
  greeting VARCHAR2(20);
  gloc VARCHAR2(20);
BEGIN
  gloc:='NYC';
  greetings('Richard',greeting,gloc);
  DBMS_OUTPUT.PUT_LINE(greeting || gloc);
END;
```

Calling Notations

Positional - The datatype and the position if the actual parameter must match with the formal parameter.
Named - The actual parameter is matched with the formal parameter using **=>** symbol
Mixed - Combination of Positional and Named notations.

```
CREATE OR REPLACE PROCEDURE greetings(name IN VARCHAR2 ,age IN NUMBER ,place IN VARCHAR2)
IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello:' || name);
    DBMS_OUTPUT.PUT_LINE('Age:' || age);
    DBMS_OUTPUT.PUT_LINE('Place:' || place);
END greetings;
/*
Calling Program
*/
EXECUTE greetings('Mohan', 26 ,'Mumbai'); --Positional
----
EXECUTE greetings(place=>'Mumbai',name=>'Mohan',age=>26); -- Named
----
EXECUTE greetings('Mohan',place=>'Mumbai',age=>26); -- Mixed - Positioned notations are not
allowed after Named notations
```

Parameter Modes

IN: [default mode] An IN parameter lets you pass a value to the subprogram. It is a read-only parameter. Inside the subprogram, an IN parameter acts like a constant. Its value cannot be changed. IN can have default value; but others cannot. In can pass expressions like (10+5) or constant. But other must be a variable. Parameters are passed by reference.
OUT: An OUT parameter returns a value to the calling program. Its value can be changed. Parameters are passed by value.
INOUT: An INOUT parameter passes an initial value to a subprogram and returns an updated value to the caller. Its value can be changed. Parameters are passed by value.

Procedure Vs. Function

Procedure	Function
Used mainly to a execute certain process	Used mainly to perform some calculation
Cannot call in SELECT statement	A Function that contains no DML statements can be called in SELECT statement
Use OUT parameter to return the value	Use RETURN to return the value
It is not mandatory to return the value	It is mandatory to return the value
RETURN will simply exit the control from subprogram.	RETURN will exit the control from subprogram and also returns the value
Return datatype will not be specified at the time of creation	Return datatype is mandatory at the time of creation
Inputs are used to *do* something, rather than *derive* something.	A function takes a set of inputs, and returns something based on those inputs. It "answers a question"

PL/SQL Packages

- Package is a collection of related variables, constants, exceptions, cursors, and subprograms.
- Objects placed in the specification are called **public objects**.
- Any objects (subprogram or variable) which are not coded in the package specification but coded in the package body is called a **private object**.
- Only the objects inside the package can call a private object.
- **Global Variables**: Global variables can be defined by creating **body less package**. Whenever a package is loaded, whole package (including variables) will be loaded into the memory for easy access. It won't be destroyed until the package is complete. These public variables in the package specification can be accessed directly from other programs. It is useful to declare constants.

```
CREATE OR REPLACE PACKAGE pkg_var IS
    v_ttemp number(2):=10;
END;
----
BEGIN
    DBMS_OUTPUT.PUT_LINE(pkg_var.v_ttemp);
END;
```

- **Overloading**: Packages allow overloading (procedures or functions having same name with different type or different number of parameters) of procedures and functions. Standalone procedures and functions cannot be overloaded.
- **Forward Declaration**: If there are any private procedure or functions in the package, as a convention (not mandatory), it can be declared in the beginning of the package body and define it in the later part of the package body.
- Avoid re-compilation of specification. Because, once the specification is compiled, the body and other objects depend on this package become invalid. So they also need to be compiled again to use it.
- The procedures and functions inside the package does not create an object. Object is created for entire package only.
- For local procedure call (inside package), no need to pass parameters because all the variables declared as public (public inside the package or outside) can be accessed anywhere in the package.

Package Specification

```
CREATE [OR REPLACE] PACKAGE package_name IS
    --Declaration of package elements;
     PROCEDURE procedure_name(param1 datatype,param2 datatype,..);
     FUNCTION function_name(param1 datatype,param2 datatype,..) RETURN datatype;
END [package_name];
```

Package Body

```
CREATE [OR REPLACE] PACKAGE BODY package_name IS
    --Declaration of variables
    -- implementation of package elements

    PROCEDURE procedure_name(param1 datatype,param2 datatype,..) IS
        --declare variables;
    BEGIN
        --executable statements;
    END procedure_name;

    FUNCTION function_name(param1 datatype,param2 datatype,..)
    RETURN datatype IS
        --declare variables;
    BEGIN
        --executable statements;
        RETURN (return_value);
    END function_name;
END [package_name];
```

Calling package elements

```
BEGIN
    package_name.procedure_name(param1,param2..);
    DBMS_OUTPUT.PUT_LINE(package_name.function_name);
END;
```

One-time only procedure

```
CREATE OR REPLACE PACKAGE BODY package_name IS
     --Functions, Procedures etc..
BEGIN
     --written at the end of package
     --whatever code is here executes one time only
     -- runs immediately when the package is loaded into the memory
     --useful for initializing variables like gold rate, USD rate etc.. which
changes every day.
END package_name; --The END of one time only procedure should be the END of package
body. There won't be any separate END for that procedure.
```

Is it possible to use OUT parameter in function? Can we call a function with OUT parameter inside a SQL statement?

Yes, you can have OUT parameter inside a function. But if you have an OUT parameter in the function then it cannot be called from SQL expression. `ORA-06572: Function FN_TEST has out arguments`

```
CREATE OR REPLACE FUNCTION FN_FIRSTNAME (NUM_RTRN_CODE OUT NUMBER)
RETURN VARCHAR2 IS
BEGIN
NUM_RTRN_CODE := 1;
RETURN 'MANDAN';
END;

CREATE OR REPLACE FUNCTION FN_LASTNAME
RETURN VARCHAR2 IS
BEGIN
RETURN 'MANDODHIRI';
END;

DECLARE
NUM_RTRN_CODE NUMBER(10);
FIRSTNAME VARCHAR2(10);
LASTNAME VARCHAR2(10);
BEGIN
     SELECT FN_LASTNAME INTO LASTNAME FROM DUAL;
     FIRSTNAME := FN_FIRSTNAME(NUM_RTRN_CODE);
     IF (NUM_RTRN_CODE = 1 ) THEN
       DBMS_OUTPUT.PUT_LINE('MY NAME IS ' || FIRSTNAME || ' ' || LASTNAME);
     END IF;
END;
```

What is PURGE clause used for?

When issuing a DROP TABLE statement in Oracle, you can specify the PURGE option.
`DROP TABLE table_name PURGE;`
The PURGE option will purge the table and the space associated with the object. It won't appear in the recycle bin also. The risk of specifying the PURGE option is that you will not be able to recover the table.

```
SELECT * FROM USER_RECYCLEBIN; --To see the contents of your recycle bin; You can use the
RECYCLEBIN synonym instead
PURGE TABLE table_name; -- removes the table from the recycle bin
PURGE RECYCLEBIN; --remove the entire contents of your recycle bin
```

What would be the result of below query?

```
SELECT SYSDATE - SYSDATE FROM DUAL; --0
SELECT SYSDATE + SYSDATE FROM DUAL; -- ORA-00975: date + date not allowed
```

department_id (foreign key) of employees table refers to the department_id (primary key) in department table. Can we delete records from department table and vise versa?

A foreign key with **ON DELETE CASCADE** means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted.

```
... REFERENCES parent_table (column1)
ON DELETE CASCADE
```

A foreign key with **ON DELETE SET NULL** means that if a record in the parent table is deleted, then the corresponding records in the child table will have the foreign key fields set to null. The records in the child table will not be deleted.

```
...REFERENCES parent_table (column1)
ON DELETE SET NULL
```

If we try to delete without these clause from a parent table, we will get error. So we need to delete the data from the child table first to delete from the parent table.
`integrity constraint violated - child record found`

Can we use DDL trigger on tables?

No. DDL triggers are only applicable for schema and database only; not for individual tables.

What is the difference between row_number() and rank()?

In row_numer, all rows will be assigned with unique number starting with 1 even if there are duplicates. But in rank, if there are duplicate values, same ranking will be assigned for those rows.

UPDATE or MERGE is faster?

The UPDATE statement will most likely be more efficient than a MERGE if the all you are doing is updating rows. Given the complex nature of the MERGE command's match condition, it can result in more overhead to process the source and target rows. However, when you need to do more than one operation, the MERGE command is most likely a better choice, as you are only making one pass through the data as opposed to multiple passes, one for each separate UPDATE, INSERT, or DELETE command, through the source data.

You have a table itemColor (Synthetic NUMBER PRIMARY KEY, ItemID NUMBER, ColorID NUMBER, LastUpdated DATE); You want to remove all duplicate pairs (ItemID, ColorID) and to leave just the latest pair. Please write a query which will achieve it.

```
DELETE FROM ITEMCOLOR CLR_A -- ROWIDS WHICH WE DON'T WANT TO KEEP
  WHERE ROWID < (SELECT MAX(ROWID) FROM ITEMCOLOR CLR_B -- ROWIDS WHICH WANT TO KEEP
      WHERE CLR_A.ITEMID = CLR_B.ITEMID
      AND CLR_A.COLORID = CLR_B.COLORID);
```

Write a function to find provided number is prime or not

```
CREATE OR REPLACE FUNCTION PRIMENUMBER(N NUMBER)RETURN VARCHAR2 AS
  I       NUMBER;
  TEMP    NUMBER;
BEGIN
  I := 2;
  TEMP := 1;
  FOR I IN 2..N / 2 LOOP
    IF MOD(N, I)= 0 THEN
      TEMP := 0;
      EXIT;
    END IF;
  END LOOP;
  IF TEMP = 1 THEN RETURN 'PRIME';
  ELSE RETURN 'NOT PRIME';
  END IF;
END PRIMENUMBER;
```

PL/SQL Triggers

- Triggers are stored programs, which are automatically executed or fired implicitly when a triggering events occur. We cannot execute it explicitly.
- Triggers can be enabled or disabled. But stored sub programs cannot.
- Triggers cannot exceed the size limit of 32Kb.
- If a trigger requires many lines of code, consider moving it to a stored procedure that is invoked from the trigger.
- For every trigger a background process is created. It will compromise the performance of the database.
- Triggers cannot span across multiple tables. But views can span across tables.
- A trigger cannot end a transaction directly or indirectly (DDL). Because a trigger is part of a larger transaction. But an autonomous transaction can contain COMMIT and ROLLBACK statements inside a trigger.
- Row-Level trigger: Fire for each affected row. The old and new values can be seen using this trigger.
- Statement-Level trigger: Fire for each statement regardless of how many rows are affected. Default trigger is statement level trigger.

Triggering Events:

DML Statement - Update, Delete, Insert...
DDL Statement - Create, Alter...
System Event - Shutdown, Start-Up of database
User Event - Logon, Logoff

```
CREATE [OR REPLACE] TRIGGER trigger_name
 {BEFORE|AFTER|INSTEAD OF} triggering_event
 [OF column_name]
 ON table_name
 [FOR EACH ROW] --Except for INSTEAD OF
triggers, if you omit this clause, then the
trigger is a statement trigger.
 [FOLLOWS another_trigger_name]
 [ENABLE/DISABLE]
 [WHEN condition]
DECLARE
  --declaration statements
BEGIN
  --executable statements
END;
```

Uses of creating Trigger

- Enforce business rules
- Gain strong control over the security
- Collect statistical information
- Automatically generate values
- Prevent invalid transactions

Order of Execution

BEFORE STATEMENT -> BEFORE ROW -> DML -> AFTER ROW -> AFTER STATEMENT

Pseudo-records in Trigger

Operation	:OLD	:NEW
INSERT	NULL	Inserted value
UPDATE	value before update	value after update
DELETE	value before update	NULL

Ex:
```
IF :NEW.sal < :OLD.sal THEN
  DBMS_OUTPUT.PUT_LINE('Salary Increased');
END IF;
```

DDL Trigger

```
CREATE OR REPLACE TRIGGER tr_schema
AFTER DDL ON SCHEMA --Trigger for all DLLs for the schema where the trigger is created
BEGIN
  INSERT INTO schema_audit VALUES(
    SYSDATE,
    SYS_CONTEXT('USERENV','CURRENT_USER'), --user who executed the DDL;
                                           --sys_context(namespace,paramenter,length)
    ORA_DICT_OBJ_TYPE, --system event attribute - type of object where DDL operation occurred
    ORA_DICT_OBJ_NAME, --system event attribute - name of the object
    ORA_SYSEVENT -- system event attribute -which DDL operation
  );
END;
----
CREATE OR REPLACE TRIGGER tr_schema
AFTER TRUNCATE ON SCHEMA --for specific DDL; 'DML' does not exist(have to specify individual
DML names)
BEGIN
  INSERT  INTO…
END;
----
CREATE OR REPLACE TRIGGER tr_schema
AFTER DDL ON DATABASE --for whole Database
BEGIN
  INSERT  INTO…
END;
```

System, User Event Triggers

```
CREATE OR REPLACE TRIGGER tr_logon
AFTER LOGON ON SCHEMA
BEGIN
  INSERT INTO hr_event_audit VALUES(
    ORA_SYSEVENT,  -- system event attribute -what system event fired
    SYSDATE
  );
END;
----
CREATE OR REPLACE TRIGGER tr_logoff
BEFORE LOGOFF ON SCHEMA
BEGIN
  INSERT INTO ...
END;
----
CREATE OR REPLACE TRIGGER tr_logoff
BEFORE LOGOFF ON DATABASE -- sysdba
BEGIN
  INSERT INTO db_event_audit VALUES(
  USER); --USER will fetch the username of the logged in db user
  COMMIT;
END;
----
CREATE OR REPLACE TRIGGER tr_startup
AFTER STARTUP ON DATABASE --sysdba privilege needed
BEGIN
  INSERT INTO …
END;
----
CREATE OR REPLACE TRIGGER tr_startup
AFTER SHUTDOWN ON DATABASE --normal or + shutdown only. Won't work on ABORT or database
crashes
BEGIN
  INSERT INTO …
END;
```

Instead of Trigger

- Using Instead of Trigger, you can control the default behavior of Insert, Update, Delete and Merge operations on Views.
- You cannot specify an INSTEAD OF trigger on a table.
- INSTEAD OF trigger statements are implicitly activated for each row.
- INSTEAD OF triggers are valid for DML events on views. They are not valid for DDL or database events.

```
CREATE OR REPLACE TRIGGER tr_io_insert
INSTEAD OF INSERT
ON v$employee
FOR EACH ROW
BEGIN
  INSERT INTO employee VALUES(:new.name);
  INSERT INTO dept VALUES(:new.dname);
END;
```

```
CREATE [OR REPLACE] TRIGGER trigger_name
INSTEAD OF operation
ON view_name
FOR EACH ROW
BEGIN
  --executable statements
END;
```

Conditional predicates

These functions helps to know what operation is going on while the trigger is called. It returns Boolean value.

```
CREATE OR REPLACE TRIGGER tr_employee
BEFORE INSERT OR DELETE OR UPDATE ON tbl_employee
FOR EACH ROW
ENABLE
DECLARE
  v_user VARCHAR2(20);
BEGIN
  SELECT user INTO v_user FROM dual;
  IF INSERTING THEN
    DBMS_OUTPUT.PUT_LINE('Inserted by:' || v_user);
  ELSIF DELETING THEN
    DBMS_OUTPUT.PUT_LINE('Deleted by:' || v_user);
  ELSIF UPDATING THEN
    DBMS_OUTPUT.PUT_LINE('Updated by:' || v_user);
  END IF;
END;
```

Compound Trigger

The Compound trigger is a trigger that allows you to specify actions for each of four timing points in the single trigger body.

```
CREATE OR REPLACE TRIGGER trigger_name
FOR
INSERT OR UPDATE
ON table_name
COMPOUND TRIGGER
BEFORE STATEMENT IS
BEGIN
    --Execution statements
END BEFORE STATEMENT;

BEFORE EACH ROW IS
BEGIN
    --Execution statements
END BEFORE EACH ROW;

AFTER EACH ROW IS
BEGIN
    --Execution statements
END AFTER EACH ROW;

AFTER STATEMENT IS
BEGIN
    --Execution statements
END AFTER STATEMENT;
END;
```

BEFORE STATEMENT – level
BEFORE EACH ROW – level
AFTER EACH ROW - level
AFTER STATEMENT – level

Mutating table error

- This error occurs when we create a row level trigger and attempts to access the same table inside the trigger body.
- Row level cannot read or write to the table from which it is triggering. However, a statement trigger can access.

```
/*Only 1 president is allowed in the table*/
CREATE OR REPLACE PACKAGE presi_pack IS
    flag NUMBER(1); --Global variable
END presi_pack;
----
CREATE OR REPLACE TRIGGER presi_check_row
BEFORE INSERT OR UPDATE OF job ON emp
FOR EACH ROW
WHEN(:NEW.job='PRESIDENT')
BEGIN
        presi_pack.flag := 1;
end presi_check_row;
----
CREATE OR REPLACE TRIGGER presi_check_stat
AFTER INSERT OR UPDATE OF job ON emp
DECLARE
    presi_count NUMBER(1);
BEGIN
    IF presi_pack.flag = 1 THEN
        presi_pack.flag := 0;
        SELECT COUNT(1) INTO presi_count FROM emp WHERE job = 'PRESIDENT';
        IF presi_count > 1 THEN
            DBMS_OUTPUT.PUT_LINE('More than 1 PRESIDENT');
        END IF;
    END IF;
END presi_check_stat;
```

? Find All Rows With Null Value(s) in Any Column

```
CREATE OR REPLACE TYPE TYP_NULL_COLS AS TABLE OF VARCHAR2(200);

CREATE OR REPLACE FUNCTION FN_GET_NULLS(VC2_TBL_NAME IN VARCHAR2)
RETURN TYP_NULL_COLS AS
  ROW_ID VARCHAR2(100);
  COL_NAME VARCHAR2(100);
  CUR_COL_NAME SYS_REFCURSOR;
  CUR_ROW_ID SYS_REFCURSOR;
  REC_COL_NAME VARCHAR2(100);
  REC_ROW_ID VARCHAR2(100);
  VC2_STR VARCHAR2(1000);
  CLTN_NULL_COLS TYP_NULL_COLS := TYP_NULL_COLS();
  COUNTER NUMBER(10);
BEGIN
COUNTER := 0;
  OPEN CUR_COL_NAME FOR SELECT COLUMN_NAME FROM ALL_TAB_COLUMNS WHERE TABLE_NAME = VC2
_TBL_NAME;
  LOOP
    FETCH CUR_COL_NAME INTO REC_COL_NAME;
    EXIT WHEN CUR_COL_NAME%NOTFOUND;

    VC2_STR := 'SELECT ROWID FROM ' || VC2_TBL_NAME ||' WHERE '|| REC_COL_NAME ||' IS
    NULL';
        OPEN CUR_ROW_ID FOR VC2_STR;
        LOOP
        FETCH CUR_ROW_ID INTO REC_ROW_ID;
          EXIT WHEN CUR_ROW_ID%NOTFOUND;
          COUNTER := COUNTER + 1;
          CLTN_NULL_COLS.EXTEND();
          CLTN_NULL_COLS(COUNTER)  := 'ROWID :' || REC_ROW_ID || ' | COLUMN NAME:' ||
    REC_COL_NAME;
        END LOOP;

  END LOOP;
  RETURN CLTN_NULL_COLS;
END FN_GET_NULLS;

SELECT * FROM TABLE(FN_GET_NULLS('EMPLOYEES'));
```

Interview (7) Questions and Answers

? Drawbacks of Procedures, Functions, Packages, Triggers

Procedures/Functions/Packages

More memory may be required on the Oracle database server when using Oracle PL/SQL packages as the whole package is loaded into memory as soon as any object in the package is accessed.
Updating one of the functions/procedures/packages may invalidate other objects which use these functions/procedures/packages.

Trigger

Triggers cannot be executed explicitly; as a result, it is difficult to test and if there are any bugs in the trigger source code, it is difficult to find. As triggers are fired on events, it is possible that the user forgets about the trigger which is working in the background.

? What are index skip, index range, index unique scanning?

Index Range scan is used when there is NON UNIQUE index or you are searching for a range of values.
In previous releases a composite index could only be used if the first column, the leading edge, of the index was referenced in the WHERE clause of a statement. In later releases (10g) **Skip Scans** are used. So, even if you have a composite index on more than one column and you use the non-prefix column alone in your SQL, it may still use index.
Unique scan is used when all columns of a unique (B-tree) index or an index created as a result of a primary key constraint are specified with equality conditions.

? What is the benefit of creating index and table on different disk?

If you place all of your indexes on a separate disk, you don't have to worry if that disk becomes corrupt because you can rebuild the indexes. Having indexes in their own tablespace will help us recover from such a catastrophic scenario in a more simplistic, efficient and ultimately faster manner. Making the index unusable, the user can still access the data. Also, by segregating related tables onto separate data files, it's easier to track I/O at the data file and tablespace level.

```
SQL> ALTER INDEX index_name UNUSABLE;
SQL> SHOW PARAMETER SKIP
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
skip_unusable_indexes                boolean     TRUE
```

? Extract First Name, Last Name, Middle Name from Full Name which is in the format: FirstName, LastName MiddleName

```
WITH FULL_NAME AS (SELECT 'HERAS, CRYSTAL C' FULL_NAME FROM DUAL
UNION SELECT 'FOX, NICHOL R.' FULL_NAME FROM DUAL
UNION SELECT 'HERAS, FATIMA' FULL_NAME FROM DUAL)

SELECT
    TRIM(SUBSTR(FULL_NAME,1, INSTR(FULL_NAME,',')-1)) FIRST_NAME,
    TRIM(SUBSTR(FULL_NAME, DECODE(INSTR(FULL_NAME,' ',1,2),0,NULL,INSTR(FULL_NAME,'
',1,2))+1, LENGTH(FULL_NAME))) MIDDLE_NAME,
    TRIM(SUBSTR(FULL_NAME, INSTR(FULL_NAME,',')+1, ABS(INSTR(FULL_NAME,' ',1,2)-
INSTR(FULL_NAME,' ',1,1)))) LAST_NAME
FROM FULL_NAME;
```

? How to search for errors and source codes in entire schema?

```
SELECT * FROM ALL_SOURCE WHERE TEXT LIKE '%part_of_source_code%';

SELECT * FROM ALL_ERRORS WHERE NAME = 'object_name'
```

? How to update multiple columns at once?

```
UPDATE EMPLOYEES
    SET (SALARY,COMMISSION_PCT) = (SELECT SALARY,COMMISSION_PCT FROM EMPLOYEES
          WHERE EMPLOYEE_ID = 200)
    WHERE EMPLOYEE_ID = 100;

UPDATE EMPLOYEES
    SET SALARY = 24000, COMMISSION_PCT = NULL
    WHERE EMPLOYEE_ID = 100;
```

NOCOPY

- By default OUT and IN OUT parameters are passed by value and IN parameters are passed by reference. When an OUT or IN OUT parameter is modified inside the procedure the procedure actually only modifies a copy of the parameter value. Only when the procedure has finished without exception is the result value copied back to the formal parameter.
- Now, if you pass a large collection as an OUT or an IN OUT parameter then it will be passed by value, in other words the entire collection will be copied to the formal parameter when entering the procedure and back again when exiting the procedure. If the collection is large this can lead to unnecessary CPU and memory consumption.
- The NOCOPY hint alleviates this problem because you can use it to instruct the runtime engine to try to pass OUT or IN OUT parameters by reference instead of by value. For example:

```
PROCEDURE GET_CUSTOMER_ORDERS(
P_CUSTOMER_ID IN NUMBER,
P_ORDERS OUT NOCOPY ORDERS_COLL
);
THEORDERS ORDERS_COLL;
GET_CUSTOMER_ORDERS(124, THEORDERS);
```

- In the absence of the NOCOPY hint the entire orders collection would have been copied into the orders variable upon exit from the procedure. Instead the collection is now passed by reference.
- Keep in mind, however, that there is a downside to using NOCOPY. When you pass parameters to a procedure by reference then any modifications you perform on the parameters inside the procedure is done on the same memory location as the actual parameter, so the modifications are visible. In other words, there is no way to ?undo? or ?rollback? these modifications, even when an exception is raised midway. So if an exception is raised inside the procedure the value of the parameter is ?undefined? and cannot be trusted.

Sequence

```
CREATE SEQUENCE sequence_name
START WITH starting_number
INCREMENT BY incrementing_value --Cannot be zero; Default is 1; if -ve, values
descends if +ve, values ascends
MAXVALUE maximum_value | NOMAXVALUE -- should be equal to or greater than START
WITH and must be greater than MINVALUE
MINVALUE minimum_value | NOMINVALUE -- should be equal to or greater than START
WITH and must be lesser than MAXVALUE
CACHE cache_num | NOCACHE -- number of integers to keep in memory; default is 20
CYCLE | NOCYCLE --default is NOCYCLE
ORDER | NOORDER; --default is NOORDER; ORDER = sequence numbers are generated in
the order of request. Useful when using sequence number as timestamp. Not required
for primary key.
```

NEXTVAL & CURRVAL

```
SELECT sequence_name.NEXTVAL FROM DUAL;
SELECT sequence_name.CURRVAL FROM DUAL;
DROP SEQUENCE sequence_name;
```

CURSORs

- Cursor is a pointer to a memory area called context area.
- Context area is a memory region inside the Process Global Area(PGA).
- **Implicit CURSORs** are automatically created by Oracle when DML statement is executed. Cursor name is 'SQL'.
- **Explicit CURSORs** are user defined cursors. It is created for any DML operation which returns more than 1 row.

Implicit CURSOR

```
DECLARE
  total_rows(2);
  emp_rec emp %ROWTYPE;
BEGIN
  SELECT * INTO emp_rec FROM emp WHERE empno=7369;
  IF SQL %FOUND THEN
    DBMS_OUTPUT.PUT_LINE(emp_rec.ename ||emp_rec.sal);
  END;
END;
```

CURSOR Attributes

```
%FOUND
%NOTFOUND
%ISOPEN
%ROWCOUNT
%BULK_ROWCOUNT
%BULK_EXCEPTIONS
```

Explicit CURSOR

```
DECLARE
  CURSOR emp_cur IS
    SELECT empno,ename FROM emp;
  emp_rec emp_cur %ROWTYPE;
BEGIN
  OPEN emp_cur;
  LOOP
    FETCH emp_cur INTO emp_rec;
    EXIT WHEN emp_cur %NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(emp_rec.empno ||emp_rec.ename);
  END LOOP;
  CLOSE emp_cur;
END;
```

```
SQLERRM - Error Message
SQLCODE - Error Number
```

Recursive Cursor

A recursive subprogram is one that calls itself. Each recursive call creates a new instance of any items declared in the subprogram, including parameters, variables, cursors, and exceptions. A recursive cursor (a pointer to a shared SQL area) is used to keep a pointer to each call of a recursive function.

REF CURSOR

- A REF CURSOR is basically a data type.
- A REF CURSOR can be associated with more than one SELECT statement at run-time. Before associating a new SELECT statement, we need to close the previous CURSOR.
- The primary advantage of using REF CURSOR is their capability to pass result sets between sub programs (like stored procedures, functions, packages etc.).

Dealing with REF CURSOR in the sub-programs of a PL/SQL block

The sub-routine gets executed for every iteration, which displays the employee information for the respective department.

```
DECLARE
  TYPE r_cursor IS REF CURSOR;
  c_emp r_cursor;
  TYPE rec_emp IS RECORD(
    name  VARCHAR2(20),
    sal   NUMBER(6));
  er rec_emp;
  PROCEDURE printemployeedetails IS
  BEGIN
    LOOP
      FETCH c_emp INTO ER;
      EXIT WHEN c_emp %NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(er.name || ' – ' || er.sal);
    END LOOP;
  END;
BEGIN
  FOR i IN (SELECT deptno,dname FROM dept) --can use SELECT statement directly
without defining a cursor
  LOOP
    OPEN c_emp FOR SELECT ename,sal FROM emp WHERE deptno = i.deptno;
    DBMS_OUTPUT.PUT_LINE(i.dname);
    DBMS_OUTPUT.PUT_LINE('–––––');
    printemployeedetails;
    CLOSE c_emp;
  END LOOP;
END;
```

Passing REF CURSOR as parameters to sub-programs

```
DECLARE
  TYPE r_cursor IS REF CURSOR;
  c_emp r_cursor;
  TYPE rec_emp IS RECORD(
    NAME  VARCHAR2(20),
    sal   NUMBER(6));
  PROCEDURE printemployeedetails(p_emp r_cursor) IS
    er rec_emp;
  BEGIN
    LOOP
      FETCH p_emp INTO er; --no need to OPEN
      EXIT WHEN p_emp %NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(er.name || ' – ' || er.sal);
    END LOOP;
  END;
BEGIN
  FOR i IN (SELECT deptno,dname FROM dept)
  LOOP
    OPEN c_emp FOR SELECT ename,sal FROM EMP WHERE deptno = i.deptno;
    DBMS_OUTPUT.PUT_LINE(i.dname);
    DBMS_OUTPUT.PUT_LINE('–––––');
    printemployeedetails(c_emp);
    CLOSE c_emp;
  END LOOP;
END;
```

Types of REF CURSORs

Strong REF CURSOR

- Any REF CURSOR which has fixed return type is called **Strong REF CURSOR**
- Strong REF CURSOR supports different type of SELECT statements but all of the same structure ,but not necessary that the table should be same.

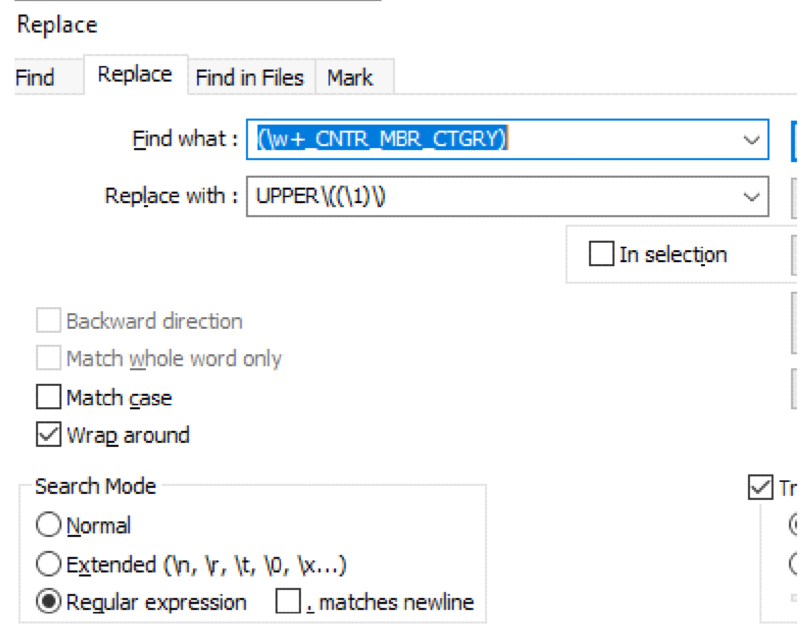
```
DECLARE
    TYPE ref_cursor_name IS REF CURSOR
    RETURN (return_type);--Return must be of RECORD datatype
----
DECLARE
    TYPE my_RefCur IS REF CURSOR
    RETURN employees %ROWTYPE;
    cur_var my_RefCur;
    rec_var employees %ROWTYPE;
BEGIN
    OPEN cur_var FOR SELECT * FROM employees WHERE employee_id =100;
    FETCH cur_var INTO rec_var;
    CLOSE cur_var;
    DBMS_OUTPUT.PUT_LINE(rec_var.first_name || rec_var.salary);
END;
```

Strong Ref Cursor with User Defined Record Datatype

- Use of this is, we can customize the number of field we want to fetch and still we can have a record datatype for Strong Ref Cursor

```
DECLARE
    TYPE my_rec IS RECORD(
        emp_sal employees.salary %TYPE;);
    TYPE my_RefCur IS REF CURSOR
    RETURN my_rec; --User defined RECORD datatype for return
    cur_var my_RefCur;
    at_var employees.salary %TYPE;
BEGIN
    OPEN cur_var FOR SELECT salary FROM employees WHERE employee_id =100;
    FETCH cur_var INTO at_var;
    CLOSE cur_var;
    DBMS_OUTPUT.PUT_LINE('Salary:' || at_var);
END;
```

? Wild card search in Notepad ++


Replace

Find what : (\w+_CNTR_MBR_CTGRY)
Replace with : UPPER\((\1)\)
☐ In selection
☐ Backward direction
☐ Match whole word only
☐ Match case
☑ Wrap around
Search Mode
○ Normal
○ Extended (\n, \r, \t, \0, \x...)
● Regular expression ☐ . matches newline
☑ Tr

? Call a function using the data from WITH clause.

```
SELECT STDTY.*,
        (SELECT SSD_MBR_ID FROM DRVTS_SWAPS_STMP_DUTY
        WHERE NVL(SSD_MBR_ID,'-') <> NVL(STDTY.SSD_MBR_ID,'-'))
    FROM DRVTS_SWAPS_STMP_DUTY STDTY
    WHERE TRUNC(SSD_CRNT_BSNS_DATE) BETWEEN
     (WITH FROMDATE AS
        (SELECT SYP_CRNT_DATE, SYP_BSNS_SGMNT,SYP_BSNS_SUB_SGMNT
        FROM SYSTM_PRM
        WHERE SYP_BSNS_SGMNT = 'INRDRV' AND SYP_BSNS_SUB_SGMNT = 'IRSWAP')
        SELECT S_FN_PREV_MONTH_END (SYP_CRNT_DATE,SYP_BSNS_SGMNT,SYP_BSNS_SUB_SGMNT)
        FROM FROMDATE)
    AND (SELECT SYP_CRNT_DATE FROM SYSTM_PRM
    WHERE SYP_BSNS_SGMNT = 'INTDRV' AND SYP_BSNS_SUB_SGMNT = 'IRSWAP')
    AND SSD_MBR_ID = :VC2_MBR_ID; -- bind variable
```

? Table with index or a GTT is faster?

Table with index is faster. GTT without index and GTT with index have more cost than a table with index. GTT is used not for performance gain rather to store values for temporary purpose.

GTT table has to be truncated in order to drop or to create index on it.

? What would be the result of below query?

```
SELECT first_name,salary,department_id FROM employees WHERE department_id = 50
UNION
SELECT first_name,salary FROM employees WHERE department_id = 60 --ORA-01789: query block has
incorrect number of result columns

What is BRS and CR?
BRS - Business Requirement Specification: This document is called as high level document and includes
the entire requirement demanded by the client. ... BRS includes list of requirements which are
demanded by the client and should be part of the proposed system.
```

? If we want to fetch 90 records out of 100 records. What scan will oracle choose? Index scan or Full table scan?

If the cardinality is above the half of the total number of rows, then full table scan is performed. Otherwise index scan is performed. Ex: Total rows in a table is 100. If the cardinality is above 50, then full table scan is used.

? Does dropping a table drops index?

Dropping a table removes the table definition from the data dictionary. All rows of the table are no longer accessible. All indexes and triggers associated with a table are dropped. All views and PL/SQL program units dependent on a dropped table remain, yet become invalid (not usable).

? Will index gets copied with SELECT * FROM old_table?

No. This will create a table with rows from specified query but only with NOT NULL and Check constraints by default. Indexes or any other constraints will not be copied / recreated.
You can get DDL for indexes by

```
SELECT DBMS_METADATA.GET_DDL ('INDEX', INDEX_NAME)
  FROM USER_INDEXES
 WHERE TABLE_NAME = 'EMPLOYEES';
```

? How can we do a partition on an existing table?

We can partition existing table using the DBMS_REDEFINITION package.

? How can we insert index to a table while a DML operation is going on? ONLINE

Index online enables you to update base tables at the same time you are building or rebuilding indexes on that table. You can perform DML operations while the index build is taking place, but DDL operations are not allowed. If you do it online, you'll need additional space to hold the changes that are made during the rebuild as well.

```
CREATE INDEX emp_name ON emp (mgr, emp1, emp2, emp3) ONLINE;

CREATE INDEX index_name REBUILD ONLINE;
```

? What is INSERT ALL?

INSERT ALL statement is used to add multiple rows with a single INSERT statement. The rows can be inserted into one table or multiple tables using only one SQL command.

```
INSERT ALL
  INTO mytable (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
  INTO mytable2 (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
  INTO mytable3 (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
SELECT * FROM dual;
```

? What is CAST function?

CAST function converts one datatype to another.

```
SELECT CAST( '22-AUG-2003' AS VARCHAR2(30) )
FROM DUAL;

SELECT CAST(SYSTIMESTAMP AS DATE) FROM DUAL;
```

Weak REF CURSOR

- Weak REF CURSORs are those cursors which do not have any return type
- These cursors are the most frequently used REF CURSOR as they are open to all SELECT statements
- This REF CURSOR allows us to fetch any type of SELECT statement irrespective of data structure .

```
DECLARE
    TYPE ref_cursor_name IS REF CURSOR;
----
DECLARE
    TYPE my_RefCur IS REF CURSOR;
    cur_var my_RefCur;

    f_name employees.first_name %TYPE;
    emp_sal employees.salary %TYPE;
BEGIN
    OPEN cur_var FOR SELECT first_name, salary FROM employees WHERE employee_id =
100;
    FETCH cur_var INTO f_name,emp_sal;
    CLOSE cur_var;
    DBMS_OUTPUT.PUT_LINE(f_name || emp_sal);
END;
```

SYS_REF CURSOR

It is a predefined weak REF CURSOR (`TYPE SYS_REFCURSOR IS REF CURSOR;`). So without declaring the ref pointer type, you can assign variable.

```
DECLARE
    cur_var SYS_REFCURSOR;
    f_name employees.first_name %TYPE;
    emp_sal employees.salary %TYPE;
BEGIN
    OPEN cur_var FOR SELECT first_name, salary FROM employees WHERE employee_id =
100;
    FETCH cur_var INTO f_name,emp_sal;
    CLOSE cur_var;
    DBMS_OUTPUT.PUT_LINE(f_name || emp_sal);
END;
```

Parameterized CURSORs

```
DECLARE
  emp_rec emp %ROWTYPE;
  CURSOR emp_cur(max_wage NUMBER :=100, emp_name VARCHAR2) IS  --Default value is
assigned to
                                                             --parameter
max_wage
    SELECT * FROM emp WHERE sal>max_wage and ename=emp_name;
BEGIN
  OPEN emp_cur(2000,'Raj');
  LOOP
    FETCH emp_cur INTO emp_rec;
    EXIT WHEN emp_cur %NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(emp_rec.ename ||emp_rec.sal);
  END LOOP;
  CLOSE emp_cur;
END;
```

CURSOR vs. REF CURSOR

REF CURSOR	CURSOR
Dynamic	Static
Can be associated with multiple SELECT statements in a PL/SQL block	Can only access single SELECT statement at a time
Can be changed at run time	Cannot be changed at run time. Can be done with parameterized cursor.
Can be returned to the client application	Cannot be returned to the client application
Cannot be global. Cannot define them OUTSIDE of a procedure / function	Can be global. Global cursors can be opened and executed outside of the package in which they are defined
Can be passed from one sub-program to other sub-program	Cannot be passed

FOR UPDATE OF, FOR UPDATE and WHERE CURRENT OF

- **FOR UPDATE** will give exclusive row-level lock on all rows retrieved by SELECT statement.
- The FOR UPDATE clause is generally used in cases where an online system needs to display a set of row data on a screen and they need to ensure that the data does not change before the end-user has an opportunity to update the data. In the real-world, many large online systems do not use the FOR UPDATE clause.
- If you try to access the rows with the **NOWAIT** clause, you will get an error message, ORA-00054 Resource busy and acquire with NOWAIT specified. NOWAIT option is just to investigate that yes i am not at all willing to wait to acquire the lock rather than hang myself, If i cannot get the lock immediately, an error is returned to signal that the lock is not possible at this time. You may try again later.
- If there are more than one table are joined for update, then the use of **FOR UPDATE OF** ... will only lock the rows in the tables that contain the columns you specify in the **OF clause**. You can never lock a single column, the minimum lock is at row level. It locks all rows in the table that contains the column, which are selected by the query.
- **WHERE CURRENT OF** clause can be used for both DELETE and UPDATE statements inside a cursor's range to make changes to the last fetched row(s)

```
CURSOR cursor_name
IS
    SELECT * FROM ..
    FOR UPDATE [OF column_list] [WAIT 15] [NOWAIT];
--NOWAIT - cursor does not wait for resources. If it is locked, it will show error
--WAIT 15 - wait up to 15 seconds for another session to release their lock. If
not, show error
```

```
DECLARE
CURSOR cur IS
    SELECT * FROM departments WHERE department_id = 210 FOR UPDATE;
emp_rec departments %ROWTYPE;
BEGIN
    OPEN cur;
    LOOP
        FETCH cur INTO emp_rec;
        EXIT WHEN cur %NOTFOUND;
        INSERT INTO emp_log VALUES emp_rec;
        DELETE FROM departments WHERE CURRENT OF cur;
    END LOOP;
    COMMIT;
    CLOSE cur;
END;
```

Modes of Locking

Exclusive Lock

If a lock is acquired on a data item to perform a write operation, then it is an exclusive lock

Shared Lock

Read locks are shared because no data value is being changed

Local Screening

If there are two variables with same name in the outer and inner block, and if we call the variable from inside the block, it always prefer the local (inner) variable. To refer to the outer variable, we need to use label.

```
<<out_label>>
DECLARE
    x NUMBER := 100;
BEGIN
    FOR x IN 1..10 LOOP
        DBMS_OUTPUT.PUT_LINE(out_label.x);
    END LOOP;
END;
```

Hard parse vs. soft parse

Hard parse

If a session executes an SQL statement that does not exist in the shared pool, then Oracle has to do a hard parse.
Oracle must then:

- Allocate memory for the statement from the shared pool.
- Check the statement syntactically
- Check if the user trying to execute the statement has the necessary rights to execute it

A hard parse is expensive in both terms of CPU used and number of shared pool latch and library cache latch it needs to acquire and release. It should be avoided whenever possible.

Soft parse

If a session executes an SQL statement that exists in the shared pool and there is a version of the statement that can be used, then this is referred to as a soft parse.

Identical Statements

A statement is identical to another statement, if there is absolutely no difference between the letters. For example select x from y and SELECT X FROM Y are not identical, although they clearly do the same thing.
Even if two statements are identical, this doesn't mean they are shareable. In order for two identical statements to be shareable, the following must be true

- Object names must resolve the same actual objects
- The optimizer goal is the same
- Types and length of bind variables is similar
- The NLS environment is the same

Versions of statements

If two statements are identical but not shareable, they have different versions of the statement. High version counts for sql statements should be avoided. The number of versions for a statement can be found in v$sqlarea:
`SELECT SQL_TEXT FROM V$SQLAREA WHERE VERSION_COUNT > 1;`

Parent and Child Cursors

For each SQL statement that you execute, Oracle engine will generate two cursors: parent and child cursor. Two cursors are generated because, there could be differences like there can be different bind variables or two different schema or different literals etc in same SQL statement. The parent cursor will hold the SQL statement and the child cursor will hold the information related to the differences. This makes the child cursor as deciding factor as to SQL statement to go for hard or soft parse.

Parameter cursor_sharing

The parameter cursor_sharing affects the behavior of un-identical sql statements. As of 9i, valid values for this parameter are: exact, force and similar.

```
ALTER SYSTEM SET CURSOR_SHARING='EXACT';
ALTER SYSTEM FLUSH SHARED_POOL;
SHO PARAMETER CURSOR_SHARING
```

EXACT:
FORCE: the requirement for two statements to be identical is relaxed: statements that differ in some literals, but are otherwise identical, are regarded as identical statements (and can share a cursor. So, the following two statements might be considered identical: select x from f where a='foo' and select x from f where a='bar'.
Sometimes, this is too much of a relaxation because a user might actually want two statements to be different (as they might have a different execution plan). This can be avoided with **SIMILAR:** Statements that differ in literals are consider identical unless the execution plan is different for the statements..
If you set CURSOR_SHARING, then Oracle recommends the FORCE setting unless you are in a DSS environment. FORCE limits the growth of child cursors that can occur when the setting is SIMILAR.

```
DROP TABLE TEST PURGE;
CREATE TABLE TEST (ID1 NUMBER, ID2 NUMBER, TXT CHAR(1000));
INSERT INTO TEST VALUES (1,1, 'one');

BEGIN
FOR I IN 1..1000 LOOP
INSERT INTO TEST VALUES (2,2, 'two');
INSERT INTO TEST VALUES (3,3, 'three');
END LOOP;
END;
/

INSERT INTO TEST SELECT * FROM TEST WHERE ID1=3;
COMMIT;

CREATE INDEX TEST_IDX1 ON TEST(ID1);
CREATE INDEX TEST_IDX2 ON TEST(ID2);
```

? What is AWR, ADDM and ASH?

The **Automatic Workload Repository** is used to collect performance statistics.
Active Session History allows you to see current and historical information about active sessions on the database.
The **Automatic Database Diagnostic Monitor** analyses data in the Automatic Workload Repository (AWR) to identify potential performance bottlenecks. For each of the identified issues it locates the root cause and provides recommendations for correcting the problem. An ADDM analysis task is performed and its findings and recommendations stored in the database every time an AWR snapshot is taken provided the STATISTICS_LEVEL parameter is set to TYPICAL or ALL.
If a performance problem lasts for a significant portion of the time between snapshots, it will be captured by ADDM. If a particular problem lasts for a very short duration, then its severity might be averaged out or minimized by other performance problems in the analysis period. Therefore, the problem may not appear in the ADDM findings.

Generate AWR Report

```
SQL> SHOW PARAMETER STATISTICS_LEVEL

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
statistics_level                     string      TYPICAL


EXECUTE dbms_workload_repository.create_snapshot(); --To create snapsot
manually

SET LINES 100 PAGES 999 --This script will list all AWR snapshots
SELECT SNAP_ID,
  SNAP_LEVEL,
  TO_CHAR(BEGIN_INTERVAL_TIME, 'DD/MM/YY HH24:MI:SS') BEGIN
FROM
   DBA_HIST_SNAPSHOT
ORDER BY 1;

Then generate the AWR between those two manual snapshots.
The awrrpt.sql SQL script generates an HTML or text report that displays
statistics for a range of snapshot Ids.

Run -> %ORACLE_HOME%\rdbms\admin\awrrpt.sql
```

awr_teni ash_teni ADDM

Generate ASH Report

```
Run -> %ORACLE_HOME%\rdbms\admin\ashrpt.sql
```

Configure Enterprise Manager

```
SQL> SELECT DBMS_XDB_CONFIG.gethttpport FROM DUAL;
GETHTTPPORT
-----------
          0

SQL> SELECT DBMS_XDB_CONFIG.gethttpsport FROM DUAL;


GETHTTPSPORT
------------
        5500

SQL> EXEC DBMS_XDB_CONFIG.sethttpsport(5500);
PL/SQL procedure successfully completed.

https://localhost:5500/em/login
```

STAMPDUTY STAMPDUTY

https://drive.google.com/open?id=1KaGD7LpYp7Jz5vaJkVR3dNyoZM78iYwH

? When we need to re-build the index?

```
SQL> ANALYZE INDEX EMP_EMP_ID_PK VALIDATE STRUCTURE;
Index analyzed.

SQL> COLUMN name FORMAT a20;
SQL> SELECT name, height,lf_rows,lf_blks,del_lf_rows FROM
  2  INDEX_STATS;

NAME                     HEIGHT    LF_ROWS    LF_BLKS DEL_LF_ROWS
-------------------- ---------- ---------- ---------- -----------
EMP_EMP_ID_PK                 1        107          1           0

There are two rules of thumb to help determine if the index needs to be
rebuilt.
1)If the index has height greater than four, rebuild the index.
2)The deleted leaf rows should be less than 20%.
```

If it is determined that the index needs to be rebuilt, this can easily be accomplished by the `ALTER INDEX <INDEX_NAME> REBUILD | REBULID ONLINE` command. It is not recommended, this command could be executed during normal operating hours. The alternative is to drop and re-create the index. Creating an index uses the base table as its data source that needs to put a lock on the table. The index is also unavailable during creation.

Interview (8) Questions and Answers

? What is the difference between alert log file and trace file?

- Alert log file is found at the background dump directory specified by the background_dump_dest parameter. Alert log file logs the information like startup or shutdown information of the database, admin activities and it's all done by the server process.
- Trace files are of 3 types – background trace files, core trace files and user trace files.
  If any background process fails to perform, it will throw error and a trace file will be generated called background trace files
  For all operating system related errors with oracle, trace files will be generated called core trace files
  For all user related errors, trace files called user trace files
  The default location of these files can be changed by defining following parameters
  BACKGROUND_DUMP_DEST
  CORE_DUMP_DEST
  USER_DUMP_DEST
- Automatic Diagnostic Repository (ADR) - The ADR is a systemwide tracing and logging central repository for database diagnostic data such as trace, the alert log, health monitor reports. The ADR root directory is known as ADR base. Its location is set by DISGNOSTIC_DEST parameter.

```
SELECT ID1,ID2, COUNT(*)
FROM TEST
GROUP BY ID1,ID2;
--Parent Cursor
COL SQL_TEXT FOR A30 WORD_WRAPPED
SELECT SQL_TEXT , SQL_ID, VERSION_COUNT, HASH_VALUE,PLAN_HASH_VALUE
FROM V$SQLAREA
WHERE UPPER(SQL_TEXT) LIKE 'SELECT COUNT(*) FROM TEST%'
AND UPPER(SQL_TEXT) NOT LIKE '%HASH%';
--Child Cursor
COL CHILD_NUMBER FOR 99
SELECT SQL_TEXT, SQL_ID, CHILD_NUMBER CHILD#, HASH_VALUE, PLAN_HASH_VALUE
FROM V$SQL
WHERE UPPER(SQL_TEXT) LIKE 'SELECT COUNT(*) FROM TEST%'
AND UPPER(SQL_TEXT) NOT LIKE '%HASH%';
```

CURSOR_SHARING = EXACT

- Two statements are identical if they are textually identical (All bind variable and value in where clause are same). This is as is described above
- Causes maximum memory usage in library cache as two cursors – one parent and one child cursor are created for each distinct value of the bind variable.
- Gives best performance as optimizer creates different execution plan for each value of the bind variable.
- If histogram on a column is created with only one bucket, i.e. it does not know about the skew in data, only one child cursor will be created. If histogram is created on a column with >1 buckets i.e. it knows about skew in data in that column, it will create one child cursor for each statement even of the execution plan is same.

CURSOR_SHARING = SIMILAR reduces the number of parent cursors.

- If there is skew (data is not evenly distributed. "Shuffled") in data
  - If histogram on the column containing skewed data is there
    - multiple child cursors may be created – one for each value of the bind variable
  - else (histogram is not available)
    - only one child cursor will be created.
- else (Data is not skewed)
  - only one child cursor will be created.
- Reduces memory usage in library cache as only one parent cursor is created .
- If data is not skewed or the optimizer is not aware of the skew (without histogram), optimizer peeks at the value of the bind variable on the first execution of the statement and that plan is used for all the values of the bind variable. Thus only one child cursor is created resulting in minimum memory usage by child cursors. In this case performance will be affected if there is skew in the data.
- If data is skewed and the optimizer is aware of the skew (with histogram), multiple child cursor are created – one for each distinct value of the bind variable. In this case performance will be the best as optimizer creates different execution plan for each value of the bind variable. But in this case we will have multiple child cursors created for the same execution plan.

CURSOR_SHARING = FORCE IN 10g

```
ALTER SYSTEM SET OPTIMIZER_FEATURES_ENABLE='10.2.0.3';
```

- Causes minimum memory usage in library cache as only one parent cursor and only one child cursor are created .
- In this case performance will be affected if there is skew in the data.

CURSOR_SHARING = FORCE IN 11g (ADAPTIVE CURSOR SHARING)

```
ALTER SYSTEM SET OPTIMIZER_FEATURES_ENABLE='11.2.0.1';
```

- Reduces memory usage in library cache as only one parent cursor and only one child cursor are created .
- If data is not skewed or the optimizer is not aware of the skew, optimizer peeks at the value of the bind variable on the first execution of the statement and that plan is used for all the values of the bind variable. Thus only one child cursor is created resulting in minimum memory usage by child cursors. In this case performance will be affected if there is skew in the data. (same scenario as cursor_sharing=similar)
- If data is skewed and the optimizer is aware of the skew, multiple child cursor are created for different values of the bind variable – one for each distinct execution plan . In this case performance will be the best as optimizer creates different execution plans for different values of the bind variable. But in this case we will have only child cursor created for the same execution plan thereby resulting in optimum memory usage by child cursors.

Histograms

- A histogram is a special type of column statistic that provides more detailed information about the data distribution in a table column.
- By default, Oracle collects general information about column data such as high value, low value, number of distinct values, but does not always collect information about the distribution of data within column.
- A histogram sorts values into "buckets," as you might sort coins into buckets.
- If the data is data is unevenly distributed, the optimizer might need a histogram to determine best plan.
- **Height-Balanced Histograms:** The columns are divided into buckets so that each bucket contains approximately the same number of rows.
- Frequency Histograms: Each value of the column corresponds to a single bucket of the histogram. Each bucket contains the number of occurrences of this single value. Oracle automatically creates frequency histograms if the number of distinct values is less than or equal to the number of histogram buckets specified (maximum buckets is 254).

```
SELECT COLUMN_NAME, NUM_DISTINCT, NUM_BUCKETS, HISTOGRAM FROM DBA_TAB_COL_STATISTICS
WHERE TABLE_NAME = 'EMPLOYEES';
```

COLUMN_NAME	NUM_DISTINCT	NUM_BUCKETS	HISTOGRAM
EMPLOYEE_ID	107	1	NONE
FIRST_NAME	91	1	NONE
LAST_NAME	102	102	FREQUENCY

```
SELECT ENDPOINT_NUMBER, ENDPOINT_VALUE
FROM DBA_TAB_HISTOGRAMS
WHERE TABLE_NAME = 'EMPLOYEES'
ORDER BY ENDPOINT_NUMBER;
```

ENDPOINT_NUMBER	ENDPOINT_VALUE
0	0.1
0	100
0	3.39535255630759E35
1	206
1	10
1	4.53868230530328E35

METHOD_OPT

```
FOR ALL [INDEXED] [HIDDEN] COLUMNS SIZE [SIZE_CLAUSE] FOR COLUMNS SIZE SIZE_VALUE
COLUMN_NAME
```

- The default, FOR ALL COLUMNS, will collects base column statistics for all of the columns (including hidden columns) in the table.
- FOR ALL INDEXED COLUMNS limits base column gathering to only those columns that are included in an index.
- FOR ALL HIDDEN COLUMNS limits base column statistics gathering to only the virtual columns that have been created on a table.
- The SIZE part of the METHOD_OPT
  - AUTO means Oracle will automatically determines the columns that need histograms based on the column usage information (SYS.COL_USAGE$), and the presence of a data skew.
  - An integer value indicates that a histogram will be created with at most the specified number of buckets. Must be in the range [1,254]. SIZE 1 means no histogram will be created.
  - REPEAT ensures a histogram will only be created for any column that already has one. This will limit the maximum number of buckets used for the newly created histograms.
  - SKEWONLY automatically creates a histogram on any column that shows a skew in its data distribution.

```
BEGIN DBMS_STATS.GATHER_TABLE_STATS('SH', 'SALES',
METHOD_OPT => 'FOR ALL COLUMNS SIZE 1 FOR COLUMNS SIZE 254 CUST_ID'); --
Histogram created for CUST_ID column and not for any other columns
END;
------
BEGIN
 DBMS_STATS.DELETE_COLUMN_STATS('SH', 'SALES', 'PROD_ID'); -- Delete statistics
END;
------
BEGIN
 DBMS_STATS.GATHER_TABLE_STATS('SH', 'SALES',
 METHOD_OPT => 'FOR COLUMNS SIZE 254 CUST_ID TIME_ID CHANNEL_ID PROMO_ID'); --
Histogram created for CUST_ID, TIME_ID, CHANNEL_ID, PROMO_ID columns
END;
------
EXEC DBMS_STATS.GATHER_TABLE_STATS(OWNNAME => 'HR',-
TABNAME => 'TEST',-
ESTIMATE_PERCENT =>NULL,-
METHOD_OPT => 'FOR COLUMNS SIZE 1 ID1');
------
EXEC DBMS_STATS.GATHER_TABLE_STATS(OWNNAME => 'HR',-
TABNAME => 'TEST',-
ESTIMATE_PERCENT =>NULL,-
CASCADE => TRUE,-
METHOD_OPT => 'FOR COLUMNS SIZE 4 ID2');
```

Bind Peeking

- Bind peeking or bind variable peeking, is when a query uses bind variable, the optimizer must select the best plan without the presence of literals in the SQL text. This plan might not be efficient for different bind variables.
- Oracle 11g Adaptive Cursor Sharing attempts to resolve this issue created by bind peeking. In this, if the optimizer detects that a SQL might perform better with different execution plans when provided with different bind variables, it will mark the SQL as **bind sensitive cursor**. So different execution plans are created for different bind variables.
- **Bind aware cursors** are bind sensitive cursors which are eligible to use different plans for different bind values. After a cursor has been made bind aware, the optimizer chooses plans for future executions based on the bind variable and its selective estimate. If the database marks the cursor as bind aware, then the next time, it generates new plan for new bind variable and marks the original cursor generated for the statement as not sharable (IS_SHAREABLE = N in V$SQL) and the cursor is no longer will be useable and will be among the first to be expired out of the shared SQL area.

Exception Handling

- SQLCODE & SQLERRM are used for logging error
- If exception is fired, it cannot go back to executable section of current block and uncommitted changes are rolled back.
- For user-defined exception, the default SQLCODE=1 and SQLERRM = 'User Defined Exception'
- If there are no errors, SQLCODE = ORA-0000 and SQLERRM = 'Normal, Successful Completion'

Predefined Exceptions

NO_DATA_FOUND - It is raised when a SELECT INTO statement returns no rows.
PROGRAM_ERROR - It is raised when PL/SQL has an internal problem.
TOO_MANY_ROWS - It is raised when a SELECT INTO statement returns more than one row.
VALUE_ERROR - It is raised when an arithmetic, conversion, truncation, or size constraint error occurs.
ZERO_DIVIDE - It is raised when an attempt is made to divide a number by zero.
DUP_VAL_ON_INDEX - Unique constraint error

Exception Declaration

- An exception declaration declares a user-defined exception
- No error message or error code can be associated with this exception

```
DECLARE
    exception_name EXCEPTION;
BEGIN
    IF condition THEN
        RAISE exception_name;
    END IF;
    --Executable statements;
EXCEPTION
    WHEN exception_name THEN
        --statement1;
    WHEN OTHERS THEN
        --statement2;
END;
```

Write a function with collection as return parameter.

```
CREATE TYPE RCD_TESTTYPE AS OBJECT(
    EMPID NUMBER(10),
    ENAME VARCHAR2(20)
);

CREATE TYPE CLT_TESTTYPE AS TABLE OF RCD_TESTTYPE;

CREATE OR REPLACE FUNCTION FN_TESTTYPE
    RETURN CLT_TESTTYPE IS
    V_RET CLT_TESTTYPE;
BEGIN
    SELECT RCD_TESTTYPE (EMPLOYEE_ID, FIRST_NAME)
    BULK COLLECT INTO V_RET
    FROM EMPLOYEES
    WHERE DEPARTMENT_ID IN (10,20,30);

    RETURN V_RET;
END;

SELECT * FROM TABLE(FN_TESTTYPE);
```

```
    EMPID ENAME
---------- --------------------
       200 Jennifer
       201 Michael
       202 Pat
       114 Den
       115 Alexander
       116 Shelli
       117 Sigal
       118 Guy
       119 Karen

9 rows selected.
```

What TRUNC(DATE,'YY'), TRUNC(DATE,'MM'), TRUNC(DATE,'DD') will result?

```
SELECT TRUNC(LAST_DAY(TO_DATE('07-APR-15','DD-MON-YY')) + 31, 'MM') FROM DUAL; --30th
April + 31 = 31st May
01-MAY-15

SELECT TRUNC(LAST_DAY(TO_DATE('07-APR-15','DD-MON-YY')) + 32, 'MM') FROM DUAL; --30th
April + 32 = 1st June
01-JUN-15

SELECT TRUNC(LAST_DAY(TO_DATE('07-APR-15','DD-MON-YY')) + 11, 'MM') FROM DUAL; --30th
April + 11 = 11th May
01-MAY-15

SELECT TRUNC(TO_DATE('07-APR-15','DD-MON-YY') + 11, 'MM') FROM DUAL; --7th April + 11 =
18th April
01-APR-15

SELECT TRUNC(LAST_DAY(TO_DATE('07-APR-15','DD-MON-YY')) + 11, 'DD') FROM DUAL; --30th
April + 11 = 11th May
11-MAY-15

SELECT TRUNC(TO_DATE('07-APR-15','DD-MON-YY') + 11, 'DD') FROM DUAL; --7th April + 11 =
18th April
18-APR-15

SELECT TRUNC(LAST_DAY(TO_DATE('07-APR-15','DD-MON-YY')) + 11, 'YY') FROM DUAL; --30th
April + 11 = 11th May 2015
01-JAN-15

SELECT TRUNC(TO_DATE('07-APR-15','DD-MON-YY') + 11, 'YY') FROM DUAL; --7th April + 11 =
18th April 2015
01-JAN-15

SELECT TRUNC(LAST_DAY(TO_DATE('07-APR-15','DD-MON-YY')) + 365, 'YY') FROM DUAL; --30th
April + 365 = 29th April 2016
01-JAN-16

SELECT TRUNC(TO_DATE('07-APR-15','DD-MON-YY') + 365, 'YY') FROM DUAL; --7th April + 365
= 6th April 2016
01-JAN-16

SELECT TRUNC(LAST_DAY(TO_DATE('07-APR-15','DD-MON-YY')) -11, 'DD') FROM DUAL; --30th
April -11 = 19th April
19-APR-15

SELECT TRUNC(TO_DATE('07-APR-15','DD-MON-YY') -11, 'DD') FROM DUAL; --7th April -11 =
27th March
27-MAR-15

SELECT TRUNC(LAST_DAY(TO_DATE('07-APR-15','DD-MON-YY')) - 11, 'MM') FROM DUAL; --30th
April -11 = 19th April
01-APR-15

SELECT TRUNC(TO_DATE('07-APR-15','DD-MON-YY') - 11, 'MM') FROM DUAL; --7th April -11 =
27th March
01-MAR-15
```

Does variable without size gives error while creating a table

```
CREATE TABLE TESTSIZE (COLUMN1 VARCHAR2); --ERROR; HAVE TO USE VARCHAR2(SIZE)
CREATE TABLE TESTSIZE (COLUMN1 NUMBER); --ALLOWS NUMBER OF ANY SIZE AND DECIMALS
CREATE TABLE TESTSIZE (COLUMN1 CHAR); --ALLOWS ONLY 1 CHARACTER
```

You need to create a table for a banking application. One of the columns in the table has the following requirements: 1) You want a column in the table to store the duration of the credit period. 2) The data in the column should be stored in a format such that it can be easily added and subtracted with date data type without using conversion functions. 3) The maximum period of the credit provision in the application is 30 days. 4) The interest has to be calculated for the number of days an individual has taken a credit for. Which data type would you use for such a column in the table?

- INTERVAL DAY TO SECOND stores a period of time in terms of days, hours, minutes, and seconds. This data type is useful for representing the precise difference between two datetime values.
- You can perform a number of arithmetic operations on date (DATE), timestamp (TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE) and interval (INTERVAL DAY TO SECOND and INTERVAL YEAR TO MONTH) data.
- It stores duration of the credit as days - The format stored is numeric format, and you know that numeric values can be easily added and subtracted with date data type without using conversion functions (i.e. SELECT SYSDATE - 1 FROM DUAL;)
- The interest has to be calculated for the number of days an individual has taken a credit for, so it will be easy to calculate interest by using the interest rate and duration of the the credit which is numeric format.

RAISE_APPLICATION_ERROR

- Using this procedure, you can associate an error number (-20,000 to -20,999) with custom error message
- No exception name can be associated with this exception

```
ACCEPT var_age NUMBER PROMPT 'Enter your age:'; -- (SQL* Plus function)Accept value from
the user with custom message
DECLARE
    age NUMBER := &var_age;
BEGIN
    IF age < 18 THEN
        RAISE_APPLICATION_ERROR (-20008, 'Age only above 18 are allowed');
    END IF;
    DBMS_OUTPUT.PUT_LINE('You are allowed');
EXCEPTION
    WHEN OTHERS THEN --No name available for the exception we raised
        DBMS_OUTPUT.PUT_LINE(SQLERRM); -- Error message of exception we raised
END;
```

PRAGMA EXCEPTION_INIT

- We can associate an exception name with an Oracle error number.
- It is not mandatory to use PRAGMA EXCEPTION_INIT with RAISE_APPLICATION_ERROR procedure.
- You can use PRAGMA EXCEPTION_INIT with RAISE statement also.

```
DECLARE
    age NUMBER := 17;
    ex_age EXCEPTION; --Declare exception name
    PRAGMA EXCEPTION_INIT(ex_age,-20008) --(exception_name, error_number)
BEGIN
    IF age < 18 THEN
        RAISE_APPLICATION_ERROR (-20008, 'Age only above 18 are allowed');
    END IF;
    DBMS_OUTPUT.PUT_LINE('You are allowed');
EXCEPTION
    WHEN ex_age THEN --Name is available because we are using PRAGMA EXCEPTION_INIT
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
```

Autonomous transactions

- Autonomous transactions allow a single transaction to be subdivided into multiple commit/rollback transactions.
- When an autonomous transaction is called, the original transaction (calling transaction) is temporarily suspended.
- **PRAGMA** is used to provide an instruction to the compiler. Pragmas are defined in the declarative section in PL/SQL.

```
CREATE OR REPLACE PROCEDURE log_errors(p_error_message IN VARCHAR2)
IS
    PRAGMA AUTONOMOUS_TRANSACTION; --PRAGMA in the called program
BEGIN
    INSERT INTO error_logs VALUES(sysdate,p_error_message);
    COMMIT;  --Only above insert will be committed
END;
----
BEGIN
    INSERT INTO..
EXCEPTION
    WHEN OTHERS THEN
        log_errors('Error'); -- Calling procedure; This will be committed
        ROLLBACK;
END;
```

Returning Clause

- Returning clause is used to return the new value after the update or insert or old value after delete.
- It is recommended to use it with delete.

```
DECLARE
    dept_no dept.deptno %TYPE;
BEGIN
    INSERT INTO dept VALUES(70 ,'FOUR','FIVE')
    RETURNING deptno INTO dept_no;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Inserted:' || dept_no);
    DELETE FROM dept WHERE dept_no = 70
    RETURNING deptno INTO dept_no;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Deleted:' || dept_no);
END;
```

AUTHID

- The AUTHID CURRENT_USER is used when you want a piece of code to execute with the privileges of the current user, and not with the privilege of the user who defined the PL/SQL code.
- AUTHID DEFINER is exactly opposite to AUTHID CURRENT_USER. Using this clause is as same as granting public access to the PL/SQL code.
- If no AUTHID clause is specified Oracle will default to AUTHID DEFINER.
- It is suggestible to set AUTHID clause. If not, an intruder may get access to privileges of the definer which an intruder should not get.

```
CREATE OR REPLACE PROCEDURE create_dept(v_deptno NUMBER)
AUTHID CURRENT_USER
IS
BEGIN
   INSERT INTO departments VALUES (v_deptno);
END;
```

PL/SQL Bulk Collect

- Bulk collect is used to reduce the context switching between SQL engine and PL/SQL engine and to improve query performance.
- Bulk collect will reduce the context switching by collecting all the SQL statement calls from PL/SQL program and sending them to SQL engine in just one go and vice versa.
- Bulk collect clause can be used with SELECT INTO, FETCH INTO, RETURNING INTO statements.

Collection of Record

```
DECLARE
    TYPE rc_emp IS RECORD(
        v_empno employees.employee_id%TYPE,
        v_ename employees.first_name%TYPE,
        v_sal employees.salary%TYPE);
TYPE cl_emp IS TABLE OF rc_emp;
v_emp cl_emp;
BEGIN
    SELECT employee_id, first_name, salary
        BULK COLLECT INTO v_emp FROM employees;
    FOR i IN 1..v_emp.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE(v_emp(i).v_empno || v_emp(i).v_ename ||
v_emp(i).v_sal);
    END LOOP;
    FORALL i IN 1..v_emp.COUNT
        INSERT INTO temp_emp (employee_id,first_name,salary)
            VALUES(v_emp(i).v_empno, v_emp(i).v_ename, v_emp(i).v_sal);
END;
```

SELECT INTO

```
DECLARE
    TYPE nt_fName IS TABLE OF VARCHAR2(20);
    fname nt_fName;
BEGIN
    SELECT first_name BULK COLLECT INTO fname FROM employees; --variable should be
a collection
    FOR i IN 1..fname.COUNT
    LOOP
        DBMS_OUTPUT.PUT_LINE(i||fname(i));
    END LOOP;
END;
```

FETCH INTO

```
DECLARE
    CURSOR exp_cur IS
    SELECT first_name FROM employees;
    TYPE nt_fNAME IS TABLE OF VARCHAR2(20);
    fname nt_fNAME;
BEGIN
    OPEN exp_cur;
    FETCH exp_cur BULK COLLECT INTO fname;--
Bulk collect does not need loop
    FOR i IN fname.FIRST..fname.LAST
    LOOP
        DBMS_OUTPUT.PUT_LINE(i||fname(i));
    END LOOP;
    CLOSE exp_cur;
END;
```

Bulk Collect with LIMIT clause

- Whenever we retrieve a large number of records using bulk collect, the program starts consuming lot of memory in order to be fast and efficient. That degrades the performance of the database.
- LIMIT clause will restrict the number of rows fetched.
- LIMIT clause can be only used with FETCH INTO statement.

```
DECLARE
    CURSOR exp_cur IS
    SELECT first_name FROM employees;
    TYPE nt_fNAME IS TABLE OF VARCHAR2(20);
    fname nt_fNAME;
BEGIN
    OPEN exp_cur;
    LOOP
    FETCH exp_cur BULK COLLECT INTO fname LIMIT 10;
    FOR i IN 1..fname.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE(i||fname(i));
    END LOOP;
    EXIT WHEN fname.COUNT = 0;
    END LOOP;
    CLOSE exp_cur;
END;
```

FORALL (Bulk Binding)

- FORALL statement reduces context switches which occur during the execution of a DML statement by sending it in batches instead of one at a time.
- With BULK COLLECT we were fetching data from table and storing it into the collection. But in FORALL statement, we will fetch the data from the collection and store it into the table.
- A FORALL statement can have only one DML statement at a time.

```
FORALL i IN bound_clause --Bound clause decides the number of iteration.
[SAVE EXCEPTIONS] --It helps the DML statements to keep running even when there is
an exception.
                  --Using this is recommended
--DML statement;
```

Lower and Upper bound

- The collection should have consecutive index numbers.
- If an element in the range is missing or was deleted, PL/SQL raises an exception.

```
DECLARE
    TYPE myArray IS TABLE OF NUMBER(2);
    col_var myArray := myArray(9,45,1,24,5,4,7,54,6,23);
BEGIN
    FORALL i IN 1..col_var.COUNT
        INSERT INTO tbl_mulpxn VALUES (col_var(i));
    DBMS_OUTPUT.PUT_LINE(SQL %ROWCOUNT);
END;
```

Indices-of bound

- The indexes need not be consecutive.
- If a subscript in the range does not exist in the collection, that subscript is skipped.
- If collection is an associative array, it must be indexed by PLS_INTEGER.

Evaluate the following SQL statements executed in the given order:

```
ALTER TABLE cust
ADD CONSTRAINT cust_id_pk
PRIMARY KEY(cust_id) DEFERRABLE INITIALLY DEFERRED;
INSERT INTO cust VALUES (1,'RAJ'); --row 1
INSERT INTO cust VALUES (1,'SAM'); --row 2
COMMIT;

SET CONSTRAINT cust_id_pk IMMEDIATE;
INSERT INTO cust VALUES (1,'LATA'); --row 3
INSERT INTO cust VALUES (2,'KING'); --row 4
COMMIT;

Row1 and Row2 will give constraint error. Row3 and Row4 will be inserted.
```

Can ORDER BY clause use column alias to sort data?

Only ORDER BY can use column alias. WHERE, GROUP BY, HAVING clause cannot use column alias.
Ex: `SELECT employee_id, salary sal FROM employees ORDER BY sal;`

Evaluate the SQL statements:

```
CREATE TABLE new_order (orderno NUMBER(4), booking_date TIMESTAMP WITH LOCAL TIME ZONE);
```

The database is located in San Francisco where the time zone is -8:00. The user is located in New York where the time zone is -5:00. A New York user inserts the following record:

```
INSERT INTO new_order VALUES (1, TO_TIMESTAMP_TZ('2007-05-10 6:00:00 -5:00','YYYY-MM-DD
HH:MI:SS TZH:TZM'));
```

When the San Francisco user selects the row, booking_date is displayed as 2007-05-10 3.00.00.000000
When the Indian user selects the row, booking_date is displayed as 2007-05-10 16.30.00.000000

Different ways to insert DEFAULT value into table

```
INSERT INTO TEST_TABLE VALUES(1,NULL);
INSERT INTO TEST_TABLE VALUES(2,DEFAULT);
INSERT INTO (SELECT ids FROM TEST_TABLE) VALUES(3);
```

What happens to other db objects when an associated table is dropped?

Dropping a table removes the table definition from the data dictionary. All rows of the table are no longer accessible. All indexes and triggers associated with a table are dropped. All views and PL/SQL program units dependent on a dropped table remain, yet become invalid (not usable).

What are the limitations of a CHECK constraint?

- Subqueries cannot be used within your Oracle check constraints.
- Check constraint cannot reference another columns.
- SYSDATE, CURRVAL, NEXTVAL, LEVEL, ROWID, UID, USER or USERENV cannot be referenced with Oracle check constraint

How to see the past operations performed in a table?

`VERSIONS BETWEEN TIMESTAMP MINVALUE AND MAXVALUE;` returns all the committed occurrences of the rows for a query of an object, while NOT displaying the UNCOMMITTED row versions.

```
CREATE TABLE digits
(id NUMBER(2),
description VARCHAR2(15));

INSERT INTO digits VALUES (1,'ONE');

COMMIT;

UPDATE digits
SET description ='TWO'
WHERE id=1;

COMMIT;

INSERT INTO digits VALUES (2,'TWO');

COMMIT;

DELETE FROM digits;

COMMIT;
```

DESCRIPTION	VERSIONS_XID	VERSIONS_OPERATION
TWO	07000A008C040000	D
TWO	030007001B030000	I
TWO	06001E003E030000	U
ONE		

```
SELECT DESCRIPTION, VERSIONS_STARTTIME, VERSIONS_STARTSCN, VERSIONS_ENDTIME, VERSIONS_ENDSCN,
VERSIONS_XID, VERSIONS_OPERATION
FROM DIGITS
VERSIONS BETWEEN TIMESTAMP MINVALUE AND MAXVALUE;

SELECT DESCRIPTION, VERSIONS_STARTTIME, VERSIONS_STARTSCN, VERSIONS_ENDTIME, VERSIONS_ENDSCN,
VERSIONS_XID, VERSIONS_OPERATION
FROM DIGITS
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE;

SELECT DESCRIPTION, VERSIONS_STARTTIME, VERSIONS_STARTSCN, VERSIONS_ENDTIME, VERSIONS_ENDSCN,
VERSIONS_XID, VERSIONS_OPERATION
FROM DIGITS
VERSIONS BETWEEN TIMESTAMP
    SYSTIMESTAMP - INTERVAL '10' MINUTE AND
    SYSTIMESTAMP - INTERVAL '1' MINUTE;


UPDATE employees SET salary =
    (SELECT salary FROM employees
    AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '2' MINUTE)
    WHERE last_name = 'Chung')
    WHERE last_name = 'Chung';
```

Fetch all objects modified or created recently

```
SELECT * FROM DBA_OBJECTS WHERE TRUNC(CREATED) = SYSDATE ORDER BY CREATED DESC;

SELECT * FROM DBA_OBJECTS WHERE TRUNC(LAST_DDL_TIME) = SYSDATE ORDER BY LAST_DDL_TIME DESC;
```

```
DECLARE
    TYPE myArray IS TABLE OF NUMBER(2);
    col_var myArray := myArray(9,45,1,24,5,4,7,54,6,23);
BEGIN
    col_var.DELETE(3 , 6);
    FORALL i IN INDICES OF col_var
        INSERT INTO tbl_mulpxn VALUES (col_var(i));
    DBMS_OUTPUT.PUT_LINE(SQL %ROWCOUNT);
END;
----
DECLARE
    TYPE myArray IS TABLE OF NUMBER(2);
    col_var myArray := myArray(9,45,1,24,5,4,7,54,6,23);
BEGIN
    col_var.DELETE(3 , 6);
    FORALL i IN INDICES OF col_var BETWEEN 1 AND 10
        INSERT INTO tbl_mulpxn VALUES (col_var(i));
    DBMS_OUTPUT.PUT_LINE(SQL %ROWCOUNT);
END;
```

Values-of bound

- It requires two collection; source collection and indexing collection
- The subscripts for the FORALL indexing collection are taken from the values of the elements in source collection
- The indexing collection must be a nested table, or an associative array
- The elements of the indexing collection must be of either PLS_INTEGER or BINARY_INTEGER
- If it is associate array, then it must be indexed by PLS_INTEGER or BINARY_INTEGER
- Indexing collection is a group of indexes that the FORALL statement can loop through

```
DECLARE
    TYPE myArray IS TABLE OF NUMBER(2);--Source collection
    src_col myArray := myArray(9,45,1,24,5,4,7,54,6,23);
    TYPE yourArray IS TABLE OF PLS_INTEGER; --Indexing collection
    idx_col yourArray:=yourArray();
BEGIN
    idx_col.EXTEND(2);
    idx_col(1):=4; --24 inserted here; src_col(4)
    idx_col(2):=8; --54 inserted here; src_col(8)

    FORALL i IN VALUES OF idx_col --idx_col(1) = 4; i =4
        INSERT INTO tbl_mulpxn VALUES (src_col(i)); --src_col(4) = 24
    DBMS_OUTPUT.PUT_LINE(SQL %ROWCOUNT);
END;
```

SQL%BULK_ROWCOUNT

The SQL%BULK_ROWCOUNT cursor attribute gives granular information about the rows affected by each iteration of the FORALL statement. Every row in the driving collection has a corresponding row in the SQL%BULK_ROWCOUNT cursor attribute. In the below code, we can see that rows affected for the username "BANANA" is zero.

```
CREATE TABLE bulk_rowcount_test AS
SELECT *
FROM   all_users;
----
DECLARE
  TYPE t_array_tab IS TABLE OF VARCHAR2(30);
  l_array t_array_tab := t_array_tab('SCOTT', 'SYS',
                                     'SYSTEM', 'DBSNMP', 'BANANA');
BEGIN
  -- Perform bulk delete operation.
  FORALL i IN l_array.FIRST .. l_array.LAST
    DELETE FROM bulk_rowcount_test
    WHERE username = l_array(i);

  -- Report affected rows.
  FOR i IN l_array.FIRST .. l_array.LAST LOOP
    DBMS_OUTPUT.PUT_LINE('Element: ' || RPAD(l_array(i), 15, ' ') ||
      ' Rows affected: ' || SQL %BULK_ROWCOUNT(i));
  END LOOP;
END;
/
Element: SCOTT           Rows affected: 1
Element: SYS             Rows affected: 1
Element: SYSTEM          Rows affected: 1
Element: DBSNMP          Rows affected: 1
Element: BANANA          Rows affected: 0
```

Native Dynamic SQL

Execute Immediate

- Using Execute Immediate, we can parse and execute any SQL statement or a PL/SQL block dynamically in Oracle database
- Use of bind variable: Security against SQL injections and performance enhancement by reducing hard parsing.
- Generally dynamic SQL is slower than static SQL so it should not be used unless absolutely necessary.
- Main advantage of dynamic SQL is that it allows to perform DDL commands that are not supported directly within PL/SQL

Single Row Queries

```
DECLARE
    l_sql VARCHAR2(100);
    l_ename emp.ename%TYPE;
BEGIN
    l_sql:='SELECT ename FROM emp WHERE
emp_no=1234';
    EXECUTE IMMEDIATE l_sql INTO l_ename;
END;
```

DDL Operations

```
BEGIN
    EXECUTE IMMEDIATE 'TRUNCATE TABLE
my_table';
END;
```

PL/SQL Block using EXECUTE IMMEDIATE

```
DECLARE
    plsql_blk VARCHAR2(400);
BEGIN
    plsql_blk := 'DECLARE
    var_user VARCHAR2(10);
    BEGIN
        SELECT user INTO var_user
FROM DUAL;

DBMS_OUTPUT.PUT_LINE(''User:'' ||
var_user);
    END;';
    EXECUTE IMMEDIATE plsql_blk;
END;
```

Bind Variable

```
DECLARE
    l_sql VARCHAR2(100);
    l_ename employees.first_name%TYPE;
BEGIN
    l_sql := 'SELECT first_name FROM employees WHERE employee_id = :empno and department_id
= :deptno'; --:empno is a bind variable
    EXECUTE IMMEDIATE l_sql INTO l_ename USING 100,90;-- passing 100 into empno and 90 into
deptno as bind variable
    DBMS_OUTPUT.PUT_LINE(l_ename);
END;
```

BULK COLLECT INTO with EXECUTE IMMEDIATE

```
DECLARE
    TYPE nt_Fname IS TABLE OF VARCHAR2(60);
    fname nt_Fname;
    sql_qry VARCHAR2(150);
BEGIN
    sql_qry := 'SELECT first_name FROM
employees';
    EXECUTE IMMEDIATE sql_qry BULK COLLECT
INTO fname;
END;
```

SAVE EXCEPTIONS and SQL%BULK_EXCEPTION

The following code creates a collection with 100 rows, but sets the value of rows 50 and 51 to NULL. Since the "exception_test" table does not allow nulls, these rows will result in an exception. The SAVE EXCEPTIONS clause allows the bulk operation to continue past any exceptions, but if any exceptions were raised , it will jump to the exception handler once all the operations are complete. In this case, the exception handler just loops through the SQL% BULK_EXCEPTION cursor attribute to see what errors occurred.

```
CREATE TABLE exception_test (
  id  NUMBER(10) NOT NULL);
----
DECLARE
  TYPE t_tab IS TABLE OF exception_test%ROWTYPE;
  l_tab t_tab := t_tab();
  l_error_count NUMBER(10);
  ex_dml_errors EXCEPTION;
  PRAGMA EXCEPTION_INIT(ex_dml_errors, -24381);
BEGIN
  -- Fill the collection.
  FOR i IN 1..100 LOOP
    l_tab.EXTEND;
    l_tab(i).id := i;
  END LOOP;
  -- Cause a failure.
  l_tab(50).id := NULL;
  l_tab(51).id := NULL;

  EXECUTE IMMEDIATE 'TRUNCATE TABLE exception_test';
  -- Perform a bulk operation.
  BEGIN
    FORALL i IN l_tab.FIRST..l_tab.LAST SAVE EXCEPTIONS
      INSERT INTO exception_test
      VALUES l_tab(i);
  EXCEPTION
    WHEN ex_dml_errors THEN
      l_error_count := SQL%BULK_EXCEPTIONS.COUNT;
      DBMS_OUTPUT.PUT_LINE('Number of failures: ' || l_error_count);
      FOR i IN 1 .. l_error_count LOOP
        DBMS_OUTPUT.PUT_LINE('Error: ' || i ||
          ' Array Index: ' || SQL%BULK_EXCEPTIONS(i).ERROR_INDEX ||
          ' Error Code: ' || SQL%BULK_EXCEPTIONS(i).ERROR_CODE ||
          ' Message1: ' || SQLERRM(SQL%BULK_EXCEPTIONS(i).ERROR_CODE) ||
          ' Message2: ' || SQLERRM(-SQL%BULK_EXCEPTIONS(i).ERROR_CODE)) ;
      END LOOP;
  END;
END;
/

Number of failures: 2
Error: 1 Array Index: 50 Error Code: 1400 Message1:  -1400: non-ORACLE exception  Message2:
ORA-01400: cannot insert NULL into ()
Error: 2 Array Index: 51 Error Code: 1400 Message1:  -1400: non-ORACLE exception  Message2:
ORA-01400: cannot insert NULL into ()
```

UTL_MAIL

The UTL_MAIL package was introduced in Oracle 10g to provide a simple API to allow email to be sent from PL/SQL.

PIVOT

The PIVOT operator takes data in separate rows, aggregates it and converts it into columns. The UNPIVOT operator converts column-based data into separate rows.

```
SELECT *
FROM   (SELECT department_id, salary
        FROM   employees)
PIVOT  (SUM(salary) AS sum_salary
FOR (department_id) IN (10,50,60));
--FOR work like GROUP BY and IN will
filter the result
----
SELECT *
FROM   (SELECT job_id, department_id,
  salary FROM   employees)
PIVOT  (SUM(salary) AS sum_salary
FOR (department_id) IN (10,50,60));--
break it down by
     job_id
```

10_SUM_SALARY	50_SUM_SALARY	60_SUM_SALARY
4400	234600	28800

JOB_ID	10_SUM_SALARY	50_SUM_SALARY	60_SUM_SALARY
AC_ACCOUNT	NULL	NULL	NULL
AC_MGR	NULL	NULL	NULL
AD_ASST	4400	NULL	NULL
IT_PROG	NULL	NULL	28800
SH_CLERK	NULL	96450	NULL
ST_CLERK	NULL	83550	NULL
ST_MAN	NULL	54600	NULL

? Retrieve a table after drop

```
DROP TABLE test_table;
FLASHBACK TABLE test_table TO BEFORE DROP;
FLASHBACK TABLE test_table TO BEFORE DROP RENAME TO test_emp; -- Data moved
toanother table. Old table is still in recyclebin
```

The clause TO BEFORE DROP retrieves back the dropped table and all the constraints except referential integrity constraints that reference other tables are retrieved automatically after the table is flashed back. However, the tables dropped using PURGE option cannot be retrieved back. TO BEFORE DROP clause is also unable to recover tables dropped TRUNCATE TABLE command.

? You want to add a constraint on the CUST_FIRST_NAME column of the CUSTOMERS table so that alphabets but not numbers can be inserted in the column.

```
ALTER TABLE CUSTOMERS
ADD CONSTRAINT cust_f_name
CHECK (REGEXP_LIKE(cust_first_name,'^[[:alpha:]]*$'))
NOVALIDATE ;
```

? Use a view without storing it as a view?

```
INSERT INTO (SELECT PRODUCT_NAME,PRODUCT_STATUS FROM PRODUCT_INFORMATION WITH CHECK
OPTION)
VALUES('FOOD1','GOOD1'); -- It will insert into the table
-------
INSERT INTO (SELECT PRODUCT_NAME,PRODUCT_STATUS FROM PRODUCT_INFORMATION WHERE
PRODUCT_NAME = 'FOOD3' WITH CHECK OPTION)
VALUES('FOOD1','GOOD1'); -- It will not insert into the table as it does not
have the rows in the select statement which is going to be inserted. Hence
violated with check option.
```

? What is CASE_NOT_FOUND Exception?

The CASE_NOT_FOUND exception (ORA-06592) is raised when none of the choices in the WHEN clauses of a CASE statement is selected, and there is no ELSE clause.

? What will happen if cursor is not open and we tried to do %NOTFOUND?

It will throw an exception 'INVALID CURSOR'

? What error will come if you try to assign a larger value than the declared one?

```
DECLARE
    names VARCHAR2(10);
BEGIN
    names:= '1234567891011'; --ORA-06502: PL/SQL: numeric or value error:
character string buffer too small
END;
```

? What will be the output of this block (NOCOPY)?

```
DECLARE
   n NUMBER := 10;
   PROCEDURE do_something (
      n1 IN NUMBER,
      n2 IN OUT NUMBER,
      n3 IN OUT NOCOPY NUMBER) IS
   BEGIN
      n2 := 20; -- Pass by value; If there are no exception,value is
assigned back to n after procedure is complete.
      DBMS_OUTPUT.PUT_LINE(n1);   -- prints 10
      n3 := 30; -- Pass by reference; referenced variable n got changed to
30 even if any exception happens or before completing the procedure
      DBMS_OUTPUT.PUT_LINE(n1);   -- prints 30
   END;
BEGIN
   do_something(n, n, n);
   DBMS_OUTPUT.PUT_LINE(n);   -- prints 20
END;
```

? What are the conditions a column can be dropped?

- The column would be dropped provided at least one or more columns remain in the table.
- The column can be dropped even if it is part of a composite PRIMARY KEY provided the CASCADE option is used.

? What would result COUNT(NULL)?

```
SELECT COUNT(DEPARTMENT_ID) FROM EMPLOYEES WHERE DEPARTMENT_ID IS NULL;--0
SELECT COUNT(NVL(DEPARTMENT_ID,0)) FROM EMPLOYEES WHERE DEPARTMENT_ID IS NULL; --1
SELECT COUNT(NULL) FROM DUAL; --0
```

? Different syntax to delete from table?

```
DELETE PRODUCT_INFORMATION WHERE PRODUCT_NAME = 'FOOD1';
DELETE FROM PRODUCT_INFORMATION WHERE PRODUCT_NAME = 'FOOD2';
```

? Types of functions in Oracle?

String/Char Functions
Numeric/Math Functions
Date/Time Functions
Conversion Functions
Analytic Functions
Aggregate Functions
Advanced Functions

? SubProgram overloading in Oracle

Subprograms with same name and different type or number of parameters.

? What is the maximum index allowed?

Unlimited

ID	CUSTOMER_ID	PRODUCT_CODE_A	PRODUCT_CODE_B	PRODUCT_CODE_C	PRODUCT_CODE_D
1	101	10	20	30	
2	102	40		50	
3	103	60	70	80	90
4	104	100			

```
SELECT *
FROM   unpivot_test
UNPIVOT INCLUDE NULLS (quantity FOR
product_code IN (product_code_a AS
'A', --default is
     exclude NULLS
product_code_b AS 'B',
product_code_c AS 'C',
product_code_d AS 'D'));
```

ID	CUSTOMER_ID	P	QUANTITY
1	101	A	10
1	101	B	20
1	101	C	30
1	101	D	
2	102	A	40
2	102	B	
2	102	C	50
2	102	D	
3	103	A	60
3	103	B	70
3	103	C	80
3	103	D	90
4	104	A	100
4	104	B	
4	104	C	
4	104	D	

UTL_FILE

In Oracle PL/SQL, UTL_FILE is an Oracle supplied package which is used for file operations (read and write) in conjunction with the underlying operating system.

```
DECLARE
    tc_logfile UTL_FILE.FILE_TYPE;
    filedir VARCHAR2(10);
    v_dir VARCHAR2(100);
    read_line VARCHAR2(200);
BEGIN
    filedir := 'E:\';
    v_dir := 'CREATE OR REPLACE DIRECTORY TEMP_TEXT as '''|| filedir||'''';
    EXECUTE IMMEDIATE v_dir;
    tc_logfile := UTL_FILE.FOPEN('TEMP_TEXT','TestFile.txt','W'); -- A to append; W
to write; R to read
    UTL_FILE.PUT_LINE(tc_logfile, 'STARTING..' || SYSTIMESTAMP);
    UTL_FILE.PUT_LINE(tc_logfile, 'This is a test file');
    IF UTL_FILE.IS_OPEN(tc_logfile) THEN
        UTL_FILE.FCLOSE(tc_logfile);
    END IF;
    tc_logfile := UTL_FILE.FOPEN('TEMP_TEXT','TestFile.txt','R');
    LOOP
        BEGIN
            UTL_FILE.GET_LINE(tc_logfile,read_line);
            DBMS_OUTPUT.PUT_LINE(read_line);
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                EXIT;
        END;
    END LOOP;
    UTL_FILE.FCLOSE(tc_logfile);
END;
```

LISTAGG

The LISTAGG analytic function was introduced in Oracle 11g Release 2, making it very easy to aggregate strings. The nice thing about this function is it also allows us to order the elements in the concatenated list.

```
SELECT department_id, LISTAGG(first_name, ',')
WITHIN GROUP (ORDER BY first_name) AS employees
FROM   employees
GROUP BY department_id;
```

External tables

```
CREATE OR REPLACE DIRECTORY directory_name AS 'C\Users'; -- Directory object
CREATE TABLE students(name VARCHAR2(20),college VARCHAR2(20), major VARCHAR2(20))
    ORGANIZATION EXTERNAL(
    TYPE ORACLE_LOADER
--The ORACLE_LOADER access driver is the default. It can perform only data loads,
and the data must come from text datafiles. Loads from external tables to internal
tables are done by reading from the external tables' text-only datafiles.
--The ORACLE_DATAPUMP access driver can perform both loads and unloads. The data
must come from binary dump files. Loads to internal tables from external tables are
done by fetching from the binary dump files. Unloads from internal tables to
external tables are done by populating the external tables' binary dump files.
    DEFAULT DIRECTORY directory_name
    ACCESS PARAMETERS (
        RECORD DELIMITED BY NEWLINE
        FIELDS TERMINATED BY ',' -- comma separated
        MISSING FIELD VALUE ARE NULL -- if any data is missing, add them as NULL
        ( name CHAR(20),
          collage CHAR(20),
          major CHAR(20)
        )
    )
    LOCATION ('major.txt') -- filename
)
REJECT LIMIT UNLIMITED; -- stop the process if error comes more than specified
value.
```

10	Jennifer
20	Michael,Pat
30	Alexander,Den,Guy,Karen
40	Susan

Import data from Excel to Oracle using SQL Developer

```
Right click the table you want to insert data -> select import data -> select excel
file -> click open
```

XMLELEMENT

The XMLELEMENT function is the basic unit for turning column data into XML fragments.

```
SQL> SELECT XMLELEMENT("name", e.first_name) AS employee FROM HR.employees e
     WHERE e.employee_id = 160;
---------------------------------------------------------------------------
<name>Louise</name>
----
SQL> SELECT XMLELEMENT("employee",
         XMLELEMENT("employee_no", e.employee_id),
         XMLELEMENT("name", e.first_name)
       ) AS employee
FROM   employees e
WHERE  e.employee_id = 160;
---------------------------------------------------------------------------
<employee><employee_no>160</employee_no><name>Louise</name></employee>
----
SQL> SELECT XMLELEMENT("employee",
  2          XMLATTRIBUTES(  --XMLATRIBUTES function converts column data into
attributes of the parent element
  3            e.employee_id AS "employee_id",
  4            e.first_name AS "name")
  5        ) AS employee
  6  FROM   employees e
  7  WHERE  e.employee_id = 160;
---------------------------------------------------------------------------
<employee employee_id="160" name="Louise"></employee>
----
SQL> SELECT XMLELEMENT("employee",
  2          XMLFOREST( --Like XMLATTRIBUTES, the XMLFOREST function allows you to
process multiple columns at once.
  3            e.employee_id AS "employee_id",
  4            e.first_name AS "name")
  5        ) AS employee
  6  FROM   employees e
  7  WHERE  e.employee_id = 160;
---------------------------------------------------------------------------
<employee><employee_id>160</employee_id><name>Louise</name></employee>
----
SQL> SELECT XMLAGG( --XMLAGG function allows to aggregate separate
fragments(multiple rows of data) into a single fragment
  2          XMLELEMENT("emp",
  3            XMLFOREST(
  4              e.employee_id AS "empid",
  5              e.first_name AS "name")
  6            )
  7        ) AS employees
  8  FROM   employees e
  9  WHERE  e.department_id = 50;
-------------------------------------------------------------------
<emp><empid>120</empid><name>Matthew</name></emp><emp><empid>121</empid><name>
Adam</name></emp><emp><empid>122</empid><name>Payam</name></emp><emp>
<empid>123</empid><name>Shanta</name></emp><emp><empid>123</empid><name>Kevin</name>
</emp>
```

SQL Loader

SQL*Loader is a bulk loader utility used for moving data from external files into the Oracle database.

SQL Loader Control File

```
LOAD DATA
    INFILE 'C:\text_file.csv' --input file
    BADFILE 'C:\textfile_bad.bad'  --contains the records that are rejected by sql
loader or oracle database becuase of invalid format. After a data is accepted by sql
loader, it is sent to oracle database for insertion
    DISCARDFILE 'C:\textfile_discard.dis'   --records which are filtered out of the
load because they don't meet the criteria specified in WHEN clause
TRUNCATE INTO TABLE table_name  --TRUNCATE to delete previous data in table
--INSERT: Loads only if the target table is empty
--APPEND: Loads rows if the target table is empty or not
--REPLACE: First it delete the rows in the existing table and then Loads the data
--TRUNCATE: First it truncate the table and then Loads the data

WHEN OBJECT_TYPE <> 'INDEX' --i don't need any record with 'INDEX' in it
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY "#"  --delimited by , or #
TRAILING NULL COLS  --if the last column is empty, then treat this as NULL value;
otherwise SQL Loader will treat this record as bad if the last column is empty
(deptno,    --column names in table
 dname,
 jdate date'mm/dd/yyyy', --formating the date
 loc"TRIM(:OBJECT_TYPE)")   -- TRIM the white space if any

 or
(deptno position(1:3), dname position(4:8), jdate position(9:18), loc
position(19:22))

or
(deptno "deptno+100", --add 100 to the deptno
dname "upper(:dname)",  --upper case
jdate,
loc "decode(:loc,'Delhi','New Delhi',:loc)")

sqlldr control=C:\textfile_control.ctl log=C:\textfile_log.log  --given the datafile
in INFILE of control file

or

sqlldr datafile=C:\text_file.csv control=C:\textfile_control.ctl log=C:
\textfile_log.log
```

SELECT AS OF Clause

```
SELECT * FROM table_name
AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL '1' MINUTE);
```

Interview (9) Questions and Answers

? How the execution of the correlated subqueries works?

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query. The inner query executes after the outer query returns the row. Each row returned by the outer query is evaluated for the results returned by the inner query.

```
SELECT employee_id, last_name, job_id, department_id
FROM employees outer
WHERE EXISTS ( SELECT ’X’
     FROM employees
     WHERE manager_id = outer.employee_id);
---------
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT ’X’
     FROM employees
     WHERE department_id = d.department_id);
```

? Right angle triangle using numbers?

```
SELECT TO_NUMBER(RPAD('9',ROWNUM*2,'9'))
     FROM ALL_OBJECTS
     WHERE ROWNUM <= 19;
---------
SELECT ROWNUM, 'X', 'Y' FROM ALL_OBJECTS WHERE ROWNUM < 1001;
```

TO_NUMBER(RPAD('9',ROWNUM*2,'9'))
99
9999
999999
99999999
9999999999
999999999999

ROWNUM	'X'	'Y'
1	x	y
2	x	y
3	x	y
4	x	y

? How much storage a NUMBER datatype takes?

When Oracle stores a number, it does so by storing as little as it can in order to represent that number. It does this by storing the significant digits, an exponent used to place the decimal place and information regarding the sign of the number (positive or negative). So, the more digits a number contains, the more storage it consumes.

```
CREATE TABLE T ( X NUMBER, Y NUMBER );
---------
INSERT INTO T ( X )
  SELECT TO_NUMBER(RPAD('9',ROWNUM*2,'9'))
    FROM ALL_OBJECTS
    WHERE ROWNUM <= 19;
---------
UPDATE T SET Y = X+1;
---------
SET NUMFORMAT 99999999999999999.99
SET LINE  120
---------
SELECT X, Y, VSIZE(X), VSIZE(Y)
 FROM T ORDER BY X;
```

X	Y	VSIZE(X)	VSIZE(Y)
99	100	2	2
9999	10000	3	2
999999	1000000	4	2
99999999	100000000	5	2
9999999999	10000000000	6	2
999999999999	1000000000000	7	2
99999999999999	100000000000000	8	2

```
CREATE TABLE T ( X NUMBER, Y NUMBER );
---------
INSERT INTO T ( X )
  SELECT TO_NUMBER(RPAD('1',ROWNUM*2,'0'))
    FROM ALL_OBJECTS
    WHERE ROWNUM <= 19;
---------
UPDATE T SET Y = X+1;
---------
SET NUMFORMAT 99999999999999999.99
SET LINE  120
---------
SELECT X, Y, VSIZE(X), VSIZE(Y)
 FROM T ORDER BY X;
```

X	Y	VSIZE(X)	VSIZE(Y)
10	11	2	2
1000	1001	2	3
100000	100001	2	4
10000000	10000001	2	5
1000000000	1000000001	2	6
100000000000	100000000001	2	7
10000000000000	10000000000001	2	8

```
CREATE TABLE T ( X NUMBER, Y NUMBER );
---------
INSERT INTO T ( X )
  SELECT TO_NUMBER(RPAD('0',ROWNUM*2,'1'))
    FROM ALL_OBJECTS
    WHERE ROWNUM <= 19;
---------
UPDATE T SET Y = X+1;
---------
SET NUMFORMAT 99999999999999999.99
SET LINE  120
---------
SELECT X, Y, VSIZE(X), VSIZE(Y)
 FROM T ORDER BY X;
```

X	Y	VSIZE(X)	VSIZE(Y)
1	2	2	2
111	112	3	3
11111	11112	4	4
1111111	1111112	5	5
111111111	111111112	6	6
11111111111	11111111112	7	7
1111111111111	1111111111112	8	8

```
CREATE TABLE T ( X NUMBER, Y NUMBER );
---------
INSERT INTO T ( X )
  SELECT TO_NUMBER(RPAD('0',ROWNUM*2,'0'))
    FROM ALL_OBJECTS
    WHERE ROWNUM <= 19;
--------
UPDATE T SET Y = X+1;
---------
SET NUMFORMAT 99999999999999999.99
SET LINE  120
---------
SELECT X, Y, VSIZE(X), VSIZE(Y)
 FROM T ORDER BY X;
```

X	Y	VSIZE(X)	VSIZE(Y)
0	1	1	2
0	1	1	2
0	1	1	2
0	1	1	2
0	1	1	2
0	1	1	2
0	1	1	2

? How to convert a string to CLOB?

```
CREATE TABLE product_reviews(
  product_id       NUMBER    PRIMARY KEY,
  product_review   CLOB);

INSERT INTO product_reviews VALUES (1, TO_CLOB('Imagine this is a long string of chars.'));

INSERT INTO product_reviews VALUES (2, EMPTY_CLOB());

INSERT INTO product_reviews VALUES (3, NULL);
```

? What result will these below queries give?

```
SELECT TO_CHAR(1123.90 ,'$9,999') FROM DUAL; -- $1,124
SELECT TO_CHAR(11235.90 ,'$9,999') FROM DUAL; --#######
```

Regular Expressions

- Enables you to search for patterns in string
- Regular expression functions can result in full-table scans that lead to horribly inefficient queries.

SQL Element	Description
REGEXP_LIKE	Searches a character column for a pattern. Use this function in the WHERE clause of a query to return rows matching a regular expression. The condition is also valid in a constraint or as a PL/SQL function returning a boolean. The following WHERE clause filters employees with a first name of Steven or Stephen: `WHERE REGEXP_LIKE(first_name, '^Ste(v|ph)en$')`
REGEXP_REPLACE	Searches for a pattern in a character column and replaces each occurrence of that pattern with the specified string. The following function puts a space after each character in the country_name column: `REGEXP_REPLACE(country_name, '(.)', '\1 ')`
REGEXP_INSTR	Searches a string for a given occurrence of a regular expression pattern and returns an integer indicating the position in the string where the match is found. You specify which occurrence you want to find and the start position. For example, the following performs a boolean test for a valid email address in the emailcolumn: `REGEXP_INSTR(email, '\w+@\w+(\.\w+)+') > 0`
REGEXP_SUBSTR	Returns the substring matching the regular expression pattern that you specify. The following function uses the x flag to match the first string by ignoring spaces in the regular expression: `REGEXP_SUBSTR('oracle', 'o r a c l e', 1, 1, 'x')`

```
REGEXP_LIKE(<source_string>,<pattern>,<match_parameter>)

REGEXP_REPLACE(<source_string>,<pattern>,<replace_string>,
<position>,<occurrence>,<match_parameter>)

REGEXP_INSTR(<source_string>,<pattern>[[,<start_position>][, <occurrence>]
[,<return_option>][,<match_parameter>][,<sub_expression>]])

REGEXP_SUBSTR(source_string,pattern[, position [, occurrence[, match_parameter]]])


REGEXP_COUNT(<source_string>,<pattern>[[,<start_position>],[<match_parameter>]])
```

Metacharacters

Syntax	Description	Example
.	Matches any character in the database character set.	The expression **a.b** matches the strings **abb**, **acb**, and **adb**, but does not match **acc**.
+	Matches one or more occurrences of the preceding subexpression.	The expression **a+** matches the strings **a**, **aa**, and **aaa**, but does not match **bbb**.
?	Matches zero or one occurrence of the preceding subexpression.	The expression **ab?c** matches the strings **abc** and **ac**, but does not match **abbc**.
*****	Matches zero or more occurrences of the preceding subexpression.	The expression **ab*c** matches the strings **ac**, **abc**, and **abbc**, but does not match **abb**.
{m}	Matches exactly ***m*** occurrences of the preceding subexpression.	The expression **a{3}** matches the strings **aaa**, but does not match **aa**.
{m,}	Matches at least ***m*** occurrences of the preceding subexpression.	The expression **a{3,}** matches the strings **aaa** and **aaaa**, but does not match **aa**.
{m,n}	Matches at least ***m***, but not more than ***n*** occurrences of the preceding subexpression.	The expression **a{3,5}**matches the strings **aaa**, **aaaa**, and **aaaaa**, but does not match **aa**.
[…]	Matches any single character in the list within the brackets.	The expression **[abc]** matches the first character in the strings **all**, **bill**, and **cold**, but does not match any characters in **doll**.
[^ …]	Matches any single character not in the list within the brackets. Characters not in the non-matching character list are returned as a match.	The expression **[^abc]** matches the character **d** in the string **abcdef**, but not the character **a**, **b**, or **c**. The expression **[^abc]+** matches the sequence **def** in the string **abcdef**, but not **a**, **b**, or **c**. The expression **[^a-i]**excludes any character between **a** and **i** from the search result. This expression matches the character **j** in the string **hij**, but does not match any characters in the string **abcdefghi**.
|	Matches one of the alternatives.	The expression **a|b** matches character **a** or character **b**.
(…)	Treats the expression within parentheses as a unit. The subexpression can be a string of literals or a complex expression containing operators.	The expression **(abc)?def** matches the optional string **abc**, followed by **def**. Thus, the expression matches **abcdefghi** and **def**, but does not match **ghi**.
\*n*	Matches the nth preceding subexpression, that is, whatever is grouped within parentheses, where ***n*** is an integer from 1 to 9. The parentheses cause an expression to be remembered; a backreference refers to it.	The expression **(abc|def)xy\1** matches the strings **abcxyabc** and **defxydef**, but does not match **abcxydef** or **abcxy**. the expression **^(.*)\1$** matches a line consisting of two adjacent instances of the same string.
\	Treats the subsequent metacharacter in the expression as a literal. Use a backslash (\\) to search for a character that is normally treated as a metacharacter. Use consecutive backslashes (\\) to match the backslash literal itself.	The expression **\\+** searches for the plus character (**+**). It matches the plus character in the string **abc+def**, but does not match **abcdef**.
^	Matches the beginning of a string (default). In multiline mode, it matches the beginning of any line within the source string.	The expression **^def** matches **def** in the string **defghi** but does not match **def** in **abcdef**.
$	Matches the end of a string (default). In multiline mode, it matches the beginning of any line within the source string.	The expression **def$** matches **def** in the string **abcdef** but does not match **def** in the string **defghi**.
[:*class*:]	Matches any character belonging to the specified POSIX character ***class***.	The expression **[[:upper:]]+** searches for one or more consecutive uppercase characters. This expression matches **DEF** in the string **abcDEFghi** but does not match the string **abcdefghi**.

Posix Characters

[:alnum:]	Alphanumeric characters
[:alpha:]	Alphabetic characters
[:blank:]	Blank space characters
[:cntrl:]	Control characters (non-printing)
[:digit:]	Numeric digits
[:graph:]	Any [:punct:], [:upper:], [:lower:], [:digit:] chars
[:lower:]	Lowercase alphabetic characters
[:upper:]	Uppercase alphabetic characters
[:print:]	Printable characters
[:punct:]	Punctuation characters
[:space:]	Space characters(non-printing), such as carriage return, newline, vertical tab and form feed
[:xdigit:]	Hexadecimal characters

Reg. Exp.	Matches . . .	Example
\d	A digit character. It is equivalent to the POSIX class **[[:digit:]]**.	The expression **^\\(\d{3}\\) \d{3}-\d{4}$**matches **(650) 555-1212** but does not match **650-555-1212**.
\D	A non-digit character. It is equivalent to the POSIX class **[^[:digit:]]**.	The expression **\w\d\D** matches **b2b** and **b2_**but does not match **b22**.
\w	A word character, which is defined as an alphanumeric or underscore (_) character. It is equivalent to the POSIX class **[[:alnum:]_]**. Note that if you do not want to include the underscore character, you can use the POSIX class **[[:alnum:]]**.	The expression **\w+@\w+(\.\w+)+** matches the string **jdoe@company.co.uk** but not the string **jdoe@company**.
\W	A non-word character. It is equivalent to the POSIX class **[^[:alnum:]_]**.	The expression **\w+\W\s\w+** matches the string **to: bill** but not the string **to bill**.
\s	A whitespace character. It is equivalent to the POSIX class **[[:space:]]**.	The expression **\\(\w\s\w\s\\)** matches the string **(a b)** but not the string **(ab)**.
\S	A non-whitespace character. It is equivalent to the POSIX class **[^[:space:]]**.	The expression **\\(\w\S\w\S\\)** matches the string **(abde)** but not the string **(a b d e)**.
\A	Only at the beginning of a string. In multi-line mode, that is, when embedded newline characters in a string are considered the termination of a line, **\A** does not match the beginning of each line.	The expression **\AL** matches only the first **L** character in the string **Line1\nLine2\n**, regardless of whether the search is in single-line or multi-line mode.
\Z	Only at the end of string or before a newline ending a string. In multi-line mode, that is, when embedded newline characters in a string are considered the termination of a line, **\Z** does not match the end of each line.	In the expression **\s\Z,** the **\s** matches the last space in the string **L i n e \n**, regardless of whether the search is in single-line or multi-line mode.
\z	Only at the end of a string.	In the expression **\s\z**, the **\s** matches the newline in the string **L i n e \n**, regardless of whether the search is in single-line or multi-line mode.
***?**	The preceding pattern element 0 or more times (non-greedy). Note that this quantifier matches the empty string whenever possible.	The expression **\w*?x\w** is "non-greedy" and so matches **abxc** in the string **abxcxd**. The expression **\w*x \w** is "greedy" and so matches **abxcxd in the string abxcxd**. The expression **\w*?x\w** also matches the string **xa**.
+?	The preceding pattern element 1 or more times (non-greedy).	The expression **\w+?x\w** is "non-greedy" and so matches **abxc** in the string **abxcxd**. The expression **\w+x \w** is "greedy" and so matches **abxcxd in the string abxcxd**. The expression **\w+?x\w** does not match the string **xa**, but does match the string **axa**.
??	The preceding pattern element 0 or 1 time (non-greedy). Note that this quantifier matches the empty string whenever possible.	The expression **a??aa** is "non-greedy" and matches **aa** in the string **aaaa**. The expression **a?aa** is "greedy" and so matches **aaa** in the string **aaaa**.
{n}?	The preceding pattern element exactly **n** times (non-greedy). In this case **{n}?** is equivalent to **{n}**.	The expression **(a|aa){2}?** matches **aa** in the string **aaaa**.
{n,}?	The preceding pattern element at least **n** times (non-greedy).	The expression **a{2,}?** is "non-greedy" and matches **aa** in the string **aaaaa**. The expression **a{2,}** is "greedy" and so matches **aaaaa**.
{n,m}?	At least **n** but not more than **m** times (non-greedy). Note that **{0,m}?** matches the empty string whenever possible.	The expression **a{2,4}?** is "non-greedy" and matches **aa** in the string **aaaaa**. The expression **a{2,4}** is "greedy" and so matches **aaaa**.

Match Parameter

c	Case sensitive
i	Case insensitive
m	Treat the source string as multiple lines
n	Allow the period(.) to match any new line character
x	Ignore white space

```
SELECT col1
FROM table1
WHERE REGEXP_LIKE (col1,‘dr(ink(ing)?|(unk)|(ank)|(unkard))’,’i’);
----
REGEXP_LIKE (email_address, '[a-zA-Z0-
9._%-]+@[a-zA-Z0-9._%-]+\.[a-zA-Z]
{2,4}');
----
REGEXP_LIKE(first_name, 'Dav(e|id)');
```

Encrypt in Oracle

```
GRANT EXECUTE  ON DBMS_CRYPTO TO HR;
CREATE OR REPLACE PROCEDURE ENCRYPTION(TEXT IN VARCHAR2, ENCRYPTEDTEXT OUT
VARCHAR2) AS
RAW_SET RAW(100);
RAW_PASSWORD RAW(100);
ENCRYPTION_RESULT RAW(100);
ENCRYPTION_PASSWORD VARCHAR2(100)  := 'teni';
OPERATION_MODE NUMBER;
    BEGIN
        RAW_SET:=UTL_I18N.STRING_TO_RAW(TEXT,'AL32UTF8');
        RAW_PASSWORD := UTL_I18N.STRING_TO_RAW(ENCRYPTION_PASSWORD,'AL32UTF8');
        OPERATION_MODE:=DBMS_CRYPTO.ENCRYPT_DES + DBMS_CRYPTO.PAD_ZERO +
DBMS_CRYPTO.CHAIN_ECB;
        ENCRYPTION_RESULT:
=DBMS_CRYPTO.ENCRYPT(RAW_SET,OPERATION_MODE,RAW_PASSWORD);
        DBMS_OUTPUT.PUT_LINE(ENCRYPTION_RESULT);
        ENCRYPTEDTEXT := RAWTOHEX (ENCRYPTION_RESULT);
END;

VARIABLE RESULT_ENCRYPTION VARCHAR2(200)
EXEC ENCRYPTION('Clear Text', :RESULT_ENCRYPTION);
PRINT RESULT_ENCRYPTION

RESULT_ENCRYPTION
---------------------------------------------------------------------------
DA581222F8FE175B6B98867727F9CF19


CREATE OR REPLACE PROCEDURE DECRYPTION (ENCRYPTED_TEXT IN VARCHAR2, DECRYPTED_TEXT
OUT VARCHAR2) AS
RAW_SET RAW(100);
RAW_PASSWORD RAW(100);
DECRYPTION_RESULT RAW(100);
DECRYPTION_PASSWORD VARCHAR2(100)  := 'teni';
OPERATION_MODE NUMBER;
    BEGIN
        RAW_SET:=HEXTORAW(ENCRYPTED_TEXT);
        RAW_PASSWORD :=UTL_I18N.STRING_TO_RAW(DECRYPTION_PASSWORD,'AL32UTF8');
        OPERATION_MODE:=DBMS_CRYPTO.ENCRYPT_DES + DBMS_CRYPTO.PAD_ZERO +
DBMS_CRYPTO.CHAIN_ECB;
        DECRYPTION_RESULT:
=DBMS_CRYPTO.DECRYPT(RAW_SET,OPERATION_MODE,RAW_PASSWORD);
        DBMS_OUTPUT.PUT_LINE(DECRYPTION_RESULT);
        DECRYPTED_TEXT := UTL_I18N.RAW_TO_CHAR (DECRYPTION_RESULT,'AL32UTF8');
    END;


VARIABLE RESULT_DECRYPTION VARCHAR2(200);
EXEC DECRYPTION(:RESULT_ENCRYPTION,:RESULT_DECRYPTION);
PRINT RESULT_DECRYPTION;

RESULT_DECRYPTION
---------------------------------------------------------------------------
CLEAR TEXT
```

Performance Tuning Tools

Execution Plan:

It is the sequence of steps that the optimizer determines how to execute a SQL statement. Using Explain plan command, you can identify the execution plan oracle applies to a particular SQL statement.

```
EXPLAIN PLAN
[SET STATEMENT_ID = 'statement_id'] [INTO table_name]
FOR SELECT * FROM employees;

SELECT * FROM PLAN_TABLE; -- PLAN_TABLE is a Global Temporary table. Users cannot
see the plans inserted by other sessions.

-- Below query will show execution plan without considering the affects of bind
variables used in the query.
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);--for better formatted output - like a report

--To see actual execution plan used by the database
GRANT SELECT ON v_$session TO hr;
GRANT SELECT ON v_$sql_plan_statistics_all to hr;
GRANT SELECT ON v_$sql_plan to hr;
GRANT SELECT ON v_$sql to hr;

SELECT * FROM V$SQL WHERE LOWER(SQL_TEXT) LIKE 'SELECT * FROM EMPLOYEES %';
--SQL ID: 1XTYZRSTP37SU
--CHILD NUMBER: 0

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('1XTYZRSTP37SU',0));

--We can access these dynamic performance views individually by:
SELECT * FROM v$session;
select * from v$sql_plan_statistics_all;
select * from v$sql_plan;
select * from v$sql;
```

Query Optimization

Rule Based Optimizer (RBO)

Predefined set of conditions to determine what plan it is going to use. Default optimization is Rule based.

Cost Based Optimizer(CBO)

Pick up a plan according to least cost.

```
SQL> SET AUTOTRACE ON
--If any error comes, that mean, PLAN_TABLE is not defined.
--utlxplan.sql will create schema for this PLAN_TABLE
SQL> SET TIMING ON
--To turn on time elapsed
```

Table Compression

Oracle 9i onward allows whole tables or individual table partitions to be compressed to reduce disk space requirements.

```
CREATE TABLE test_tab (
  id              NUMBER(10)    NOT NULL,
  description    VARCHAR2(100) NOT NULL,
  created_date   DATE          NOT NULL,
  created_by     VARCHAR2(50)  NOT NULL,
  updated_date   DATE ,
  updated_by     VARCHAR2(50)
)COMPRESS;
```

```
CREATE TABLE test_tab (
  id              NUMBER(10)    NOT NULL,
  description    VARCHAR2(100) NOT NULL,
  created_date   DATE          NOT NULL,
  created_by     VARCHAR2(50)  NOT NULL,
  updated_date   DATE ,
  updated_by     VARCHAR2(50)
)COMPRESS
PARTITION BY RANGE (created_date) (
  PARTITION test_tab_q1 VALUES LESS THAN (TO_DATE('01/04/2003', 'DD/MM/YYYY')),
  PARTITION test_tab_q2 VALUES LESS THAN (MAXVALUE) --Default partition
);
```

```
ALTER TABLE test_tab NOCOMPRESS;
ALTER TABLE test_tab COMPRESS; -- Basic Compression
ALTER TABLE test_tab COMPRESS FOR ALL OPERATIONS; -- Advanced Compression
```

Advanced compression enables table for all operations including DML with minimal performance loss. Advanced compression is a licensed feature.

This doesn't affect the compression of existing data, but will affect the compression of new data that is loaded by direct path loads. If you want to compress existing data, you must perform a move operation, so the data gets compressed during the copy.

```
ALTER TABLE test_tab MOVE NOCOMPRESS;
ALTER TABLE test_tab MOVE COMPRESS;
```

A similar action could be performed for an individual partition of a partitioned table.

```
ALTER TABLE test_tab MOVE PARTITION test_tab_q2 COMPRESS;
ALTER TABLE test_tab MOVE PARTITION test_tab_q2 NOCOMPRESS;
```

If you use the keyword MOVE and there are indexes on the table, those indexes will become corrupt. This corruption occurs because you're changing the row location in the table when you proactively compress the data. To fix this problem, after a MOVE compression action, rebuild the indexes. This is one reason you may choose to compress the data for future operations now and then move it later when you can incur downtime to rebuild the indexes.

```
SELECT compression, compress_for FROM   USER_TABLES WHERE  table_name = 'TEST_TAB';

SELECT partition_name, compression, compress_for FROM   USER_TAB_PARTITIONS
WHERE  table_name = 'TEST_TAB' ORDER BY 1;
```

```
SELECT SEGMENT_NAME, BYTES FROM USER_SEGMENTS WHERE SEGMENT_NAME = 'EMP'; -- To check
size of table
```

Partitioning an existing table

Create a new partition table with same structure of the table which you want to partition.
The DBMS_REDEFINITION package provides an interface to perform an online redefinition of tables.

```
BEGIN
DBMS_REDEFINITION.CAN_REDEF_TABLE('DEVIROTR', 'IRD_TXN_DTS_AUD'); -- Determines
if a given table can be redefined online
END;

BEGIN
DBMS_REDEFINITION.START_REDEF_TABLE(     -- Initiates the redefinition process
UNAME       => 'DEVIROTR', -- Schema name
ORIG_TABLE => 'IRD_TXN_DTS_AUD',  -- Original Table
INT_TABLE  => 'IRD_TXN_DTS_AUD_P'); -- Temporary table created with partition
END;

BEGIN
DBMS_REDEFINITION.SYNC_INTERIM_TABLE(  -- Keeps the interim table synchronized
with the original table
UNAME       => 'DEVIROTR', ORIG_TABLE => 'IRD_TXN_DTS_AUD',
INT_TABLE  => 'IRD_TXN_DTS_AUD_P');
END;

BEGIN
DBMS_REDEFINITION.FINISH_REDEF_TABLE(  -- Completes the redefinition process
UNAME       => 'DEVIROTR', ORIG_TABLE => 'IRD_TXN_DTS_AUD',
INT_TABLE  => 'IRD_TXN_DTS_AUD_P');
END;

BEGIN
DBMS_REDEFINITION.ABORT_REDEF_TABLE(  -- Cleans up errors that occur during the
redefinition process and removes all temporary objects created by the
reorganization process
UNAME       => 'DEVIROTR', ORIG_TABLE => 'IRD_TXN_DTS_AUD',
INT_TABLE  => 'IRD_TXN_DTS_AUD_P');
END;

BEGIN
DBMS_STATS.GATHER_TABLE_STATS('DEVIROTR', 'IRD_TXN_DTS_AUD', CASCADE => TRUE,
GRANULARITY=>'ALL'); -- Gather statitstics for new table
END;

ALTER TABLE DEVIROTR.IRD_TXN_DTS_AUD ENABLE ROW MOVEMENT; -- If any row is
updated, ROW MOVEMENT enables that row to be moved to its corresponding
partition.
```

Split Partition

SPLIT PARTITION command to divide a single partition into two partitions, and redistribute the partitions contents between the new partitions

Splitting Range Partitions

```
ALTER TABLE IRD_TXN_DTS_AUD SPLIT PARTITION "PMAX" AT
(TO_DATE('01/01/2021','DD/MM/YYYY')) INTO (PARTITION "2020", PARTITION
"PMAX"); -- Suppose PMAX is the default partition name and we want to move the
data from default partition to different partition. 2020 will receive the
rows that meet the subpartitioning constraints specified (Rows LESS THAN
01/01/2021) and PMAX will receive the rows are not directed to 2020.
```

Splitting List Partitions

```
ALTER TABLE IRD_TXN_DTS_AUD SPLIT PARTITION "PMAX" VALUES
(10,20,30) INTO (PARTITION "2020", PARTITION "PMAX");
```

Splitting Hash Partitions

SPLIT PARTITION command cannot add a partition to a HASH partitioned table. There is no upper limit to the number of partitions that a table may have.

Oracle Partitions

- Partitioning allows tables, indexes and index-organized tables to be subdivided into smaller pieces, enabling these database objects to be managed and accessed at a finer level of granularity (Indexing will be ineffective for large amount of data).
- **Partition pruning** is the act of eliminating, or ignoring partitions that are irrelevant to the SQL statements' selection criteria
- Mainly used in data warehousing area (More SELECT less DML)

List Partition

- Enables rows to be allocated to partitions based on nominated lists of values
- This is similar but more flexible than range partitioning and enables non-adjacent partition key rows to be stored in the same partition. Ex: partition based on sales region; US sales, Mumbai sales; department wise

```
CREATE TABLE table_name (deptno NUMBER(4), dname VARCHAR2(20))
PARTITION BY LIST(deptno)
(PARTITION p1 VALUES ('10'), -- PARTITION "2018" VALUES ('10'), -- It will
throw error ("invalid partition name") if we try to give partition name
starting with number without quotes
 PARTITION p2 VALUES ('20'), -- PARTITION "2019" VALUES ('20'),
 PARTITION p3 VALUES ('30'), -- PARTITION "2020" VALUES ('30'),
 PARTITION p4 VALUES ('40'), -- PARTITION "2021" VALUES ('40')
 PARTITION region_null VALUES (NULL), -- NULL values
 PARTITION region_unknown VALUES (DEFAULT) -- Default values
)ENABLE ROW MOVEMENT;
 ----

SELECT * FROM table_name PARTITION(p1);
```

Hash Partition

- Allocates rows based on a mathematical hash function. This helps to ensure that each partition is of the same but tends to reduce the possibility of partition elimination for range scans.
- Have to specify number of partitions and oracle will divide the data into partitions using its algorithm.

```
CREATE TABLE table_name(empno, sal...)
PARTITION BY HASH(empno)
PARTITIONS 4
ENABLE ROW MOVEMENT;
----
SELECT * FROM table_name;;
```

Range Partition

- Allows rows to be allocated to partitions based on contiguous ranges of the partition key
- Range partition on a time-based column is common because it enables us to quickly purge older data by dropping a partition. Ex: keep only last 3 years of data; delete partition for 2014; Employee ID from 1 to 10,000 - 10,000 to 40,000

```
CREATE TABLE table_name (empid, sal...)
PARTITION BY RANGE(sal)
INTERVAL (1000) --In future if the data is increasing, oracle will create partitions when
it reach this interval
(PARTITION p1 VALUES LESS THAN (1000),
 PARTITION p2 VALUES LESS THAN (2000),
 PARTITION p3 VALUES LESS THAN (3000),
 PARTITION p4 VALUES LESS THAN (4000),
 PARTITION p5 VALUES LESS THAN (MAXVALUE)-- default value
)ENABLE ROW MOVEMENT; -- inside partition, rows are allowed to move when the value changes.
Ex: when the sal increases, that row moves to different partition. Otherwise it won't allow
to update sal.
----
SELECT * FROM table_name PARTITION(p1);
```

Composite Partitioning (Sub-partitioning)

Doing partitioning inside a partition is called composite partitioning. Ex: Range then hash; Range then list etc.

```
CREATE TABLE table_name (empid, sal...)
PARTITION BY RANGE(hiredate)
SUBPARTITION BY LIST(deptno)
(PARTITION p1 VALUES LESS THAN (TO_DATE('01-JAN-1980','dd-MON-yyyy'))
(SUBPARTITION sp1 VALUES ('10'),
 SUBPARTITION sp2 VALUES ('20'),
 SUBPARTITION sp3 VALUES ('30'),
 SUBPARTITION sp4 VALUES ('40')),
PARTITION p2 VALUES LESS THAN (TO_DATE('01-JAN-1990','dd-MON-yyyy'))
(SUBPARTITION sp1 VALUES ('10'),
 SUBPARTITION sp2 VALUES ('20'),
 SUBPARTITION sp3 VALUES ('30'),
 SUBPARTITION sp4 VALUES ('40'))
)ENABLE ROW MOVEMENT;
```

```
ALTER TABLE employee ADD PARTITION p5 VALUES LESS THAN (50);
----
ALTER TABLE employee DROP PARTITION p4;
----
ALTER TABLE employee RENAME PARTITION p3 tp p4;
----
ALTER TABLE employee TRUNCATE PARTITION p4;
----
ALTER TABLE employee SPLIT PARTITION p2 AT(15) INTO (PARTITION p21, PARTITION p22);
----
ALTER TABLE employee EXCHANGE PARTITION p21 WITH TABLE employee2
----
ALTER TABLE employee MOVE PARTITION p21 TABLESPACE SYSAUX;
```

Oracle Hints

Hints are used to explicitly instruct the oracle optimizer to use certain execution plan rather than oracle chooses by itself. More than one hints can be used at the same time. Use the hints when you are absolutely sure that the hints lead to a significant performance boost. Oracle recommends to use hints sparingly (less), and only after you have collected statistics on the relevant tables and evaluated the optimizer plan without hints using the EXPLAIN PLAN statement.

Hints for Optimization Approaches and Goals

HINT	DESCRIPTION
ALL_ROWS	The ALL_ROWS hint explicitly chooses the cost-based approach to optimize a statement block with a goal of **best throughput** (that is, minimum total resource consumption). `SELECT /*+ ALL_ROWS */ * FROM employees; -- Cardinality: 107`
FIRST_ROWS(n)	Instructs Oracle to choose the plan that returns the first n rows most efficiently (**best response time**). `SELECT /*+ FIRST_ROWS(10) */ * FROM employees; -- Cardinality: 10; All rows will be selected but first 10 rows will be returned most effectively`

Oracle ignores FIRST_ROWS in all DELETE and UPDATE statements and in SELECT statement blocks that include sorts and/or groupings, as it needs to fetch all relevant data anyway.

Hints for Access Paths

Cluster Index

An table cluster that uses an index to locate data. The cluster index is a B-tree index on the cluster key.

Cluster Scan

A **cluster** is a schema object that contains data from one or more tables, all of which have one or more columns in common. Oracle Database stores together all the rows from all the tables that share the same cluster key. In an indexed table cluster, Oracle Database first obtains the rowid of one of the selected rows by scanning the cluster index. Oracle Database then locates the rows based on this rowid.

Hash Scan

The database uses a hash scan to locate rows in a hash cluster based on a hash value. In a hash cluster, all rows with the same hash value are stored in the same data block. Oracle Database first obtains the hash value by applying a hash function to a cluster key value specified by the statement, and then scans the data blocks containing rows with that hash value.

Hints for Join Operation

Hashing

Hashing is the process of converting an input of any length into a fixed size string of text using an algorithm (hash function). Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value. It can be also used for encrypting password and data.

Nested Loop Join

A Nested Loops join works in the same way as nested loops. One of the joining tables is designated as the outer table (small table) and another one as the inner table (large table). For each row of the outer table, all the rows from the inner table are matched one by one if the row matches it is included in the result-set otherwise it is ignored.

Merge Join

Merge join requires both inputs to be sorted. Because the rows are pre-sorted, a Merge join immediately begins the matching process. It reads a row from one input and compares it with the row of another input. If the rows match, that matched row is considered in the result-set.

To alter the degree of parallelism of a table

```
ALTER TABLE employees PARALLEL 4;
```

Hash Join

A Hash join is performed in two phases; the Build phase (small table) and the Probe phase(large table). During the build phase, joining keys of all the rows of the build table are scanned. Hashes are generated and placed in an in-memory hash table. No rows are returned until this point. During the probe phase, joining keys of each row of the probe table are scanned. Again hashes are generated (using the same hash function as above) and compared against the corresponding hash table for a match. A Hash function requires significant amount of CPU cycles to generate hashes and memory resources to store the hash table.

HINT	DESCRIPTION
FULL	The FULL hint explicitly chooses a full table scan for the specified table. `SELECT /*+ FULL */ * FROM employees;`
ROW_ID	The ROWID hint explicitly chooses a table scan by rowid for the specified table. `SELECT /*+ROWID(employees)*/ * FROM employees WHERE rowid > 'AAAAtkAABAAAFNTAAA';`
CLUSTER	The CLUSTER hint explicitly chooses a cluster scan to access the specified table. It applies only to clustered objects. `SELECT /*+ CLUSTER */ employees.last_name, department_id FROM employees, departments WHERE department_id = 10 AND employees.department_id = departments.department_id;`
HASH	The HASH hint explicitly chooses a hash scan to access the specified table. It applies only to tables stored in a cluster. `/*+ HASH (table_name) */`
INDEX	The INDEX hint explicitly chooses an index scan for the specified table. If a list of indexes specified, the optimizer chooses the one with lowest cost. If no index is specified, then the optimizer chooses the available index for the table with the lowest cost. You can use the INDEX hint for domain, B-tree, bitmap, and bitmap join indexes. However, Oracle recommends using INDEX_COMBINE rather than INDEX for bitmap indexes, because it is a more versatile hint. `SELECT /*+ INDEX(e,emp_dept_idx) */ * FROM emp e;`
INDEX_ASC	The INDEX_ASC hint explicitly chooses to scan the index entries in ascending order of their indexed values.
INDEX_DESC	The INDEX_DESC hint explicitly chooses to scan the index entries in descending order of their indexed values. In a partitioned index, the results are in descending order within each partition.
INDEX_COMBINE	The INDEX_COMBINE hint explicitly chooses a bitmap access path for the table. If no indexes are given as arguments for the INDEX_COMBINE hint, then the optimizer uses whatever Boolean combination of bitmap indexes has the best cost estimate for the table. If certain indexes are given as arguments, then the optimizer tries to use some Boolean combination of those particular bitmap indexes.
NO_INDEX	The NO_INDEX hint explicitly disallows a set of indexes for the specified table. `SELECT /*+ NO_INDEX(emp emp_dept_idx) */ * FROM emp, dept WHERE emp.deptno = dept.deptno;`

HINTS	DESCRIPTION
USE_HASH	The USE_HASH hint causes Oracle to join each specified table with another row source, using a hash join. `SELECT /*+ USE_HASH(employees departments)*/ * FROM employees, departments WHERE employees.department_id = departments.department_id;`
USE_MERGE	The USE_MERGE hint causes Oracle to join each specified table with another row source, using a sort-merge join. `SELECT /*+USE_MERGE(employees departments)*/ * FROM employees, departments WHERE employees.department_id = departments.department_id;`
USE_NL	The USE_NL hint causes Oracle to join each specified table to another row source with a nested loops join, using the specified table as the inner table. `SELECT /*+USE_NL(employees departments)*/ * FROM employees, departments WHERE employees.department_id = departments.department_id;`
USE_NL_WITH_INDEX	The USE_NL_WITH_INDEX hint instructs the optimizer to join specified table to another row source with a nested loops join, using the specified table as the inner table and will use index `SELECT /*+ USE_NL_WITH_INDEX(e emp_department_idx) */ FROM employees e,departments d WHERE e.department_id = d.department_id and e.department_id= 40;`
NO_USE_NL	It instructs the optimizer to not to use nested loop joins when joining each specified table to another row source using the specified table as the inner table. `SELECT /*+NO_USE_NL(employees departments)*/ * FROM employees, departments WHERE employees.department_id = departments.department_id;`
NO_USE_HASH	It instructs the optimizer to not to use hash join. `SELECT /*+ NO_USE_HASH(employees departments)*/ * FROM employees, departments WHERE employees.department_id = departments.department_id;`
NO_USE_MERGE	It instructs the optimizer to not to use sort merge join . `SELECT /*+NO_USE_MERGE(employees departments)*/ * FROM employees, departments WHERE employees.department_id = departments.department_id;`

Hints for Join Orders

HINTS	DESCRIPTION
LEADING	The LEADING hint causes Oracle to use the specified table as the first table in the join order. If you specify two or more LEADING hints on different tables, then all of them are ignored. If you specify the ORDERED hint, then it overrides all LEADING hints. `SELECT /*+ LEADING(e j) */ * FROM employees e, departments d, job_history j WHERE e.department_id = d.department_id AND e.hire_date = j.hire_date;`
ORDERED	The ORDERED hint causes Oracle to join tables in the order in which they appear in the FROM clause. If you omit the ORDERED hint from a SQL statement performing a join, then the optimizer chooses the order in which to join the tables. You might want to use the ORDERED hint to specify a join order if you know something about the number of rows selected from each table that the optimizer does not. Such information lets you choose an inner and outer table better than the optimizer could.

Hints for Parallel Execution

Parallel hint enables a SQL statement to be simultaneously processed by multiple threads or processes.

HINTS	DESCRIPTION
PARALLEL	The PARALLEL hint lets you specify the desired number of concurrent servers that can be used for a parallel operation. The hint applies to the SELECT, INSERT, UPDATE, and DELETE portions of a statement, as well as to the table scan portion. The database computes the degree of parallelism which can be 2 or greater. `SELECT /*+ PARALLEL */ last_name, first_name, salary FROM employees;`
NO_PARALLEL	The NO_PARALLEL hint overrides a PARALLEL parameter in the DDL that created or altered the table. It uses SERIAL plan. `SELECT /*+ NO_PARALLEL(e) */ FROM employees e;`

DATA Pump

- Export data using expdp utility
- Import data using impdp utility

expdp

- A full export does not export system schemas that contain Oracle managed data and meta data. Ex: SYS
- Grants on objects owned by SYS are never exported

Full database Export

```
--Make a directory in server
--Make directory object
SQL> CREATE DIRECTORY directory_obj_name AS 'actual_directory_address' -- won't
create any physical directory
SQL> GRANT READ , WRITE ON DIRECTORY directory_obj_name;
SQL> GRANT DATAPUMP_EXP_FULL_DATABASE TO username;--DATAPUMP_EXP_FULL_DATABASE is a role
CMD> EXPDP username/password@oracle_service_name DIRECTORY=directory_obj_name
DUMPFILE=dump_file_name.dmp LOGFILE=log_file_name.log FULL=YES;
--Dumpfile = One or more disk file contains table data,database objects, meta data and
control information. These files are written in binary format. This can be imported only
by datapump utility IMPDP.
--FULL = all the data and metadata has to be exported
```

Exporting Tablespace

```
--Make a directory in server
--Make directory object
SQL> CREATE DIRECTORY directory_obj_name AS 'actual_directory_address'
SQL> GRANT READ , WRITE ON DIRECTORY directory_obj_name;
SQL> GRANT DATAPUMP_EXP_FULL_DATABASE TO username;
CMD> EXPDP username/password@oracle_service_name DIRECTORY=directory_obj_name
DUMPFILE=dump_file_name.dmp LOGFILE=log_file_name.log TABLESPACES =
tablespace_name1,tablespace_name2;
```

Exporting Schema

Schema is a collection of logical structure of data such as views, synonyms, tables etc.. Schema object has similar name of the user. Every user owns a schema.

```
--Make a directory in server
--Make directory object
SQL> CREATE DIRECTORY directory_obj_name AS 'actual_directory_address'
SQL> GRANT READ , WRITE ON DIRECTORY directory_obj_name;
SQL> GRANT DATAPUMP_EXP_FULL_DATABASE TO username;
CMD> EXPDP username/password@oracle_service_name DIRECTORY=directory_obj_name
DUMPFILE=dump_file_name.dmp LOGFILE=log_file_name.log SCHEMAS =
schema_name1,schema_name2;
```

Export using PARFILE

```
--Make a directory in server
--Make directory object
--PARFILE contains details of the DUMPFILE location, LOGFILE location etc. So no need to
specify that explicitly if we are using PARFILE for export.

SQL> CREATE DIRECTORY directory_obj_name AS 'actual_directory_address'
SQL> GRANT READ , WRITE ON DIRECTORY directory_obj_name;
SQL> GRANT DATAPUMP_EXP_FULL_DATABASE TO username;
CMD> EXPDP username/password@oracle_service_name PARFILE='parameter_file_address'
```

Creating Parameter File

Make a text file -> Enter the details of export

```
DIRECTORY = directory_obj_name
DUMPFILE = dump_file_name.dmp
LOGFILE = log_file_name.log
TABLES = table_name1, table_name2
```

Save as file_name.par

impdp

```
CMD> IMPDP username/password@oracle_service_name DIRECTORY=directory_obj_name
DUMPFILE=dump_file_name.dmp LOGFILE=log_file_name.log TABLES = table_name1, table_name2;
```

Rename the Table while importing

```
CMD> IMPDP username/password@oracle_service_name DIRECTORY=directory_obj_name
DUMPFILE=dump_file_name.dmp LOGFILE=log_file_name.log TABLES = table_name
REMAP_TABLE=schema_name.old_table_name:new_table_name;
```

Import Table to different Schema

```
CMD> IMPDP username/password@oracle_service_name DIRECTORY=directory_obj_name
DUMPFILE=dump_file_name.dmp LOGFILE=log_file_name.log TABLES = 'table_name'
REMAP_SCHEMA=old_schema_name:new_schema_name;
```

Note: Specify the name of the table in single quotes and all in upper case

Import Schema to new Schema

```
CMD> IMPDP username/password@oracle_service_name DIRECTORY=directory_obj_name
DUMPFILE=dump_file_name.dmp LOGFILE=log_file_name.log SCHEMAS = schema_name
REMAP_SCHEMA=old_schema_name:new_schema_name;
```

SQLFILE Parameter

- It generates a SQL script based on the other parameter in impdp data pump import.
- This script contain all the DDL(only DDL) that is executed by impdp data pump in background to perform the import.
- Whenever you use SQLFILE parameter in your impdp data pump import, it does not actually perform the import rather it just generates SQL script for the import and the target system remains unchanged

```
CMD> IMPDP username/password@oracle_service_name DIRECTORY=directory_obj_name
DUMPFILE=dump_file_name.dmp LOGFILE=log_file_name.log TABLESPACES = tablespace_name
SQLFILE=sql_file_name
```

Cold Backup and Restore

```
/*Source DB -> CMD ->*/
CMD> SET ORACLE_SID=EZTGDC
CMD> SQLPLUS / AS SYSDBA
SQL> SHOW PARAMETER SPFILE; -- shows directory of spfile
SQL> CREATE PFILE='D:\pfile\pfileclone.ora' FROM SPFILE;
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE AS 'D:\pfile\TRACE.SQL';
SQL> SHUTDOWN IMMEDIATE;
SQL> EXIT
/*Create folders adump,bdump,cdump,dpdump,pfile,scripts,udump in target DB*/
/*Create folder flash_recovery_area and oradata,arch in target DB*/
/*Copy datafile, pfile, trace file to target database folder*/
CMD> SET ORACLE_SID=EZTGDC
CMD> SQLPLUS / AS SYSDBA
SQL> STARTUP
----
/*Target DB -> CMD ->*/
CMD> ORADIM -NEW -SID EZTGDC  --create a new SID in target system
CMD> SET ORACLE_SID=EZTGDC
/*Edit TRACE.SQL as below*/
    /*Delete all till "STARTUP NOMOUNT" and delete all after " ;"*/
    /*Change db_name and directory names*/
    /*Change REUSE to SET and NORESETLOGS to RESETLOGS*/
/*Edit PFILE as well*/
SQL> CREATE SPFILE FROM PFILE='C:\pfile\pfileclone.ora';
CMD> SQLPLUS / AS SYSDBA
/*Delete all control files from the folder*/
SQL> @C:\pfile\TRACE.SQL
SQL> ALTER DATABASE OPEN RESETLOGS
```

Hot Backup and Restore

```
/*Create all required folders in target system (adump,bdump…)*/
/*The exact copy of pfile is needed from source db*/
CMD> ORADIM -NEW -SID EZTGDC2 --new SID in target system
CMD> SET ORACLE_SID = EZTGDC --old SID
SQL> SELECT LOG_MODE FROM V$DATABASE;
SQL> SELECT * FROM V$BACKUP; -- If not ACTIVE
SQL> ALTER DATABASE BEGIN BACKUP; -- If not ACTIVE
/*Copy all dbf(only dbf) files to db folder*/
SQL> ALTER DATABASE END BACKUP;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT; -- Run 2-3 times
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE AS 'C:\pfile\TRACE.SQL' REUSE;
/*Edit TRACE.SQL as before*/
/*Connect to the target database*/
SQL> CREATE SPFILE FROM PFILE='C:\pfile\init.ora';
SQL> STARTUP NOMOUNT
SQL> @C:\pfile\TRACE.SQL
SQL> SET LOGSOURCE 'C:\oracle\product\10.2.0\oradata\EZTGDC\arch';
SQL> RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL;
SQL> ALTER DATABASE OPEN RESETLOGS;
```

RMAN Simple Backup

```
CMD> SET ORACLE_SID=EZTGDC
CMD> RMAN TARGET /
RMAN> SHOW ALL; -- Shows RMAN configuration parameters
RMAN> CONFIGURE DEFAULT DEVICE TYPE CLEAR; --resetting to default values
RMAN> BACKUP DATABASE TAG'DB_SMPL_BKP' PLUS ARCHIVELOG TAG'ARC_SMPL_BKP';
RMAN> SHUTDOWN IMMEDIATE
/*Delete all dbf, control files and archive logs*/
RMAN> STARTUP NOMOUNT;
RMAN> RESTORE CONTROLFILE FROM 'C:\oracle\product\10.2.0\flash_recovery_area\EZTGDC
\BACKUPSET\2016_08_24\O1_MF_NCSNF_DB_SMPL_BKP_CVTZ8LFL_.BKP';
RMAN> ALTER DATABASE MOUNT;
RMAN> RESTORE DATABASE;
/*Press ENTER if it is looks like it stuck somewhere.*/
RMAN> RECOVER DATABASE;
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

RMAN Incremental backup

Differential (Default)- Backup all the blocks changed after recent incremental backup level 0 or level 1. Save time and space.

```
RMAN> BACKUP INCREMENTAL LEVEL=0 DATABASE TAG='DB_LEVEL0-SUN' PLUS ARCHIVELOG
TAG='ARC_LEVEL0-SUN';
RMAN> BACKUP INCREMENTAL LEVEL=1 DATABASE TAG='DB_LEVEL1-MON' PLUS ARCHIVELOG
TAG='ARC_LEVEL1-MON';
SQL> SHUTDOWN IMMEDIATE;
/*Delete all source files*/
SQL> STARTUP NOMOUNT;
RMAN> RESTORE CONTROLFILE FROM'C:\oracle\product\10.2.0\flash_recovery_area\EZTGDC
\BACKUPSET\2016_08_25\O1_MF_NCSN1_DB_LEVEL1-MON_CVX87XPO_.BKP';
RMAN> ALTER DATABASE MOUNT;
RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

RMAN Cumulative Backup

Cumulative- Backup all the blocks changed after recent backup at level 0

```
RMAN> BACKUP INCREMENTAL LEVEL=0 CUMULATIVE DATABASE TAG='DB_LEVEL0-CUM-SUN' PLUS
ARCHIVELOG TAG='ARC_LEVEL0-SUN';
RMAN> BACKUP INCREMENTAL LEVEL=1 CUMULATIVE DATABASE TAG='DB_LEVEL1-CUM-MON' PLUS
ARCHIVELOG TAG='ARC_LEVEL1-MON';
SQL> SHUTDOWN IMMEDIATE;
/*Delete all source files*/
SQL> STARTUP NOMOUNT;
RMAN> RESTORE CONTROLFILE FROM 'C:\oracle\product\10.2.0\flash_recovery_area\EZTGDC
\BACKUPSET\2016_08_25\O1_MF_NCSN1_DB_LEVEL1-MON_CVX87XPO_.BKP';
RMAN> ALTER DATABASE MOUNT;
RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

Turn Flashback Mode On

Database must be in ARCHIVELOG mode. Flashback uses its own logging mechanism and uses flashback logs in the fast_recovery_area. You should also have the archive logs up to the days which you are going to rewind the database. This is because, flashback logs contain old versions of the block before a change is made. When a rewind is performed, the database restores the version of each block that is immediately before the target time. The database then uses redo logs to reapply changes that were made after these blocks were written to the flashback logs.
However if you want to flashback a table or check the state of a table as of some point in time instead of the whole database, then you don't need to enable any option on your database. All you need to do is resize your undo tablespace according to your flashback query requirements.

```
SQL>ALTER SYSTEM SET UNDO_RETENTION=10800 SCOPE=BOTH;--(60*60*3) (3 hrs)
```

```
SQL> SHUTDOWN IMMEDIATE
SQL> STARTUP MOUNT
SQL> SHOW PARAMETER SPFILE;
SQL> ALTER SYSTEM SET DB_RECOVERY_FILE_DEST='C:\oracle\product\10.2.0
\flash_recovery_area' SCOPE = SPFILE;
SQL> ALTER SYSTEM SET DB_RECOVERY_FILE_DEST_SIZE=5G SCOPE=SPFILE;
SQL> ALTER SYSTEM SET DB_FLASHBACK_RETENTION_TARGET=300 SCOPE=SPFILE;
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE FLASHBACK ON;
SQL> ALTER DATABASE OPEN;
SQL> SELECT FLASHBACK_ON, LOG_MODE FROM V$DATABASE;
SQL> SHOW PARAMETER DB_RECOVERY_FILE_DEST;
```

Guaranteed Restore point

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER DATABASE OPEN;
SQL> CREATE RESTORE POINT CLEAN_DB GUARANTEE FLASHBACK DATABASE; --CLEAN_DB is the name
given to the guaranteed restore point.
SQL> SELECT * FROM V$RESTORE_POINT; --viewing the guaranteed restore point
----
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> SELECT * FROM V$RESTORE_POINT;
SQL> FLASHBACK DATABASE TO RESTORE POINT CLEAN_DB;
SQL> ALTER DATABASE OPEN RESETLOGS;
SQL> SELECT CURRENT_SCN FROM V$DATABASE;
```

Point in Time Recovery

```
SQL> SELECT TABLE_NAME,ROW_MOVEMENT from all_tables where ROW_MOVEMENT='ENABLED'
SQL> ALTER TABLE table_name ENABLE ROW MOVEMENT;
SQL> FLASHBACK TABLE table_name to TIMESTAMP(to_timestamp('2016/08/24 05:00:00',
'YYYY/MM/DD HH24:MI:SS'))
```

Oracle 18c

```
SQL> SHOW CON_NAME;
CON_NAME
------------------------------
CDB$ROOT -- Connection Name; Connected to container database ROOT

SQL> SELECT NAME,OPEN_MODE,CDB FROM V$DATABASE;
NAME      OPEN_MODE            CDB
--------- -------------------- ---
XE        READ WRITE           YES

SQL> SHOW PDBS
    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB                        READ WRITE NO

SQL> SELECT name, con_id FROM v$pdbs; --Name and container ID of all pluggable
database inside CBD$ROOT container
NAME                     CON_ID
-------------------- ----------
PDB$SEED                      2 -- Oracle system defined
ORCLPDB                       3 -- Default pluggable database
```

```
SQL> CL SCR -- Clear screen -- SQL*plus
SQL> CLS -- Clear screen -- CMD
SQL> COLUMN name FORMAT a20; -- Format column "name" to 20 character
```

```
SQL> SELECT name as "Service Name" FROM V$ACTIVE_SERVICES
WHERE con_id = 3;-- Find service name
Service Name
----------------------------------------------------------------
xepdb1
```

```
--Add entry in tnsnames.ora file
ORCLPDB =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orclpdb.local)
    )
  )
```

```
--Open CMD as admin
C:\Windows\system32>lsnrctl reload

LSNRCTL for 64-bit Windows: Version 18.0.0.0.0 - Production on 05-JUN-2019
17:02:16

Copyright (c) 1991, 2018, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521)))
The command completed successfully
```

```
--Switch from container database to pluggable database
--All high privilage users are kept under container database (Ex: SYS, SYSTEM)
--All sample schemas are kept under pluggable database
```

```
SQL> ALTER SESSION SET CONTAINER = ORCLPDB;
Session altered.

SQL> SHOW con_name;
CON_NAME
------------------------------
ORCLPDB
```

```
SQL> sho user
USER is "HR"
SQL> show user
USER is "HR"
```

```
SQL> COLUMN name FORMAT a20;
SQL> SELECT name, open_mode FROM v$pdbs;
NAME                 OPEN_MODE
-------------------- ----------
ORCLPDB              MOUNTED

SQL> ALTER PLUGGABLE DATABASE OPEN; --SQL> ALTER PLUGGABLE DATABASE CLOSE;
Pluggable database altered.

SQL> SELECT name, open_mode FROM v$pdbs;
NAME                 OPEN_MODE
-------------------- ----------
ORCLPDB              READ WRITE
```

```
SQL> ALTER USER hr IDENTIFIED BY hr ACCOUNT UNLOCK;
User altered.

-- Connect to HR user
C:\Users\TeniMathew>sqlplus hr/hr@//localhost:1521/orclpdb

SQL*Plus: Release 18.0.0.0.0 - Production on Wed Jun 5 17:22:31 2019
Version 18.4.0.0.0

Copyright (c) 1982, 2018, Oracle.  All rights reserved.

Connected to:
Oracle Database 18c Express Edition Release 18.0.0.0.0 - Production
Version 18.4.0.0.0

-- Login using TNS alias
C:\Users\TeniMathew>sqlplus hr/hr@ORCLPDB

SQL*Plus: Release 18.0.0.0.0 - Production on Wed Jun 5 17:25:04 2019
Version 18.4.0.0.0

Copyright (c) 1982, 2018, Oracle.  All rights reserved.

Last Successful login time: Wed Jun 05 2019 17:22:32 +05:30

Connected to:
Oracle Database 18c Express Edition Release 18.0.0.0.0 - Production
Version 18.4.0.0.0
```

```
--Check DB_HOME and SID
Win+R -> REGEDIT -> HKEY_LOCAL_MACHINE -> SOFTWARE -> ORACLE -> KEY_OraDB18Home1 ->
All registry entries will be here
```

```
--Uninstall Oracle 18C
Delete Oracle Home path
Delete regedit entry -> REGEDIT -> HKEY_LOCAL_MACHINE -> SYSTEM ->
CurrentControlSet -> Services -> Oracle% -> Delete all registeries for Oracle
                         REGEDIT -> HKEY_LOCAL_MACHINE -> SOFTWARE -> ORACLE ->
Delete
Restart your system
Delete Oracle Group and User from Windows 10 -> Win+R -> lusrmgr.msc -> Users ->
Delete user for ORACLE
                                                                     -> Groups ->
ORA_% -> Delete all Oracle entries
Delete DB Home and Oracle Base folders
```

Oracle Apex

```
SQL> SELECT VERSION FROM DBA_REGISTRY WHERE COMP_ID = 'APEX';
no rows selected

@apxremov.sql -- to uninstall apexins

@apexins USER_TENI USER_TENI TEMP /i/
```

@apxrtins tablespace_apex tablespace_files tablespace_temp images
Where:

- tablespace_apex is the name of the tablespace for the Oracle Application Express application user.
- tablespace_files is the name of the tablespace for the Oracle Application Express files user.
- tablespace_temp is the name of the temporary tablespace.
- images is the virtual directory for Oracle Application Express images. To support future Oracle Application Express upgrades, define the virtual image directory as /i/.

When Oracle Application Express installs it creates three new database accounts:

- APEX_030200 - The account that owns the Oracle Application Express schema and metadata.
- FLOWS_FILES - The account that owns the Oracle Application Express uploaded files.
- APEX_PUBLIC_USER - The minimally privileged account used for Oracle Application Express configuration with Oracle HTTP Server and mod_plsql.

```
SQL> @apxchpwd.sql --To change the password for the ADMIN account:
...set_appun.sql
================================================================================
This script can be used to change the password of an Application Express
instance administrator. If the user does not yet exist, a user record will be
created.
================================================================================
Enter the administrator's username [ADMIN] admin
User "admin" does not yet exist and will be created.
Enter admin's email [admin] tenimathew@gmail.com
Enter admins password [] Tenimathew@15!
Created instance administrator ADMIN.
```

```
SQL> @apxldimg.sql E:\apex_19.1_en
SQL> @apex_epg_config.sql E:\apex_19.1_en
SQL> ALTER USER ANONYMOUS ACCOUNT UNLOCK;
SQL> SELECT DBMS_XDB.gethttpport FROM DUAL;
SQL> EXEC DBMS_XDB.sethttpport(8080);
SQL> COMMIT;
```

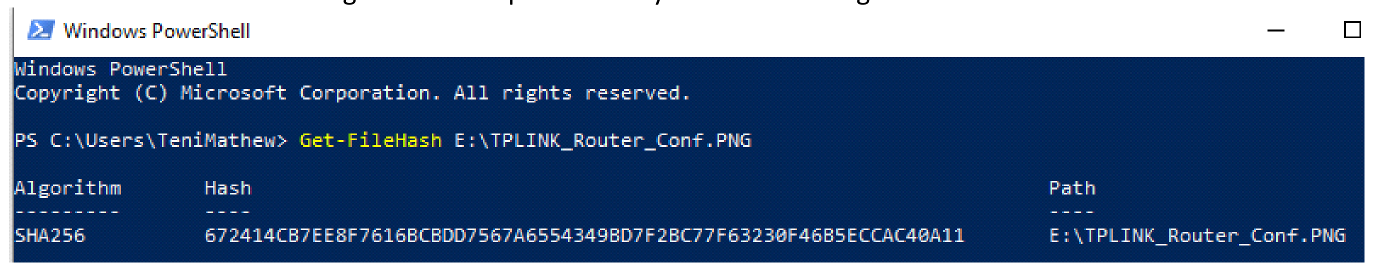
http://localhost:8080/apex/apex_admin

? Create a script to drop all the tables

```
SELECT 'DROP TABLE "' || TABLE_NAME || '" CASCADE CONSTRAINTS;' FROM user_tables;
```

? What is a checksum?

A checksum is a sequence of numbers and letters used to check data for errors. If you know the checksum of an original file, you can use a checksum utility to confirm your copy is identical.
To produce a checksum, you run a program that puts that file through an algorithm. Typical algorithms used for this include MD5, SHA-1, SHA-256, and SHA-512. The algorithm uses a cryptographic hash function that takes an input and produces a string (a sequence of numbers and letters) of a fixed length. The input file can be a small 1 MB file or a massive 4 GB file, but either way, you'll end up with a checksum of the same length. Checksums may also be called "hashes.". Small changes in the file produce very different looking checksums.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\TeniMathew> Get-FileHash E:\TPLINK_Router_Conf.PNG

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          672414CB7EE8F7616BCBDD7567A6554349BD7F2BC77F63230F46B5ECCAC40A11       E:\TPLINK_Router_Conf.PNG
```

SQL Functions

Single-row Functions

Character Functions

Function Name	Examples	Return Value
LOWER(string_value)	LOWER('Good Morning')	good morning
UPPER(string_value)	UPPER('Good Morning')	GOOD MORNING
INITCAP(string_value)	INITCAP('GOOD MORNING')	Good Morning
CONCAT(col1,col2)	CONCAT(‘God’,’Father’)	God Father
	‘God’||' '||’Father’	
SUBSTR (string_value, starting position, length)	SUBSTR ('Good Morning', 6, 5)	Morni
	SUBSTR ('Good Morning', 13, 5)	NULL
	SUBSTR ('Good Morning', 4, 13)	d Morning
	SUBSTR (50047+3, 2, 4)	0050
	SUBSTR (sysdate, 4, 3)	SEP
	SUBSTR ('Good Morning', 0, 6)	Good M
	SUBSTR ('Good Morning', -9, 6)	d Morn
	SUBSTR ('Good Morning', 3)	od Morning
LENGTH (string_value)	LENGTH ('Good Morning')	12
INSTR(string, substring [, start_position [, th_appearance]])	INSTR('Good Morning', 'o' , 3 , 2)	7
LPAD (string_value, total_length, pad_value)	LPAD ('Good', 6, '*')	**Good
RPAD (string_value, total_length, pad_value)	RPAD ('Good', 6, '*')	Good**
TRIM (trim_character trim_text FROM string_value)	TRIM (leading 'o' FROM 'ood Morning')	d Morning
	LEADING/ TRAILING/ BOTH	
LTRIM(string_value, trim_text)	LTRIM ('Good Morning', 'Good')	Morning
RTRIM (string_value, trim_text)	RTRIM ('Good Morning', ' Morning')	Good
TRANSLATE(string_value, search_value, translate_value)	TRANSLATE ('HOW IS YOUR DOG','DO','CA')	HAW IS YAUR CAG
REPLACE(string_value, search_value, replace_value)	REPLACE ('HOW IS YOUR DOG','DO','CA')	HOW IS YOUR CAG
DECODE (search_value, search_value1, ifsame, search_value2, ifsame,search_value3, ifsame, default)	DECODE('string2','string2','same','default')	same
	DECODE('string1','string2','same','string1','same2','default')	same2

Date Functions

Function Name	Examples	Return Value
ADD_MONTHS (date, number of months)	ADD_MONTHS ('16-Sep-81', 3)	16-Dec-81
MONTHS_BETWEEN(from_date, to_date)	MONTHS_BETWEEN ('16-Sep-81', '16-Dec-81')	3
NEXT_DAY(date, day_of_week)	NEXT_DAY ('01-Jun-08', 'Wednesday') or NEXT_DAY ('01-Jun-08', 4) *--Sunday = 1, Saturday = 7*	04-JUN-08
LAST_DAY(date)	LAST_DAY ('01-Jun-08')	30-Jun-08
NEW_TIME(date, from_time_zone, to_time_zone)	NEW_TIME ('01-Jun-08', 'CST', 'EST')	31-May-08

Conversion Functions

Function Name	Examples	Return Value
TO_CHAR (value, format)	TO_CHAR (3000, '$9999')	$3000
	TO_CHAR (SYSDATE, 'Day, Month YYYY')	Monday, June 2008
TO_DATE (date, format)	TO_DATE ('01-Jun-08','DD-MON-YY')	01-Jun-08
NVL (exp,if_null_display)	NVL (null, 1)	1
NVL2(exp,if_not_null_display,if_null_display)	NVL2 (null, 1,2)	2
	NVL2 (5, 1,2)	1
COALESCE(exp1,exp2,..expN)	COALESCE (null, null,2)	2
-- Returns first NOT NULL value	COALESCE (null, null,null)	NULL
NULLIF(exp1,exp2)	NULLIF(2,2)	NULL
	NULLIF(2,3)	2

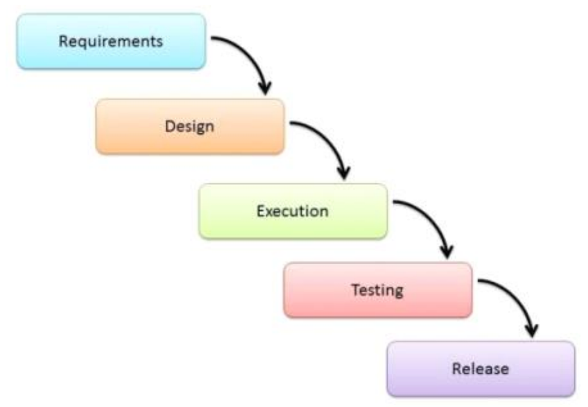
Number Functions

Function Name	Examples	Return Value
ABS (x)	ABS (1)	1
	ABS (-1)	1
CEIL (x)	CEIL (2.83)	3
	CEIL (2.49)	3
	CEIL (-1.6)	-1
FLOOR (x)	FLOOR (2.83)	2
	FLOOR (2.49)	2
	FLOOR (-1.6)	-2
ROUND (x, y)	ROUND (25.67,0)	26
	ROUND (25.67,1)	25.7
	ROUND (25.34,1)	25.3
	ROUND (25.34,2)	25.34
	ROUND (25.67,-1)	30
TRUNC (x, y)	TRUNC (25.67,0)	25
	TRUNC (25.3445,1)	25.3
	TRUNC (25.345,2)	25.34
	TRUNC (25.67,-1)	20
	TRUNC (35.20,-1)	30

SDLC

Software development life cycle (SDLC) is a series of phases that provide a common understanding of the software building process.

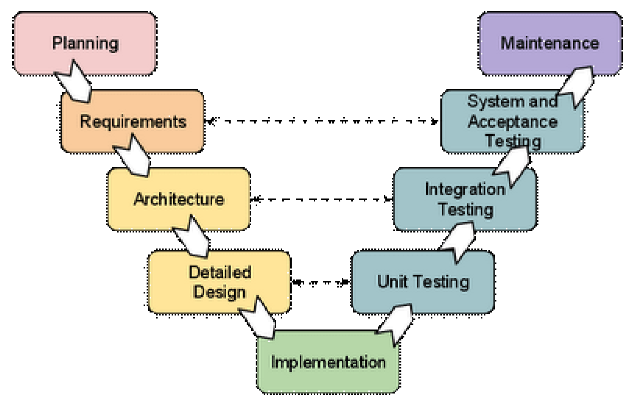
Waterfall Model



The Waterfall Model is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement. The waterfall approach is the earliest approach and most widely known that was used for software development.
The usage

- Projects which not focus on changing the requirements

V-Shaped Model



It is an extension of the waterfall model, Instead of moving down in a linear way, the process steps are bent upwards after the implementation and coding phase, to form the typical V shape. The major difference between the V-shaped model and waterfall model is the early test planning in the V-shaped model.
The usage

- Software requirements clearly defined and known
- Software development technologies and tools are well-known

Prototyping Model

It refers to the activity of creating prototypes of software applications, for example, incomplete versions of the software program being developed. It is an activity that can occur in software development and It used to visualize some component of the software to limit the gap of misunderstanding the customer requirements by the development team. This also will reduce the iterations may occur in the waterfall approach and hard to be implemented due to the inflexibility of the waterfall approach. So, when the final prototype is developed, the requirement is considered to be frozen.
The usage

- This process can be used with any software developing life cycle model. While this shall be chosen when you are developing a system has user interactions. So, if the system does not have user interactions, such as a system does some calculations shall not have prototypes.

Spiral Model (SDM)

It is combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. This model of development combines the features of the prototyping model and the waterfall model. The spiral model is favored for large, expensive, and complicated projects. This model uses many of the same phases as the waterfall model, in essentially the same order, separated by planning, risk assessment, and the building of prototypes and simulations.
The usage

- It is used in the large applications and systems which built-in small phases or segments.

Iterative and Incremental Model

It is developed to overcome the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during the development of earlier parts or versions of the system. It can consist of mini waterfalls or mini V-Shaped model
The usage

- It is used in shrink-wrap application and large system which built-in small phases or segments. Also, can be used in a system has separated components, for example, ERP system. Which we can start with the budget module as a first iteration and then we can start with the inventory module and so forth.

Agile Model

It is based on iterative and incremental development, where requirements and solutions evolve through collaboration between cross-functional teams.
The usage

- It can be used with any type of the project, but it needs more engagement from the customer and to be interactive. Also, we can use it when the customer needs to have some functional requirement ready in less than three weeks and the requirements are not clear enough. This will enable more valuable and workable piece for software early which also increase the customer satisfaction.

Testing

Unit Testing (White-Box Testing) - Verification

It focuses on smallest unit of software design. In this we test an individual unit or group of inter related units. It is often done by programmer by using sample input and observing its corresponding outputs. In this we focus on internal mechanism i.e. how the output is achieved.

System Testing (Black-Box Testing) (ST) - Validation

In this software is tested such that it works fine for different operating system. In this we just focus on required input and output without focusing on internal working.

Alpha Testing (In-house Testing)

This testing performed by the test team and possibly other interested, friendly insiders. The point of an Alpha Test is often to assess readiness of the system to be exposed to external stakeholders (such as customers). Usually performed not by clients, but by internal testers.

Beta Testing

Testing conducted at one or more customer sites by the end-user of a software product or system . This is usually a "friendly" user and the testing is conducted before the system is made generally available. Usually performed by clients, sometimes with the help of internal testers.

User Acceptance Testing (UAT)

A formal product evaluation performed by a customer as a condition of purchase usually performed by clients. This is carried out to determine whether the software satisfies its acceptance criteria and should be accepted by the customer. User acceptance testing (UAT) is one of the final stages of a software project. Often, once UAT is complete, the project is done, and payment is due.

Regression Testing

Every time new module is added leads to changes in program. This type of testing make sure that whole component works properly even after adding components to the complete program.

Smoke Testing

This test is done to make sure that software under testing is ready or stable for further testing. It is called smoke test as testing initial pass is done to check if it did not catch the fire or smoked in the initial switch on.

Stress Testing

In this we gives unfavorable conditions to the system and check how they perform in those condition.

Performance Testing

It is designed to test the run-time performance of software within the context of an integrated system. It is used to test speed and effectiveness of program.

The Basic Difference between SRS, BRS, CR, PR & FRS Documentation

SRS - System Requirement Specification:

It describes entire system flow, how data is going to flow into the system and overall functionality of the system. SRS says brief about each module's functionality and doesn't include in-depth functionality of each page and module.

SRS documentation includes a set of use cases that explain all the interactions the users will have with the software. Use cases are also known as functional requirements. In addition to use cases, the SRS also contain non-functional requirements.

NOTE: The Software Requirements Specification should contain descriptive labels for and references to every figure, table, and diagram included within the document.

BRS - Business Requirement Specification:

This document is called as high level document and includes the entire requirement demanded by the client. Ideally, this document simply includes all the requirements should be part of proposed system. BRS includes list of requirements which are demanded by the client and should be part of the proposed system.

Business Requirements Specification general Contents

- The purpose of the project
- The Client, the Customer, and other stakeholders
- Users of the product
- The scope of the work
- The scope of the product
- Features and Functionalities
- Usability and Humanity Requirements
- Performance Requirements

FRS - Functional Requirement Specification:

FRS includes requirements, converted into functionality and says that how this requirement is going to work as a part of a proposed system. FRS includes requirement converted into the way it is going to work as a part of proposed system.

Change Request (CR)

A change request is a document containing a call for an adjustment of a system; it is of great importance in the change management process. A change request is declarative, i.e. it states what needs to be accomplished, but leaves out how the change should be carried out.

Problem Report (PR)

A PR is a Problem Report. It's not only used for problems but as a way to track any user-submitted issues including bugs, ideas and patches related to the base system

What does SDR stand for?

SDR stands for Service Desk Request (computer support; various organizations)

Documents prepared before release of a software

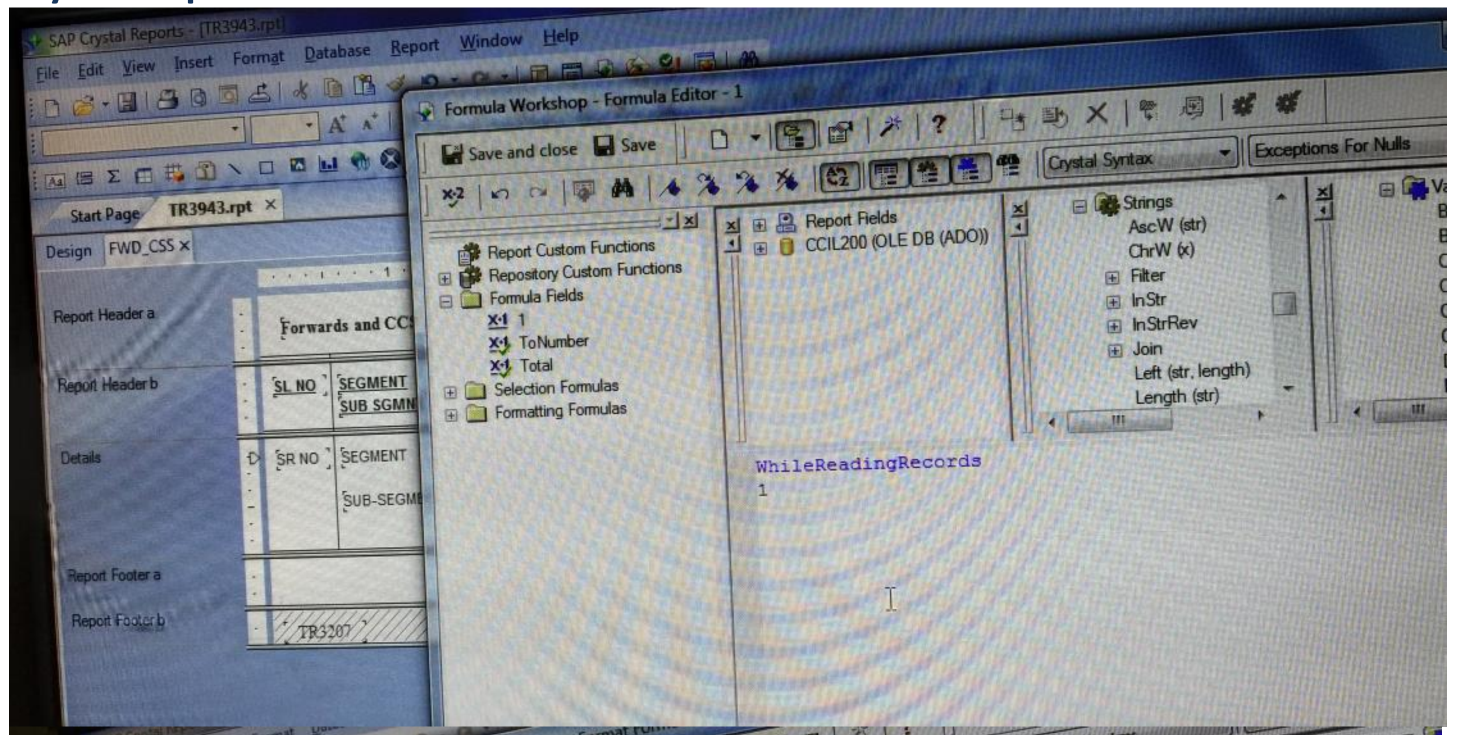
USDP(Unified Software Development Process)

Agile Frameworks
XP, Scrum, Crystal, DSDM, FDD

Kanban Board

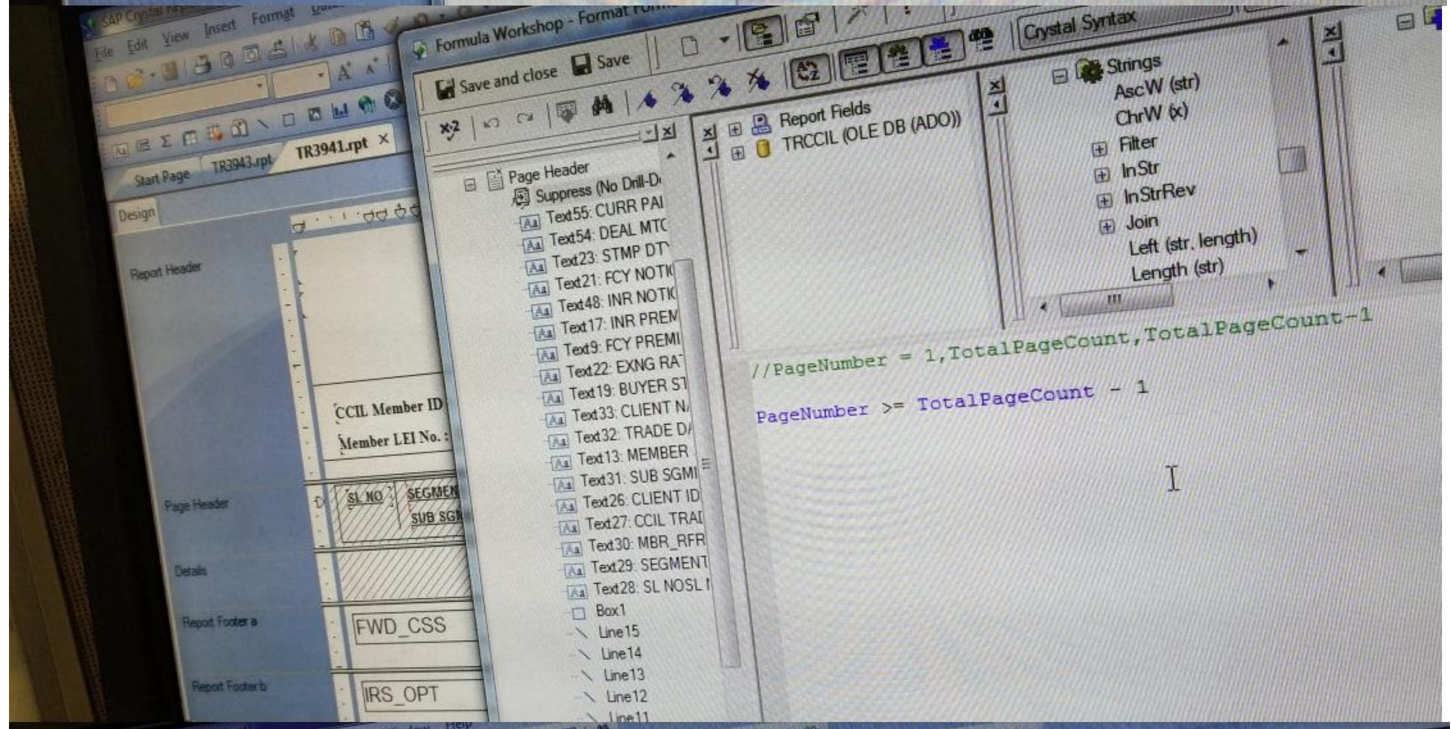
Burn down Charts

Crystal Report



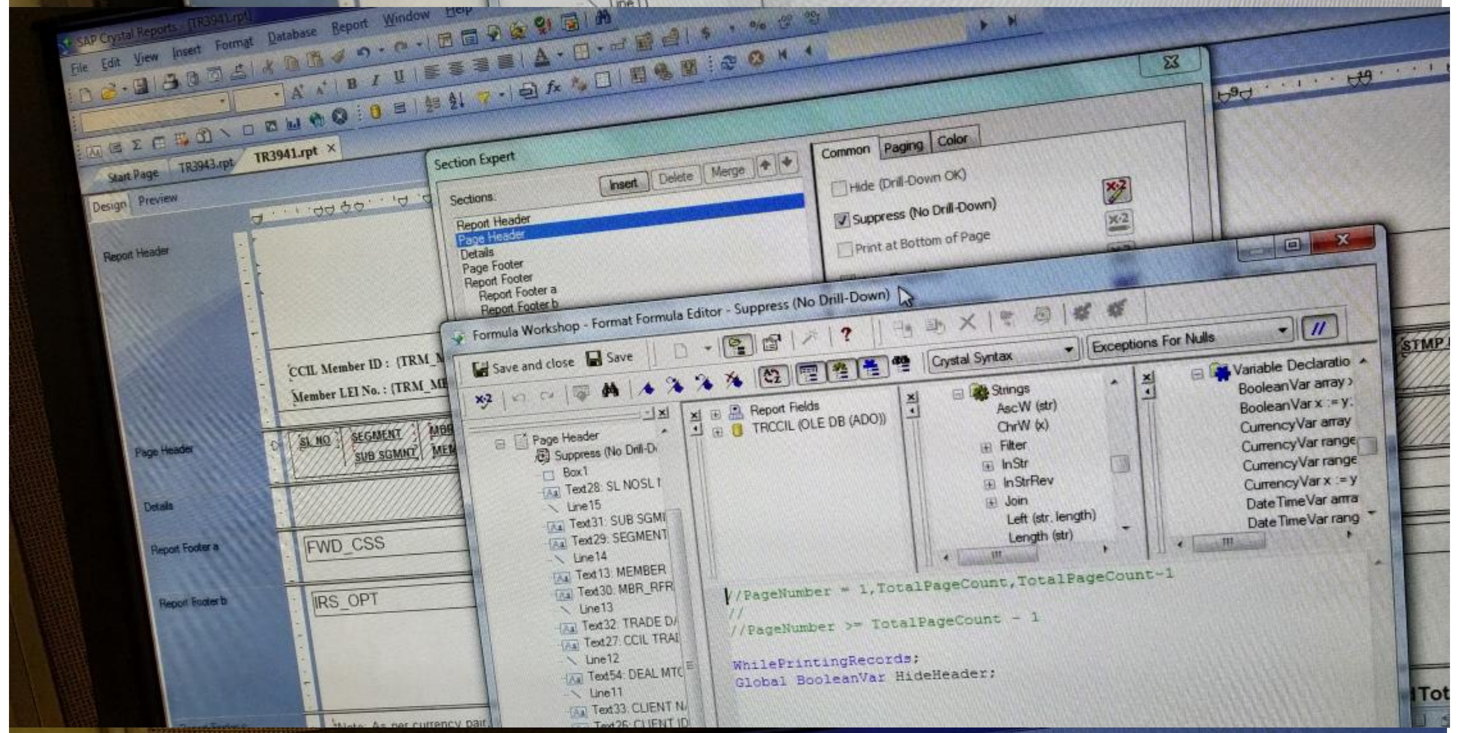
Formula Workshop - Formula Editor - 1

```
WhileReadingRecords
1
```



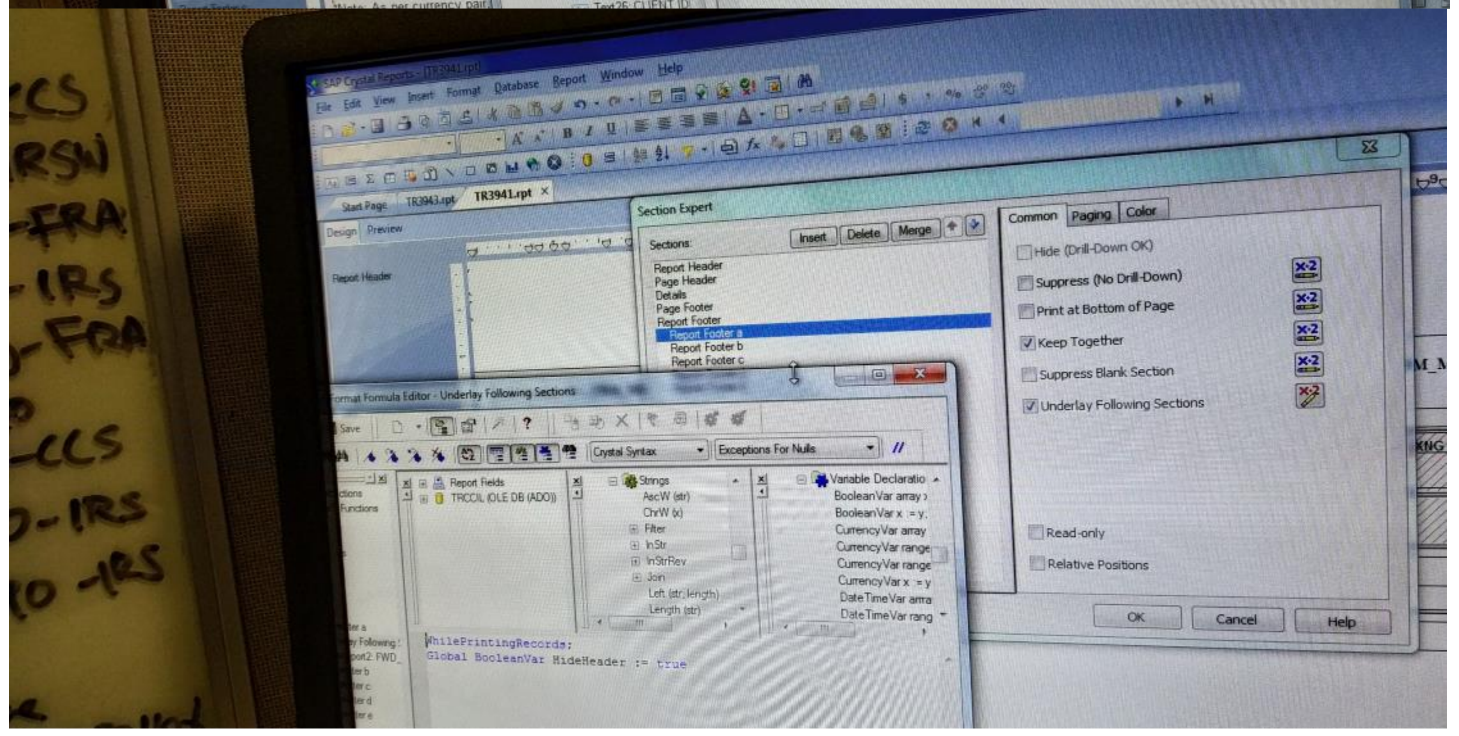
Formula Workshop - Format Formula Editor

```
//PageNumber = 1,TotalPageCount,TotalPageCount-1

PageNumber >= TotalPageCount - 1
```



Formula Workshop - Format Formula Editor - Suppress (No Drill-Down)

```
//PageNumber = 1,TotalPageCount,TotalPageCount-1
//
//PageNumber >= TotalPageCount - 1

WhilePrintingRecords;
Global BooleanVar HideHeader;
```



Section Expert — Sections: Report Header, Page Header, Details, Page Footer, Report Footer, Report Footer a, Report Footer b, Report Footer c

Common:
- [ ] Hide (Drill-Down OK)
- [ ] Suppress (No Drill-Down)
- [ ] Print at Bottom of Page
- [✓] Keep Together
- [ ] Suppress Blank Section
- [✓] Underlay Following Sections
- [ ] Read-only
- [ ] Relative Positions

Format Formula Editor - Underlay Following Sections

```
WhilePrintingRecords;
Global BooleanVar HideHeader := true
```

Shell

Shell is an environment in which we can run our commands, programs, and shell scripts.

Kernal

The kernel is a program that constitutes the central core of a computer operating system. The kernel itself does not interact directly with the user, but rather interacts with the shell and other programs as well as with the hardware devices on the system. The kernel is the first part of the operating system to load into memory during booting (i.e., system startup), and it remains there for the entire duration of the computer session because its services are required continuously.

Commands

Command	Description
clear	clear screen
who am i	Displays username, terminal id, logged in time and date
pwd	Present working directory. To save the present working directory to a variable 'd': d=$(pwd)
cal	Calendar. To display calendar of July 2016: cal 7 2006 or cal jul 2016
date	Displays date, time, timezone. To format date: date '+DATE:%d-%m-%y%nTIME:%H:%M:%S'. %n = newline
touch	Creates empty text files: touch *filename*
mkdir	Make directory: mkdir *foldername*
cd	Change directory: cd *foldername*
cat	To create file: cat > *filename* To view the text inside the file: cat < *filename* To merge two files: cat *filename1 filename2* > *filename3* > < means direction of data flow Append a file: cat >> *filename*
ctrl+d	End of document + Save
mv	Rename: mv *oldfilename newfilename*
rm	Remove file: rm *filename* To remove directory: rm -r *foldername* To remove directory: rmdir *foldername*
cp	Copy: cp *oldfilename newfilename*
ln	Links (duplicate) ln *originalfile linkedfile* --hard link; creates actual file ln -s *originalfile linkedfile* --soft link; creates shortcut to a file Number of hardlink -rw-rw-r--. 1 bert bert 0 Nov 5 21:55 orig -- 1 represents hardlink to itself (File) -rw-rw-r--. 2 bert bert 0 Nov 5 21:55 link -- 2 represents 1 hardlink has created (File) -rw-rw-r--. 2 bert bert 0 Nov 5 21:55 orig drwxrwxr-x. 2 bert bert 4096 Nov 5 21:26 a -- 2 represents link to itself and one to parent directory drwxrwxr-x. 3 bert bert 4096 Nov 5 21:26 a -- 3 represents one more directory has created inside.
ls	Lists files and folders To display the file/folder permissions(long listing): ls -l In permissions, 'd' indicates directory; '-' indicates file To display all files including hidden files: ls -a Read - 4 Write - 2 Execute - 1 4+2+1=7. So maximum value for a file/folder is 777. Owner, Group, Other Users.
umask	It is a system variable that stores a 3 digit number: When a file is created, unix will subtract this number from 666 for file and 777 for directory. Default number is 022 Ex: 666-022 = 644; Owner: Read, Write; Group: Read; Other Users: Read
chmod	Change permissions: chmod 777 *filename* chmod +x -- add execute
hide a file	To hide a file: insert a dot in-front of filename To show the hidden files: ctrl+h
uname	Displays Kernal name, Machine name, Kernal Version, Date etc. Display all information: uname -a
file	Shows type of files. Ex:ASCII text, Unicode text, Directory etc. All files in directory: file *
wc	(Word Count) Displays Number of lines, Number of words, Number of characters, filename: wc *filename* Carriage return will be also counted (which are not displayed \n etc.) To print only number of lines: wc -l *filename* To print only number of words: wc -w *filename* To print only number of characters: wc -c *filename*
sort	Arrange the lines in alphabetical order: sort *filename* To arrange the lines which are typed: sort Numerical sort: sort -n Sort in reverse order: sort -r Sort according to a specific column: sort -k*fromcolumn*, *tocolumn* . Ex: sort -k2,2 (sort by column 2) Sort using a designated tmp directory: $ sort -T *sortdirectory filename* Dictionary sort: 1,2,10 gets sorted as 1,10,2
cut	Used to display data from a delimited text file d=delimiter, fields columnname(s): cut -d"-" -f 1,3 *filnename*
dd	Used to convert and format Convert to upper case: dd if=*inputfile* of=*outputfile* conv=ucase Convert to European Coding Format: dd if=*inputfile* of=*outputfile* conv=ebcdic
man	Manual for commands press 'q' to come to terminal back
banner	Display text in # Display in single line (limit 10 characters): banner "text"
compress	Zip the file: compress -v *filename* To view the zipped file: zcat *filename* To uncompress: uncompress *filename*
echo	Print something To print variable: echo $*variablename*
#	Comment
sh	execute shell scripts: sh *filename*
read	To read the value into a variable: read *variablename1 variablename2*
Accent Graves (Reverse Quote)	Ex: There are some text in file1 . I want them in positional parameters Set will assign value to positional parameters: set cat file1; This will take $1: cat, $2: file1 To do that: set \`cat file1\` ; This will execute cat command first and then set command; This will take $1 from the file1 and $2 from the file1 etc.
expr	Instructs shell to perform arithmetic operation. Ex: expr $a + $b
\	Escape character. Ex: Multiplication: echo \`expr $a /* $b\` echo \`expr $a \\*  \\( $b + $c \\) / $d\`
bc	Used for real numbers instead of expr a = 10.5 b = 3.5 c = \`echo $a + $b | bc\` --pass echo to input of bc echo $c
Escape sequences	Newline: \n Carriage return: \r --the cursor comes back to initial position. So, if anything is print before, it will be replaced. Tab(5 spaces): \t Backspace: \b Same line: \c Bold : echo "\033[1m statement_to_print \033[0m" Print in reverse color: echo "\033[7m statement_to_print \033[0m" Back to normal format: \033[0m
tput	To clear the terminal (not even $): tput clear Calculate number of rows on screen: tput lines Calculate number of columns on screen: tput cols Position the cursor: tput cup *rownumber columnnumber* Activate bold: tput bold
Command Success or not	To check the command was success or not. Success= 0 or Failed=1. echo $?

If - Else	if (*condition*) then
	        statements
	else
	        statements
	fi

If - Else - If	if (*condition*) then
	        statements
	elif(*condition*) then
	        statements
	else
	        statements
	fi

-lt	Less than: [$num -lt 10]
-gt	Greater than: [$num -gt 10]
-eq	Equal to: [$num -eq 10]
-le	Less than or equal to: [$num -le 10]
-ge	Greater than or equal to: [$num -ge 10]
-ne	not equal to: [$num -ne 10]
-f	Check whether file (not directory) exists: [-f $*variablename*]
-d	Check whether directory exists: [-d $*variablename*]
-c	Check whether character special files exists: [-c $*variablename*]
-b	Check whether block special files (images, videos) exists: [-b $*variablename*]
-r	Check whether there is read permission: [-r $*variablename*]
-w	Check whether there is write permission: [-w $*variablename*]
-x	Check whether there is execute permission: [-x $*variablename*]
-s	Check whether the file size >0: [-s $*variablename*]
String checks	equal: ["$str1" = "$str2"] not equal : ["$str1" != "$str2"] length of string > 0 : [-n "$str1"] length of string = 0 :[-z "$str1"]
and operator	if [$num -le 100 -a $num -ge 50]
or operator	if [$num -le 100 -o $num -ge 50]

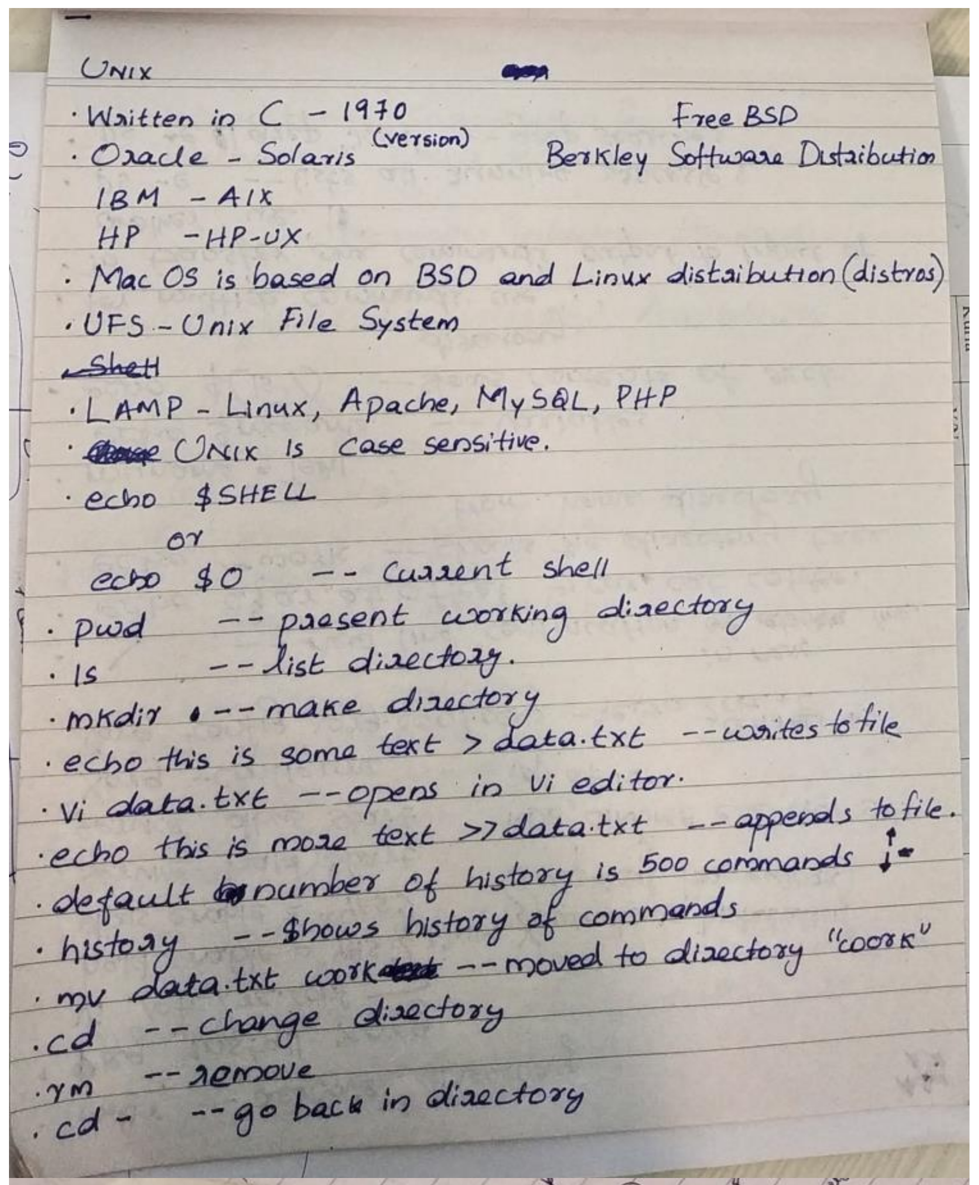
case operator	case $*variable* in
	        [a-z])
	                statement ;;
	        [A-Z])
	                statement ;;
	        ?) --one character
	                statement ;;
	        *) --default case
	                statement ;;
	esac
	
	case $word in
	        [aeiou]* | [AEIOU]*) --match the first character
	                statement ;;
	        [0-9]*)
	                statement ;;
	        *[0-9]) --match the end character
	                statement ;;
	        ???) --match a three letter word
	                statement ;;
	        *) --default case
	                statement ;;
	esac

while loop	count=1
	while [$count -le 10]
	do
	        statement
	done

until loop	count=1
	until [$count -ge 10]
	do
	        statement
	done

for loop	for item in *
	do
	        if [-d $item] then
	                echo("directory")
	        fi
	done

grep	Used to search patterns in text file grep *findword filename* grep -i money text_file -- case insensitive grep -i -n money text_file -- line numbers grep -i -n -c money text_file --number of lines grep -i -v money text_file --lines which are not matching will be displayed
passwd file	username, password(encrypted), userid, groupid, computer_name, home_folder, default shell (bash)
IFS	delimiter for set command; default is space IFS =: --change delimiter (define before set command)
Change default input to file	terminal = \`tty\` --tty contains settings of the terminal exec < $*filename* while read line -- line is a variable do echo $line --line from the text file done exec < $terminal --change back the settings of the terminal which is saved in the variable
sleep	sleep *number_of_seconds* sleep 2
break, exit, continue	loop control statements break: forces the loop to stop working continue: skips the statements after the continue and continue the loop from next value exit: exit from loop
multiple commands	ls ; cal; banner "hello" cat text_file && echo "prev command was Success" cat text_file || echo "Doesn't matter if the prev command was success"
write	communicate with different user mesg y -- enable message reception mesg n -- disable message reception write *username* finger --show the users who are logged in. It shows * symbol to the terminal name who have disabled message reception.
function	function_name() {echo " hello"} . func.sh -- dot space *function_file_name* (now functions will be stored in memory) enter the function name to run the function: *function_name* unset *functionname* -- remove the function
multiple scripts	Type sh scipt_name.sh inside the script
set	Set parameters to positional parameters Ex: set hi this is a test echo $1 : hi echo $2 : this echo $*: hi this is a test To see number of positional parameter created: echo $# $1,$2,$3 indicates the position of the argument

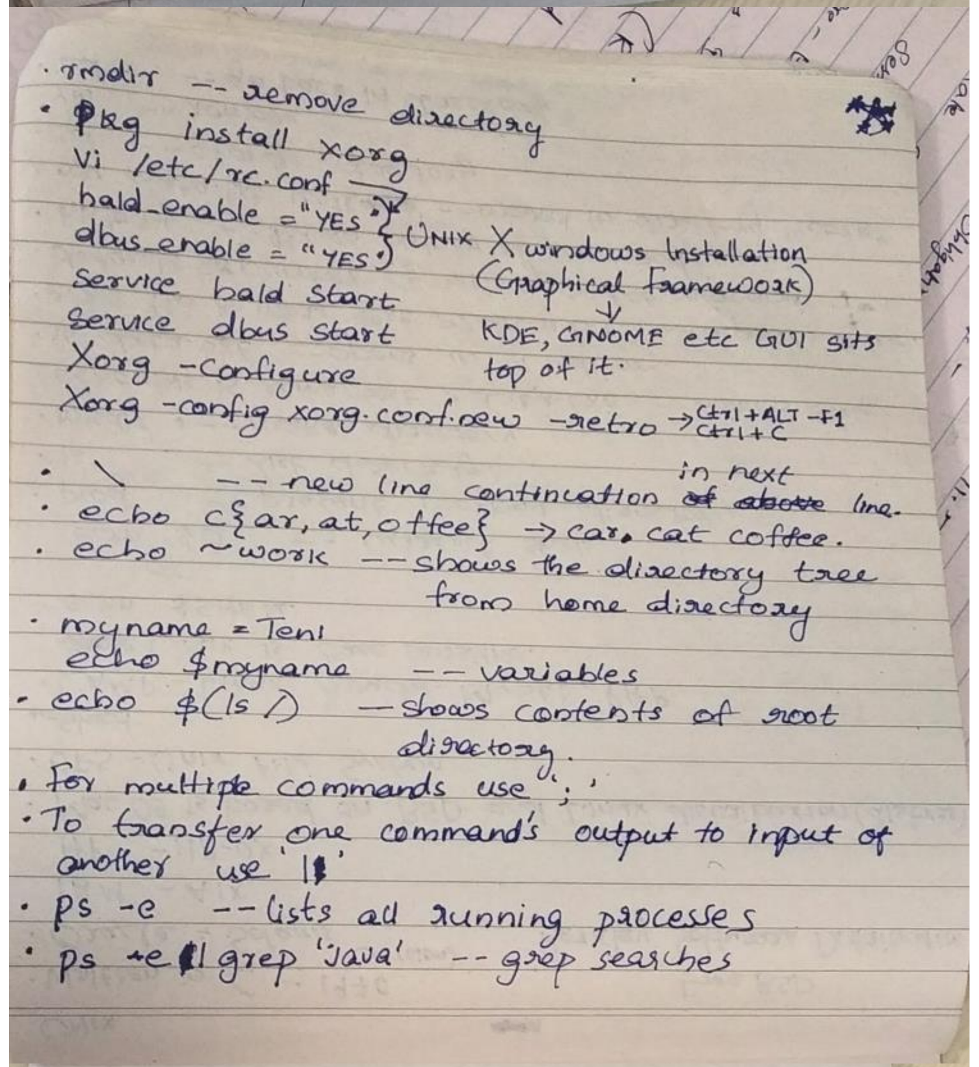


Unix

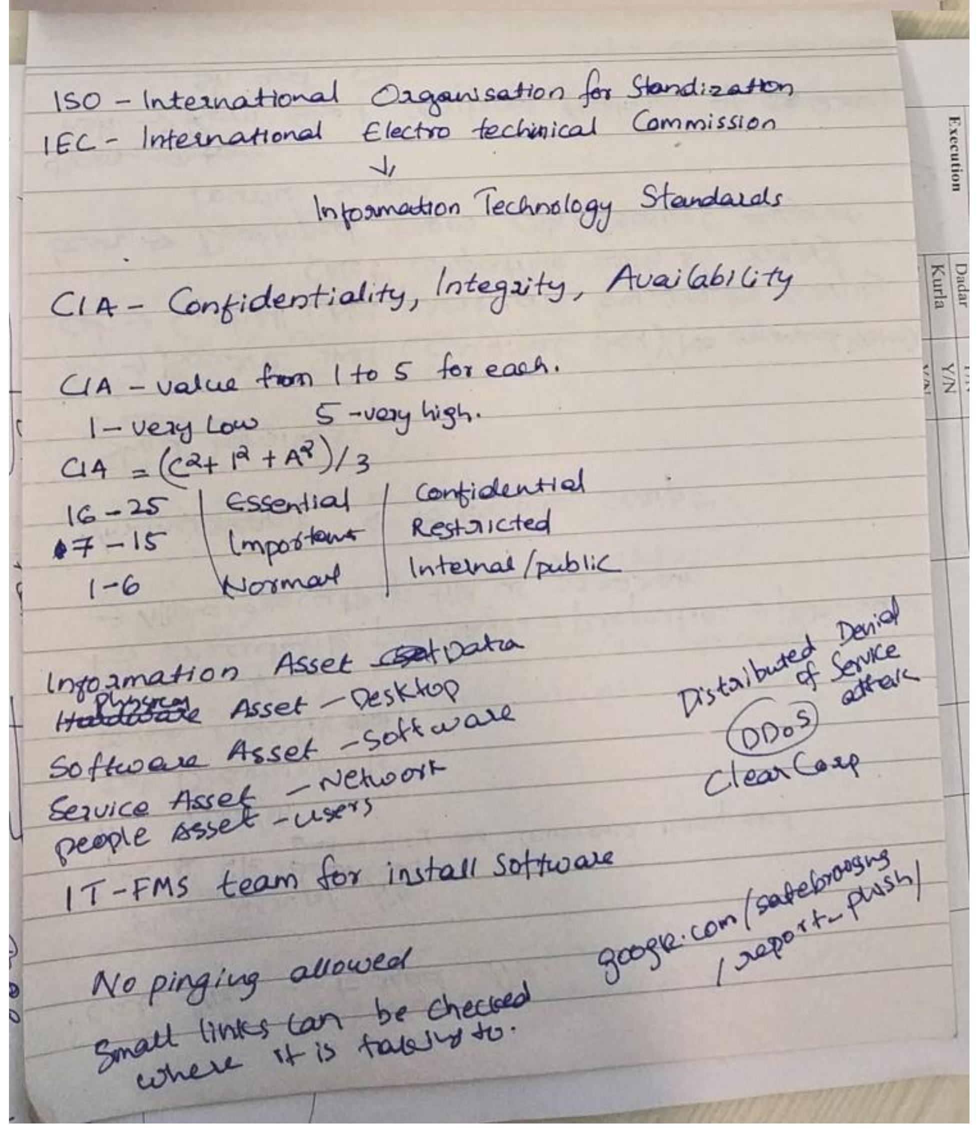
- Written in C - 1970 — Free BSD
- Oracle - Solaris (version) — Berkley Software Distaibution
  - IBM - AIX
  - HP - HP-UX
- Mac OS is based on BSD and Linux distaibution (distros)
- UFS - Unix File System

~~Shell~~

- LAMP - Linux, Apache, MySQL, PHP
- UNIX is case sensitive.
- echo $SHELL
  or
  echo $0 -- Current shell
- pwd -- present working directory
- ls -- list directory.
- mkdir -- make directory
- echo this is some text > data.txt -- writes to file
- vi data.txt -- opens in vi editor.
- echo this is more text >> data.txt -- appends to file.
- default number of history is 500 commands
- history -- shows history of commands
- mv data.txt work -- moved to directory "work"
- .cd -- change directory
- .rm -- remove
- .cd - -- go back in directory



- rmdir -- remove directory
- pkg install xorg
  - vi /etc/rc.conf
  - hald_enable = "YES" } UNIX X windows Installation (Graphical framework)
  - dbus_enable = "YES"
  - service hald start
  - service dbus start — KDE, GNOME etc GUI sits top of it.
  - Xorg -configure
  - Xorg -config xorg.conf.new -retro → Ctrl+ALT-F1, Ctrl+C
- \ -- new line continuation ~~of above~~ in next line.
- echo c{ar, at, offee} → car, cat coffee.
- echo ~work -- shows the directory tree from home directory
- myname = Teni
  echo $myname -- variables
- echo $(ls /) -- shows contents of root directory.
- For multiple commands use ';'
- To transfer one command's output to input of another use '|'
- ps -e -- lists all running processes
- ps -e | grep 'Java' -- grep searches



ISO - International Organisation for Standization
IEC - International Electro techinical Commission
↓
Information Technology Standards

CIA - Confidentiality, Integrity, Availability

CIA - value from 1 to 5 for each.
1 - very Low   5 - very high.

$CIA = (C^2 + I^2 + A^2)/3$

16 - 25	Essential	Confidential
7 - 15	Important	Restricted
1 - 6	Normal	Internal / public

Information Asset - Data
~~Hardware~~ Physical Asset - Desktop
Software Asset - Software
Service Asset - Network
people Asset - users

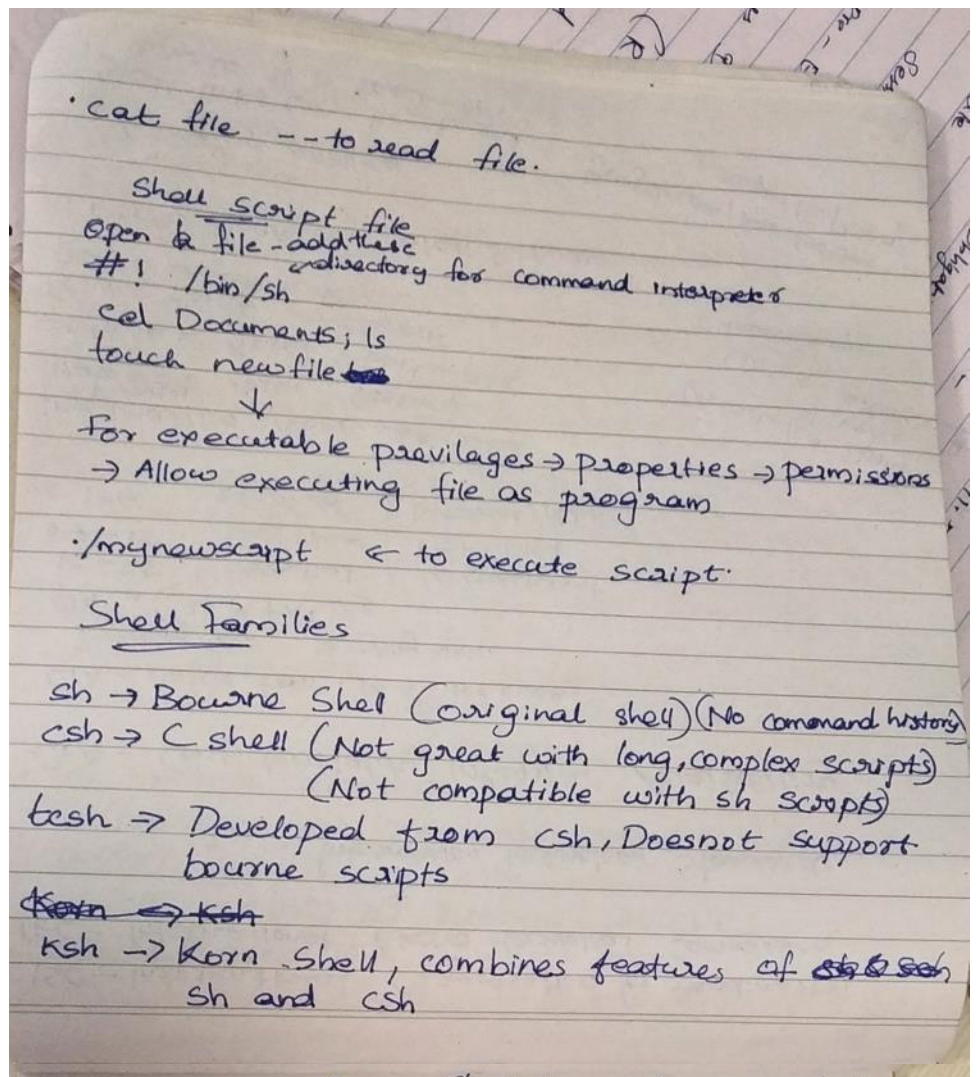
Distributed Denial of Service attack
DDoS
ClearCorp

IT-FMS team for install Software

No pinging allowed

Small links can be checked where it is taking to.

google.com/safebrowsing/report_phish/



- cat file -- to read file.

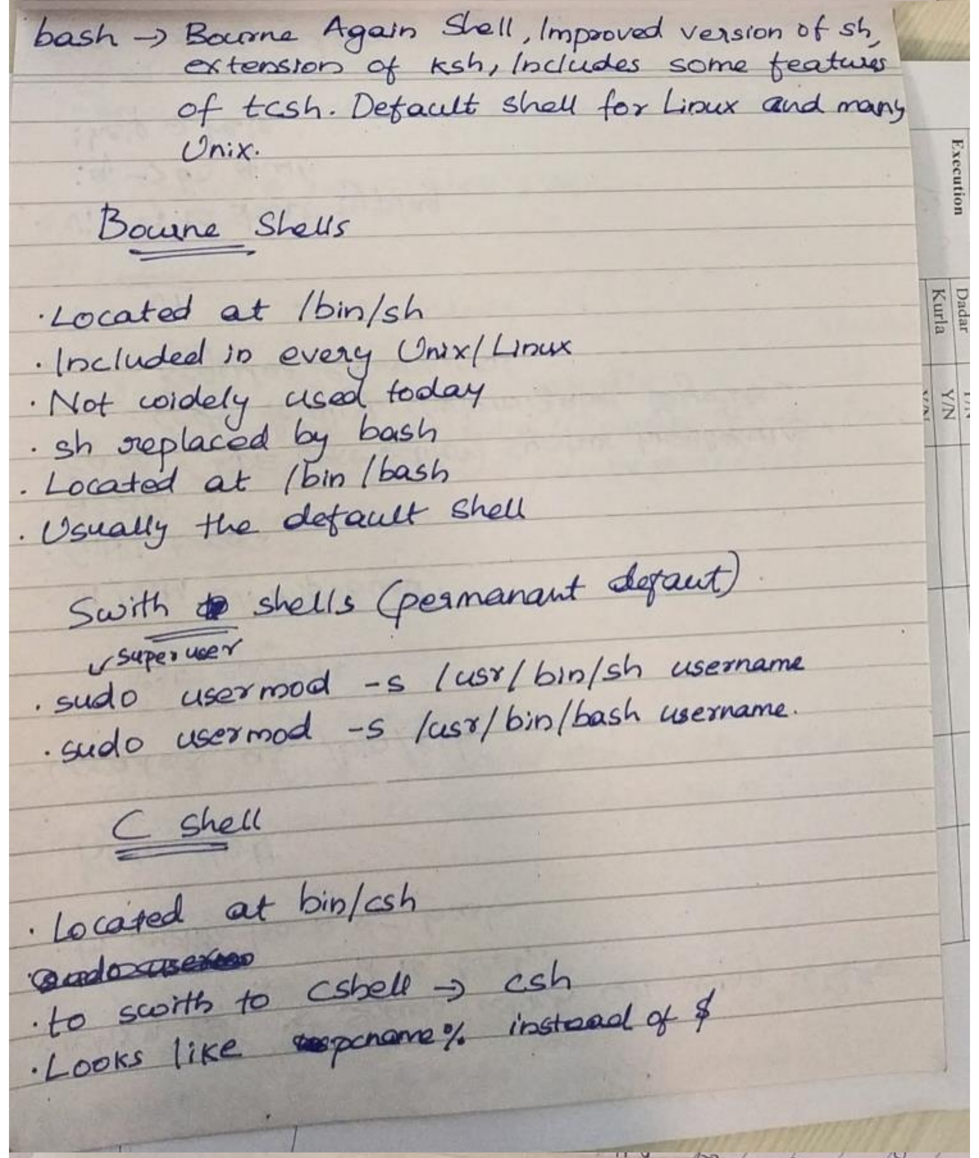
Shell script file

Open a file - add these
`#! /bin/sh` ← directory for command interpreter
cd Documents; ls
touch newfile
↓
For executable previlages → properties → permissions → Allow executing file as program

./mynewscript ← to execute script.

Shell Families

- sh → Bourne Shell (original shell) (No command history)
- csh → C shell (Not great with long, complex scripts) (Not compatible with sh scripts)
- tcsh → Developed from csh, Doesnot support bourne scripts
- ~~Korn → ksh~~
- ksh → Korn Shell, combines features of sh and csh



- bash → Bourne Again Shell, Improved version of sh, extension of ksh, Includes some features of tcsh. Default shell for Linux and many Unix.

Bourne Shells

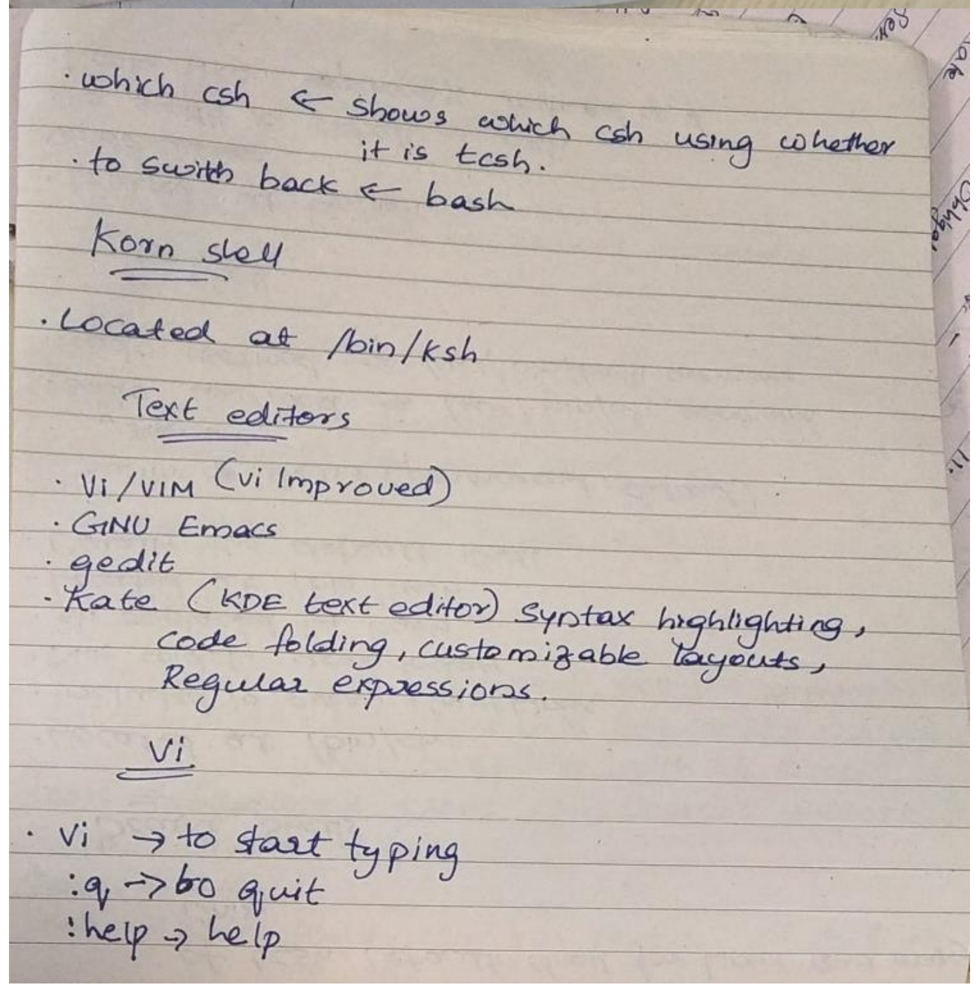
- Located at /bin/sh
- Included in every Unix/Linux
- Not widely used today
- sh replaced by bash
- Located at /bin/bash
- Usually the default shell

Swith shells (permanant default)

✓ super user
- sudo usermod -s /usr/bin/sh username
- sudo usermod -s /usr/bin/bash username.

C shell

- Located at bin/csh
- to swith to cshell → csh
- Looks like pcname% instead of $



- which csh ← shows which csh using whether it is tcsh.
- to swith back ← bash

Korn shell

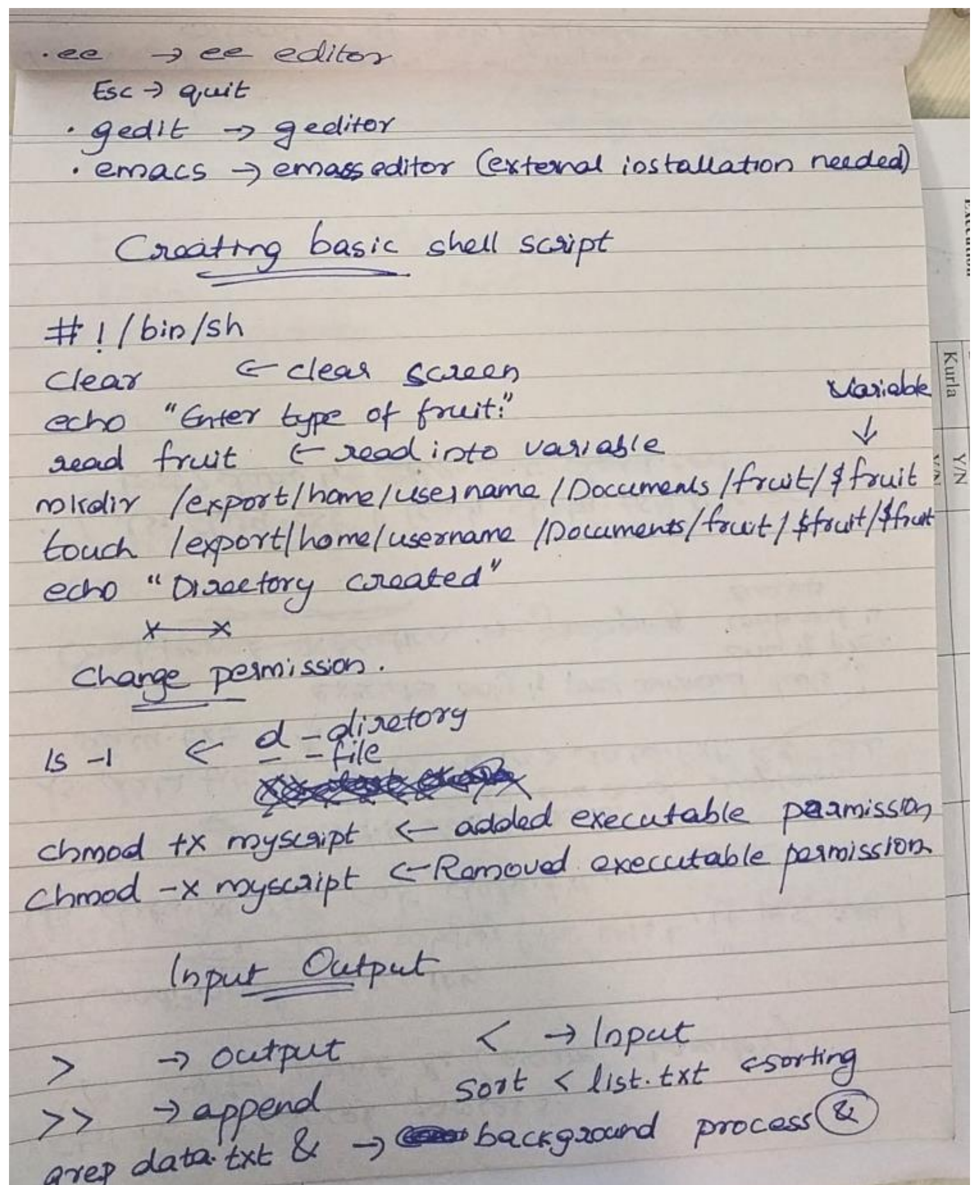
- Located at /bin/ksh

Text editors

- Vi/VIM (vi Improved)
- GNU Emacs
- gedit
- Kate (KDE text editor) syntax highlighting, code folding, customizable layouts, Regular expressions.

Vi

- vi → to start typing
  - :q → to quit
  - :help → help



- ee → ee editor
  - Esc → quit
- gedit → geditor
- emacs → emacs editor (external installation needed)

Creating basic shell script

```
#!/bin/sh
clear        ← clear screen
echo "Enter type of fruit:"
read fruit   ← read into variable
mkdir /export/home/username/Documents/fruit/$fruit
touch /export/home/username/Documents/fruit/$fruit/$fruit
echo "Directory created"
```
Variable

Change permission.

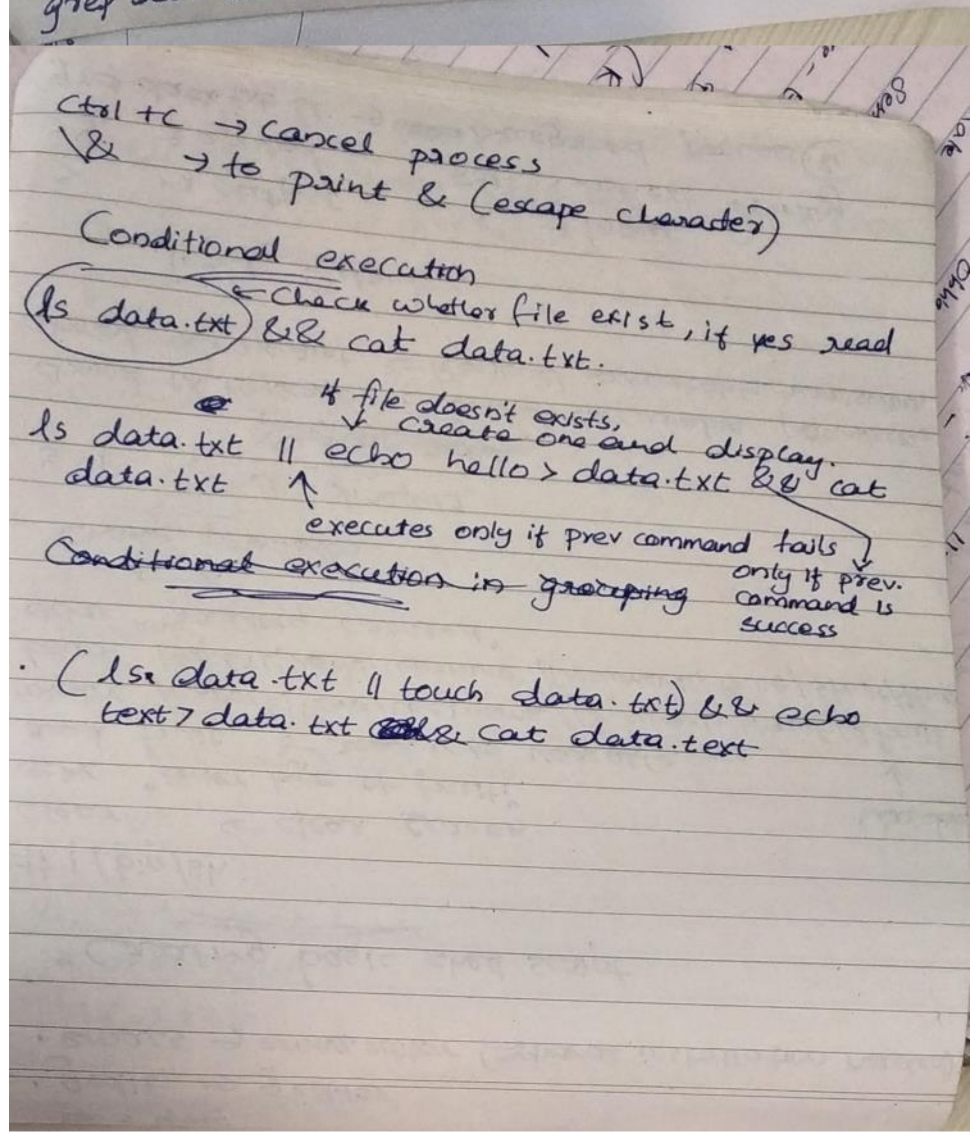
ls -l ← d - diretory, - - file

chmod +x myscript ← added executable permission
chmod -x myscript ← Removed executable permission

Input Output

> → output	< → Input
>> → append	sort < list.txt ← sorting

grep data.txt & → background process &



Ctrl + C → cancel process
\& → to print & (escape character)

Conditional execution

(ls data.txt) && cat data.txt. ← Check whether file exist, if yes read

ls data.txt || echo hello > data.txt && cat data.txt

If file doesn't exists, create one and display.
executes only if prev command fails
only if prev. command is success

~~Conditional execution in grouping~~

- (ls data.txt || touch data.txt) && echo text > data.txt && cat data.text