Reference Notes of Oracle 11g SQL Part 1

Oracle Trainer :- Sekhar

Oracle is a RDBMS.

RDBMS means Relational Database Management System

Dr. E.F codd is the father of RDBMS.

Examples of RDBMS

Oracle, MySQL, SQL Server, Access, IBM-SQL, Paradox, Posgre SQL & MariaDB.

Latest versions of Oracle

Oracle 11g, Oracle 12c, Oracle 18c & Oracle 19c.

SQL (structed query language)

PL/SQL : procedural language sturctured query language.

Database enginee is the interactor between oracle and the user.

Important queries are as follows

To connect to oracle user for example system is the super user.

Double click on SQL *plus and oracle will ask you the user name and password.

Give it accordingly.

```
SQL> connect
Enter user-name: system
Enter password:
Connected.
```

****

sql> cl scr;

It is used to clear the screen.

****

2) sql> select * from tab;

It is used to see the list of current tables in the user.

****

Date in oracle sql is always in the format of DD-Mon-YY

To check today current the query is as follows.

```
SQL> select sysdate from dual;

SYSDATE
---------
02-APR-20
```

****

To create a table called salespeople query is as follows.

```
SQL> create table salespeople
  2  (snum number(5) primary key,
  3    sname char(25),
  4    city varchar2(20),
  5    comm number(11,2));
```

Creating a basic table involves naming the table and defining its columns and each column's data type.

The SQL CREATE TABLE statement is used to create a new table.

==

Primary key

****

Primary key is used to give uniqueness to that table through the attribute which will be declared as primary key.

Is short primary key is allowed only once in a table.

Null has a special values in Oracle.

****

To see the structure of the table.

SQL> desc salespeople;

```
 Name                              Null?    Type
 --------------------------------------- -------- ----------------------------
 SNUM                              NOT NULL NUMBER(5)
 SNAME                                      CHAR(25)
 CITY                                       VARCHAR2(20)
 COMM                                       NUMBER(11,2)
```

SQL> describe salespeople;

```
 Name                              Null?    Type
 --------------------------------------- -------- ----------------------------
 SNUM                              NOT NULL NUMBER(5)
 SNAME                                      CHAR(25)
 CITY                                       VARCHAR2(20)
 COMM                                       NUMBER(11,2)
```

****

To insert or add a record to the table. The query will be as follows.

sql> INSERT INTO salespeople values(1001,'James Bond','New York',7788.55);

The above query will add 1 record to the table;

or

sql> insert into salespeople values

(&snum,'&sname','&city',&comm);

In the above query after adding the record you can give the / command to add many more records

/ command is used to repeat the last query.

In case you want leave certain fields or attributes blank in case that field does not have any constraint or any keys associated with it in that case the query will be as follows.

sql> INSERT INTO salespeople values(1234,'Dr. Dinesh', 'New York', Null);

in the above query you will use null value to leave a particular attribute blank.

****

sql> commit;

to save the tupples permanently

commit command should be given if auto commit is off.

Commit should be given after updating or deleting or adding new records.

----

sql> set autocommit on;

This is used to commit every query given by the user.

----

sql> set autocommit off;

This is used to off the auto commit;

****

To see the number of records or tupples in the table the query is

SQL> select * from salespeople;

****

sql> select snum, sname, city from

salespeople;

The above query will display only particular attributes from the table (in short the above query is for display particular attributes.

****

12)DROP TABLE table_name;

sql> DROP TABLE employees;

The table will be dropped

The SQL DROP TABLE statement is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

****

Where clause

It is used for giving conditions and fetching the records you want.

The SQL WHERE clause is used to specify a condition while fetching the data from single table or joining with multiple table.

If the given condition is satisfied then only it returns specific value from the table.

The WHERE clause not only used in SELECT statement, but it is also used in UPDATE, DELETE statement

write a query to display the salespeople who reside in london.

```
SQL> select * from salespeople
  2   where city ='London';


     SNUM SNAME                   CITY                    COMM
---------- ----------------------- ------------------- ----------
COUNTRY
------------------------------
     1090 Dr. Jun Jun Wala         London                 1810.14
     1400 Dr. Rahul               London                  9579


     SNUM SNAME                   CITY                    COMM
---------- ----------------------- ------------------- ----------
COUNTRY
------------------------------
     8977 Shri amit               London
--------------------------------------------------------------------------------
```

Waq to print particular attributes for salesperson residing in london.

```
SQL> select snum, sname, city  from salespeople
  2  where city = 'London';

     SNUM SNAME                    CITY
---------- ------------------------ --------------------
      1001 Kalia                   London
      1090 Dr. Jun Jun Wala         London
      1400 Dr. Rahul               London
      8977 Shri amit               London
-------------------------------------------------------------------------
```

Write a query where you will print snum, sname, city for a person whose salesman no is 1400;

```
SQL> select snum, sname, city
  2  from salespeople
  3   where snum =1400;

     SNUM SNAME                    CITY
---------- ------------------------ --------------------
      1400 Dr. Rahul               London
-------------------------------------------------------------------
```

Write a query where you will print all salesperson residing in London or Mumbai.

```
SQL> select snum, sname, city
  2  from salespeople
  3  where city = 'London' or city ='Mumbai';

     SNUM SNAME                    CITY
```

```
---------- ------------------------ --------------------
     1001 Kalia                London
     1090 Dr. Jun Jun Wala         London
     1400 Dr. Rahul              London
     8977 Shri amit              London
----------------------------------------------------------
```

<u>Write a query where you will print all salespeople whose name is "Dr. Rahul"</u>

```
SQL> select * from salespeople
  2   where sname ='Dr. Rahul';
```

Reference Notes of Oracle 12c SQL Part 2

Oracle Trainer :- Sekhar

Constraints

****

There are 6 types of constraints in oracle.

1st one is Primary key,

2nd one is Null.

Not null

Foreign key,

check clause

Default

----

query

******

waq where you will create a table called employees which will have the following constraints, Empno primary key, ename cannot be left blank and basic salary has to be minimum Rs. 2,400/- and city default 'Mumbai'

```
SQL> create table  employees
  2  (empno number(5) primary key,
  3   ename char(20) Not Null,
  4   doj    date,
  5   basic number(9,2) check(basic >=2400),
  6   city   varchar2(19) Default 'Bengaluru');

Table created.
```

While trying to add records in following scenarios

****

```
SQL> insert into employees
  2   values(1004, null, sysdate, 8999, Default);
 values(1004, null, sysdate, 8999, Default)
              *
ERROR at line 2:
ORA-01400: cannot insert NULL into
```

```
SQL> insert into employees
  2   values(1004, 'DinDayal', sysdate, 1200, Default);
insert into ggemployee
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C007585) violated
```

```
SQL> insert into employees
  2   values(1004, 'DinDayal', sysdate, 1450, 'London');


1 row created.
SQL> commit;
```

===

<u>IN Clause</u>

****

In Clause works faster for fetching records and when table has huge database.

It can be used with all char, varchar2, date and number attributes.

waq where you will print all details of snum 1001, 1008, 1004, 1090

```
SQL> select snum, sname, city
  2  from salespeople
  3   where snum in (1001, 1008, 1004, 1090);

      SNUM SNAME                    CITY
---------- ------------------------ --------------------
      1001 Bill gates               Navi Mumbai
      1090 Dr. Jun Jun Wala         London
      1008 James Bond               Mumbai
```

----

waq where u will print all salesperson residing in London or newyork or chicago or mumbai.

```
SQL> select snum, sname, city
  2  from salespeople
  3   where  city in ('London', 'New York', 'Chichago', 'Mumbai');

      SNUM SNAME                    CITY
---------- ------------------------ --------------------
      1090 Dr. Jun Jun Wala         London
      1008 James Bond               Mumbai
      1400 Dr. Rahul                London
      8977 Shri amit                London
```

Waq where you will display snum, sname and city of salesperson not residing in

London, newyork or chicago or mumbai.

```
SQL> select snum, sname, city
  2  from salespeople
  3   where  city not in ('London', 'New York', 'Chichago', 'Mumbai');

      SNUM SNAME                    CITY
---------- ------------------------ --------------------
      1001 Kalicharan               Navi Mumbai
```

```
1234 seema              bihar
1456 Ranjit singh         Jaipur
1040 Rana Pratap           Los Angeles
```

----

waq to print snum, sname, and city of salespeple whose sales number should not be 1008, 1001, 1004 and 1090

```
SQL> select snum, sname, city
  2  from salespeople
  3   where snum not  in (1001, 1008, 1004, 1090);
```

****

<u>Important string Functions</u>

****

<u>Uppper and lower functions in the same query.</u>

```
SQL> select upper(sname) name, lower(city) city, comm
  2  from salespeople;
```

----

lpad( ) Function

```
SQL> select lpad('Rama was a great king  ', 72, '*') lpad
  2  from dual;
```

RPAD()

****

```
SQL> select rpad('Rama was a great king  ', 72, '*') rpad
  2  from dual;
```

InitCap() : will print every words first letter in capital

```
SQL> select initcap(sname) from salespeople;

SQL> select initcap(sname)sname,city  from salespeople;
```

Ltrim() : will remove the left trailing blank spaces from the string.

****

```
SQL> select ltrim('       Suresh is the V.C of Bangalore University
  2  ') from dual;
```

Rtrim( ) :-

****

```
SQL> select rtrim('      Suresh is the V.C of Bangalore University   ') from dual;
```

Length() : -

```
SQL> select length(' India wins world cup of football in 2040      ')
  2  from dual;
```

```
SQL> select length(' Jaipur is a nice city      ') from dual;
```

```
SQL> select length(trim(' Jaipur is a nice city      ')) from dual;
```

```
SQL> select trim(' Jaipur is a nice city      ') from dual;

TRIM('JAIPURISANICECI
---------------------
Jaipur is a nice city
```

```
SQL> select substr('   White house  is a nice fort  ', 5, 8) substr
  2  from dual;
```

Alter Table.

The ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

```
sql>ALTER TABLE salespeople
   ADD DateofJoin Date;
```

The above query will add a column to the table employee;

To drop a column in a table

```
sql>ALTER TABLE salespeople
   DROP COLUMN DateOfjoin
```

----

to modify a column in a table query is

```
sql>alter table salespeople
     modify comm number(12,2);
```

Delete : is used to delete all records.

****

waq to delete all records in orders table.

```
sql> delete from orders;
```

If commit has not been given deleted records can be rolled back.

```
sql> rollback;
```

sql>select * from orders;

<u>waq to delete all salespeople of london city;</u>

```
sql>  delete from salespeople
          where city = 'London';


sql> commit;
```

==

waq to remove details of salesman no 1004;

```
sql> delete from salespeople
         where snum =1004;
```

<u>Update</u>

Update is use to modify the records provided you have permission.

waq to update all records where commission is increased by 200 rupees for all employees

```
sql>update salespeople
       set comm = comm +200;


sql> update salespeople
        set comm = comm -100
         where city = 'London';


sql> update customers
        set city ='New York',  Name = 'Rama'
          where cnum = 2009;
```

<u>Foreign Key</u>

****

Foreign key is a key which is a primary key in another table.

```
SQL> desc salespeople;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SNUM                                      NOT NULL NUMBER(5)
 SNAME                                              CHAR(25)
 CITY                                               VARCHAR2(20)
 COMM                                               NUMBER(12,2)
 ----------------------------------------- -------- ----------------------------
```

Creating customer table with snum as foreign key connecting to parent table salespeople;

```
SQL> create table customers
  2  (cnum number(5) primary key,
  3    cname char(28),
  4    city varchar2(20),
  5     snum number(5) references salespeople(snum));

Table created.
```

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

Creating Orders table with snum and cnum as foreign key connecting to respected parent tables salespeople and customers.

```
SQL> create table orders
  2    (onum number(5) primary key,
  3     odate date,
  4     oamount number(11,2),
  5     snum number(5) references salespeople (snum),
  6     cnum number(5) references customers (cnum));
Table created.
```

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

<u>SQL - LIKE Clause</u>

****

The SQL LIKE clause is used to compare a value to similar values using wildcard operators.

There are two wildcards used in conjunction with the LIKE operator:

The percent sign (%) & The underscore (_)

The percent sign represents zero, one, or multiple characters.

The underscore represents a single number or character.

==

```
sql> SELECT * FROM salespepeople
       WHERE empname like 'D%';
```

The above query will display whose names begins with D

% sign represent any characters but the first character must begins with character D.

--

waq where u will display salespeople whose city name begins with A

```
sql>SELECT * FROM salespeople
      WHERE city like 'A%';
```

---

w.a.q wher you will print the sname, city, comm for all people residing in London(Use like operator)

```
sql>Select snum, sname, city  FROM salespeople
    WHERE city  like 'L%';
```

==

_ (underscore) in like operator represent 1 character or number or space or special symbol.

```
sql>
select * from salespeople
where sname like '_____';
```

In the above query we will display only those names which are of 5 characters.

one _ underscore represent one character.

```
sql> select * from salespeople
where city like '______';
```

The above query will print city whose name size is of 6 characters.

```
sql>select * from customers
where cname like 'a%';
```

The above query will display all those cnames that begins with a

<u>Between Operator</u>

****

The BETWEEN operator is used to select values within a range.

```
sql> SELECT * FROM salespeople
    WHERE comm between 10000 and  20000;
```

The following SQL statement selects all salespeople whose commission is between 10000 and 20000

<u>To display the employees outside the range of the previous example, use NOT BETWEEN:</u>

<u>Example</u>

```
sql> SELECT * FROM customers
    WHERE comm NOT BETWEEN 10000 AND 20000;
```

The above query will print only those salespeople whose salary does not fall in the above range.

<u>ORDER BY Clause</u>

****

The SQL ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns.

You can use more than one column in the ORDER BY clause.

Make sure whatever column you are using to sort, that column should be in column-list.

waq where you will sort on employee name sorting based on salary in ascending order.

```
SQL> SELECT * FROM employee
```

```
    ORDER By SALARY;
```

<u>**Following is the query where we sort only by name in ascending order.**</u>

```
SQL> SELECT * FROM EMPLOYEE
    order by empname;
```

```
sql> select empno, city, basic from employee
    order by city;
```

in the above query the records are sorted based on city in ascending order.

<u>**Following is an example which would sort the result in descending order by city:**</u>

```
SQL> SELECT * FROM employee
    ORDER BY city DESC;
```

or

```
sql> select empname, city from employee
    order by city desc;
```

****

Reference Notes of Oracle 12c SQL Part 3

Oracle Trainer :- Sekhar

Oracle uses ROWNUM to fetch limited number of records.

To execute in oracle the command is

```
sql> select * from salespeople
    where rownum <=4;
```

The above query will execute the top 4 records.

===

SQL - Distinct Keyword

****

The SQL DISTINCT keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

Syntax:

```
sql>select distinct city from salespeople;

sql> select count(distinct city) from salespeople;
```

===

Between Operator

****

The BETWEEN operator is used to select values within a range.

The SQL BETWEEN Operator

The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.

===

<u>NOT BETWEEN Operator Example</u>

BETWEEN Operator with Text Value Example

****

The following SQL statement selects all salespeople whose name begins

with any of the letter BETWEEN 'C' and 'M':

```
sql> SELECT * FROM salespeople;
    WHERE Name  BETWEEN 'C' AND 'M';
```

BETWEEN Operator with IN Example

****

The following SQL statement selects all employees with a salary

BETWEEN 50000 AND 150000

but employee number of 99, 2 or 390 should not be displayed:

```
sql> SELECT * FROM employee
    WHERE (basic BETWEEN 50000 AND 150000)
             AND NOT empno IN (99,2,390,66);
```

In the above query we are using the a sub query and also the in operator

with not operator so that only those employees are print who draw salary

in the given range but particular employees no should not be printed due to

as per client requirement.

NOT BETWEEN Operator with Text Value Example

****

The following SQL statement selects all employees with empno

beginning with any of the letter NOT BETWEEN 'C' and 'M':

```
sql>SELECT * FROM employee
   WHERE Name not   BETWEEN 'C' AND 'M';
```

==

Assume for the following examples we have an employee table.

****

```
SQL> SELECT * FROM employee
        ORDER BY NAME;
```

The above query will sort the record based on name in ascending order.

```
sql> select empname, basic from employee
     order by basic desc;
```

the abovery query will sort in descending order.

****

<u>Group By</u>

****

The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

If you want to know the total amount of salary on each designation,

then GROUP BY query would be as follows:

```
SQL> SELECT desig, SUM(basic) FROM employee
    GROUP BY desig;
```

In the above query group by is used so that the query in the memory of the computer will group the records based on designation and then each group will have only 1 output in the screen along with the salary total of each group.

---

write a query where you will print the maximum salary drawn in each group using aggregate functions.

```
sql> SELECT desig, Max(basic) FROM employee
    GROUP BY desig;
```

```
sql> SELECT city SUM(basic) FROM employee
    GROUP BY city;
```

in the above query we are showing city wise salary using the group by clause.

```
sql> SELECT city, min(basic) FROM employee
    GROUP BY city;
```

in the above query we are showing city wise minimum salary using the group by clause.

```
sql> SELECT city, avg(basic) FROM employee
    GROUP BY city;
```

In the above query we will be showing city wise average salary.

```
sql>SELECT city, avg(basic)
    max(basic), min(basic), sum(basic)
    FROM employee
    GROUP BY city;
```

The above query will print city wise sum, max, min & average salary.

```
sql> select city, count(basic)
    from employee
    group by city;
```

the above query will print the number of employees in each city.using count function.

<u>The HAVING Clause</u>

****

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

The WHERE clause places conditions on the selected columns, where as the HAVING clause places conditions on groups created by the GROUP BY clause.

```
sql> SELECT Desig, count(desig)
FROM employee
GROUP BY Desig
HAVING COUNT(Desig) >= 2;
```

The above query will display those designation that appears more than twice in the table and for that we are using the group by and having clause and the agregate function.

==

waq where you will print all those city where their are 2 or less than 2 salespeople;

```
sql> select city, count(city)
       from salespeople
        group by city
          having count(city) <=2
          order by city desc;
```

<u>Views</u>

****

Views are logical table based on real table which are called base tables.

In views our query is stored which can be executed from time to time.
You can add records, modify records through views.

Write a view called london to see salespeople who residing in london.

```
sql> create view london
         as select * from salespeople where city = 'London';


sql> select * from london;
```

The above query will display those salespeople who stay in london.

To add records through view london

```
sql> insert into london values
        (1900, 'James Bond', 'New York', null);


sql>  insert into london values
        (1450, 'Jack Patel', 'London', 654.34);
```

To see all the london records through view

```
sql> select * from london;
```

To drop a view london

```
sql> drop view london
```

Reference Notes of Oracle 12c SQL Part 4

Oracle Trainer :- Sekhar

Set Operators

****

Union

*****

union is a set operator which is used for combining 2 queries.

The SQL UNION operator is used to combine the result sets of 2 or more SELECT statements. It removes duplicate rows between the various SELECT statements.

```
SQL> select snum from salespeople
  2  union
  3  select snum from customers;


 SNUM
      444
     1001
     1008
     1013
     1040


  SNUM
     1577
     1666
     1777
     1899
     1982
     3453
```

==

Union all

****

This operator is used for combining both the queries and both queries will get executed.

rules while using any set operator is it should have a common attribute name and

data type and data type size

in the following query both the table output would come

```
SQL> select snum from salespeople
  2  union all
  3  select snum from customers;
```

Minus operator

****

Minus operator is used for removing the common values from both the tables.

Query

write a query where u will print all salespeople who have still not been able to bring a single customer

```
SQL> select snum from salespeople
  2  minus
  3  select snum from customers;
```

===

Intersect operator

****

This operator which is a set operator will get printed if their is common records in both the tables.

Query

******

Write a query where u will print all salespeople who have booked at least 1 customer

```
SQL> select snum from salespeople
  2  intersect
  3  select snum from customers;

 SNUM
     1001
     1040
     1456
```

Joins

****

Joins is a facility in oracle sql to combine 2 or more tables in to a single query as per the logical requirement of the project.

You can also use 4 to 7 tables also in the same query if required as per the business requirements of the project while writing joins queries.

The are many types of joins.

1) Equi Join : In equi join you need a common values in 2 or more tables.

Then those common values may be printed or not as per the logical requirement of the query.

query

****

write a query where you will print the salesman no, name and who are his customer along with their cname and no

```
SQL> select salespeople.snum, sname, customers.cnum, cname
  2  from salespeople, customers
  3  where salespeople.snum = customers.snum;
```

```
     SNUM SNAME                CNUM CNAME
---------- ------------------------ ---------- --------------------
      1001 Kalia                      2019       Haynes
      1456 Ranjit singh               2007       Grass
      1040 Rana Pratap                2044       Diana
==========================================================
```

query

```
*******
```

Write a query where you will print the snum, sname, and his cnum, and cname and also print the salesman no from sales table. (Hint use allias table names)

```
SQL> select s.snum, sname, cnum, cname, c.snum
  2   from salespeople s, customers c
  3    where s.snum = c.snum;

     SNUM SNAME                     CNUM CNAME                SNUM
---------- ------------------------ ---------- -------------------- ----------
      1456 Ranjit singh               2007 Grass                1456
      1040 Rana Pratap                2044 Diana                1040
      1013 Dr. Batli Wala             2891 Janaki R             1013
      9001 James Singh                2828 Suganya Gowda        9001
====================================================================
```

Query

```
******
```

Write a query where you will print snum, name and cnum and cname and print only

those salesperson where customer and salespersons reside in the same city.

```
SQL> select s.snum, sname, s.city, c.cnum, c.cname, c.city
    from salespeople s, customers c
    where s.snum = c.snum
    and
     rtrim(s.city)=rtrim(c.city);
```

==

<u>Inner Joins</u>

****

Inner joins are also known as equi joins.

The Inner Join keyword selects all rows from both tables as long as there is a match

between the columns. If there are rows in the "Customers" table that do not have matches in "Orders", these customers will NOT be listed or displayed.

query

****

Write a query where using inner join print cnum from order table , cname from its master table, onum, amount and print only those customers who placed a orders.

```
sql>select o.cnum, c.cname, onum, oamount
  2  from customers c
  3    inner join orders o
  4     on c.cnum = o.cnum;
```

---

query

****

write a query where using inner join print snum, sname, and which customer they are giving service along with customer name and their customer number and print only those salespeople who are servicing any customers.

```
SQL> select s.snum, s.sname, c.cnum, c.cname
  2  from salespeople s
  3   inner join customers c
  4   on s.snum = c.snum;

    SNUM SNAME                      CNUM CNAME
---------- ------------------------ ---------- --------------------
    1456 Ranjit singh             2007 Grass
    1456 Ranjit singh             2001 Kalia
    1040 Rana Pratap                2044 Diana
    1013 Dr. Batli Wala             2891 Janaki R
    9001 James Singh                2828 Suganya Gowda
=================================================================
```

<u>LEFT JOIN</u>

****

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table

The result is NULL in the right side when there is no match.

query

****

write a query using left join print all cname, their cnum from orders table if they have placed orders and also the onum and oamount.

```
SQL> select c.cname, o.cnum, o.onum, o.oamount
  2  from customers c
  3  left join orders o
  4   on c.cnum = o.cnum
  5   order by c.cname desc;

CNAME                      CNUM       ONUM    OAMOUNT
-------------------- ---------- ---------- ----------
Suganya Gowda
Lucy Singh                 2014       3067    6543.34
Lucy Singh                 2014       3029    9494.33
Kalia
Janaki R
Haynes
Grass                      2007       3002      87366
Grass                      2007       3004  234564.45
Grass                      2007       3024   44425.44
Diana                      2044       3007     425425
Diana                      2044       3078   87345.33

11 rows selected.
```

The above query will also print customers who have not placed orders by giving null value their.

===

<u>RIGHT JOIN</u>

****

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1).

The result is NULL in the left side table when there is no match.

query

****

write a query where u will print all customers name and the orders number they have placed use right join.

```
SQL> select customers.cname, orders.onum
  2  from customers
  3    right join orders
  4     on customers.cnum = orders.cnum;

CNAME                ONUM
-------------------- ----------
Grass                3002
Grass                3004
Lucy Singh           3029
Lucy Singh           3067
Diana                3078
Diana                3007
```

==

<u>FULL OUTER JOIN</u>

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

Query

****

Write a query where you will print the cname, cnum from customer table and onum, cnum and order amount from order tables

using full outer joins

```
SQL> select c.cname, c.cnum, o.cnum, o.onum, oamount
  2  from customers c
  3   full outer join orders
  4    o on c.cnum = o.cnum;

CNAME                     CNUM       CNUM       ONUM    OAMOUNT
-------------------- ---------- ---------- ---------- ----------
Grass                 2007       2007       3004  234564.45
Diana                 2044       2044       3007     425425
Grass                 2007       2007       3002      87366
Lucy Singh              2014       2014       3067    6543.34
Lucy Singh              2014       2014       3029    9494.33
Grass                 2007       2007       3024   44425.44
Diana                 2044       2044       3078   87345.33
Janaki R                2891
Haynes                  2019
Kalia                 2001
Suganya Gowda              2828
```

<u>Sub Query</u>

****

A Subquery or Inner query or Nested query is a query within another SQL query, and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that subqueries must follow:

--

a)Subqueries must be enclosed within parentheses.

b)A subquery can have only one column in the SELECT clause, unless multiple

columns are in the main query for the subquery to compare its selected columns.

c)An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY.

d)Subqueries that return more than one row can only be used with multiple value operators,

such as the IN operator.

e) The BETWEEN operator cannot be used with a subquery; however, the BETWEEN can be used within the subquery.

<u>Query</u>

****

Write a sub query where you will print all order details for customer name called grass.

```
SQL> select * from orders
  2   where cnum in
  3     (select cnum from customers
  4       where cname = 'Grass');

ONUM ODATE       OAMOUNT      CNUM      SNUM
---------- --------- ---------- ---------- --------------------------------
     3004 09-JAN-15  234564.45       2007       1456
     3002 16-FEB-16      87366       2007       1456
     3024 04-JAN-16   44425.44       2007       1456
```

----

query

*****

Write a sub query where you will print all customers details of salesman name is Ranjit singh.

```
SQL> select * from customers
  2   where snum in
  3    (select snum from salespeople
  4       where rtrim(sname) = 'Ranjit singh');

     CNUM CNAME              CITY                  SNUM
---------- ------------------- ------------------- ----------
     2001 Kalia              Patna                 1456
     2007 Grass              New York              1456
```

Query

******

Write a sub query where you will print all orders details of customers who reside in Los Angeles.

```
SQL> select * from orders
  2   where cnum in
  3     (select cnum from customers
  4       where city = 'Los Angeles');


ONUM ODATE       OAMOUNT      CNUM      SNUM
---------- --------- ---------- ---------- ----------
     3007 16-FEB-16     425425       2044       1040
     3078 05-MAY-16   87345.33       2044       1040
```

Query

******

Write a sub query where you will print all the customer number, name and city who have not placed any orders.

```
SQL> select cnum, cname, city
  2    from customers
  3     where cnum not in
  4         (select cnum from orders);
```

```
      CNUM CNAME                CITY
---------- -------------------- --------------------
      2891 Janaki R             Mumbai
      2019 Haynes               Cariro
      2001 Kalia                Patna
      2828 Suganya Gowda        Mumbai
```

Important Notes Of Oracle 12c SQL Part 5
****
*****

Oracle Trainer :- Sekhar

****

Sub Query
****
A Subquery or Inner query or Nested query is a query within another SQL query, and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that subqueries must follow:

----

a)Subqueries must be enclosed within parentheses.

b)A subquery can have only one column in the SELECT clause, unless multiple
  columns are in the main query for the subquery to compare its selected columns.

c)An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY.

d)The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.

e)Subqueries that return more than one row can only be used with multiple value operators,
  such as the IN operator.

f) A subquery cannot be immediately enclosed in a set function.

g) The BETWEEN operator cannot be used with a subquery; however, the BETWEEN can be used within the subquery.

Query
****
Write a sub query where you will print all order details for customer name called grass.

```
SQL> select * from orders
  2    where cnum in
  3      (select cnum from customers
  4        where cname = 'Grass');

ONUM ODATE          OAMOUNT        CNUM
  SNUM
---------- --------- ----------
----------
----------------------------------
      3004 09-JAN-15  234564.45
2007       1456
      3002 16-FEB-16      87366
2007       1456
      3024 04-JAN-16   44425.44
2007       1456

----------------------------------------
----------------------------------------
----------
SQL> select * from customers;

      CNUM CNAME                CITY
                 SNUM
---------- --------------------
-------------------- ----------
      2001 Kalia                Patna
                 1456
      2007 Grass                New York
                 1456
      2019 Haynes               Cariro
                 1001
      2014 Lucy Singh           Jaipur
                 1001
      2044 Diana                Los
Angeles                1040
      2891 Janaki R             Mumbai
```

```
                  1013
      2828 Suganya Gowda        Mumbai
                  9001

7 rows selected.
```


----

query
*****
Write a sub query where you will print all customers details of salesman name is Ranjit singh.

```
SQL> select * from customers
  2    where snum in
  3     (select snum from salespeople
  4        where rtrim(sname) = 'Ranjit
singh');

      CNUM CNAME                CITY
                  SNUM
---------- -------------------
---------------------- ----------
      2001 Kalia                Patna
                  1456
      2007 Grass                New York
                  1456
```


Query
******

Write a sub query where you will print all orders details of customers who reside in
Los Angeles.

```
SQL> select * from orders
  2    where cnum in
  3       (select cnum from customers
  4         where city = 'Los Angeles');

      ONUM ODATE        OAMOUNT
CNUM       SNUM
---------- --------- ----------
---------- ----------
      3007 16-FEB-16     425425
2044       1040
      3078 05-MAY-16   87345.33
2044       1040
---------------------------------------------
---------------------------------------------
-------------------------
```

Query
******
Write a sub query where you will print all the customer number, name and city who have not placed any orders.

```
SQL> select cnum, cname, city
  2     from customers
  3      where cnum not in
  4           (select cnum from orders);

      CNUM CNAME                     CITY
---------- --------------------
--------------------
```

```
      2891 Janaki R                 Mumbai
      2019 Haynes                   Cariro
      2001 Kalia                    Patna
      2828 Suganya Gowda            Mumbai
----------------------------------------
----------------------------------------
-------------------------
```

Query
******
Write a sub query where you will print salesman no, name and city
who are servicing atleast 1 or more customers.

```
SQL> select snum, sname, city
  2     from salespeople
  3      where snum in
  4            (select snum from
customers);

      SNUM SNAME
CITY
---------- -------------------------
--------------------
      1456 Ranjit singh
Jaipur
      1001 Kalia
Mumbai
      1040 Rana Pratap                  Los
Angeles
      1013 Dr. Batli Wala
Mumbai
      9001 James Singh
Mumbai
```

Query
****
Write a sub query where you will print all salespeople who are not giving service even to a single customer.

```
SQL> select * from salespeople
  2    where snum not in
  3      (select snum from customers);
```

Correlated Subquery
****
If there is any correlation between main query and subquery then subquery is called as correlated subquery.

A correlated subquery is a subquery that receives some input from main query and sends
result back to main query. Unlike normal subquery, a correlated subquery receives value from
main query. It uses the value (generally in condition) and sends the results of the query back
to main query.

write a co-related sub query where you will print the
third highest commission from salespeople.

```
SQL> select sname, comm from salespeople
s1
  2     where 2 =
  3       (select count(*) from
salespeople
  4          where comm >s1.comm);

SNAME                          COMM
------------------------- ----------
ram                            4949
```

PL/SQL - Notes Part 1 for students
References
------------------------------=---------------------------

The PL/SQL programming language was developed by Oracle Corporation as procedural extension language for SQL and the Oracle relational database.

Following are notable facts about PL/SQL:
****

PL/SQL is a completely portable, high-performance transaction-processing language.

PL/SQL provides a built-in interpreted and OS independent programming environment.

PL/SQL can also directly be called from the command-line SQL*Plus interface.

Direct call can also be made from external programming language calls to database.

Features of PL/SQL
****

PL/SQL is tightly integrated with SQL.
It offers extensive error checking.
It offers numerous data types.
It offers a variety of programming structures.
It supports structured programming through functions and procedures.
It supports developing web applications and server pages.

Advantages of PL/SQL
****

SQL is the standard database language and PL/SQL is strongly integrated with SQL.

PL/SQL supports both static and dynamic SQL.

Static SQL supports DML operations and

transaction control from PL/SQL block.

Dynamic SQL is SQL allows embedding DDL statements in PL/SQL blocks.

PL/SQL allows sending an entire block of statements to the database at one time.

PL/SQL give high productivity to programmers as it can query, transform, and update data in a database.

PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.

Applications written in PL/SQL are fully portable.

PL/SQL provides high security level.

PL/SQL provides access to predefined SQL packages.
PL/SQL provides support for

Object-Oriented Programming.
PL/SQL provides support for Developing Web Applications and Server Pages
----

PL/SQL is not a stand-alone programming language;
it is a tool within the Oracle programming environment.

SQL* Plus is an interactive tool that allows you to type SQL and PL/SQL statements at the command prompt.

These commands are then sent to the database for processing.
Once the statements are processed, the results are sent back and displayed on screen.
=====================================
==========================
=====================================
==========================

1st Program in PL/SQL Block -- First.sql

```
************************************************
/*
==========================================
1) Declare Section
It is a optional section.
we declare variables, cursors, procedures,
functions and

2)Execution Section
THis section is the second section and the
compulsary section. it has logic of the
program.

Begin
.......
............
End;
3) exceptional handling.
*/

set serveroutput on;
DECLARE

 messg  varchar2(40) := 'Good Morning
India.';
```

```
 name2 varchar2(89) := 'I will get
Peformance bonus of
              $333535 every year.';

BEGIN

   dbms_output.put_line(messg);
   dbms_output.put_line(name2);
END;
/
==================================
=========================
==================================
=========================
```

Program using if conditions.
****

```
/* write a pl/sql block to intialize the rating
and if
 it is less than 215 flash a message YOU
ARE selected in the process.
*/

DECLARE
   a number(4) := 33;
```

```
BEGIN
  -- check the boolean condition using if
statement
IF( a < 215 )
   THEN
      dbms_output.put_line('you are selected
in the process ' );
END IF;

   dbms_output.put_line('value of a is : ' ||
a);
END;
/
```

=======================================
==========================
=======================================
==========================

PL/SQL Another program on accepting values using if conditons.
****
****

/* write a pl/sql block where you will intalize a salesman no and if found and current

```
insenitve less than 25 k increase the salary
by 1000 rupees
*/

set serveroutput on;
DECLARE
   tempno salespeople.snum%type := 1019;
   tsal  salespeople.comm%type;

BEGIN
   SELECT  comm INTO  tsal
   FROM salespeople
   WHERE snum = tempno;

   IF (tsal <= 25000)
     THEN

        UPDATE salespeople
         SET comm =  comm + 1000
           WHERE snum = tempno;
         commit;
        dbms_output.put_line ('Salary
updated');
   END IF;
```

```
END;
/
====================================
==========================
====================================
==========================
```

Program using while loops
****

```
DECLARE
   a number(2) := 1;

BEGIN

   WHILE a < 20
      LOOP
          dbms_output.put_line('value of a: ' ||
a);
          a := a + 1;
   END LOOP;

END;
/
====================================
```

=========================
===================================
=========================

A program using for loops
****

```
DECLARE
   a number(2);

BEGIN

   FOR a in 10 .. 20
      LOOP
         dbms_output.put_line('value of a: ' ||
a);
   END LOOP;

END;
/
```

===================================
=========================
===================================
=========================

A program using %type
****

```
set serveroutput on ;
DECLARE

   tempno employee.empno%type ;
   tempname  employee.sname%type;
   tdoj employee.doj%type;
   tdesig employee.desig%type;
   tbasic  employee.basic%type;

 BEGIN
   tempno := &snum;

   SELECT empno, sname, doj, desig,
basic into
   tempno, tempname,tdoj, tdesig, tbasic
   FROM employee
   WHERE empno = tempno;

dbms_output.put_line('employee ' ||
tempname || ' Posted as  ' || tdesig || ' earns
' || tbasic);
END;
/
```

====================================
=========================
====================================
==========================

Program using % Rowtypes
****

```
DECLARE
customer_rec customers%rowtype;

BEGIN
   SELECT * into customer_rec
   FROM customers
   WHERE cnum = 2001;

   dbms_output.put_line('Customer Cnum: '
|| customer_rec.cnum);
   dbms_output.put_line('Customer Name: '
|| customer_rec.cname);
   dbms_output.put_line('Customer City: ' ||
customer_rec.city);
   dbms_output.put_line('Customer
Salesman: ' || customer_rec.snum);
```

```
END;
/
=================================
=======================
=================================
=======================
```

PL/SQL - Notes Part 2 for students References

Cursors
****

Oracle creates a memory area, known as context area, for processing an SQL statement,
which contains all information needed for processing the statement,
for example, number of rows processed etc.

Implicit Cursors
****

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement.

Programmers cannot control the implicit cursors and the information in it.

Implicit cursors will process 1 records in a table.

Implicit Cursor attributes.
****

%FOUND will aways return true if insert, update or delete is sucusseful.
    else it returns false.

%NOTFOUND it will true statement if insert , update or delete is not
    sucessful else it return false.

%ISOPEN
%ROWCOUNT

access the attributes with the following syntax
sql%attribute_name
for example sql%rowcount

==
======================
==
======================

Implicit cursor programs.
****

```
/*
a pl/sql block  using  implicit cursor where
commission is decreased by Rs. 200 for
all SALESPEOPLE  AND BLOCK WILL display
how many person insentive decreased
if the query is sucessfull.
*/
Set serveroutput on;
```

```
DECLARE
 total_rows number(4);

BEGIN

   UPDATE salespeople SET comm = comm -
200 ;

   IF  sql%found
      THEN
          total_rows := sql%rowcount;
          dbms_output.put_line( total_rows || '
Salespeople Insentive  Decreased');

          commit;

   END IF;
END;
/
```

==
=====================
==
=====================

Explicit cursor
****

****
Explicit cursors are programmer defined cursors for gaining more control over the context area.
An explicit cursor should be defined in the declaration section of the PL/SQL Block.
It is created on a SELECT Statement which returns more than one row.

Explicit cursor has to be declared in the declare section, cursor has to be opend and records fetched from cursor and then cursor has to be closed in the end.

attributes.
****
%found
%notfound
%isopen
%rowcount

=====================================
============

/*
write a pl/sql block using explicit cursors to declare a explict cursor fetch all the customer table tupples and print the report using selected attrributes.

```
*/

set serveroutput on;
DECLARE
   c_id customers.cnum%type;
   c_name customers.cname%type;
   c_addr customers.city%type;

   CURSOR c_customers is
      SELECT cnum, cname, city FROM
customers;

BEGIN

   OPEN c_customers;

  LOOP
      FETCH c_customers into c_id, c_name,
c_addr;
          EXIT WHEN c_customers%notfound;

      dbms_output.put_line(c_id || ' ' || c_name || ' '
|| c_addr);

 END LOOP;

  CLOSE c_customers;
```

```
END;
/
```

==
=====================
==
=====================

explicit cursor using for loop
****

write a pl/sql block to print all customers staying in Bengaluru
using for loop only.

```
set serveroutput on;

DECLARE
  CURSOR cus IS SELECT  cnum, cname, city
FROM customers
                          where city = 'London' or city
= 'Bengaluru'
                          order by cname desc;

BEGIN
       FOR r in cus
             LOOP
                DBMS_OUTPUT.PUT_LINE('cnum
```

is ' || r.cnum);

DBMS_OUTPUT.PUT_LINE('Customer name is ' || r.cname);
                DBMS_OUTPUT.PUT_LINE(' City is ' || r.city);
          END LOOP;
END;
/

===
===

Exceptions
****
An error condition during a program execution is called an exception in PL/SQL.

PL/SQL supports programmers to catch such conditions using EXCEPTION block in
the program and an appropriate action is taken against the error condition.

/*
Exception examples.
****

```
write a pl/sql block to fetch the details of
customer number 88
use the in built exceptions to print "CUsotmer not
found if the cutomer no does not exist."
*/

set serveroutput on;

DECLARE
   c_id customers.cnum%type := 88;
   c_name  customers.cname%type;
   c_addr customers.city%type;
BEGIN
   SELECT  cname, city INTO  c_name, c_addr
   FROM customers
   WHERE cnum = c_id;

   DBMS_OUTPUT.PUT_LINE ('Name: '||
c_name);
   DBMS_OUTPUT.PUT_LINE ('Address: ' ||
c_addr);

EXCEPTION
   WHEN no_data_found THEN
      dbms_output.put_line('This customers no
does not exist in  table!');

   WHEN others THEN
```

```
      dbms_output.put_line('some other Error!');
END;
/
```

==
==

PL/SQL – Procedures
****

A subprogram is a program unit/module that performs a particular task.

These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program, which is called the calling program.

examples of creating a stand alone Procedure
****

```
sql>
CREATE OR REPLACE PROCEDURE greetings
AS
```

```
BEGIN
   dbms_output.put_line('Welcome to the world of
PL/SQL Programming');
END;
/
```

to execute the above procedure

```
sql>execute greetings;
```

===
===

write a pl/sql block where you will declare a procedure within a block called findMin which will

receive 2 variables values and return the lowest of 2 numbers.
****

```
DECLARE
   a number;
   b number;
   c number;
```

```
PROCEDURE findMin(x IN number, y IN number,
z OUT number) IS
BEGIN

   IF x < y
      THEN
         z :=  x;
 else
      z :=  y;
   END IF;

END;

BEGIN
   a:= &a;
   b:= &b;

   findMin(a, b, c);
   dbms_output.put_line(' Minimum of  both
numbers is ' || c);
END;
/
```

```
===================================
====================
===================================
====================
```

to see the user defined procedures
****

To list all stored procedures in the database you're connected to

```
sql> select object_name from user_procedures;
```

===

To list stand alone procedures in the database you're connected to

```
SQL> select object_name from user_procedures
          where object_name = 'GREETINGS';
```

===

Functions
****

Creating a Function
A standalone function is created using the CREATE FUNCTION statement.

Following stand alone function will print the number of records from a customer table.
****

****

```
CREATE OR REPLACE FUNCTION
totalCustomers
RETURN number IS
   total number(4) := 0;
BEGIN
   SELECT count(*) into total
   FROM customers;

   RETURN total;
END;
/
```

==================================
==================

The above function can be called from the following pl/sql block.

****
****

```
set serveroutput on;

DECLARE
   c number(4);

BEGIN
```

```
   c := totalCustomers();

   dbms_output.put_line('Total no  of Customers
is : ' || c);
END;
/
```

==
==================

==
==================

to list all the functions in PL/SQL
****

```
sql> select object_name  from user_objects
where object_type = 'FUNCTION';
```

==
=================

Advance PL/SQL - Notes for students References

Packages
****
Packages are schema objects that groups logically related PL/SQL types, variables, and subprograms.

A package will have two mandatory parts −
Package specification
Package body or definition

Advantage of Packages in PL/SQL
****
Packages let you encapsulate logically related types, items, and subprograms in a named PL/SQL module.

Each package is easy to understand, and the interfaces between packages are simple, clear, and well defined.

Encapsulation. Packages enable you to encapsulate or group stored procedures, variables, data types, and so on in a named, stored unit. ...
Data security. The methods of package definition enable you to specify which variables, cursors, and procedures are public and private. ...
Better performance.
==
Package Specification
****
The specification is the interface to the package.

It just DECLARES the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms.

All objects placed in the specification are called public objects.
Any subprogram not in the package specification but coded in the package body is called a private object.
===

Package Body
****
The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from the code outside the package.

The CREATE PACKAGE BODY Statement is used for creating the package body

==
=============
==
=============

Example of using packages
****

Question
****
Create a package called sales_salary which will have a user defined procedure called find_sal.

Packages Specification created
****

```
CREATE PACKAGE sales_salary AS
   PROCEDURE find_sal(s_no salespeople.snum%type);
END sales_salary;
/
```

Question
****
Create a package body called sales_salary in which a procedure called find_sal will fetch the details of saleman which will be called from a different pl/sql block.

Packages body or defenition (ppbody.sql)
****

```
CREATE OR REPLACE PACKAGE BODY sales_salary AS
   PROCEDURE find_sal(s_no salespeople.snum%type) IS
    s_salary salespeople.comm%TYPE;
    BEGIN
       SELECT comm into s_salary
       FROM salespeople
       WHERE snum = s_no;
       dbms_output.put_line('Insentive is : '|| s_salary);
   END find_sal;
END sales_salary;
/
```

=========================

example of using package elements to execute the code from a different pl/sql block to accept sales man number and call the procedure declared in the above block.

****
```
/*
following code from sql *plus or  execute it from following
edit f:\packsales.sql
*/
DECLARE
   code salespeople.snum%type := &snum;
BEGIN
 sales_salary.find_sal(code);
END;
```

==
==================================
==
==================================
Dropping the pacakge
****

```
sql> DROP PACKAGE BODY sales_salary ;
```

to drop specificatons and body
```
sql>DROP PACKAGE sales_salary ;
```
==
==================================
==
==================================

To Recompile a pacakge the query is
****

```
sql>  alter  PACKAGE sales_salary compile;
```

To Recompile a pacakge body the query is

****
sql> alter PACKAGE sales_salary compile body;
===
===

Another examples of packages
****
package specification
****
/* create a package which will have procedure and a functions
*/

```
CREATE PACKAGE packdemo1
  AS
            PROCEDURE spDemo5;
            FUNCTION fnDemo3 RETURN NUMBER;
  END packdemo1;
  /
```

=======================================

Package body
****
package specification
****
/* create a package body which will have procedure code and a functions code & functions will
return a value
*/

```
  CREATE OR REPLACE PACKAGE BODY packdemo1
  AS
```

```
            PROCEDURE spDemo5 Is
            BEGIN

DBMS_OUTPUT.PUT_LINe('Procedure in package');
            END spDemo5;

            FUNCTION fnDemo3 RETURN NUMBER IS
            BEGIN
                        DBMS_OUTPUT.PUT_LINe('Function
in package');
                        RETURN 1;
            END fnDemo3;
 END packdemo1;
 /
```
==

following queries will call the procedure & then the function
****

```
SQL> exec packdemo1.spDemo5;
Procedure in package

PL/SQL procedure successfully completed.

SQL> SELECT packdemo1.fnDemo3 FROM dual;
```
==
==

Collections
****

A collection is an ordered group of elements having the

same data type.
Each element is identified by a unique subscript that represents its position in the collection.

PL/SQL provides three collection types −
• Index-by tables or Associative array
• Nested table
• Variable-size array or Varray

Varray

PL/SQL programming language provides a data structure called the VARRAY, which can store a
fixed-size sequential collection of elements of the same type.
 A varray is used to store an ordered
collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Important points
****
You can initialize the varray elements using the constructor method of the varray type, which has the same name as the varray.
Varrays is a one-dimensional arrays.
A varray is automatically NULL when it is declared and must be initialized before its elements can be referenced.

==

Creating a Varray Type

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

example of varray.
****

```
DECLARE
/*
examples of  varray in pl/sql which is part of collections.
*/

   type namesarray IS VARRAY(5) OF VARCHAR2(10);
   type grades IS VARRAY(5) OF INTEGER;

   names namesarray;
   marks grades;
   total integer;

BEGIN
   names := namesarray('Grass', 'John', 'Suresh K', 'Rita', 
'Lucy');
   marks:= grades(9, 45, 54, 73, 99);

   total := names.count;

   dbms_output.put_line('Total '|| total || ' Students');

   FOR i in 1 .. total LOOP

         dbms_output.put_line('Student : ' || names(i) || '
         Marks: ' || marks(i));
   END LOOP;
END;
/
```

==
=====================================
==
=====================================

Index-By Table
****
An index-by table (also called an associative array) is a set of key-value pairs.
Each key is unique and is used to locate the corresponding value.

The key can be either an integer or a string.

An index-by table is created using the following syntax.
Here, we are creating an index-by table named table_name, the keys of which will be of the subscript_type and associated values will be of the element_type

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

Index by table example where you will declaer a table called salary and store employees names and salary
and print the table details using a loop.
****
****

```
DECLARE

   TYPE salary IS TABLE OF NUMBER INDEX BY
```

```
VARCHAR2(20);
   salary_list salary;
   name   VARCHAR2(20);

BEGIN
   -- adding elements to the table
   salary_list('Hillary') := 62000;
   salary_list('Mike') := 75000;
   salary_list('Lucy') := 100000;
   salary_list('James') := 78000;

   -- printing the table
   name := salary_list.FIRST;

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

==
==================================
==
==================================

Nested Tables
****

A nested table is like a one-dimensional array.
However, a nested table differs from an array in the following aspects −

****
An array has a declared number of elements, but a nested table does not.
The size of a nested table can increase dynamically.

An array is always dense, i.e., it always has consecutive subscripts.
A nested array is dense initially, but it can become sparse when elements are deleted from it.

A nested table can be stored in a database column.
An associative array cannot be stored in the database.
==

An example of Nested table block
****
```
/*
this block will has 2 tables
   names_table and grades which we will use for storing 
college students name and marks.
*/

DECLARE

   TYPE names_table IS TABLE OF VARCHAR2(10);
   TYPE grades IS TABLE OF INTEGER;

   names names_table;
   marks grades;
   total integer;
```

```
BEGIN
   names := names_table('Sunita', 'Jackson', 'Jalps',
'Ranveer', 'Premji');
   marks:= grades(98, 97, 78, 87, 92);

   total := names.count;
   dbms_output.put_line('Total '|| total || ' Students');

   FOR i IN 1 .. total LOOP
      dbms_output.put_line('Student:'||names(i)||', Marks:' ||
marks(i));
   end loop;
END;
/
```

==
=============
==
=============

PRAGMA:
****

A pragma is generally a line of source code prescribing an action you want the compiler to take. It’s like an option that you give the compiler.
it can result in different run time behavior for the program, but it doesn’t get translated directly into byte-code.

Types of pragma
****

PRAGMA AUTONOMOUS_TRANSACTION
****

Once started, an autonomous transaction is fully independent. It shares no locks, resources, or commit-dependencies with the main transaction.
You can log events, increment retry counters, and so on, even if the main transaction rolls back.
Unlike regular triggers, autonomous triggers can contain transaction control statements such as COMMIT and ROLLBACK.
==
========

example of PRAGMA AUTONOMOUS_TRANSACTION
****

```
/*
First create a procedure called logerror22
*******************************************************
*/
CREATE OR REPLACE PROCEDURE logerror22
AS
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   INSERT INTO errorlog
VALUES(errorid.nextval,'dff',sysdate);
   COMMIT;
END;
/
```

==
=
Then excute the following anonymous block.
****

```
BEGIN
  INSERT INTO salespeople VALUES(4564, null, null, null);
```

```
   INSERT INTO  salespeople VALUES(NULL,'ee','rr', null);
EXCEPTION
   WHEN OTHERS THEN
          logerror22;
                 rollback;
END;
/
```

==
=
==
=

PRAGMA EXCEPTION_INIT
****

The EXCEPTION_INIT pragma associates a user-defined exception name with an Oracle Database error number. You can intercept any Oracle Database error number and write an exception handler for it,
instead of using the OTHERS handler.

The EXCEPTION_INIT pragma associates a user-defined exception name with an Oracle Database error number. You can intercept any Oracle Database error number and write an exception handler for it, instead of using the OTHERS handler.

==
============
==
============

An example of PL/SQL Block to handle the validation of NULL Value in Primary Key
for employee table using exception _init.

```
*************************************************************************
*********************************
/*
To handle error conditions (typically ORA- messages) that
have no predefined name,
you must use the OTHERS handler or the pragma
EXCEPTION_INIT.

A pragma is a compiler directive that is processed at
compile time, not at run time.

Steps to be followed to use unnamed system exceptions
1.They are raised implicitly
2.If they are not handled in WHEN OTHERS they must be
handled explicitly
3.To handle the exception explicitly they must be declared
using PRAGMA EXCEPTION_INIT
*/

DECLARE

insert_exep EXCEPTION;
-- Handles Cannot Insert NULL into Primary Key Constraint
--

PRAGMA EXCEPTION_INIT(insert_exep,- 01400);

BEGIN
INSERT INTO empl(ename,sal)VALUES(' Jack Patel
',2340);

EXCEPTION
WHEN insert_exep THEN
DBMS_OUTPUT.PUT_LINE('Insert Operation Failed check
```

the employee table rules');

END;
/
===
===
Functions for Trapping Exceptions
****
When an exception occurs, you can retrieve the associated
    error code or error message by using two functions.

• Based on the values of the code or the message, you can decide which subsequent actions to take.
****
– SQLERRM returns character data containing the message
  associated with the error number.
– SQLCODE returns the numeric value for the error code. (You
   can assign it to a NUMBER variable.)

Functions for Trapping Exceptions
****
• You cannot use SQLCODE or SQLERRM directly in an SQL statement.
• Instead, you must assign their values to local variables,then use the variables in the
SQL statement

Another pl/sql block using exceptions
****
/*

When we really dont know what type of Exception gets raised during execution of PL/SQL
Block we use the WHEN OTHERS.

```
-- PL/SQL Block to handle multiple Exceptions inside each
Inner Blocks --
-- In the Main Block handle other Exceptions --
*/

DECLARE
l_name VARCHAR2(60);
BEGIN
		BEGIN
		SELECT ename INTO l_name FROM empl
WHERE empno = 8000;

		EXCEPTION
		WHEN NO_DATA_FOUND THEN
		DBMS_OUTPUT.PUT_LINE('Employee No. is
invalid.Please provide an existing Employee No.');

		END;

		BEGIN
		SELECT ename INTO l_name FROM emp
WHERE deptno = 30;

		EXCEPTION
		WHEN TOO_MANY_ROWS THEN
	DBMS_OUTPUT.PUT_LINE('Your SELECT statement
retrieved Multiple Rows.Consider using a Cursor');
		END;

INSERT INTO emp(ename)VALUES('Sarita K');
```

```
EXCEPTION

WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('SQL Error Message :
'||sqlerrm);
END;
/
```

==
==============================
==
==============================

Oracle Architecture Notes

- Introduction to ORACLE & its products

- Introduction to Oracle Architecture
  - Oracle Physical structure-Data Files, Control Files and Redo Log Files.
  - Oracle Logical Structure- Tablespaces, Segments, Extents and Blocks
  - Schema objects-Tables, Sequences, Synonyms, Views
  - Oracle Memory Structures and Background Processes, Data Dictionary

▪Through handouts (lecture 1 & 2):

- Revision of SQL
- Introduction to Advanced SQL & PL/SQL

What is Oracle ?

Oracle is a relational database management system.

- It is a management system which uses the relational data model.
- In the relational data model, data is seen by the users in form of tables alone.

Oracle Server:

- Is a database management system that provides an open, comprehensive, integrated approach to information management.
- Consists of an Oracle Instance and an Oracle database

Database Architecture - Introduction

Three Major Instances:

1. Database instance
2. File Structure
3. Data Structures

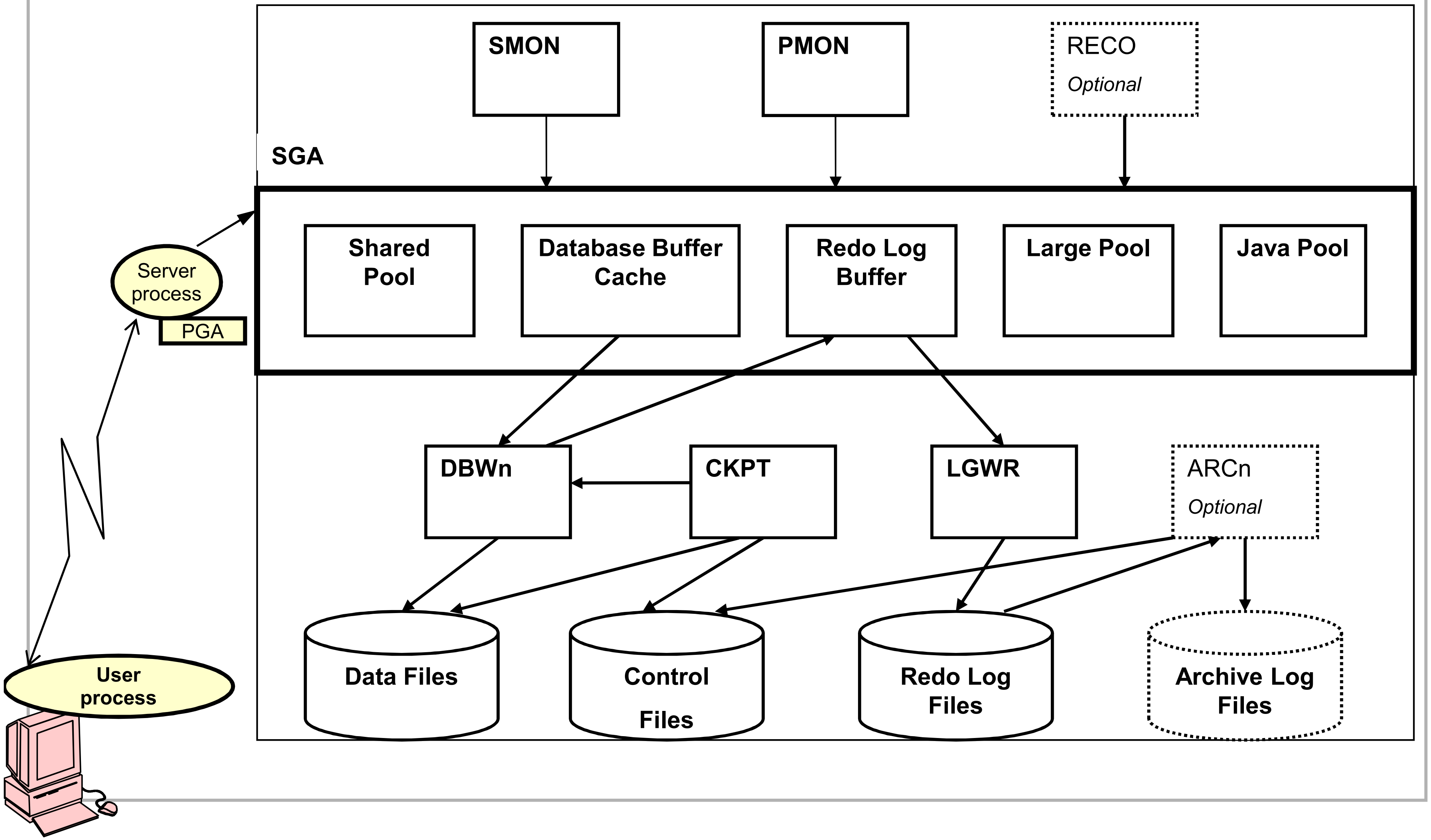
Database Instance:

- Oracle Database consists of Software Modules & Database Files
- Instance –After the complete installation of Oracle , when you start the Oracle database , then you have what is referred to as an*"Oracle Database Instance". *It is the actual execution of DBMS software that manages data in the databases tablespace.

Properties Of Database Instance

1. Created on loading the software from disk to memory.
2. It is an aggregation of processes and memory structures
3. It is sharable thus allowing multiple users to access the same database.

Oracle Instance



Memory Components and Background Processes

- Two Main Components:

1. SGA(System Global Area)

   -a group of shared memory structures that contain data and control information for one Oracle database instance.

   -the data in the instance's SGA is shared among the multiple concurrent users.

   -allocated when you start the database instance.

   -de-allocated when the instance is shutdown.

2. PGA (Program Global Area)

   -Each server process has a PGA allocated that is a private area for each server

   -Work area for each application.

SGA Memory Areas

- **Shared pool** contains machine-language code and execution plans for frequently used SQL commands.
- **Database Buffer Cache** stores data values which are written later to the data files by the database writer (DBWn).
- **Redo Log Buffer** stores a copy of the changed data from user transaction. This data is periodically written to the Redo Log Files by the Log Writer (LGWR).
- **Large Pool** is a work area given for backup and recovery operations.
- **Java Pool** stores the machine-language and execution plans for Java commands used in application programs and database operations.

PGA Memory Areas

- Each server process has a PGA allocated that is a <u>private area</u> for each server. This is the work area for each application. The application code, along with copies of the data, is located here.

- There are various background processes that support and monitor the server processes. These background processes also handle the data management and keep the database running smooth and efficiently.

Processes

- **System Monitor** (SMON) :

  -general server housekeeping functions.
- **Process Monitor** (PMON) :

  - monitors and manages individual user sessions .

  -performs database locking/unlocking functions on UPDATE and DELETE query.
- **Database Writer** (DBWn) :

  -writes changed data from the database buffer cache to data files.

- **Log Writer** (LGWR) :

  -writes the redo log data from the Redo Log Buffer to the Redo Log Files.

  -Redo Log files aid in database recovery.

  -keep track of the database changes whenever they are committed

Processes (contd.)

- **Checkpoint** (CKPT) :

  -responsible for signaling DBWn and LGWR to write the contents of the Database Buffer Cache and the Redo Log Cache to the data files and Redo Log files respectively.

- **Archiver** (ARCn) :

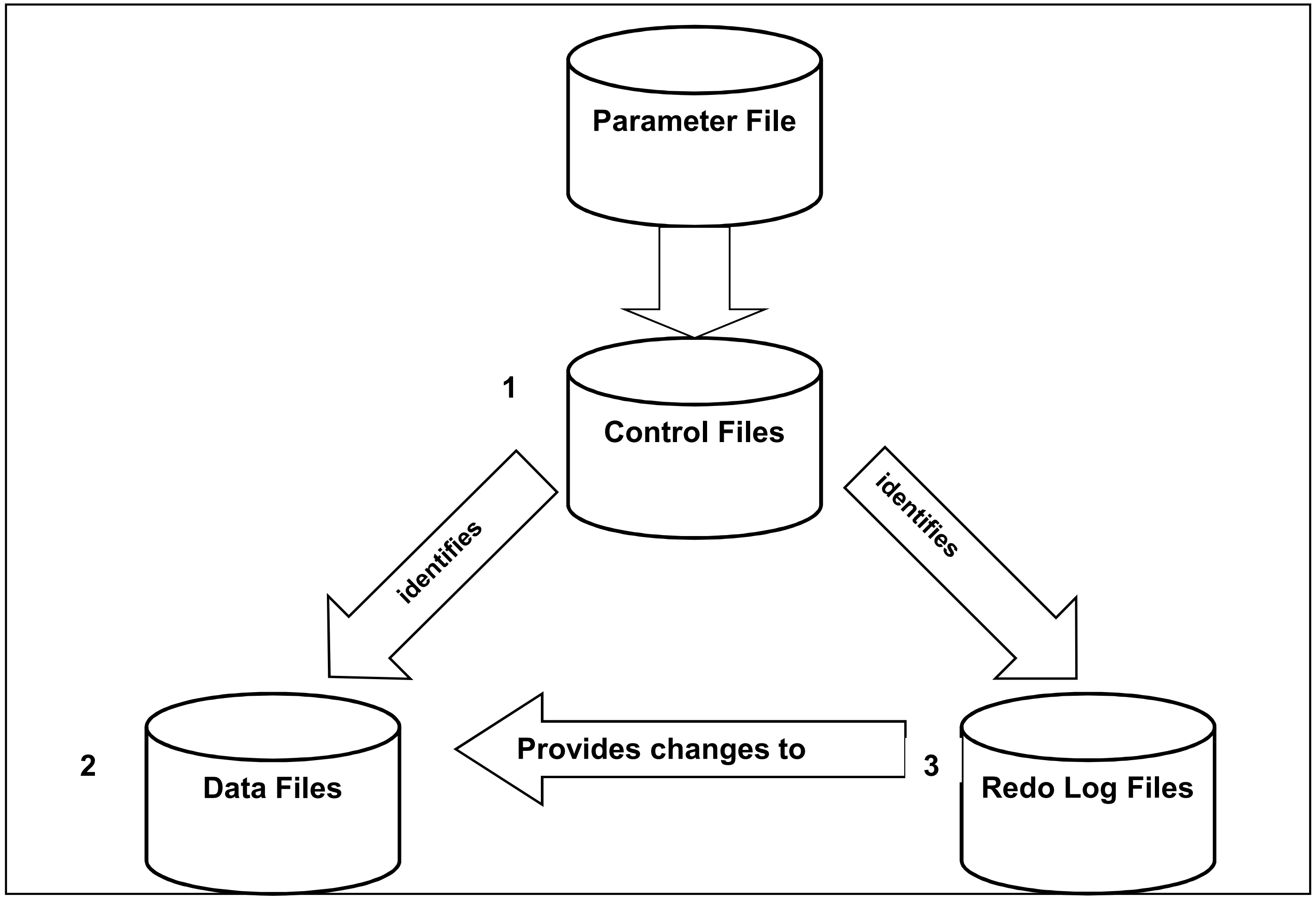
  -reads the Redo Log files after they are filled & copies it to a corresponding Archive Log File.

  -there can be up to 10 separate archive processes per instance Arc0-Arc9.

- **Recoverer** (RECO) :

  -detect and correct errors as a result of communications problems in a distributed database environment.

File Structure- Three Basic Oracle Files



Parameter File – the init.ora file

- Purpose:

  - specifies the configuration information about the database instance.

- The parameters include:

1. Names and locations of the control files
2. Block size
3. Cache sizes
4. Database name
5. Instance name
6. Domain name
7. Is read each time a database instance is started
8. Has a **.ora** suffix

Data Files

- Purpose:

  -contain the actual data stored in the database.

  -contains user data stored in tables + includes indexes, data dictionary, and rollback segments.

- Characteristics:

1. Data files are composed of Oracle blocks, which are in turn composed of operating system blocks
2. Oracle block sizes range from 2 Kb to 32 Kb – average size is 8 Kb
3. Data files belong to only one database and to only one tablespace within that database
4. Data files are the lowest level of granularity between an Oracle database and the operating system
5. When you map out a database onto the OS I/O sub-systems, the smallest unit you can put in any location is a data file
6. Have a **.dbf** suffix

Redo Log Files

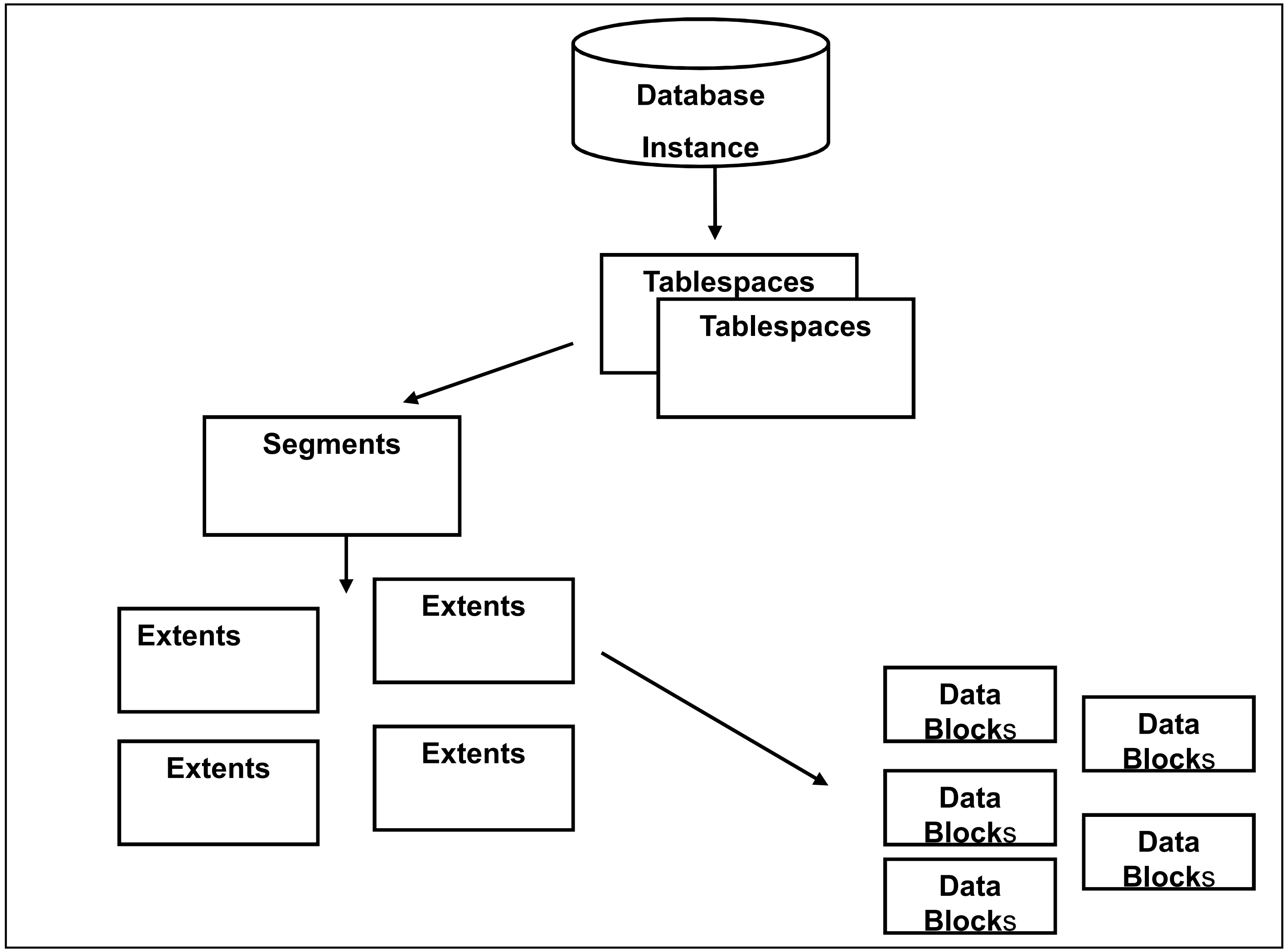
- **Purpose:**

  - store changes made to the database as a result of transaction and internal Oracle activities.

- **Characteristics:**

1. By default, an Oracle database contains three redo log groups, REDO01.log, REDO02.log and REDO03.log
2. Every Oracle database must have at least two redo log groups
3. The database will write log entries to a subsequent redo log group when the previous redo log group fills up
4. As a general rule , there should be one redo log group for approximately every four database users that create action queries
5. Oracle keeps track of the Redo Log file by using a redo log sequence number, this number is recorded inside the file as they are used
6. The redo log sequence number is different than the operating system file name that is used to identify the physical file
7. If the database is in ARCHIVELOG mode full Redo Log files are copied to Archive Log files before they are reused, otherwise they are written over
8. Have a **.log** suffix

Data Structures



TABLESPACE SEGMENT EXTENTS and DATA BLOCKS

- **Tablespace** is used to store related database objects. One tablespace is used to store all of the system tables; another tablespace may be created for all indexes or a tablespace may be created to store all of the tables for a specific application. The idea is to store data that has something in common or has similar characteristics. The database server stores the data in each tablespace in data files with **.dbf** extensions.

- **Segments** are used to organize tablespace data within a tablespace. A segment stores an individual database object like a table or index.

- **Extents** are contiguous units of storage, usually disk space, within a segment. Oracle uses extents for performance reasons by storing data that needs to be retrieved in a single disk I/O. An extent is made up of multiple data blocks

- **Data Blocks** are the smallest unit of Oracle database storage. Oracle stores 8,192 bytes (8K) in one data block. A data block is comprised of multiple operating system blocks. Depending on the operating system an operating system block can store 512 to 4K bytes. A data block contains header, directory and row data:

1. Block Header - operating system block address
2. Table Directory - identifies the database table for which the following data belongs
3. Row Directory - identifies the database rows for which the data belongs
4. Row Data - stores the actual row values