Oracle 11g Notes of
Murali (Naresh Technology)
Hyderabad

Written By

Saket

- Please Send your Suggestions at – saketdew@gmail.com
- Mail me latest notes of other subjects.
- Contact me for other subjects notes.

ORACLE 11G(grid)

Data: 27/2/2015

There are mainly two different types of database language is ORACLE. They are:

1) SQL (STRUCTURED QUERY LANGUAGE)
2) PL/SQL (PROCEDURAL LANGUAGE extension of SQL)
3) Dynamic SQL.(optional)

ORACLE is a Relational Database product which is used to store data permanently in secondary storage devices. If you want to operate ORACLE then we are using following languages:

1) SQL: It is non-procedural language.
2) PL/SQL: It is a procedural language.

All organizations store same type of data.

DATA: It is a collection of Raw facts.

Example: 101 dinesh 2000

102 mahesh 3000

In above example, there no meaningful data such data is known as Raw facts.

INFORMATION: It is a collection of meaningful data or processed data.

Example:

EmpID	Ename	Salary
101	dinesh	2000
102	naresh	3000

In the above example, there is meaningful data which is in table format which consist of three different fields.

DATA STORE: It is a place where we can store data or information.

1) Books & Papers
2) Flat files
3) Database

FLAT FILES: This is a traditional mechanism which is used to store data or information in individual unrelated files. These files are also called as Flat Files.

Drawbacks of Flat files:

1) Data Retrieval
2) Data Redundancy
3) Data Integrity
4) Data Security
5) Data Indexing

1) **Data Retrieval:** If we want to retrieve data from flat files then we must develop application program in high level languages, where as if we want to retrieve data from databases then we are using Sequel Language.

SEQUEL (Structured English Query Language)

Date: 28/2/2015

2) **Data Redundancy:** Sometimes we are maintaining multiple copies of the same data in different locations this data is also called as Duplicate data or Redundant data. In Flat files mechanism when we are modifying data in one location it is not effected in another location. This is called INCONSISTENCY.

Date: 2/3/2015

In databases, every transaction internally having 4 properties. These properties are known as ACID properties.
ACID Properties:
A mean Atomicity (ROLLBACK)
C mean Consistency
I mean Isolation
D mean Durability (COMMIT)
These properties only automatically maintains consistent data in databases.

3) **Data Integrity:** Integrity means to maintain proper data. If we want to maintain proper data then we are defining set of rules, these rules are also called as " Business rules". In databases, we are maintaining proper data using 'constraints', 'triggers'. If we want to maintain proper data in flat files we must develop application programs in high level languages like COBOL, JAVA, ETC…..
4) **Data Security:** Data stored in flat files cannot be secured because flat files doesn't provides security mechanism. Whereas databases provides "ROLE based security".
5) **Data Indexing:** If we want to retrieve data very quickly from database then we are using indexing mechanism. Whereas flat files doesn't provide indexing mechanism.

To overcome all the above problems, a new software used by all organization to store data or information in secondary storage devices. This is called DBMS software.

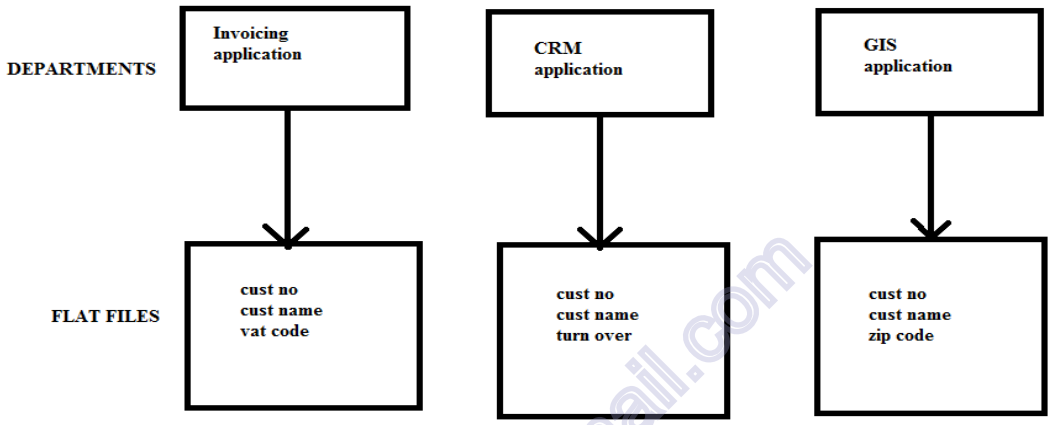
DBMS (DATABASE MANAGEMENT SYSTEM):

It is a collection of programs (S/W) written to manage database.

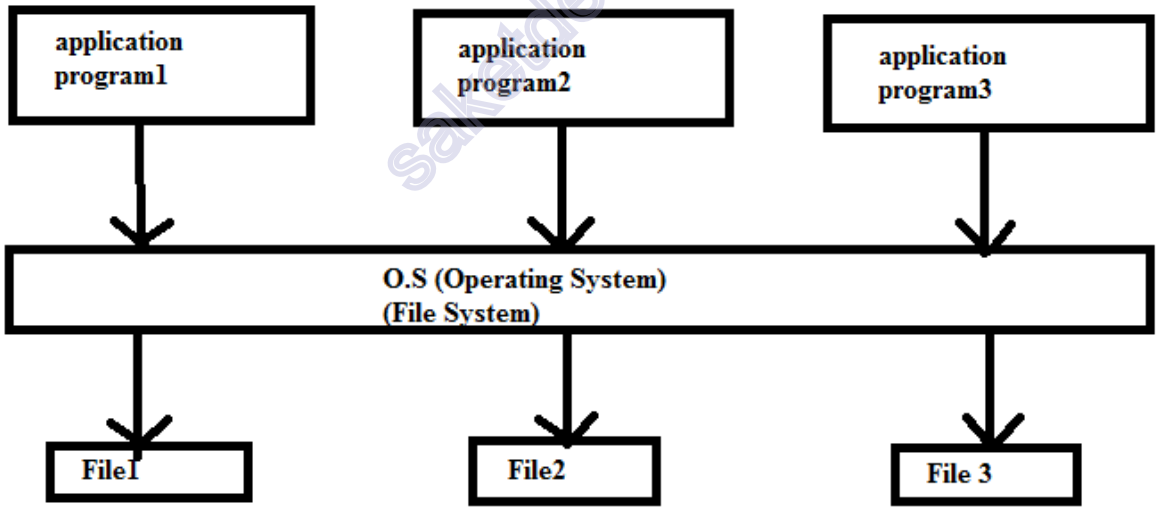
Example: ORACLE, FOXPRO, DB2, TERADATA, SQLSERVER, SYBASE, MYSQL, INGRESS, INFORMIX, SQLLITE…… etc;

In file based approach, every application program in the organization maintain its own file separate from other application program.

FILE based approach:



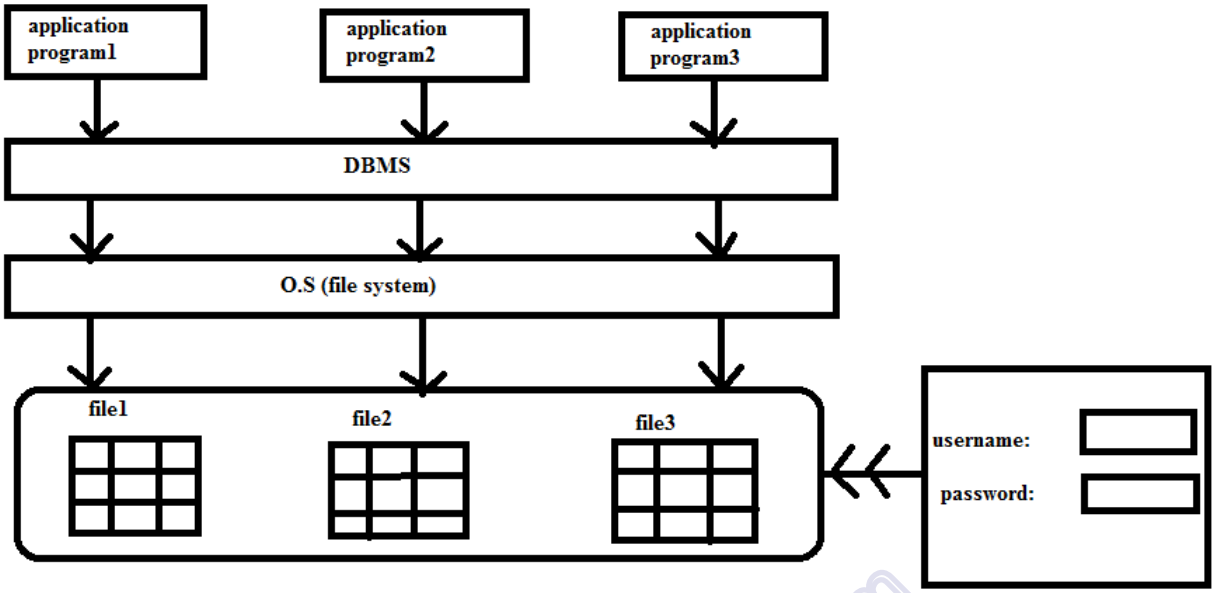
Here custno and cust name attributes are duplicate data.



THIS IS OLD APPROACH

Once we are installing DBMS software into our system then automatically some place is created in hard disc. This is called "Physical Database". And also automatically an user interface is created. Through the user interface we can also directly interactive with the

database or indirectly interactive with the database using application programs in high level languages.



"AFTER INSTALLING DBMS SOFTWARE"

Date: 3/3/2015

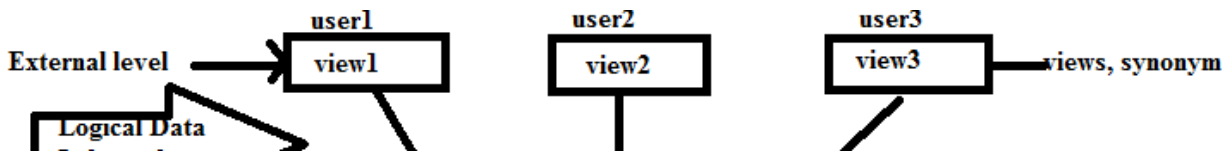
DATABASE: It is an organized collection of interrelated data used by application program in an organization. Once data stored in database it can be shared by number of users simultaneously and also this data can be integrated.

DBMS Architecture: American National Standard Institute(ANSI) has established three level architecture for database. This architecture is also called as "ansi/sparc" (Standard Planning And Requirements Committee) architecture.

Main objective of DBMS architecture is to separate users view of the database from the where physically it is stored.

This architecture mainly consist of three levels. They are:

1) External level
2) Conceptual level
3) Internal level



DBMS Architecture or Three Level Architecture

DBMS architecture provides "**DATA INDEPENDENCE**".

Data Independence: Upper levels are unaffected by changes in the lower levels is called as "Data Independence". DBMS architecture have two types of Data Independences:

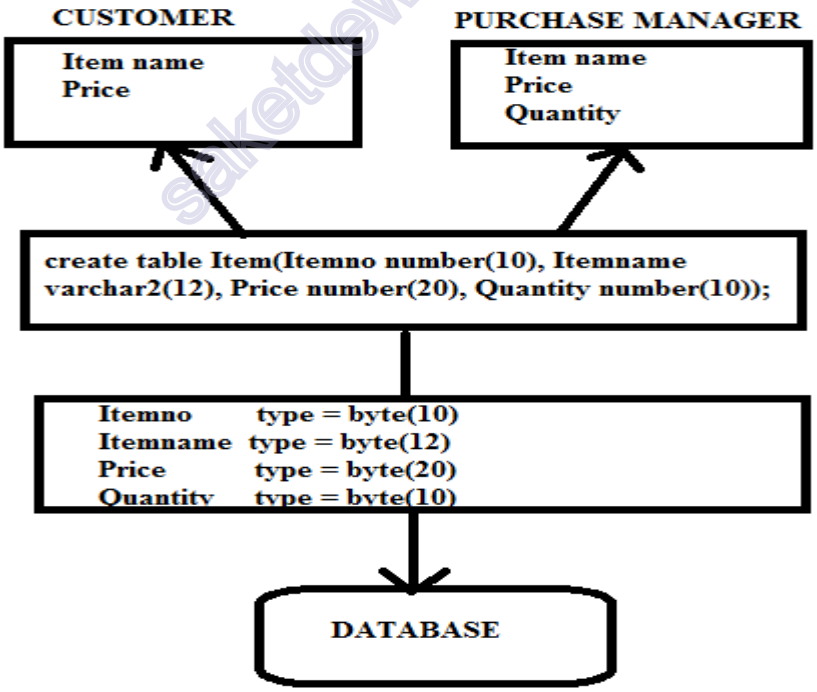
1) Logical Data Independence
2) Physical Data Independence

1) **Logical Data Independence:** Changes to the conceptual level do not required to change to the external level this is called Logical Data Independence.
   Example: Adding a new entity in conceptual level does not effect in external level.
2) **Physical Data Independence:** Changes to the internal level do not required to changes in conceptual level. This is called Physical Data Independence.
   Example: Adding an index to the internal level it is not affected in conceptual level.

CONCEPTUAL LEVEL: It describes logical representation of the database. Conceptual level defines type of data storing in database and also defines what type of data does not store in database using constraints and also specifies the relationship between data items.

NOTE: This level does not define how data is stored in database. In relational databases we are defining conceptual levels through tables.

EXTERNAL LEVEL: This level describes end user view of the database. i.e., in this level some group of users access only part of the database. In this level only we are defining the view and those views given to the number of users.

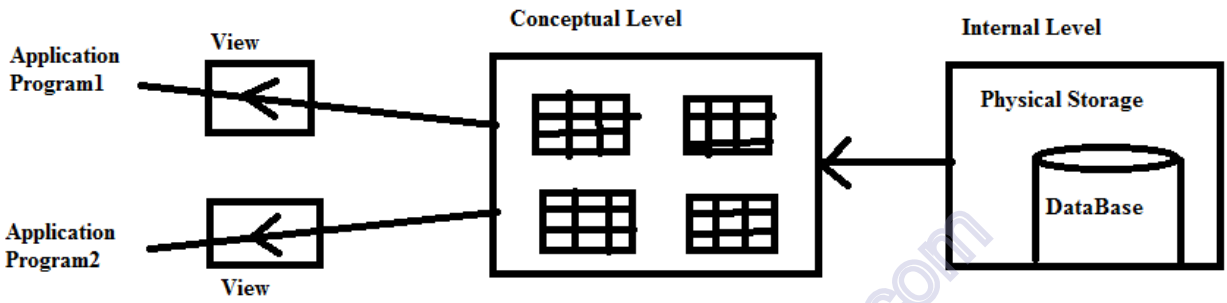


INTERNAL LEVEL: Internal level describes how physically data is stored in database. This level is handled by database administrator only. In relational databases indexes, cluster are available in internal level.

Data: 4/3/15

DATA MODELS: How data is represented at the conceptual level defined by means of "Data Model" in the history of database design three data models have been used.

1) Hierarchical Data Model
2) Network Data Model
3) Relational Data Model



DBMS ARCHITECTURE

Hierarchical Data Model: This model introduced in 1960. In this data model organizes data in tree like structure, we are representing data in parent child hierarchy. In this data model also data is represented in the format of records and also record type is also same as table in relational data model.

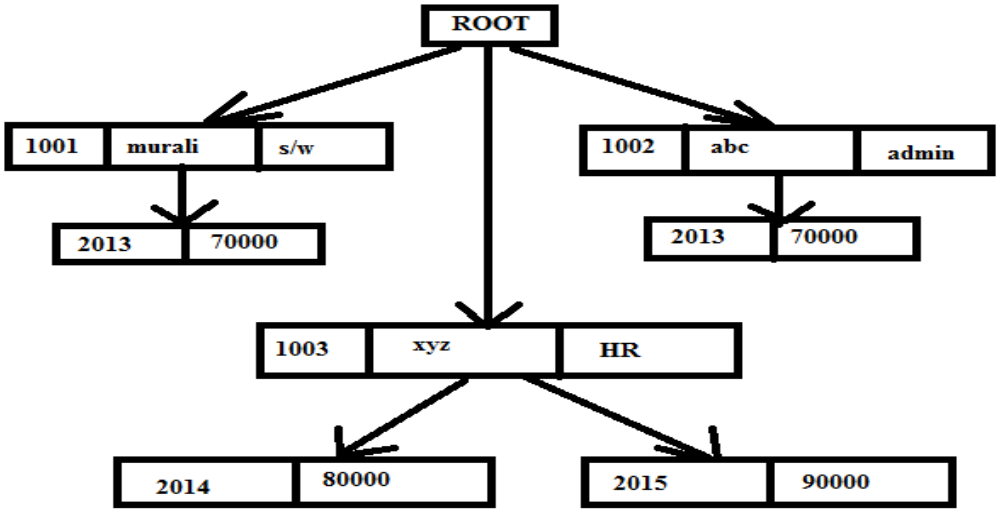
This Data model having more duplicate records because this data model is implemented based on one- to – many relationships. That is why in this data model always child segments are repeated.

In this data model products, we are retrieved data very slowly because in this data model products data base servers searching data based on root node onwards.

In 1960, IBM introduced IMS(Information Management System) product based on Hierarchical Data Model.

If we want to operate hierarchical data model products then we are using procedural language.

HIERARCHICAL DATA MODEL



RELATIONAL DATA MODEL

Primary key

EMP MASTER TABLE

empno	empname	deptname
1001	murali	s/w
1002	abc	admin
1003	xyyz	HR

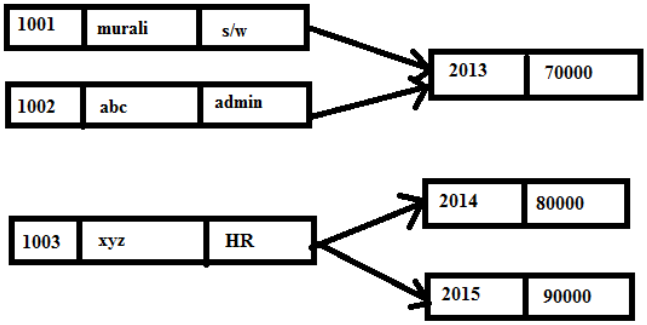
Foreign key

LOAN DETAILS CHILD TABLE

empno	year	loan amount
1001	2013	70000
1002	2013	70000
1003	2014	80000
1003	2015	90000

Date: 5/3/15

Network Data Model: In 1970's, CODASYL(Conference Or Data System Language) committee introduced network data model. This data model is implemented based on many – to – many relationships. In this data model also data is stored in format of records. And also records type is also same as table in Relational Data model. In 1970's IBM company introduced IDMS(Information Data Management System) based on network data model. If we want to operate network data model products then we must use procedural language.

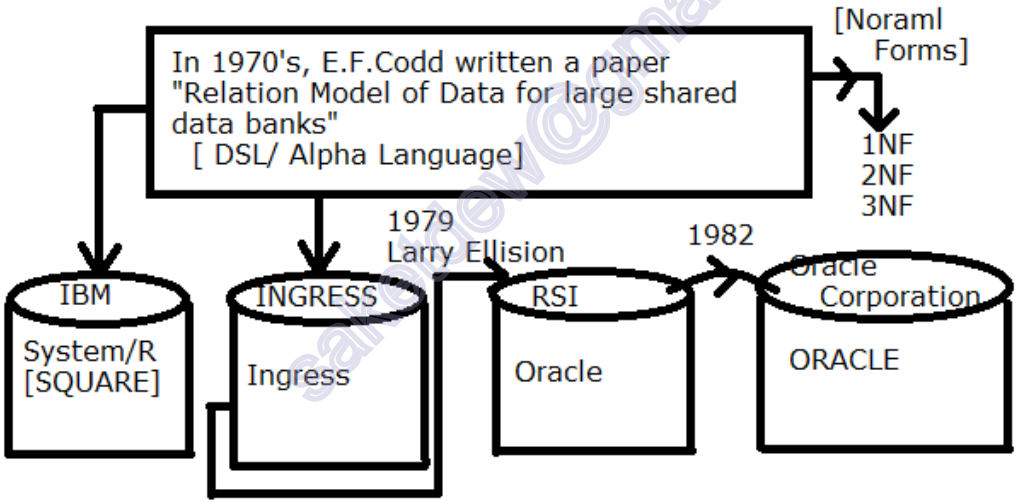


NETWORK DATA MODEL

Relational Data Model: In 1970, E.F.Codd introduced Relational Data Model. In this data model we are storing data in 2- dimensional tables. Relational data model mainly consist of 3 components.

1) Collection of Objects.
2) Set of Operators.
3) Set of Integrity rules.

If we want Relational Data Model products then we are using Non- Procedural Language "SQL".



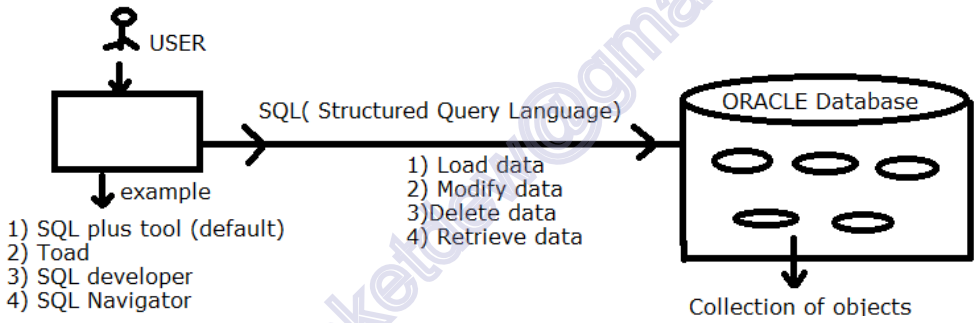
➔ In 1977 Larry Ellision, Bob Miner, Ed Oates founded new company " Software Development Laboratories" (SDL)
➔ In 1978 SDL Introduced oracle version1 (never released).
➔ In 1979 SDL name changed into RSI(Relational Software Incorporation).
➔ In 1982 RSI name changed into Oracle Corporation.

ORACLE VERSIONS

1)**Oracle 2.0** -> 1979
-> First public release
-> In this 2.0 only basic SQL functionality is there "joins".

2) **Oracle 3.0** -> 1983
-> Rewritten in C language.
-> Commit, Roll back.
3) **Oracle 4.0** -> 1984
-> Read Consistancy
-> exp/imp utility programs [export/import].
4) **Oracle 5.0** -> 1985
-> Client server architecture.
5) **Oracle 6.0** -> 1988
-> Introduced PL/SQL
-> Row level locks.
6) **Oracle 7.0** -> 1992
-> Roles
-> Integrity constraints.
-> Stored Procedures
-> Stored Functions
-> Packages
-> Triggers
-> Data type "varchar" changed into "varchar2"
-> Truncate table.
7) **Oracle 7.1** -> 1994
-> Introduced dynamic SQL
-> ANSI/ISO SQL-92.
8) **Oracle 7.2** -> 1995
-> Inline views or sub queries used in from clause.
-> ref cursor (cursor variable)
9) **Oracle 7.3** -> 1996
-> Bit map indexes.
-> utl_file package.
10) **Oracle 8.0** -> 1997
-> Object technology
-> Columns increased per a table upto "1000".
-> nested table, varray.
-> Instead of triggers.
11) **Oracle 8i** (i- internet) -> 1999
-> Materialized views.
-> Function based indexes
-> Case conditional statements.
-> Analytical functions.
-> Autonomous trasactions
-> rollup, cube.
-> BULK BIND
12) **Oracle 9i** -> 2001
-> 9i joins or ansi joins

-> merge statements.
-> multi table insert
-> flash back queries.
-> Renaming a column
13) **Oracle 10g** (g- grid) -> 2003
-> recycle bin
-> flash back table
-> indices of clause
-> regular expressions
-> wm_concat().
14) **Oracle 11g** -> 2007
-> Introduced continue statement in PL/SQL loops
-> Read only tables
-> Virtual Columns
-> Pivot() fuction.
-> Compound trigger
-> enable, disable clauses used in trigger specification
-> follow clause
-> Sequences used in PL/SQL without using dual table.
-> Named, mixed notations are used in a subprogram executed used select statement.
Date: 6/3/15



SQL (STRUCTURE QUERY LANGUAGE)

SQL is an Non- procedural language, which is used to operate all relational database products.

- ➔ In 1970, E.F.Codd introduced Relational Data Model. And also DSL/ Alpha language. In IBM , "System/R" team modified DSL/Alpha language into "SQUARE". Again IBM company change SQUARE into SEQUEL (Structured English Query Language). Then only SEQUEL became "SQL".
- ➔ In 1986 introducing ANSI in SQL
- ➔ In 1987 introducing ISO in SQL
- ➔ In 1989 releasing first version of ANSI/ISO SQL 89 that is SQL 89.
- ➔ In 1992 ANSI/ISO SQL92 -> SQL92.

➔ In 1999 ANSI/ISO SQL 99 -> SQL99.
➔ In 2003 ANSI/ISO SQL 03 -> SQL03.

SQL Sub Languages in every database:

1) Data Definition Language (DDL):
➔ Create
➔ Alter
➔ Drop
➔ Truncate
➔ Rename(oracle 9i)
2) Data Manipulation Language (DML):
➔ Insert
➔ update
➔ delete
➔ merge(oracle 9i)
3) Data Retrieval Language(DRL) or Data Query Language(DQL):
➔ SELECT
4) Transactional Control Language (TCL):
➔ Commit
➔ Rollback
➔ Savepoint
5) Data Control Language(DCL):
➔ Grant
➔ Revoke

Oracle 10g, 11g, 12c.(Enterprise Edition)
Username: scott
Password: tiger
Error: Account locked message displayed for first time. So, again we have to connect.
Username: \ sys as sysdba
Password: sys
SQL> alter user scott account unlock:
SQL> conn scott/tiger
Enter password: tiger
Confirm password: tiger
Date: 7/3/2015

Data Types: Data types specifies type of data within a table column Oracle have following data types:

1. Number(p,s)
2. Char→ varchar2(maxsize)
3. Date.

1) Number(p,s):
p→ precision (total number of digits)
s→ scale (it is used to store fixed, floating point numbers).

Syntax: columnname number(p,s)
Example: SQL> create table test(sno number(7,2));
SQL> insert into test values (12345.67);
SQL> select * from test;

```
   Sno
---------
12345.67
```

SQL> insert into test values(123456.7)
Error: value larger than specified precision allows for this column.
Note: whenever we are using number(p,s) then total number of digits before decimal places upto "p-s" number of digits.
[eg:- p-s => 7-2=5]
After Decimal:
SQL> insert into test values(12345.6789);
SQL> select * from test;

```
     Sno
-------------
12345.68
```

SQL> insert into test values(12345.6725);
SQL> select * from test;

```
    Sno
------------
12345.67
```

Note: whenever we are using number(p,s) and also if we are passing more number of digits after decimal place then oracle server only automatically rounded that number based on specified "scale".
Number(p): It is used to store fixed numbers only. Maximum length of the precision is up to "38".
Example: SQL> create table test1(sno number(7));
SQL> insert into test1 values(99.9);
SQL> select * from test1;

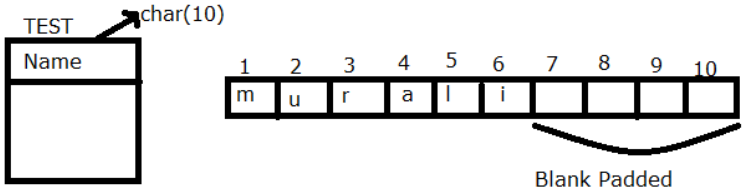
```
  Sno
-------
100
```

SQL> insert into test1 values(99.4);
SQL> select * from test1;

```
     Sno
-----------
    99
```

2) **Char:** It is used to store fixed length "alpha numeric" data in bytes. Maximum limit is upto 2000 bytes.
   Syntax: columnname char (size);
   By default character data type having one byte. Character data type automatically "Blank Padded".

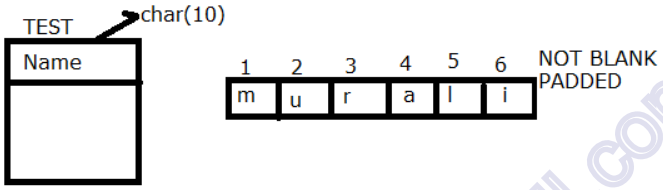
Blank Padded: Whenever we are passing less number of characters than the specified data type size then oracle server automatically allocates blank spaces after the value.
Example:



Varchar2(maxsize): It is used to store variable length alphanumeric data in bytes. Maximum limit is upto 4,000 bytes. Whenever we are using varchar2() data type oracle server not blank padded. After the value passed into that column.

Oracle 7.0 introduced "varchar2()" data type prior to oracle 7.0 we are using varchar2 data type. It is also same as varchar2() data type. But it will stores upto 2000 bytes.

Syntax: columnname varchar2(maxsize);



3) **Date:** It is used to store dates in oracle date format.
Syntax: columnname date;
In oracle by default date format is
DD- MON-YY

Date: 9/3/15

Data Definition Language(DDL):

➔ Create
➔ Alter
➔ Drop
➔ Truncate
➔ Rename (orace 9i)

These commands are used to define structure of the table.

1) **Create:** It is used to create database objects like tables, views, sequences, indexes.

Creating a table:

Syntax: create table tablename(columnname1 datatype(size), columnname2 datatype(size)…..);

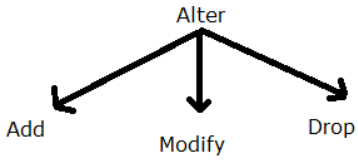
Example: SQL> create table first(sno number(10), name varchar2(10));

To view structure of the table:

Syntax: desc tablename;

Example: SQL> desc first;

2) **Alter:** It is used to change structure of the existing table.



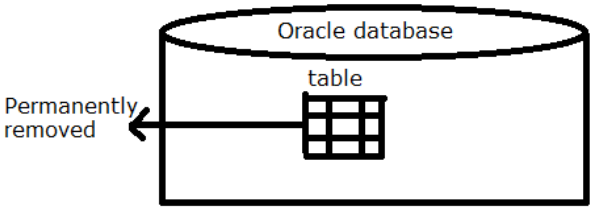
Add: It is used to add number of columns into the existing table.
Syntax: Alter table tablename add(col1 datatype(size));
Example: SQL> alter table first add sal number(10);
SQL> desc table;

Modify: It is used to change column datatype or column datatype size only.
Syntax: alter table tablename modify(col datatype(size), col2 datatype(size));
Example: SQL> alter table first modify sno date;

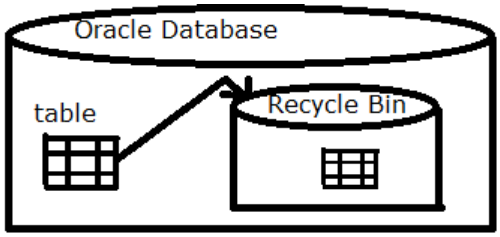
Drop: It is used to remove columns from the existing table.
Method1: If we want to drop a single column at a time without using parenthesis"()" then we are using following syntax:
Alter table tablename drop column columnname;
Example: SQL> alter table first drop column sal;
SQL> desc first;
Method2: If we want to drop single or multiple columns with using parenthesis then we are using following syntax.
Syntax: alter table tablename drop(colname1, colname2, colname3….);
Example: SQL> alter table first drop(sal);
Important:
SQL> alter table first drop column sno;
Error: cannot drop all columns in a table.
NOTE: In all database systems we cannot drop all columns in a table.
3) **DROP:** It is used to remove database objects from database.
Syntax: drop objecttype objectname;
1. Drop table tablename;
2. Drop view viewname;

Droping a Table:

Before Oracle 10g:
Syntax: drop table tablename;



Oracle 10g (Enterprise Edition)
Syntax: drop table tablename;



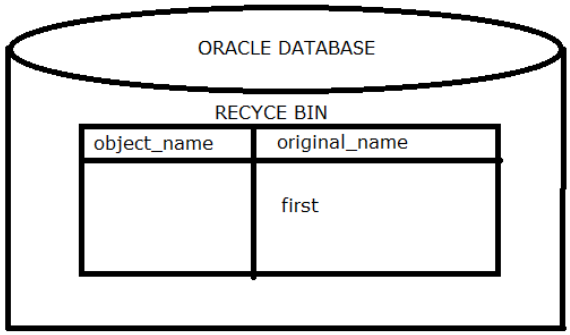
Get it back from recycle bin.
Syntax: flashback table tablename to before drop;
To drop permanently:
Syntax: drop table tablename purge;
Example: (dropping a table → oracle 10g Enterprise Edition)
SQL> drop table first;
Table dropped.
Get it back the table:
SQL> flashback table first to before drop;

Testing:
Method1: SQL> desc first;
Error: Object first does not exist.

Method2: SQL> flashback table first to before drop;
Error: object not in Recycle bin.

Date: 10/3/15

RECYCLE BIN: Oracle 10g introduced recycle bin which is used to store dropped tables. Recycle bin is a read only table whenever we are installing oracle then automatically so many read only tables are created. These read only tables are also called as " Data Dictionaries".



In oracle, we can also drop tables from recycle bin using "purge" command.

<u>To Drop particular table from Recycle bin:</u>

Syntax: purge table tablename;

Example: SQL> create table first(sno number(10));

SQL> drop table first;

SQL> desc recycle bin;

SQL> select original_name from recyclebin;

```
  Original_name
--------------------
    First
```

SQL> purge table first;

Testing:

SQL> select original_name from recyclebin;

No rows selected;

<u>To Drop all tables from Recycle bin:</u>

Syntax: SQL> purge recyclebin;

Example: SQL> create table first(sno number(10));

SQL> create table second(sno number(10));

SQL> drop table first;

SQL> drop table second;

SQL> desc recyclebin;

SQL> select original_name from recyclebin;

```
  Original_name
----------------------
      First
     Second
```

SQL> purge recyclebin;

Testing:

SQL> select original_name from recyclebin;

No rows selected

3) **Truncate:** Oracle 7.0 introduced truncate command , whenever we are using "truncate" command total data permanently deleted from table.

Syntax: truncate table tablename;
Example: create table first as select * from emp;
SQL> select * from first;
SQL> truncate table first;
Testing: SQL> select * from first;
No rows selected.
SQL> desc first;

4) **Rename:** It is used to rename a table and renaming a column also.
Renaming a Table:
Syntax: rename oldtablename to new tablename;
Example: rename first to last;

Renaming a Column: (oracle 9i)
Syntax: alter table tablename rename column oldcolumnname to newcolumnname;
Example: SQL> alter table emp rename column empno to sno;
SQL> select * from emp;
Note: In all database systems by default all DDL commands are automatically committed.

2. DATA MANIPULATION LANGUAGE (DML):

- ➔ Insert
- ➔ Update
- ➔ Delete
- ➔ Merge (oracle 9i)

These commands are used to manipulate data in a table.
1) **Insert:** It is used to insert data into a table.
Method1: Insert into tablename values(value1, value2, value3…..);
Example: SQL> create table first(sno number(10), name varchar(20));

Inserting data into table:
SQL> insert into first values(1, 'abc');
SQL> insert into first values(2, 'sachin');
SQL> select * from first;

Sno	name
1	abc
2	sachin

Method2: (using substitutional operator "&")
Syntax: insert into tablename values(&col1, &col2, &col3……);
Example: SQL> insert into first values(&sno, '&name');
Enter value for sno: 3
Enter value for name: xyz
SQL> /
Enter value for sno: 4

Enter value for name: sachin
SQL> select * from first;

Date: 11/3/15

Method3: (skipping columns)
Syntax: insert into tablename(columnname1, columnname2,…..)values(value1,value2,…);
Example: SQL> insert into first(name)values(‘www’);
SQL>select * from first;

3) **Update:** It is used to change data within a table.
Syntax: update tablename set columnname=newvalue where columnname=oldvalue;
Example: SQL> update emp set sal=1000 where ename=’SMITH’;
SQL> select * from first;
Note: In all database if we want to insert data into particular cell then we must use update command.
Example: SQL> alter table first add address varchar2(10);
SQL> update first set address=’mumbai’ where name=’sachin’;
SQL> select * from first;

Output:

sno	name	address
1	abc	
2	sachin	Mumbai
3	xyz	
4	zzz	

Example: SQL> update first set address=null where name=’sachin’;
SQL> select * from first;

5) **Delete:** It is used to delete particular rows or all rows from a table;
Syntax:
- ➔ Delete from tablename;
- ➔ Delete from tablename where condition;

First command is to delete all rows
Second command is to delete particular rows.
Example: SQL> delete from first;
Rows deleted.
SQL> rollback;
SQL> select * from first;

Difference between ‘delete’ and ‘truncate’:
Whenever we are using ‘delete from tablename’ or ‘truncate table tablename’ then automatically total data is deleted. Whenever we are using ‘delete from tablename’ then deleted data automatically stored in a buffer. We can get it back, this deleted data using

"rollback". Whenever we are using 'truncate table tablename' then total data permanently deleted. Because truncate is an DDL command. That's why we cannot get it back this data using "rollback".

3) DATA RETRIVAL LANGUAGE(DRL) (OR) DATA QUERY LANGUAGE(DQL):

-> **Select**

In relational databases we can retrieve data fromtable using select statement.

```
Syntax: select col1,col2,…. From tablename where condition
        group by columnname
        having condition
        order by columnname[asc/desc];
```

1. Select all cols and all rows.
2. Select all cols and particular rows.
3. Select particular cols and all rows.
4. Select particular cols and particular rows.

Date: 12/3/15

Creating a new table from another table:

Syntax: create table newtablename as select * from existingtablename;

Example: SQL> create table test as select * from emp;

SQL> select * from test;

Note: In all database systems whenever we are copying a table from another table constraints are never copied. (Primary table, Foreign key,…..);

Creating a new table from existing table without copying data:

Syntax: create table newtablename as select * from existingtablename where falsecondition;

Example: SQL> create table test1 as select * from emp where 1=2;

Testing: SQL> select * from test1;

No rows selected.

SQL> desc test1;

Operators Used in "Select" statement:

1. Arthematic operator(+,-,*,/)
2. Relational operator(=,<,<=,>,>=,[<> or!=]not equal).
3. Logical operator (AND,OR,NOT)
4. Special operator.

Arithmetic operator is used for "select" (eg: select column1, column2,…)

Relational and Logical and Special operators are used for "where" (eg: where condition)

We can also use arithmetic operators in "where" conditions.

➔ Arithmetic operators are used in number, data datatype column.

Q) write a query to display ename, sal, annsal from emp table;

Ans: SQL> select ename, sal, sal*12 annsal from emp;

ENAME	SAL	ANNSAL
SMITH	700	8400

Q) write a query to display the employees except job as clerk from emp table.

Ans: select * from emp where job<>'CLERK'; {or job != 'CLERK'}

Q) Write a query to display the employees who are getting more than 2000 salary from emp table?

Ans: SQL> select * from emp where sal>2000;

Example: SQL> select * from emp where job='CLERK' and sal>2000;

Note: In databases if we want to retrieve multiple values within a single column then we must use "OR" operator.

Example: SQL> select * from emp where job='CLERK' or job='SALESMAN';

Q) write a query to display the employees who are belongs to the department numbers 20,50,70,90?

Ans: SQL> select * from emp where deptno=20 or deptno=50 or deptno=70 or deptno=90;

⇨ SQL> select * from emp where sal>2000 and sal<5000;

**** Special operators:**

1) 'In' opposite 'not in'
2) 'between' opposite 'not between'
3) 'is null' opposite 'is not null'
4) 'like' opposite 'not like'

1. **In:** It is used to pick the values one by one from list of values. 'In' operator performance is very high compared to 'OR' operator. If we want to retrieve multiple values in a single column then we are using 'IN' operators in place of 'OR' operator because 'IN' operator performance is high.

Syntax: select * from tablename where columnname in(list of values);

Columnname belongs to any data type is possible.

Example: SQL> select * from emp where deptno in(20,30,50,70,90);

Date: 13/3/15

Eg: SQL> select * from emp where ename in('SMITH','FORD);

Note: In all datebase systems whenever we are using multiple row sub queries then we must use "in" operator.
Eg: select * from emp where deptno not in(10,20,null);
Note: In all relational databases "not in" operator doesn't work with "null" values.
Null: Null is an undefined, unavailable, unknown value. It is not same as "zero". Any arithmetic operations performed on null values again it becomes "null".
Eg: null+50=> null.
Q) Write a query to display ename, sal, comm, sal+comm of the employee SMITH from emp table.
Ans: SQL> select ename, sal, comm, sal+comm from emp where ename="SMITH";

o/p: ename sal comm. Sal+comm

SMITH	1100	null	null

To overcome this problem oracle provided "NVL()" function.
NVL(): NVL() is a predefined function which is used to replace or substitute user defined value in place of "null".
Syntax: NVL(exp1,exp2);
Here exp1,exp2 must belongs to same datatype. If exp1 is null then it returns exp2. Otherwise it returns exp1.
Eg: 1) NVL(null,30) -> 30.
2) NVL(10,20) -> 10.
Solution: select ename, sal, comm, sal+NVL(comm, 0) from emp where ename= 'SMITH";

ENAME	SAL	COMM	SAL+COMM
SMITH	1100	Null	1100

⇨ Sal+ nvl(comm.,0)
⇨ 1100 + nvl(1100,0)
⇨ 1100 + 0
⇨ 1100

NVL2(): Oracle 9i introduced NVL2() function. This function accepts three parameters.
Syntax: NVL2(exp1,exp2,exp3).
Here if exp1 is null, then it returns exp3. Otherwise it returns exp2.
Eg: SQL> select nvl2(null,10,20) from dual;
20
SQL> select nvl2(30,40,50) from dual;
40
Q) Update employee commission as follows using nvl2() from emp table;
1) If comm is null then update comm.->500
2) If comm is not null then update comm-> 500
Ans: SQL> update emp set comm=nvl2(comm, comm+500,500);

SQL> update emp set comm=nvl2(comm,comm+500,500);
SQL> select * from emp;

2) Between: This operator is used to retrieve range of values. This operator is also call as Betweem….. And operator.
Syntax: Select * from tablename where columnname between lowvalue and highvalue.
Eg: select * from emp where sal between 2000 and 5000;

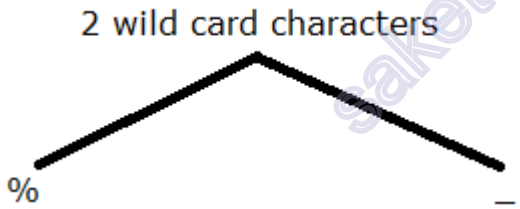
Date: 14/3/15

While using "DATE":
SQL> select * from emp where hiredate between '01-jan-1981' and '01-jan-1982';

3) Is null, is not null: These two operators used in 'where" condition only. These two operators are used to test weather a column having null values or not.
Syntax: select * from tablename where columnname is null;
Syntax: select * from tablename where columnname is not null;
Q) write a query to display the employees who are not getting commission from emp table.
Ans: SQL> select * from emp where comm is null;

Q) write a query to display the employees who are getting commission from emp table?
Ans: SQL> select * from emp where comm is not null;

4) Like: It is used to search strings based on character pattern. "Like" operator performance is very high compare to searching functions. In all databases along with like operator we are using 2 wild card charcters.



% -> group of characters if want to match or string
_ -> single character to match.

Syntax: select * from tablename where columnname like 'characterpattern';
Q) write a query to display the employees whose ename start with "M" from emp table using like operator.

Ans: SQL> select * from emp where ename like 'M';
MARTIN
MILLER
Q) write a query to display the employees where ename second letter will be 'L' from emp table using 'like' operator?
Ans: SQL> select * from emp where ename like '_ _L%';
ALLEN
CLARK
BLAKE

Q) write a query to display the employees who are joining in the month December from the emp table using like operator.
Ans: SQL> select * from emp where hiredate like '_ _ _D%'; (or) '%DEC%";

Q) write a query to display the employees who are joining in the year "81" from emp table using like operator?
Ans: SQL> select * from emp where emp table like '%81';

"Escape" function used in like operator:
Whenever wild card characters available in character data then we are trying to retrieve the data using like operator then database servers treated these wild card characters as special characters to overcome this problem ansi/iso SQL provided a special function "Escape" along with 'like' operator. Which is used to escape special characters and also this function treated as _,% as same meaning as _,%.
Syntax: select * from tablename where columnname like 'character pattern' escape 'escape character';
Note: Escape character must be an single character having length '1' within character pattern we must use escape character before wild card character.
Eg: SQL> insert into emp(empno, ename) values(1,'S_ MITH');
SQL> select * from emp;
Q) write a query to display the employees whose ename start with 'S_' from emp table using 'like' operator?
Ans: SQL> select * from emp where ename like 'S_%';
SMITH
SCOTT
S_MITH
Solution: SQL> select * from emp where ename like 'S?_%' escape'?';
S_MITH
Eg: insert into emp(empno, ename) values(2,'S__MITH');
SQL> select * from emp;
Solution: SQL> select * from emp where ename like 'S?_?_%' escape'?';
S__MITH

➔ **Concatenation Operator(|| double pipe):** It is not a special operator rarely used in "SQL" and regularly used in "PL/SQL". If we want to display column data along with literal strings then we must use concatenation operator.
Eg: select 'my employee names are'|| ename from emp;
If we want to display our own space in between the columns then also we can use concatenation operator.
Eg: select ename||' '||sal from emp;

Date: 16/3/15
Functions: Functions are used to solve particular task and also functions must return a value. Oracle have two types of functions.

1. Predefined functions
2. User defined functions

1) **Predefined functions:** there are 4 types.
   - ➔ Number function
   - ➔ Character function
   - ➔ Date function
   - ➔ Group function(or) Aggregate function

1) **Number function:** These functions operate over "number" data.

- **Abs() :** It is used to convert negative values into positive values.
  Eg: SQL> select abs(-50) from dual;
  50
- **Dual():** Dual is a predefined virtual table which contains only one row and one column Dual table is used to test predefined, user defined functions functionality.
  Example for testing predefined functions:
  SQL> select nvl(null,30) from dual;
  30
  SQL> select nvl(20,30) from dual;
  20
  SQL> select * from dual;

D
X

  Note: In oracle by default dual table column datatype is varchar2().
  Eg: select * from dual;

D ➔ Dummy column
X name varchar2()

  Generally, this table is also used to perform mathematical operations.
  Eg: SQL> select 10+50 from dual;
  60
  Eg: SQL> select ename, comm, sal, abs(comm-sal) from emp where comm is not null;

- **Mod(m,n):** It will gives remainder after 'm' divided by 'n'.

  Eg: SQL> select mod(10,5) from dual;
  0
- **Round(m,n):** It rounds given floated valued number 'm' based on 'n'.
  Eg: SQL> select round(1.8) from dual;
  2
  SQL> select round(1.23456,3) from dual;
  1.235
  Note: Round always checks remaining number if remaining number is above 50% then one added to the rounded number.
  Eg: SQL> select round(1285.456,-1) from dual;
  1290
  Execution:
  Step1: 1280->5 is replace with 0. Out of 10, 5 is above 50%(>=)
  Step2: 1280

- **Trunc(m,n):** It truncates given floated values number 'm' based on 'n'. This function doesn't check remaining number is above 50% or below 50%.
  Eg: SQL>select trunk(1.8) from dual;
  Eg: SQL> select trunk(1.23456,3) from dual;
  1.234
- **Greatest(exp1,exp2,…. expn), Least(exp1,exp2,…. expn):**
  Greatest returns maximum value among given expressions. Where as Least returns minimum value among given expressions.
  Eg: SQL> select greatest(3,5,8,9) from dual ;
  9
  SQL> select ename, sal, comm, greatest(sal,comm)from emp where comm is not null;
  SQL> select max(sal) from emp;
  6600

Date : 18/3/15

- **Ceil() and Floor():** These two functions always return integer. Ceil() returns nearest greatest integer where as floor() returns nearest lowest integer.
  Eg: select ceil(1.3) from dual;
  2
  Eg: select floor(1.9) from dual;
  1

2) CHARCTER FUNCTIONS:

- **Upper():** It is used to convert a string or column values into " upper case ".
  Eg: select upper('abc') from dual;
  ABC
  For column: SQL> select upper(ename) from dual;
- **Lower():**
  Eg: select lower(ename) from emp;
  Updating or modifying data within table:
  SQL> update emp set ename=lower(ename);
- **Initcap():** It returns first letter is capital and all remaining letters are small.
  Eg: SQL> select initcap(ename) from emp;
  Eg: SQL> select initcap('ab cd ef') from emp;
  Ab Cd Ef
- **Length():** It returns number datatype that is ., it returns total length of the string including spaces.
  Eg: select length('AB_CD') from dual;
  5

❖ **Substr():** It will extract portion of the string within given string based on last two parameters.

Eg: select substr('ABCDEF',2,3) from dual;
BCD
SQL> select substr('ABCDEF',-2,3) from dual;

EF
SQL> select substr('ABCDEF',-4) from dual;
CDEF
Syntax: substr(columnname (or) 'string name', starting position, no. of characters position);
Starting position-> can be +ve or –ve
Starting position and no. of characters position-> must be numbers

Q) write a query to display the employees whose ename second letter would be "LA" from emp table using substring function?
And: SQL> select * from emp where substr(ename,2,2)='LA';
BLAKE
CLARK
Note: In all database systems we are not allowed to use group functions in where clause but we are allowed to use number functions, character functions, date functions in where clause.
Eg: select * from emp where sal=max(sal);
Error: group function is not allowed here.
Q) write a query to display the employee whose employees length is 5 from emp table.
Ans: SQL> select * from emp where length(ename)=5;

➔ **Instr():** instr() always returns number datatype that is it returns position of the delimiter, position of the character, position of the string within given string .
Eg: SQL> select instr('ABC*D','*') from dual;

Date: 19/3/15
Eg: SQL> select instr('ABCDEFGHCDKMNHCDJL','CD',-5,2) from dual;
3
SQL> select instr('ABCDEFGHCDKMNHCDJL','CD',-4,2) from dual;
9
Syntax: instr(columnname(or)'string name', 'str', searching position, No. of occurrences from searching position);
Searching position -> +ve or –ve
Searching and No. of occurrences -> must be numbers.
Always in string returns position based on last two parameters but oracle server counts no. of characters left side first position onwards.

Lpad(): It will fills remaining spaces with specified character on the left side of the given string. Here always second parameter returns total length of the string. And also third parameter is an optional.Syntax: Lpad(columnname (or) 'string name', total length, 'filled character');
Total length -> number

Eg: SQL> select lpad('ABCD',10,'#') from dual;
######ABCD
SQL> select lpad('ABC',5) from dual;

(space)(space)ABC
SQL> select lpad('ABC',2,'*') from dual;
AB

Rpad():
Eg: SQL> select rpad('ABCD',10,'#')from dual;
ABCD######
SQL> select rpad(ENAME,50,'-')||sal from emp;

Ltrim: It removies specified character on the left side of the given string.
Syntax: Ltrim(columnname(or)'string name', {set of characters});

Eg: SQL> select ltrim('SSMISSTHSS','S')from dual;
MISSTHSS
SQL> select job,ltrim(job,'CSM') from emp;

JOB	LTRIM(JOB)
CLERK	LERK
SALESMAN	ALESMAN
MANAGER	ANAGER

Rtrim:
Eg: SQL> select rtrim('SSMISSTHSS','S') from dual;
SSMISSTH
Trim(): Oracle 8i introduced trim function it is used to remove left and right side specified characters and also it is used to remove leading and trailing spaces.
Syntax: trim('character' from 'string name');
Eg: SQL> select trim('S' from 'SSMISSTHSS')from dual;
MISSTH
This function also behaves like a ltrim, rtrim based on leading and trailing claused.
Eg: select trim(leading 's' from 'SSMISSTHSS') from dual;
SSMISSTH
Eg: select length(trim(' welcome '))from dual;
7

Translate() and Replace():
Translate() is used to replaces character by character where as replace() is used to replaces character by string (or) string by string.

Eg: select translate('india','in','xy'),replace('india','in','xy') from dual;

Translate	Replace
Xydxa	xydia

Syntax: translate('str1','str2','str3'….);
Eg: select translate('ABCDEF','FEDCBA',123456);
654321
Eg: select replace('ABC',' ','india')from dual;
AindiaBindiaC
Eg: select job,replace(job, 'SALESMAN','MARKETING') from emp;
Note: In replace function if we are not specifying third parameter specified character permanently removed from the string.
Eg: select replace('SSMISSTHSS','S') from dual;
MITH

Date: 20/3/15
If you want to count number of times particular character occurs within a given string then also we are using replace function along with length function.
Q) Write a query to count number of times that particular 'I' occurred within given string 'india' using replace function?
Ans: SQL> select length('india')-length(replace('india','I')) from dual;
2
Concat(): It is used to concatenate given two strings.
Eg: select concat('wel','come')from dual;
Welcome

DATE functions: In oracle by default date format is DD-MON-YY. Oracle having following date functions.
1. Sysdate
2. Add_months()
3. Last_day()
4. Next_day()
5. Months_between()

1) **Sysdate:** It returns current date of the system in the oracle date format
SQL> select sysdate from dual;
20-MAR-15
2) **Add_months():** It is used to add or subtract number of months to the specified date based on second parameter.
Syntax: add_months(date, number);
Number-> can be positive or negative.
Eg: SQL> select add_months(sysdate,1)from dual;
20-Apr-15
SQL> select add_months(sysdate,-1) from dual;
20-FEB-15
3) **Last_day():** It returns last date of the given months based on the passed date. This function always accepts one parameter.
Syntax: last_day(date);

Eg: SQL> select last_day(sysdate)from dual;
31-Mar-15

4) **Next_day():** It returns next occurrence day from the specified date based on second parameter.

Syntax: next_day(date,'day');
Eg: select next_day(sysdate,'monday')from dual;
23-MAR-15

5) **Months_between():** These function always returns "number" data type. i.e., it returns number of months between two specified dates.

Syntax: months_between(date1,date2)
Here date1 is more than date2 otherwise this function returns "negative" sign.
Eg: SQL> select ename, round(months_between(sysdate,hiredate))from emp;

DATE arithmetic:

1. Date + number
2. Date – number
3. Date1 + date2
4. Date1 – date2

Eg: SQL> select sysdate+1 from dual;
SQL> select sysdate-1 from dual;
SQL> select sysdate-sysdate from dual;
0
Q) write a query to display first date of the current month using predefined date functions sysdate, add_months(), last_date() functions.
Ans: SQL> select last_day((add_months(sysdate,-1)))+1 from dual;
OR
SQL> select add_months(last_day(sysdate),-1)+1 from dual;
01-MAR-15

DATE CONVERSION FUNCTIONS:
Oracle provided two date conversion function. They are:

1. To_char()
2. To_date()

Date: 23/3/15

1. **to_char():** It is used to convert date time into character type i.e., it converts date type into date string.

Eg: SQL> select to_char(sysdate,'dd/mm/yy')from dual;
23/03/15
Note: Basically "to_char" character formats are case sensitive formats.
Eg: SQL> select to_char(sysdate,'DAY')from dual;
MONDAY
Day-> Monday(lowercase letters)

DAY
DY
MONTH
MON
YEAR
RETURNS CHARACTER AS OUTPUT
D
DD
DDD
MM
YY
YYYY
HH
MI
SS
RETURNS NUMBER AS OUTPUT
Eg: SQL> select to_char (sysdate,'DY')from dual;
MON

Eg: SQL> select to_char (sysdate,'D')from dual;
2
D->day of the week(sun-1,mon-2,tue-3,….sat-7)
DD->day of the month
DDD->day of the year(tells how many days completed)
DDTH-> used as date with 'rd' or 'th' (eg: 5th 19th 23rd)
Eg: SQL> select to_char(sysdate,'DDSPTH') from dual;
Twenty third
SP-> spell out
Eg: SQL> select to_char(sysdate,'MON') from dual;
MAR
Eg: SQL> select to_char(sysdate,'HH:MI:SS') from dual;
09:51:42
By default 12-hours format
to convert into 24-hours format we have to "HH24:MI:SS" (24-hours format)

Q) SQL> select to_char('15-JUN-05','15-JUNE-05') from dual;
Error: invalid number
Note: whenever we are using to_char() always first parameter must be oracle "date" type otherwise oracle server returns an error.

2. **to_date():** It is used to convert date string into oracle "date" type(oracle date format)
Eg: SQL> select to_date('27/JUNE/05') from dual;
27-JUN-05
Eg: SQL> select to_date('27/06/05') from dual;

Error: invalid month (or) not a valid month
Whenever are using to_date() passed parameter return values match with the default date format return values. Otherwise oracle server returns an error. To overcome this problem use a second parameter as same as first parameter format then only oracle server automatically converts "date string" into "date type".
Solution: SQL> select to_date('27/06/05','DD/MM/YY') from dual;
27-JUN-05

- ➔ SQL> select to_date('09-feb-15')+5 from dual;
  14-feb-15
- ➔ SQL> select to_date('09-feb-15','DD-MM-YY')+5 from dual;
  14-FEB-15

Q) Write a query to convert given date string into client requirement format using to_char()
Given date is -> '15-jun-05
Expected format-> '15/jun/05'
SQL> select to_char('15-jun-05','dd/month/yy') from dual;
Error: invalid number
SQL> select to_char(to_date('15-jun-05'),'dd/month/yy') from dual;
15/jun /05
Date: 24/3/15

Fill mode(FM): Whenever we are using to_char() if format is either month(or) day then oracle server automatically returns "spaces" when the month (or) day value less than the maximum length of the month (or) day. To overcome this problem oracle provided Fill Mode(FM) which suppress blank spaces and leading zeros(0). This mode is used in second parameter of the to_char().
Eg: SQL> select to_char(to_date('15-JUN-05'),'DD-MON-YY') from dual;
15-JUNE -05
SQL> select to_char(to_date('15-JUN-05'), 'DD/FMMONTH/YY') from dual;
15/JUNE/05
Q) Write a query to display the employees who are joining in the month December from "emp" table using to_char()?
Ans: SQL> select * from emp where to_char(HIREDATE,'MON')= 'DEC';
OR
SQL> select * from emp where to_char(HIREDATE,'MM')='12';
OR
SQL> select * from emp where to_char(HIREDATE,'FMMONTH')='DECEMBER';

Q) Write a query to display the employees who are joining in the year '81' from emp table using to_char()?
Ans: SQL> select * from emp where to_char(hiredate,'yy')='81';
SQL> select hiredate, to_char(hiredate,'yyyy') from emp;
In Oracle, whenever we are passing date string into date functions then oracle server automatically converts date string into date type.

Eg: SQL> select last_day('12-JUN-05') from dual;
30-JUN-05
But here passed parameter format must be default date format otherwise oracle server returns an error. To overcome this problem we must use to_date().
Eg: SQL> select last_day('12-06-05') from dual;
Error: invalid month
Solution: SQL> select last_day(to_date('12-06-05','DD-MM-YY')from dual;
30-JUN-05
In Oracle whenever we are inserting dates into date data type column then oracle server automatically converts date string into date type when inserted date string is in oracle date format. Otherwise oracle server returns an error. To overcome this problem also then we must use to_date().
Eg: SQL> create table test(col1 date);
SQL> insert into test values(sysdate);
SQL> select * from test;
COL1

24-MAR-15
SQL> insert into test values('15-aug-05');
SQL> select * from test;
COL1

24-MAR-15
15-AUG-05
SQL> insert into test values('15-08-05');
Error: not a valid month
SQL> insert into test values(to_date('15-08-05','DD-MM-YY'));
1 row inserted
SQL> select * from test;
COL1

24-MAR-15
15-AUG-05
15-AUG-05

Round(), Trunc() used in dates:
In Oracle, "date" data type contains both date and time. Whenever we are using the round() or trunc() then automatically date part changed based on time portion.

Date: 25/3/15

Round() adds one day to the given date if time portion is greater than or equal to 12 noon and also time portion automatically set to "zero".
Eg: SQL> select to_char(sysdate,'DD-MON-YYYY HH24:MI:SS') from dual;

25-MAR-2015 12:24:59
When applying round():
SQL> select to_char(round(sysdate),'DD-MON-YYYY HH24:MI:SS') from dual;
26-MAR-2015 00:00:00
Whenever we are using trunc() oracle server automatically returns same date if time portion is greater than or equal to 12 noon also and also trunc() time portion is set to "zero"(0)
Eg: SQL> select to_char(trunc(sysdate),'DD-MON-YYYY HH24:MI:SS') from dual;
25-MAR-2015 00:00:00
Q) Write a query to display the employees who are joining today from emp table?
Ans: SQL> select insert into emp(empno,ename,hiredate)values(1, 'MURALI',25-MAR-15);
SQL> select * from emp;
SQL> select * from emp where trunc(hiredate) = trunc(sysdate);

GROUP FUNCTION (OR) AGGREGATE FUNCTION:
Oracle having following group function
1. max()
2. min()
3. avg()
4. sum()
5. count(*)
6. count(column name)

In all databases group functions are operate over number of values within a column and returns a single value

1. **Max():** It returns maximum value within a column.

Eg: SQL> select max(sal) from emp;
6600
SQL> select max(hiredate) from emp;
23-MAY-87
SQL> select max(ename) from emp;
WARD (A=1…… Z=26)

2. Min():
Eg: SQL> select min(sal) from emp;
750
To know the senior most employee
SQL> select min(hiredate) from emp;
17-DEC-80
In all database systems we are not allowed to use group functions in "where" clause.
SQL> select * from emp where sal= min(sal);
Error: group function is not allowed in "where"

3. **Avg():** It returns average from number data type column.

Eg: SQL> select avg(sal) from emp;
2473.21429

SQL> select avg(comm) from emp;
550
NOTE: In all database systems by default all group function ignores null values except "count(*)".
To count all null values also:
SQL> select avg(nvl(comm,0)) from emp;
157.142857

4. **Sum():** It returns total from number data type column.
Eg: SQL> select sum(sal) from emp;
34625

5. **Count(*):** It counts number of rows in a table(including null values)
Eg: SQL> select count(*) from emp;
6. **Count(column name):** It counts number of not null values within a column.
SQL> select count(comm) from emp;
4
SQL> select count(distinct(dept no)) from emp;
3
GROUP BY: This clause is used to divide similar data items into set of logical groups.
Syntax: select columnname….. from tablename group by columnname;

Data: 26/3/15
Q) Write a query number of employees in each department from emp table using "group by"?
Ans: SQL> select deptno,count(*) from emp group by deptno;

DEPTNO	COUNT(*)
30	6
20	5
10	3

Q) Write a query to display number of employees in each job from emp table using group by?
Ans: SQL> select job, count(*) from emp group by job;

JOB	COUNT(*)
CLERK	4
SALESMAN	4
PRESIDENT	1
MANAGER	3
ANALYST	2

Q) Write a query to display deptno, minimum and maximum salary from emp using group by?
Ans: SQL> select deptno, min(sal), max(sal) from emp group by deptno;

DEPTNO	MIN(SAL)	MAX(SAL)
30	950	2850
20	800	3000

10 1300 5000

Note: In all database systems we can also use "group by" clauses without using "group" functions.

Eg: SQL> select deptno from emp group by deptno;

RULE: Other than group function columns specified after select those all columns must be used in after "group by". Otherwise oracle server returns an error "not a GROUP BY expression".

Eg: SQL> select deptno, max(sal), ename from emp group by deptno,ename;

ERROR at line 1:
ORA-00979: not a GROUP BY expression

SQL> select deptno from emp group by deptno,job;

"Reverse also possible".

Note: Whenever we are submitting "group by" clauses then database servers select the data from the table from based on the group by clause columns. These columns space database servers group the data in the result set, from this result only we are selecting the data using number of columns after select keyword.

Note: Whenever we are using group functions without using "group by" clause then database servers executes these group functions based on all values in a column. Whereas when we are using "group by" then these group functions executed each and every group wise in a table.

Note: Generally in all database systems we are not allowed to display group function with another columns to overcome this problem we must use "group by" clause.

Eg: step1: SQL> select sum(sal) from emp;

34627

Step2: SQL> select deptno, sum(sal) from emp;

Error: not a single-group group function

Solution: SQL> select deptno, sum(sal) from emp group by deptno;

```
DEPTNO   SUM(SAL)
------- ----------
    30       9400
    20      10875
    10       8750
```

Eg: SQL> select deptno, job, sum(sal), count(*) from emp group by deptno, job;

```
DEPTNO JOB        SUM(SAL)   COUNT(*)
------- --------- ---------- ----------
    20 CLERK          1900          2
    30 SALESMAN       5600          4
    20 MANAGER        2975          1
    30 CLERK           950          1
    10 PRESIDENT      5000          1
    30 MANAGER        2850          1
    10 CLERK          1300          1
    10 MANAGER        2450          1
    20 ANALYST        6000          2
```

Date: 27/3/15

Q) Write a query to display those departments having more than 3 employees from emp table?
Ans: SQL> select deptno,count(*) from emp group by deptno where count(*)>3;
Error: SQL commond not properly ended.
Solution: SQL> select deptno, coutn(*) from emp group by deptno having count(*)>3;

HAVING Clause: In all databse systems after "group by" clause we are not allowed to use "where" clause. In place of this one ansi/iso SQL provided another clause "having". Generally if we want to restrict groups after "group by" then we must use "having" clause. Generally in "where" clause we are not allowed to use group functions where as in having clause we can also use group functions.
Q) Write a query to display those departments having more than 10,000 sum(sal) from emp table?
And: SQL> select deptno, sum(sal) from emp group by deptno having sum(sal)>10,000;

Deptno	sum(sal)
10	13550
20	12675

Q) Write a query to display year, number of employees per year in which more than one employee was hired from emp table using "group by"?
And: SQL> select to_char(hiredate,'YYYY'), count(*) from emp group by(to_char(hiredate,'YYYY'));
OR
SQL> select to_char (hiredate,'YYYY') year, count(*) from emp
group by to_char(hiredate,'YYYY') having count(*)>1;

year	count(*)
1981	10
1982	2

Invisible columns:
Eg: SQL> select deptno, sum(sal) from emp
group by deptno
having sum(sal)>10000;
Here count(*) is invisible column.
SQL> select deptno, sum(sal) from emp
Group by deptno
Having count(*)>3;
Deptno sum(sal)

---
20 12675
30 8400

ORDER BY: This clause is used to arrange the data in sorting order along with "order by" clause we are using two keywords.

asc(ascending

desc(descending)

By default 'order by' clause having "Ascending order".
Syntax: SQL> select * from tablename order by columnname[asc/desc];
Eg: SQL> select sal from emp order by sal;
SQL> select deptno, sal from emp
order by deptno, sal;
Eg: SQL> select deptno, count(*) from emp
Where sal>1000
group by deptno
having count(*)>3
order by depnto desc;

Deptno count(*)
---
30 4
20 5

Date: 28/3/15

Note: In oracle we can also use column position in place of columnname within "order by" clause which is used to improve performance query.
Eg: SQL> select * from emp order by 6 desc;

ROLLUP, Cube:
Oracle 8i introduced rollup, cube clauses. These clauses are used along with group by clause only. These clauses are used to calculate subtotal, grand total automatically.
Syntax: select col1, col2,…….. from tablename
group by rollup(col1,col2,..);
Syntax: select col1, col2,…. From tablename group by cube(col1,col2,…);
Rollup is used to calculate subtotal value based on a single column where as if we want to calculate subtotal value based on number of column wise then we must use cube.
Eg: SQL> select deptno, job, sum(sal)from emp group by rollup(deptno, job);
SQL> select deptno, job, sum(sal), count(*) from emp group by cube(deptno,job);
Eg: SQL> select ename, sum(sal) from emp group by rollup(ename);

JOINS: Joins are used to retrieved data from multiple tables. In all databases when we are joingin "n" tables then we must use "n-1" joining conditions. Oracle having following types of joins.

1. Equi joing (or) Inner join
2. Non equi join
3. Self join
4. Outer join

These 4 joins are also called as "**8i joins**".

9i joins or ansi joins.

1. Inner join
2. Left outer join
3. Right outer join
4. Full outer join
5. Natural join

In oracle, we can also retrieve data from multiple tables without using join condition in this case oracle server internally uses cross join. But cross join is implemented based on Cartesian product that's why this internal join returns more duplicate data.
Eg: SQL> select ename, sal, dname, loc from emp, dept;

1. **Equi join (or) Inner join:** Based on equality condition we are retrieving data from multiple tables. Here joining conditional columns must belongs to same datatype. Generally, when tables having common columns then only we are using this "join".

Syntax: select col1, col2,….
from tablename1, tablename2
where tablename1.commoncolumnname= tablename2. Commoncolumnname;

tablename. Commoncolumnname is a join condition
Eg: SQL> select ename, sal, deptno, dname, loc from emp, dept where emp.deptno= dept.deptno;
Error: column ambiguously defined.
Solution: SQL> select ename, sal, dept.deptno, dname, loc from emp, dept where emp.deptno = dept.deptno;

Date: 30/3/15

NOTE: Generally, to avoid ambiguity in future we must specify every "column name" along with table name using '.' (Full stop or dot) operator.
NOTE: In all databases we can also create alias names for the table in "from" clause of the "join conditions". These alias names are also called as "Reference names". These reference names are used in select list, used in joining condition.

Syntax: from tablename aliasname1, tablename aliasname2.
These alias names must be different names.

USING ALIAS NAMES:

SQL> select ename, sal, d.deptno,dname, loc from emp e, dept d
Where e.deptno = d.deptno;

ENAME	SAL	DEPTNO	DNAME	LOC
CLARK	2450	10	ACCOUNTING	NEW YORK
KING	5000	10	ACCOUNTING	NEW YORK
MILLER	1300	10	ACCOUNTING	NEW YORK
JONES	2975	20	RESEARCH	DALLAS
FORD	3000	20	RESEARCH	DALLAS
ADAMS	1100	20	RESEARCH	DALLAS
SMITH	800	20	RESEARCH	DALLAS
SCOTT	3000	20	RESEARCH	DALLAS
WARD	1250	30	SALES	CHICAGO
TURNER	1500	30	SALES	CHICAGO
ALLEN	1600	30	SALES	CHICAGO
JAMES	950	30	SALES	CHICAGO
BLAKE	2850	30	SALES	CHICAGO
MARTIN	1250	30	SALES	CHICAGO

In the above query , here deptno("40") doesn't displayed if we are using dept.deptno also.
NOTE: In all databases always equi joins returns matching rows only.
Q) Write a query to display the employees who are working in the location Chicago from emp, dept tables using equi join?
Ans: SQL> select loc, ename from emp, dept where emp.deptno=dept.deptno group by loc having loc='chicago';
error: not a GROUP BY expression
Solution: SQL> select ename, loc from emp, dept
where emp.deptno=dept.deptno
and loc='chicago';

ENAME	LOC
WARD	CHICAGO
TURNER	CHICAGO
ALLEN	CHICAGO
JAMES	CHICAGO
BLAKE	CHICAGO
MARTIN	CHICAGO

Note: If we want to filter the data after joining condition then we must use "and" operator in 8i joins where as in 9i joins we are using either "and" operator or "where" clause also.
Q) Write a query to display dname, sum(sal) from emp, dept tables using equi join?
Ans: SQL> select dname,sum(sal) from emp e,dept d where e.deptno=d.deptno group by dname;

DNAME	SUM(SAL)
ACCOUNTING	8750
RESEARCH	10875
SALES	9400

SQL> select dept.deptno, dname, sum(sal) from emp, dept where emp.deptno=dept.deptno group by dept.deptno, dname;

DEPTNO	DNAME	SUM(SAL)
10	ACCOUNTING	8750
20	RESEARCH	10875
30	SALES	9400

Q) Write a query to display number of employees, min(sal), max(sal), sum(sal) in each location from emp, dept tables using equi join?

Ans: select loc, count(*), min(sal), max(sal),sum(sal) from emp, dept where emp.deptno=dept.deptno group by loc;

LOC	COUNT(*)	MIN(SAL)	MAX(SAL)	SUM(SAL)
NEW YORK	3	1300	5000	8750
CHICAGO	6	950	2850	9400
DALLAS	5	800	3000	10875

SQL> select dname, sum(sal) from emp e, dept d where e.deptno=d.deptno group by dname having dname;

DNAME	SUM(SAL)
RESEARCH	10875

SQL> select dname, sum(sal) from emp e, dept d where e.deptno=d.deptno group by rollup(dname);

DNAME	SUM(SAL)
ACCOUNTING	8750
RESEARCH	10875
SALES	9400
	29025

2. **<u>Non Equi Join</u>**: Based on other than equality condition(not equal to, >,<,>=,<=,between, in,….) we can retrieve data from multiple tables. In oracle, this join is also called as "Between….. and Join".

Date: 31/3/15

Eg: SQL> create table test1(deptno number(10),ename varchar2(20));
SQL> insert into test1 values(10,'a');
SQL> insert into test1 values(20,'b');
SQL> select * from test1;

DEPTNO	ENAME
10	a
20	b

SQL> create table test2(deptno number(10),dname varchar2(20));
SQL>insert into test2 values(10,'c');
SQL>insert into test2 values(20,'d');
SQL>insert into test2 values(30,'e');
SQL> select * from test2;

DEPTNO DNAME
---
10 c
20 d
30 e

SQL> select * from test1,test2 where test1.deptno>test2.deptno;

DEPTNO ENAME DEPTNO DNAME
---
20 b 10 c

NOTE: In oracle, we can also use non-equi join when tables doesnot have common columns.

Eg: select * from salgrade

SQL> select ename,sal,losal,hisal from emp,salgrade where sal between losal and hisal;

OR/

Where sal>=losal and sal<=hisal;

3.**SELF JOIN:** Joining a table to itself is called "self join". Here joining conditional columns must belongs to same datatype. Before we are using self join we must create alias names for the table in "from" clause.

Syntax: from tablename aliasname1,tablename aliasname2;

Generally, if we want to compare two different column values in a single table then we must use self-join. But here these two columns must belong to same datatype.

Q) Write a query to display employee names and their manager names from emp table using self join?

Ans: SQL> select e1.ename "employees", e2.ename "managers" from emp e1, emp e2
Where e1.mgr=e2.empno;

SQL> select e1.ename "employees", e1.mgr, e2.empno, e2.ename "managers" from emp e1, emp e2
Where e1.mgr=e2.empno;

Q) Write a query to display the employees who are getting more salary than their manager salary from emp table using "self join"?

Ans: SQL> select e1.ename "employees", e1.sal, e2.sal, e2.ename "manager"
from emp e1, emp e2
where e1.mgr=e2.empno
and e1.sal>e2.sal;

employees sal sal manager

SCOTT 3600 3375 JONES
FORD 3600 3375 JONES

Q) Write a query to display the employees who are joining before their managers from emp table using self join?

Ans: SQL> select e1.name "employees",e1.hiredate, e2.hiredate,e2.ename "manager"
from emp e1,emp e2
where e1.mgr=e2.empno

and e1.hiredate< e2. Hiredate;

Date: 1/4/15

When to use Alias names:

In all databases systems whenever we are refer same table more than one time foe wuery results then we must create "table alias" names. These alias names are also called as '"Reference names". These alias names internally behaves like a tables when query execution time. These alias names must be different names.

Syntax: from tablename alias name1, tablename aliasname2;

Q) Write a query to display the employees who are getting same salary as scott salary from emp table using table alias names?

Ans: SQL> select e2.ename,e1.sal from emp e1, emp e2 where e1.sal=e2.sal and e1.ename='SCOTT';

4. **OUTER JOIN:** This join is used to retrieve all rows from one table and matching rows from another table. Generally, using "equi join" we are retrieving matching rows only. If we want to retrieve non-matching rows also then we are using join operator(+).

Within joining condition of "equi join" this join is called "ORACLE 8i Outer join".

NOTE: Join operator can be used only one side at a time one joining condition.

Eg: SQL> select ename, sal, d.deptno,dname,loc from emp e, dept d where e.deptno(+)=d.deptno;

e.deptno-→ matching rows

d.deptno-→ all rows

9i JOINS or ANSI JOINS:

1. Inner join
2. Left outer join
3. Right outer join
4. Full outer join
5. Natural join

1. **Inner join:** This join also returns matching rows only. Whenever tables having common columns then only we are using this join. Here also joining conditional columns must belongs to some data type. Inner join (9i) performance is very high compare to oracle 8i equi join.

Eg: SQL> select ename, sal, d.deptno, dname, loc from emp e join dept d on e. deptno=d.deptno;

Q) Write a query to display the employees who are working in the location 'CHICAGO' from emp, dept tables using 9i inner join.
Ans: SQL> select ename, loc from emp e join dept d on e.deptno=d.deptno where loc='CHICAGO';

Eg: SQL> create table t1(a varchar2(10), b varchar2(10),c varchar2(10));
SQL > insert into t1 values('x','y','z');
SQL > insert into t1 values('p','q','r');
SQL> select * from t1;
SQL> create table t2(a varchar2(10), b varchar2(10));
SQL>insert into t2 values('x','y');
SQL.> select * from t2;

```
A          B      C          A      B
-------------------------    --------------------
X          Y      Z          X      Y
P          q      r
```

8i equi join:
SQL> select * from t1,t2 where t1.a=t2.a and t1.b=t2.b;
9i inner join:
SQL> select * from t1 join t2 on t1.a=t2.a where t1.b=t2.b;
Output:

```
A          B      C          A      B
-------------------------    --------------------
X          Y      Z          X      Y
```

Date: 2/4/15

Using clause:
In 9i joins we can also use "USING" clauses in place of "ON" clause. "Using" clause performance is very high compare to "ON" clause and also when we are using "USING" clause then oracle server automatically returning common columns on time only.
Syntax: select * from tablename1 join tablename2 using (common columnnames);
Eg: SQL> select * from t1 join t2
using(a,b);
Output:

```
A      B      C
----------------
X      y      z
```

Note: whenever we are using "Using" clause then we are not allowed to use alias name or joining conditional columns.
Eg: Select ename,sal,deptno,dname,loc

from emp e join dept d
using (deptno);

2. Left Outer Join:

This join always reuturns all rows from left side table and matching rows from reight side table and also returns "NULL" values in place of non-matching rows in another table.

Eg: select * from t1;

```
A       B       C
-----------------
X       y       z
P       q       r
SQL> select * from t2;


A       B
---------
X       y
S       t
SQL> select * from t1 left outer join t2 on t1.a=t2.a and t1.b=t2.b;
A         B     C              A       B
-------------------       ---------------------
X         Y     Z              X       Y
P         q     r            (null)   (null)
```

3. Right Outer Join:

This Join returns all rows from right side table and matching rows from left side table and also returns null values in place of "non-matching" rows in another table.

4. Full Outer Join:

This join returns all data from all the tables it is the combination of left and right outer joins. This join always returns matching and non-matching rows. And also this join returns null values in place of non-matching rows.

Eg: SQL> select * from t1 full outer join t2 on t1.a=t2.a and t1.b=t2.b;

```
A       B       C              A       B
------------------          -------------
X       y       z              x       y
P       q       r             null     null
Null    null    null           s       t
```

5. **Natural Join:** This join also returns matching rows only. When we are using "Natural Join" then we are not allowed to used to use joining condition. In this case oracle server only internally uses joining condition. But here resource tables must contain common column name.

Syntax: select * from tablename1 natural join tablename2.

Note: Natural Join performance is very high compared to inner join.
Note: This join internally uses "USING" clause. That's why this join also returns common columns one time only.
Eg: SQL> select * from t1 natural join t2;
Output:

```
A       B       C
----------------
X       y       z
```

Note: When we are using "Natural" join also then we are not allowed to use alias name on joining conditional column because natural join internally uses "USING" clause.
Eg: SQL> select ename,sal,deptno,dname,loc from emp e natural join dept d;

CROSS JOIN:
Eg: select ename,sal,dname, loc from emp cross join dept;
Joining more than 2-tables.(example 3 tables):
8i JOINS
Syntax: SQL> select col1,col2,….. from table1,table2,table3
Where table1.commoncolname=table2.commcolname
and table2. Commcolname=table3.commcolname;
9i JOIN:
Syntax: SQL> select col1,col2,… from table1 join table2
on table1.commcolname=table2.commcolname join table3
on table2.commcolname=table3.commcolname;

CONSTRAINTS:
Constraints are used to prevents or stops invalid data entry into our tables. Generally constraints are created on table columns. Oracle server having following types of constraints.

1. Not null
2. Unique
3. Primary key
4. Foreign key
5. Check

These are called "Constraints (or) Constraint types".

Date: 3/4/15

All the above constraints are created in two ways:

1. Using column level
2. Using table level

1. **Using Column level:** In this methods we are defining constraints on individual columns. That is whenever we are defining the column then only immediately we are specifying constraint type.

Syntax: create table tablename(col1 datatype(size) constraint type, col2 datatype(size) constraint type,…..);

2. **Using Table level:** In this method we are defining constraints on group of columns i.e., in this method first we are defining all columns and last only we are specifying constraint type along with group of columns.

Syntax: create table tablename(col1 datatype(size),col2 datatype(size),…. , constrainttype (col1,col2,….));

1. **Not NULL:** This constraint doesn't support table level. This constraint doesn't accepts null values. But it will accepts duplicate values.

Column level: SQL> create table t1(sno number(10) not null, name varchar2(10));
Testing: SQL> insert into t1 values(null,'abc');
ORA-1400: cannot insert null into SNO;
Table level: SQL> create table t1(sno number(10), name varchar2(10), not null(sno,name));

2. **UNIQUE:** This constraint is defined on column level, table level. This constraint doesn't accepts duplicate values. But it will accepts null values.

Note: Whenever we are creating unique constraints then oracle server internally automatically creates b-tree index on those columns.
Column level: SQL> create table t2(sno number(10) unique, name varchar2(10));
Table level: SQL> create table t2(sno number(10), name varchar2(10), unique(sno,name));
Table created
Eg: SQL> insert into t3(….)

SNO	NAME
1	murali
1	abc

SQL> select * from t3;
Error: ORA-00001: Unique constraint violated.

3**. Primary Key:** Primary key "Uniquely identifying a record in a table". There can be only one primary key in a table and also primary key doesn't accepts duplicate null values.
Note: Whenever we are creating primary key then oracle server automatically create an "b- tree" indexes on those columns.
Column level: SQL> create table t4(sno number(10) primary key, name varchar2(10));
Table created
Table level: SQL> create table t4(sno number(10), name varchar2(10), primary key(sno,name));

This is also called as "Composite Primary Key" i.e., it is the combination of columns as a "single primary key".

4. **Foreign Key:** If we want to establishes relationship between tables then we are using "Referential Integrity Constraints" (Foreign Key). One table foreign key must belongs to another table "primary key". Here these two columns must belong to same data types. Always foreign key values based on "primary key" values only. Generally, primary key doesn't accepts duplicate, null, values where as foreign key accepts duplicate, null values.

Date: 4/4/15

Column level ("References"):
Syntax: create table tablename(col1 datatype(size) references master tablename(primary columnname),….);
Eg: SQL> create table h4(sno number(10) references t4);
Or
SQL> create table g4(x number((10) references t4(sno));

Table Level: (Foreign key references)
Syntax: create table tablename(col1 datatype(size),col2 datatype(size),…., foreign key(col1,col2));
Eg: SQL> create table h5(sno number(10), name varchar2(10), foreign key(sno,name) references t5);
Whenever we are establishing relationship between tables then oracle violates following two rules:-
1. Deletion in "MASTER" table
2. Insertion in "CHILD" table

1. **Deletion in "Master" table:** When we try to delete master table record if child table records are exist, then oracle server returns an error "ORA-2292" to overcome this problem first we are deleting child table records in "child table" and then only we are deleting appropriate "master table" records within master table. Otherwise using "on delete cascade" clause.
   ON DELETE CASCADE: When we are using this clause in child table then if we are deleting a "master table" record within master table then oracle server automatically deletes "master table" records in master table and also deletes corresponding records in "child table".
   Syntax: create table tablename(col1 datatype(size) references mastertablename (primary key colname) on delete cascade,…..);
   Eg: SQL> create table mas(sno number(10) primary key);
   SQL> insert into mas values(……..);
   SQL> select * from mas;
   SNO

1
2
3

SQL> create table child(sno number(10) references mas(sno) on delete cascade);
SQL> insert into child values(……..);
SQL> select * from child;
SNO

1
1
2
2
3
3

Testing : (deletion in master table)
SQL> delete from mas where sno=1;
1 row deleted
SQL> select * from mas:
SQL> select * from child;

SNO	SNO
2	2
3	2
	3
	3

Oracle also supports another clause along with foreign key "ON DELETE SET NULL".

ON DELETE SET NULL:

Whenever we are using this clause the "child table" then if we deleting a master table record then oracle server automatically deletes primary key value in master table and also corresponding foreign key values are automatically set to null in child table.

Syntax:

create table tablename(col datatype(size)
reference master tablename(primary key colname) on delete set null,……);

2. **INSERTION IN CHILD TABLE:** In Oracle, when we are trying to insert other than primary key values into Foreign key then oracle server returns an error "ORA-2291". Because in all database systems always foreign key values are based on primary key values only.

Eg: SQL> insert into child values(5);
ORA-2291: Intergrity constraint violated-parent key not found.
Solution: SQL> insert into mas values(5);
SQL> select * from mas;

SQL> insert into child values(5);
SQL> select * from mas;

Example based on dept(master table), emp(child table):

1) Deletion in Master table:

SQL> delete from dept where deptno=10;
ORA-2292: integrity constraint violated- child record foun.

2) Insertion in child table:

SQL> insert into emp(empno,deptno) values(5,90);
ORA-2291: Integrity constraint violated – parent key not found

Date: 6/4/15

In oracle, we are not allowed to define logical condtions using "SYSDATE" within check constraint.

COLUMN LEVEL:

Syntax: create table tablename(col1 datatype(size) check(logical condition),col2 datatype(size),…..);
Eg: create table t6(sal number(10) check (sal>5000));
SQL> insert into t6 values(3000);
Error: check constraint(scott.syst_c005513) violated
SQL> insert into t6 values(9000);
SQL> select * from t6;
SAL

9000
Eg: create table t7(name varchar2(20) check(name=upper(name)));
SQL> insert into t7 values('murali');
Error: check constraint(scott.syst_c00514) violated
SQL> insert into t7 values ('MURALI');
SQL> select * from t7;
NAME

MURALI

Table level:

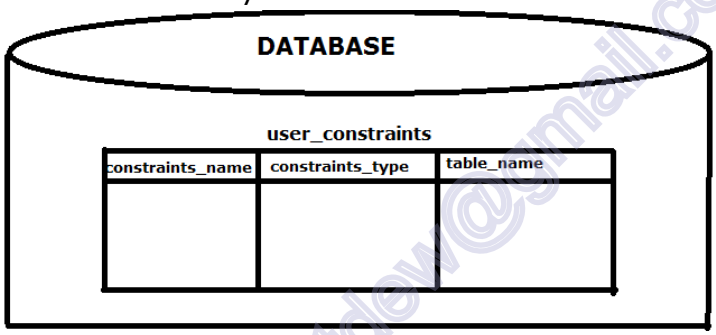
Eg: SQL> create table t8(name varchar2(20), sal number(10), check(name=upper(name) and sal>5000));

Assign User defined names to Constraints (OR) User defined Constraint Names:

In all database systems whenever we are creating constraints then database servers internally automatically generates an unique identification number for indentifying a constraint uniquely.

In Oracle also whenever we are creating constraints then oracle server automatically generates an unique identification number in the format of "SYS_Cn". This is called "Predefined Constraint Name". In place of this one we can also create our own name using "Constraint Keyword". This is called "Used defined Constraint Name".

Syntax: Constraint user defined name constraint_type;
Here, constraint-> keyword
Constraint_type-> Not null, unique, primary key, foreign key, check
User defined name->constraint name
Eg 1: (Predefined Constraint Name)
SQL> create table t10(sno number(10) primary key;
Testing:
SQL> insert into t10 values(1);
SQL> insert into t10 values(1);
Error: unique constraint (SCOTT.SYS_C005516) violated
Eg 2: User defined Constraint Name
SQL> create table t11 (sno number(10) constraint p_abc primary key);
Testing:
SQL> insert into t11 values(1);
SQL> insert into t11 values(1);
Error: unique constraints (SCOTT.P_ABC) violated
Data Dictionaries:
Whenever we are installing oracle server then oracle server automatically creates some read only tables. These read only tables are also called as "Data Dictionaries".



Note 1: In Oracle, all constraints information stored under "user_constraint" data dictionary.
Eg: SQL> desc user_constraints;
SQL> select CONSTRAINT_NAME, CONSTRAINT_TYPE from USER_CONSTRAINTS where table_name='EMP';
Output:

CONSTRAINT_NAME	CONSTRAINT_TYPE
PK_EMP	P
FK_DEPTNO	R

➔ Table_name='EMP'

Here, EMP is capital letter
NOTE 2: In Oracle, if you want to view column names along with constraint names then we are using "user_cons_columns".
Eg: SQL> desc user_cons_columns;

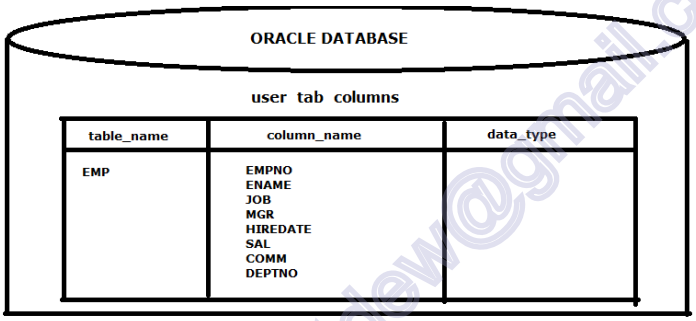
SQL> select constraint_name,column_name from user_cons_columns
where table_name='EMP';
Output:

CONSTRAINT_NAME	COLUMN_NAME
FK_DEPTNO	DEPTNO
PK_EMPNO	EMPNO

NOTE 3: In Oracle, if you want to view logical conditions of the check constraint then we are using "search_condition" from "user_constraints" data dictionary.
Eg: SQL> desc user_constraints;
SQL> select search_condition from user_constraints
where table_name='T7';
SEARCH_CONSTRAINTS
---
name=upper(name)

Date: 7/4/15

In Oracle, all columns information stored under "user_tab_columns" data dictionary.



SQL> desc user_tab_columns;
SQL> select column_name from user_tab_columns
where table_name='EMP';
Q) Write a query to display number of columns from emp table?
Ans: SQL> select count(*) from user_tab_columns where table-name='EMP';

DEFAULT: In Oracle, we can also provide "default" values for a column using "DEFAULT" constraint.
Syntax: columnname datatype(size) default actual value;
Eg: SQL> create table b1(name varchar(10), sal number(10) default 5000);
SQL> insert into b1(name) values ('abc');
SQL> select * from b1;
NOTE: In Oracle, if you want to view default values for a column then we are using "data_default" property (predefined column) from "user_tab_columns" data dictionary.
Eg: SQL> desc user_tab_columns;
SQL> select column_name,data_default from user_tab_columns where table_name='B1';
COLUMN_NAME DATA_DEFAULT

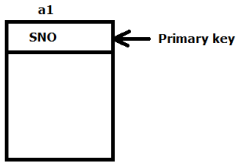
SAL 5000

Adding (or) Dropping constraints on existing table:

In Oracle, we can add or drop constraints on existing table using "ALTER" command.

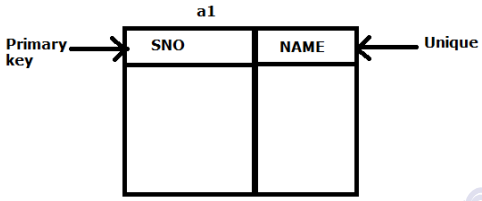
NOTE 1: If we want to add constraints on existing table or existing column then we are using "table level syntax method".

Eg: SQL> alter table a1
add primary key(sno);



NOTE 2: If we want to add a new column along with constraint then we are using "Column level syntax method".

Eg: SQL> alter table a1 add name varchar2(10) unique;



Q) Add foreign key in existing a2 references a1?

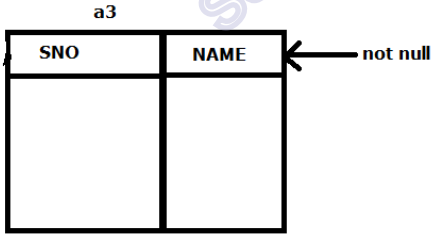
Ans: SQL> alter table a2 add foreign key(sno) references a1(sno);

NOTE 3: If we want to add "NOT NULL" constraints into existing table or existing column then we are using "alter with modify".

Syntax: alter table tablename modify columnname not null;

Eg: SQL> create table a3(sno number(10));

SQL> alter table a3 modify sno not null;



Q) Write a query to add new column with not null into a3?

Ans: SQL> alter table a3 add name varchar2(10) not null;

NOTE: In Oracle, whenever we are copying a table from another table constraints are never copied. But in all database systems "not null" constraints only copied.

Eg: SQL> create table a4 as select * from a3;

"not null is copied"

SQL> desc a4;

Eg: SQL> create table dept1 as select * from dept;
SQL> select * from dept1;
SQL> alter table dept1 add primary key(deptno);
SQL> create table emp1 as select * from emp;
SQL> select * from emp1;
SQL> alter table emp1 add primary key(empno);
SQL> alter table emp1 add foreign key deptno references emp1(deptno) on delete cascade;
Dropping Constraints: By dropping "Constraint name" two methods for dropping constraints.
Method 1: Alter table tablename drop constraint constraint name;
OR
Method 2:
A. Alter table tablename drop primary key;
B. Alter table tablename drop unique(col1,col2,….);

Date: 8/4/15

SQL> create table test(sno number(10) primary key);
SQL> alter table test drop primary key;
NOTE: In Oracle, if we want to drop primary key along with referenced foreign keys then we are using "Cascade" clause along with "Alter drop".
Syntax: alter table tablename drop primary key cascade;
SQL> alter table a1 drop primary key;
Error: this unique/primary key is referenced by some foreign key.
Solution: SQL> alter table a1 drop primary key cascade;
Eg: Not Null constraints for dropping.
SQL> desc a3;
SQL> desc user_cons_columns;
SQL> select column_name,constraint_name from user_cons_columns where table_name='A3";

```
COLUMN_NAME          CONSTRAINT_NAME
------------------------------------------------------
NAME                 SYS_C005526
SNO                  SYS_C005525
SQL> alter table a3 drop constraint SYS_C005526;
SQL> alter table a3 drop constraint SYS_C005525;
SQL> desc a3;
```

SUB QUERY (or) NESTED QUERY:

Query within another query is called "Sub Query" or "Nested Query". Sub query's are used to retrieve data from "single" or "multiple" tables based on "More than one step process".
All relational data base having two types of sub query's:

1. Non- Correlated sub query's
2. Correlated sub query's

In Non-correlated sub query's "child" query is executed first then only "parent" query is executed, whereas in correlated sub query's "parent" query is executed first then only "child" query is executed.

1. **Non-Correlated sub query's:** These sub query's mainly consist of two parts.
   I. Child query
   II. Parent query
1. **Child Query:** A query which provides values to the another query is called "Child Query or Inner Query".
2. **Parent Query:** A query which receives values from another query is called "Parent Query or Outer Query".

In non-correlated sub query maximum limit is up to "255" query's.

NON-CORRELATED SUB QUERY:

- Single row sub query's
- Multiple row sub query's
- Multiple column sub query's
- Inline view or sub query's are using in "from" clause.

Q) Write a query to display the employees who are getting more salary than the avg(sal) from emp?
Ans: SQL> select ename, sal from emp where sal>(select avg(sal) from emp);
This is a "Single row sub query" because here child query returns single value. In Single row sub query's we are using =,<,>,<=,>=, between operators.
Execution:
Step 1: SQL> select avg(sal) from emp;
2830.35714
Step 2: SQL> select * from emp where sal>2830.35714

Q) Write a query to display the employees who are working in sales department from emp, dept tables.
Ans: SQL> select ename from emp where deptno=(select deptno from dept where dname='SALES');

Date: 9/4/15

Q) Write a query to display the employees who are working in "SMITH" department number from emp table?
Ans: SQL> select ename from emp where deptno=(select deptno from emp where ename='SMITH');

Q) Write a query to display senior most employee details from emp table?
Ans: SQL> select * from emp where hiredate=(select min(hiredate) from emp);

Q) Write a query to display highest sal details emp table?
Ans: SQL> select * from emp where sal=(select max (sal) from emp);

Q) Write a query to display highest paid emp department, dname from emp, dept tables?
Ans: SQL> select dname from dept where deptno=(select deptno from emp where sal=(select sal(max) from emp));

Q) Write a query to display second max(sal) from emp table?
Ans: SQL> select max(sal) from emp where sal<(select max(sal) from emp);

Q) Write a query to display second highest sal employee details from emp table?
Ans: SQL> select * from emp where sal=(select max(sal) from emp where sal<(select max(sal) from emp));

Q) Write a query to display the employees who are working under 'BLAKE' from emp table using 'empno, mgr' columns ?
Ans: SQL> select * from emp where mgr= (select empno from emp where ename='BLAKE');

Q) Write a query to display lowest average salary job from emp tables?
Ans: SQL> select job, avg(sal) from emp
group by job
having avg(sal)=(select min(avg(sal)) from emp);
Error: nested group function without GROUP BY

NOTE: In all database systems whenever child query contains nested group functions then we must use "Group BY" clause within child query.
Ans: SQL> select job, avg(sal) from emp group by job
having avg(sal)= (select min(avg(sal)) from emp group by job);

JOB	AVG(SAL)
SALESMAN	1050

Q) Write a query to display lowest average salary job from emp table where deptno less than 30?
Ans: SQL> select deptno, min(sal) from emp
group by deptno
having min(sal)>(select min(sal) from emp where deptno= 30);

DEPTNO	MIN(SAL)
10	4400
20	1700

Date: 10/4/15

Q) Write a query to display job, avg(sal) of the employees whose job avg(sal) more than the clerk's job avg(sal) from emp table?
Ans: SQL> select avg(sal), job from emp group by job
having avg(sal) >(select avg(sal) from emp where job='CLERK');

NOTE: In all databases, whenever we are using sub query's always database servers returns parent query table columns. i.e., we are unable to use child query table columns into parent query to overcome this problem we must use joins within "Parent Query".

Q) Write a query to display the employees who are working sales department from emp, dept tables?
Ans: SQL> select ename, dname from emp e, dept d where e.deptno=d.deptno and d.deptno=(select deptno from dept where dname='SALES');

ENAME	DNAME
ALLEN	SALES
WARD	SALES
MARTIN	SALES…….

Q) Write a query to display the employees who are getting more salary than the 'BLAKE' salary from emp table?
Ans: SQL> select * from emp where sal> (select sal from emp where ename='BLAKE');

MULTIPLE ROW SUB QUERY'S:
Whenever child query returns multiple values those types of sub query's are called 'Multiple row Sub Query's'.
Q) Write a query to display the employee details who are getting max(sal) in each department from emp table?
Ans: SQL> select * from emp where sal= (select max(sal) from emp group by deptno);
Error: single row sub query returns more than one row.

This is an multiple row sub query because here query returns multiple values. In multiple row sub query's we are using 'in', 'all', 'any' operators.

NOTE: In all database systems we can also use 'in' operator in single row sub query's.

Solution: SQL> select * from emp where sal in(select max(sal) from emp group by deptno); OR
SQL> select ename, sal, deptno from emp where sal in(select max(sal) from emp group by deptno);

Q) Write a query to display the employees who are working in sales or research department from emp, dept?
Ans: SQL> select ename, deptno from emp where deptno in(select deptno from dept where dname='SALES' or dname='RESEARCH');

Q) Write a query to display the employees who are working as "supervisors" (managers) from emp table using empno, mgr?
Ans: SQL> select * from emp where empno in(select mgr from emp);

Q) Write a query to display the employees who are not working as "supervisors" (managers) from emp using empno, mgr?
Ans: SQL> select * from emp where empno not in(select nvl(mgr,0) from emp);

Date: 11/4/15

TOP N Analysis:

1. Inline view
2. Rownum

1. **Inline view:** Oracle 7.2 introduced "Inline views". In Inline views we are using "sub query's" in place of tablename with in parent query.

Syntax: select * from (select statement);
Generally, in all database systems we are not allowed to use "Order By" clause in child query's. To overcome this problem oracle 7.2 introduced sub query's in from "from" clause. These type of query's are also called as " INLINE QUERY's".
Generally, in oracle we are not allowed to use "Alias name" in "where" condition. To overcome this problem also we are using "Inline views". Whenever we are using "Inline views" this alias name automatically behaves like column name, then only this alias name we can use in "where" condition.
Eg: SQL> select ename, sal, sal*12 annsal from emp where annsal>30000;
Error: ANNSAL: Invalid Idenfier.
Solution: SQL> select * from(select ename,sal,sal*12 annsal from emp) where annsal>30000;

2. **ROWNUM:** (Magic Column) Rownum is an Psuedo column(generalised column) it behaves like a table column. Whenever we are installing oracle server then automatically some psuedo columns are created in database. These "Pseudo Columns" belongs to all tables within database. Rownum psuedo column is used to filter the data within a table.

Rownum is a psuedo column which automatically assigns numbers to each row in a table at the time of selection.

SQL> select rownum, ename from emp;
ROWNUM ENAME

```
------------------------------
1                    SMITH
2                    ALLEN
3                    WARD
```

Generally, rownum having temporarily values.
Eg: SQL> select rownum, ename from emp where deptno=10;

```
ROWNUM               ENAME
------------------------------
1                    CLARK
2                    KING
3                    XXX
```

Q) Write a query to display first row from emp table using rownum?
Ans: SQL> select * from emp where rownum=1;

Q) Write a query to display second row from emp table using rownum?
Ans: SQL> select * from emp where rownum=2;
No rows selected
Generally, rownum doesn't work with more than "1" +ve integer, i.e., it works with <, <= operators.

Q) Write a query to display first five rows from emp table using rownum?
Ans: SQL> select * from emp where rownum<=5;

Q) Write a query to display first five highest salary employees from emp using rownum?
Ans: SQL> select * from(select * from emp order by sal desc) where rownum<=5;

Q) Write a query to display 5th highest salary employee from emp table using rownum?
Ans: SQL> select * from(select * from emp order by sal desc) where rownum=5
minus
select * from(select * from emp order by sal desc) where rownum<=4;

Date: 16/4/15

Q) Write a query to display rows between 1 to 5 from emp table using "Rownum"?
Ans: SQL> select * from emp where rownum between 1 and 5;

Q) Write a query to display rows between to 5 to 9 from table?
Ans: SQL> select * from emp where rownum between 5 and 9;
No rows selected

Solution: SQL> select * from emp where rownum <=9
minus
select * from emp where rownum<=5;

Q) Write a query to display second record from emp using rownum?
Ans: SQL> select * from emp where rownum<=2
minus
select * from emp where rownum<=1;

Q) Write a query to display last two records from emp table using rownum?
Ans: SQL> select * from emp where rownum<=14
minus
select * from emp where rownum<=12;
Solution: SQL> select * from emp minus select * from emp where rownum<=
(select count(*)-2 from emp);

➔ SQL> select * from emp where rownum>1;
No rows selected
➔ SQL> select * from emp where rownum>=1;
Rows selected

NOTE: Whenever we are using alias name for rownum in "INLINE VIEWS" then that alias name works with all SQL operators because that alias name behaves like a table column.

Q) Write a query to display second record from emp table using "rownum" alias name?
Ans: SQL> select * from(select rownum r, ename, job, sal from emp) where r=2;

R	ENAME	JOB	SAL
2	ALLEN	SALESMAN	1400

NOTE: If we want to display all the columns from the table using rownum alias name then we must use table.star(*) i.e., {e.*} along with rownum alias name after select. Otherwise create an alias name for the table and use aliasname.* after select along with rownum alias name.
SQL> select * from(select rownum r, emp.* from emp) where r=2;
OR
SQL> select * from(select rownum r, e.* from emp e) where r=2;

Q) Write a query to display second, third, fifth, seventh, eighth rows from emp table using rownum alias name?
Ans: SQL> select * from (select rownum r, e.* from emp e) where r in(2,3,5,7,8);

Q) Write a query to display records from 5 to 9 from emp table using rownum alias name?
Ans: SQL> select * from(select rownum r, emp.* from emp) where r between 5 and 9;

Q) Write a query to display first and last records from emp table using rownum alias name?
Ans: SQL> select * from(select rownum r, emp.* from emp) where r=1 or r=(select count(*) from emp);

Q) Write a query to display even number of records from emp table using rownum alias name?
Ans: SQL> select * from(select rownum r, emp.* from emp) where r in(select mod(r,2)=0 from emp);
Solution: SQL> select * from(select rownum r, emp.* from emp) where mod(r,2)=0;

Q) Write a query to display skip first 5 rows from emp table using rownum alias name?
Ans: SQL> select * from (select rownum r, emp.* from emp) where r>5;

ANALYTICAL FUNCTIONS are used in INLINE VIEWS.
ANALYTICAL FUNCTIONS:
Oracle 8i introduced "Analytical Functions". These functions are similar to "Group functions" but group functions reduces number of rows within the group. Where as "Analytical Functios" doesn't reduces number of rows with in the group. That is when we are using analytical functions database servers executes for each and every row with in the group.
Using GROUP function.
Eg: SQL> select deptno, avg(sal) from emp group by deptno;

DEPTNO	AVG(SAL)
10	5883.3333
20	2935
30	1666.66667

Using Analytical Functions
SQL> select deptno, avg(sal) over(partition by deptno) from emp;

Oracle having following Analytical functions.
1. Row_number()
2. Rank()
3. Dense_rank()

These three analytical functions automatically assigns "ranks" to each row in a table either group wise or rows wise in a table.

Date : 17/4/15

Analytical Function:

Syntax:
Analytical function name over(partition by columnname order by columnname[asc/desc]);
Row_number() analytical function automatically assigns different rank numbers when values are same. Where as **rank()** and **dense_rank()** analytical functions automatically assigns same rank numbers when values are same. And also **rank()** skips next consecutive rank numbers where as **dense_rank()** doesn't skips next consecutive rank numbers.

Eg: row_number():

DEPTNO	ENAME	SAL	R
20	SCOTT	3800	1
20	FORD	3800	2
20	JONES	3475	3

Eg: rank():

DEPTNO	ENAME	SAL	R
20	SCOTT	3800	1
20	FORD	3800	1
20	JONES	3475	3

Eg: dense_rank():

DEPTNO	ENAME	SAL	R
20	SCOTT	3800	1
20	FORD	3800	1
20	JONES	3475	2

Q) Write a query to display highest salaries to lowest salaries in each department from emp table using row_number() analytical function?
Ans: SQL> select deptno, ename,sal, row_number over(partition by deptno order by sal desc) r from emp;

Q) Write a query to display 2nd highest salary employee in each department from emp table using analytical function?
Ans: SQL> select * from(select deptno, ename, sal, dense_rank() over(partition by deptno order by sal desc)r from emp) where r=2;

Q) Write a query to display 5th highest salary employee from emp table using analytical function?
Ans: SQL> select * from(select deptno, ename, sal, dense_rank() over(order by sal desc)r from emp) where r=5;

NOTE: In analytical functions "Partition By" clause is an optional clause.

Q) Write a query to display nth highest salary employee from emp table using analytical function?
Ans: SQL> select * from(select deptno, ename, sal, dense_rank() over(order by sal desc)r from emp) where r=&n;

Output: Enter value for n: 1

DEPTNO	ENAME	SAL	R
10	KING	7900	1

ROWID:
Rowid is an pseudo column, it behaves like a table column whenever we are inserting data into table then oracle server automatically generates an unique identification number for identifying a record uniquely. This is called "ROWID". Generally, using rowid's we can retrieve data very quickly from database.
Generally, rownum having temporary values where as rowed having fixed values.
Eg: SQL> select rownum, rowid, ename from emp;

Eg: SQL> select rownum, rowid, ename from emp where deptno=10;
In Oracle, by default rowid's are ascending order.

Q) Write a query to display second record from emp table using analytical function, rowid?
Ans: select * from(select deptno, ename, sal, row_number() over(order by rowid)r from emp) where r=2;

Q) Write a query to display each department second row from emp table using analytical function, rowid?
Ans: SQL> select * from(select deptno, ename, sal, row_number() over (partition by deptno order by rowid) r from emp where r=2);

Q) Write a query to display last two records from emp table using analytical function, rowid?
Ans: SQL> select * from(select deptno, ename, sal, row_number() over (order by rowid desc) r from emp)where r<=2;

Date: 18/4/15

We can also use max(), min() functions in "rowid's".
Eg: SQL> select max(rowid) from emp;
SQL> select max(rowid) from emp;

Q) Write a query to display first row from emp table using rowid?
Ans: SQL> select * from emp where rowid=(select min(rowid) from emp);

Q) Write a query to display last record from emp table using rowid?
Ans: SQL> select * from emp where rowid=(select max(rowid) from emp);

Q) Write a query to display duplicate data from the following table?
SQL> create table test(sno number(10));
SQL> insert into test values(&sno);
Enter values for sno: 10
Enter values for sno: 10
Enter values for sno: 10
Enter values for sno: 20
Enter values for sno: 20
Enter values for sno: 30
Enter values for sno: 40
SQL> select * from test;

```
SNO
-----
10
10
10
20
20
30
40
```

SQL> select sno, count(*) from test group by sno having count(*)>1;

```
SNO         COUNT(*)
--------------------------
10          3
20          2
```

Generally, rowid's are used in subquery's for deleting duplicate rows in a table.
Q) Write a query to delete duplicate rows except one row in each group from a table using rowid?
Ans: SQL> delete from test where rowid not in(select min(rowid) from test group by sno);
SQL> select * from test;

```
SNO
-----
10
20
30
40
```

Multiple Column Sub Query's:

In all database we can also compare multiple columns of the child query with the multiple columns of the parent query these type of query's are also called as "Multiple Column Sub Query's".

In all database systems when we are using multiple column sub query's then we must specify parent query where conditional column's within paranthesis "()".

Syntax: select * (or) col1, col2,…. from tablename where(col1, col2,…….) in (select col1, col2,…… from tablename where condition).

Q) Write a query to display the employees whose job, mgr match with the job, mgr of the employee 'SCOTT' from emp table?

Ans: SQL> select * from emp where(job,mgr) in(select job, mgr from emp where ename= 'SCOTT');

Q) Write a query to display highest salary employees in each department from emp table using multiple row sub querys?

Ans: SQL> select deptno, sal, ename from emp where sal in (select max(sal) from emp group by deptno);

DPETNO	SAL	ENAME
10	7900	KING
20	3800	FORD
20	3450	SCOTT
30	3450	BLAKE

Solution:

Q) Write a query to display the employees who are getting maximum salary in each department from emp table using multiple column sub query?

Ans: SQL> select deptno, sal, ename from emp where(deptno, sal) in (select deptno, max(sal) from emp group by deptno);

DPETNO	SAL	ENAME
10	7900	KING
20	3800	FORD
30	3450	BLAKE

Q) Write a query to display senior most employees in each job from emp table using multiple column sub query?

Ans: SQL> select hiredate, ename, job from emp where (hiredate, job) in (select min(hiredate), job from emp group by job);

Q) Write a query to display ename, dname, sal of the employees whose sal, comm match with the sal, comm. Of the employees working in the location 'DALLAS'?

Ans: SQL> select ename, dname, sal from emp e, dept d where e.deptno=d.deptno and (sal,nvl(comm,0)) in (select sal, nvl(comm,0) from emp where deptno=(select deptno from dept where loc='DALLAS'));

Date: 20/4/15

Q) Write a query to display the employee who are getting more salary than the highest paid employee in 20th department from emp table?
Ans: SQL> select * from emp where sal> (select max(sal) from emp where deptno=20);

Q) Write a query to display the employees who are getting more than the lowest paid emp in 10th dept?
Ans: SQL> select * from emp where sal> (select min(sal) from emp where deptno=10);

Whenever resouce table having large amount of data and also child query contains max() or min() functions and also when we are comparing then those type of sybquery's degrades performance of the application. To overcome this problem to improve performance of these type of query's ANSI/ISO SQL. These are "all","any". These sub query's special operators are used along with relational operators in parent query "where" condition.

SQL> select * from emp where sal> all(select sal from emp where deptno=20);
SQL> select * from emp where sal> any(select sal from emp where deptno=10);

Sub Query Special Operators are used in Multiple Row Sub Query:
When child query returns multiple values then those type sub query are called "Multiple Row Sub Query's". In all database systems "in", "all", "any" operators used in "Multiple Row Sub Query's".
In-> It returns same values in the list(child query)
All-> It satisfies all values in the list
Any-> It satisfies any value in the list
IN-> =ANY
Q) Write a query to display the employees who are getting more than the all salaries of the clerks from emp table using sub query special operator?
Ans: SQL> select * from emp where sal> all(select sal from emp where job='CLERK');

➔ To display the result in order(asc/desc)

SQL> select * from emp where sal> all(select sal from emp where job='CLERK') order by sal desc;

Note: whenever we are using sub query special operator "OR" internally database servers uses logical operator "AND"
Eg: SQL> select * from emp where deptno>all(10,20);
Output: 30

Note: Whenever we are using sub query special operator "ANY" internally database servers logically operators "OR"
Eg: SQL> select * from emp where deptno>any(10,20);
Output: 20
30
NOTE: "=any" is also same as "in" operator but "<>any" is not same as "not in" operator.
Eg: SQL> select * from emp where deptno not in(10,20)
Output: 30

SQL> select * from emp where deptno <> any(10,20);
Output: 10
20
30
NOTE: "<>all" is also same as "not in" operator.
Eg: SQL> select * from emp where deptno not in(10,20);
SQL> select * from emp where deptno <>all(10,20);

CORRELATED SUB QUERY'S:

In Non-Correlated sub query's 'child query' is executed first then only 'parent query' is executed. Where as in correlated sub query's 'parent query' is executed first then only 'child query' is executed. Whenever we are submitting correlated sub query then database servers get a candidate row from the 'parent query' table and then control passed into 'child query' where condition and then based on evaluation value it compare with 'parent query' where condition. Generally, in non-correlated sub query 'child query' is executed only once per 'parent query' table where as in correlated sub query.
'Child query' is executed for each row for 'parent query' table. In all database systems in correlated sub query we must create alias name for 'parent query' table within 'parent query' and use that alias name in child query "where condition".

Syntax: select * from tablename aliasname when columnname = (select * from tablename where column name =(any operator) aliasname.columnname);

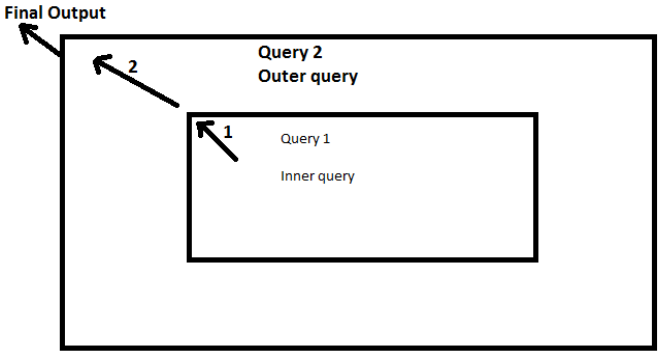
Generally, correlated sub query are used in denormalization process in denormalization process in this process we are using correlated updates. Generally, if you want to modify one table column values based on another table and also those tables are related then only we are using " Correlated update".

Syntax: update tablename1 aliasname1 set columnname = (select columnname from tablename2 aliasname2 where aliasname1.common col name);
SQL> alter table emp add dname varchar2(10);
SQL> update emp e set dname=(select dname from dept d where e.deptno=d.deptno);
SQL> select * from emp;

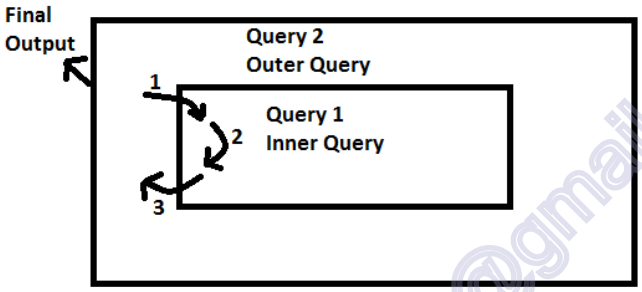
Date: 21/4/15

Q) Write a query to display the employees who are getting more salary than the average salaries of their job from emp using correleated sub query?
Ans: SQL> select * from emp e where sal> (select avg(sal) from emp where job=e.job);

Non-Correlated Sub Query:



Correlated Sub Query:



Q) Write a query to display first highest salary employee from emp table using correlated sub query??
Ans: SQL> select * from emp e1 where 1 =(select count(*) from emp e2 where e1.sal>=e2.sal);
Output: king 7900

Q) Write a query to display second highest salary employee from following table using correlated sub query?
Ans: SQL> create table test(ename varchar(20), sal number(10));
SQL> insert into test values('&e',&s);
Enter the values for e: abc
Enter the values for s: 100
SQL> /
Enter the values for e: xyz
Enter the values for s: 150
SQL> /
Enter the values for e: zzz
Enter the values for s: 200

```
SQL> /
Enter the values for e: bbb
Enter the values for s: 300

SQL> select * from test;
ENAME       SAL
-----------------------
abc         100
xyz         150
zzz         200
bbb         300
```

SQL> select * from test e1 where 2=(select count(*) from test e2 where e2.sal>=e1.sal);

Execution Process:

Phase 1:

Step 1: get a "Candidate Row" [where cursor pointer points a record is called "Candidate Row"] (abc 100)

Step 2: select count(*) from test e2 where e2.sal>=100;

4

Step 3: select * from e1. Where 2=4;

False

Phase 2:

Step 1: get a "Candidate Row" [where cursor pointer points a record is called "Candidate Row"] (xyz 150)

Step 2: select count(*) from test e2 where e2.sal>=150;

3

Step 3: select * from e1. Where 2=3;

False

Phase 3:

Step 1: get a "Candidate Row" [where cursor pointer points a record is called "Candidate Row"] (zzz 200)

Step 2: select count(*) from test e2 where e2.sal>=200;

2

Step 3: select * from e1. Where 2=2;

True

Output: zzz 200

NOTE: Whenever resource table having duplicate data based on where conditional column then the above query doesn't return any rows. To overcome this problem we must use "DISTINCT" clause within "Count(*)" function.

Eg: SQL> insert into test values('murali',200);
SQL> select * from test;

ENAME	SAL
abc	100
xyz	150
zzz	200
bbb	300
murali	200

SQL> select * from test e1 where 2=(select count(*) from test e2 where e2.sal>=e1.sal);
No Rows Selected
Solution:
SQL> select * from test e1 where 2=(select count(distinct(sal)) from test e2 where e2.sal=e1.sal);
Output:
Zzz 200
Murali 200

NOTE: We can also write above query using "n-1" method. In this case we are not allowed to use "=" operator in "Where" condition of "Child Query".

Eg: SQL> select * from test e1 where (2-1)=(select count(distinct(sal)) from test e2 where e2.sal>e1.sal);

Q) Write a query to display nth highest salary employee from emp table using "Correlated Subquery".?
Ans: SQL> select * from emp e1 where &n = (select count(distinct(sal)) from emp e2 where e2.sal>=e1.sal);

Date : 22/4/15

Exists Operator: Exists operator used in "Correlated sub query's". Exists Operator performance is very high. Compare to "in" operator. Exists operator always returns "Boolean Values" either "true" or "false". Exists Operator is used in "where" condition of the "Parent Query" only and also when we are using Exists Operator then we are not allowed to use "Column Name" along with "Exists Operator" in "Where" Condition.
Syntax:
Select * from tablename aliasname where exists(select * from tablename where columnname=aliasname. Columnname);
In all database systems if we want to test whether one table values are availabe or not availabe in another table columns then only we are using "Exists Operator".

NOTE: Exists Operator is used to test whether a given set is empty (or) non-empty Exists on non-empty set returns "true" whereas exists operator on empty set returns "false".
Eg 1: exists {1,2,3}= true
Eg 2: exists { }= false

Q) Write a query to display those departments from dept table having employees in emp table using correlated sub query exists operator?
Ans: SQL> select * from dept d where exists(select * from emp where deptno=d.deptno);
Output:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO

Q) Write a query to display the employees who are getting same salary as scott salary from emp table using correlated sub query exists operator from emp?
Ans: SQL> select * from emp e1 where exists(select * from emp e2 where ename='SCOTT' and e2.sal=e1.sal);
Output:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPNTO
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20

➔ For not displaying 'SCOTT' in the output table:

Solution: SQL> select * from emp e1 where exists(select * from emp e2 where ename='SCOTT' and e2.sal=e1.sal);

Output:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPNTO
7902	FORD	ANALYST	7566	03-DEC-81	3000		20

Q) Write a query to display those departments from dept table doesn't have employees in emp table using correlated sub query?
Ans: SQL> select * from dept d where not exists(select * from emp where deptno=d.deptno);
Output: 40 operations boston

Q) Write a query to display those department doesn't have employees in emp table using non-correlated sub query's?

Ans: SQL> select * from dept where deptno not in(select deptno from emp);
Output: 40 operations boston

Generally, "not in" operator doesn't work with null values to overcome this problem we are using "not exists" operator along with "correlated sub query".
SQL> select * from emp;
SQL> select * from dept where deptno not in(select deptno from emp);
No Rows Selected
SQL> select * from dept d where not exists(select * from emp where deptno=d.deptno);
Output: 40 Operations BOSTON

Date: 23/4/15

NOTE: In Oracle, whenever we are testing one table column values are available (or) not available in another table using not correlated and correlated query's. These type of query's are also called as "Anti Joins".
Eg: SQL> select * from dept where deptno not in(select deptno from emp);

VIEWS

View is an database object which is used to provide authority level of security. Generally, views are created by "Database Administrator".
View doesn't store data that's why view is also called as "Virtual table". View is also called as "Window of a table".
Generally views are created from "Base tables". Based on the type of base tables views are categorized into two views.
1. Simple View
2. Complex view (or) Join view

"**Simple view**" is a view which is created from only "One Base" tables where as "**Complex view**" is a view which is created from "Multiple base" tables.

Simple View:
Syntax: create or replace view viewname as
select * from tablename where condition;

DML Operations performed on Simple view:
1. When a Simple view contains "Group functions", "Group by", "Rownum", "Distinct", "Set Operators", "Joins" then we cannot perform DML operation through "Simple view" to "Base table".
2. We must include "Base table" not null column into the view then only we can perform "Insertion operation" through "View2" base table. Otherwise oracle server returns an error.

Eg 1: SQL> create or replace view v1 as select * from emp where deptno=10;
SQL> select * from v1;
SQL> insert into v1(empno,ename,deptno) values(1,'murali',30);
1 row created
SQL> select * from emp;
Eg 2: SQL> create or replace view v2 as select ename, sal, deptno from emp where deptno=10;
SQL> select * from v2;
SQL> insert into v2(ename, sal, deptno) values('abc',1000,30);
Error: Cannot insert null into empno.
In Oracle, whenever we are using "Group functions" or "rownum" in a view then we must create alias name for those expressions otherwise oracle server returns an error.
Eg: SQL> create or replace view v3 as select depnto, max(sal) a from emp group by deptno;
View created
SQL> select * from v3;
SQL> create or replace view v4 as select rownum, ename from emp;
Error: must name this expression with column alias.
SQL> create or replace view v4 as select rownum rno, ename from emp;
SQL> select * from v4;
In every database systems whenever we are creating a view then automatically view definitions (SELECT STATEMENTS…….) are automatically permanently stored in database.
In Oracle, if we want to view these definitions then we are using "user_views" data dictionary.
Eg: SQL> desc user_views;
SQL> select text from user_views where view_name='V3';
Output:
TEXT
---
select deptno,max(sal) a from emp group by deptno;

Date: 24/4/15

Complex View: Complex view is a view which is created from multiple base tables.
Eg: SQL> create or replace view v5
as
select ename, sal,dname, loc
from emp,dept
where emp.deptno=dept.deptno;
SQL> select * from v5;

DML Operations on Complex Views:

Eg: SQL> update v5 set ename='abc' where ename='SMITH';
1 row updated
SQL> update v5 set dname='xyz' where dname='SALES';
Error: Cannot modify a column which maps to a non key preserved table.
Generally, in all database systems we cannot perform DML operations through "Complex Views" to base tables. If we are try to perform DML operations then some table columns are effected and some other table columns are not effected. In Oracle, if we want to "view" effected, unaffected columns then we are using "user_updatable_columns" data dictionary.
Eg: SQL> desc user_updatable_columns
SQL> select column_name, updatable from user_updatable_columns where table_name='V5';

In Oracle, also we cannot perform DML operations through Complex Views to base table to overcome this problem Oracle 8.0 introducted instead of triggers in pl/SQL.

TRIGGERS: (PL/SQL concept)

Triggers is also same as "Stored Procedures" and also it will automatically invoked whenever a DML operations performed on a table.
All database systems having 2 types of triggers .
1. Statement level triggers
2. Row level triggers

In Statement level trigger, triggers body is executed only once per DML statements. Where as in Row lever trigger, triggers body is executed for each row for DML statements.
Syntax:
Create or replace trigger triggername
before/after Insert/Update/Delete on tablename
[for each row] -----→ row level trigger
begin

------
end;

Trigger specification

Trigger body

Difference Between Statement level and Row level Triggers:
Eg: SQL> create table test(col1 date);
Statement level trigger:
SQL> create or replace trigger tv1
after update on emp
begin
insert into test values(sysdate);
end;

/

Testing:
SQL> update emp set sal=sal+100 where deptno=10;
3 rows updated
SQL> select * from test;
SQL> delete from test;
SQL> delete trigger tv1;

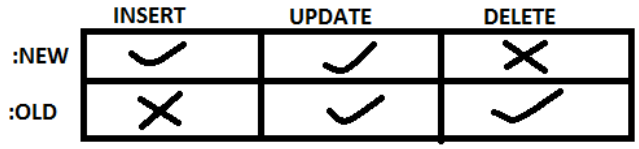
Row level triggers:
SQL> create or replace trigger tv2
after update on emp
for each row
begin
insert into test values(sysdate);
end;
/

Testing:
SQL> update emp set sal=sal+100 where deptno=10;
3 rows updated
SQL> select * from test;
SQL> drop trigger tv2;

Output:
COL1

24-APR-15
24-APR-15
24-APR-15

Row Level Triggers:
In Row level triggers, triggers body is executed per each row for DML statements. That's why we are using "for each row" clause within trigger specification and also DML transaction "roll back" segment qualifiers. These are old, new.
These qualifiers are used in either in trigger specification or in trigger body when we are using these qualifiers in trigger body then we must use ":" (column) infront of the "Qualifier name".
Syntax:
:old.columnname
Syntax:
:new.columnname



Date: 25/4/15

Q) Write a PL/SQL row level trigger on emp table whenever user deleting data then automatically those deleted records are stored in another table?
Ans: SQL> create table test as select * from emp where 1=2;
SQL> select * from test;
SQL> desc test;
SQL> create or replace trigger tg1
after delete on emp
for each row
begin
insert into test values(:old.empno, :old.ename, :old.job, :old.mgr, :old.hiredate, :old.sal, :old.comm, :old.deptno);
end;
/
View created
Testing:
SQL> delete from emp where sal>2000;
SQL> select * from test;
10 row selected

Instead of Trigger:
Generally, we cannot perform "DML operations" through complex view to "Base table". To overcome this problem Oracle 8.0 introduced "Instead of trigger" in PL/SQL. By default "Instead of Triggers" are "Row lever Triggers". "Instead of trigger" are created on "Views".
Syntax:
Create or replace trigger triggername
instead of insert/update/delete on viewname
for each row
begin

----
end;

Eg: "Instead of Trigger":
SQL> create or replace trigger tj1
instead of update on v5
for each row

```
begin
update dept set dname=:new.dname where dname=:old.dname;
update dept set loc=:new.loc where loc=:old.loc;
end;
/
Trigger created
```

Testing:

Ex 1: SQL> update v5 set dname='xyz' where dname='SALES';

Rows updated

Ex 2: SQL> desc user_updatable_columns;

SQL> select COLUMN_NAME, UPDATABLE from user_updatable_columns where table_name='V5';

COLUMN_NAMES	UPDATABLE
ENAME	YES
SAL	YES
DNAME	YES
LOC	YES

SQL> update v5 set loc='new york' where loc='DALLAS';

MATERIALIZED VIEWS:

Oracle 8i introduced "Materialized views". These views are created by database administrator. "Materialized views" are used in data warehousing applications. Generally, views doesn't store data where as "Materialized views" store data. Generally, "Materialized views" are used to improve performance of the "Joined" or "Aggregatable Query's". "Materialized views" stores replication of the remote database into local node. "Materialized views" stores data same like a table, but whenever when we are refreshing "Materialized views" it synchronizes data based on "Base Table".

Syntax:

```
create materialized view viewname
as select statement;
```

Difference Between Views, Materialized Views:

VIEWS	MATERIALIZED VIEW
1. Views doesn't store data	1. Materialized view stores data.
2. Security purpose	2. Improve performance purpose
3. When we are dropping base table	3. If we are dropping "Base table" also Materialized view can be accessible.
4. We can perform DML operations on views	4. We cannot perform DML operations on Materialized views

Before we are creating Materialized view then Database Administrator must give "Create any Materialized view" privilege to user. Otherwise oracle server returns an error "Insufficient Privilege.

Syntax:

Grant create any materialized view to username;

Eg: SQL> conn scott/tiger

SQL> create materialized view mn1

as

select * from emp;

Error: Insufficient Privileges.

Solution:

SQL> conn sys as sysdba

Enter password: sys

SQL> grant create any materialized view to scott;

SQL> conn scott/tiger

Connected

SQL> create materialized view mn1

as select * from emp;

Materialized View Created.

Date: 27/4/15

In Oracle, one of the materialized view base table must contain a primary key, otherwise oracle server returns an error.

Eg: SQL> create table test(sno number(10));

SQL> create materialized view mw1

as

select * from test;

Error: Insufficient privileges

SQL> conn sys as sysdba/sys

SQL> grant create any materialized view to scott;

Grant succeeded

SQL> conn scott/tiger

SQL> create materialized view mw1

as select * from test;

Error: table 'TEST' does not contain a primary key constraint

SQL> create table base(sno number(10) primary key, name varchar2(20));

SQL> insert into base values(&s,'&n');

SQL> select * from base;

SQL> commit;

SQL> create or replace view v1

as select * from base;

SQL> create materialized view mv1

as select * from base;

Here, materialized view also behaves like a view that is in this case materialized view is also same as "View". That is whenever we are creating materialized view then automatically materialized view definitions also permanently stored in database same like a view definition. In Oracle, if we want to view materialized view definitions then we are using "user_mviews" data dictionary.

SQL> desc user_mviews;

SQL> select query from user_mviews where mview_name='MV1';

SQL> select rowid, sno, sname from base;

SQL> select rowid, sno, name from v1;

Here, view rowid's are also same as base table rowid's that's why view does not store data. That's why view is also called as "Virtual table" or "Window of a table". Through the view only we can view (or) we can access base table data. Generally, view stores stored query in memory.

SQL> select rowid, sno, name from mv1;

Here, materialized view rowid's are different from 'Base table' rowid's are different from 'Base table' rowid's that's why materialized view's stores data.

SQL> update based set name=upper(name);

SQL> select * from base;

SQL> select * from v1;

SQL> select * from mv1;

Materialized view also stores data same like a table but, when we are refreshing materialized view it synchronizes data based on "Base tables". In Oracle, if we want to refresh materialized view then we are using "Refresh" procedure from "dbms_mview" package.

Syntax:

SQL> exec dbms_mview.refresh('materialized viewname');

Dbms_mview: predefined package in PL/SQL

Refresh('M.V name'): Predefined procedure name.

SQL> desc dbms_mview;

Eg: SQL> exec dbms_mview.refresh('mv1');

SQL> select * from mv1;

Executive Process:

In Oracle, whenever we are requesting information from the view using 'SELECT' statement, then Oracle server directly executes view definitions from the 'Data Dictionary'. That's why view does not improves performance.

Whenver, we are creating a materialized view then Oracle server automatically stores materialized view definitions in a "Data Dictionary" and also result of the query is stored in different place within database. Whenever user requesting materialized view the select statement then Oracle server directly retrieve data from materialized view whenever we are refreshing materialized view then only oracle server executes materialized view definitions based on these definitions "Base tables" are effected then only Oracle server synchronize "Base table" data into materialized view. In this process if we are not refreshing materialized view then "Base table" are never effected. This process

automatically improves performance of the query that's why materialized view improves performance.
Oracle, having two types of materialized views:
1. Complete refresh materialized views
2. Fast refresh materialized views

1. Complete refresh materialized view:
In Oracle, by default materialized views are complete refresh materialized views. These type of materialized views does not give more performance when we are refreshing materialized views number of times because in these materialized views whenever we are refresheing materialized views internally rowid's are recreated. If we are not modifying data in base table also to overcome this problem to improve more performance of query then oracle introduced "Fast refresh materialized views".
Syntax:
Create materialized view viewname
refresh complete
as
select statement;
Eg: SQL> select rowid, sno, name from mv1;
SQL> exec dbms_mview.refresh('mv1');
SQL> select rowid, sno, name from mv1;
[here rowid's are changed]

Date: 28/4/15

2. Fast Refresh Materialized View:
These type of materialized views are also called as Incremental refresh materialized views. These materialized views performance is very high compare to complete refresh materialized views because in these materialized views rowids are never changed when we are refreshing materialized views number of times also.
Syntax:
create materialized view viewname
refresh fast
as
select statement;

Before we are using "Fast Refresh materialized views" then we must capture changes made in "Base table" for this purpose oracle provided a mechanism called "Materialized view Log" i.e., when we are creating "Materialized view Log" oracle server capture the changes from base tables and storing into appropriate data dictionaries. That's why before we are using fast refresh materialized views then we must create materialized view log on "Base table".
Syntax:
Create materialized view log on basetablename;

Eg:
SQL> select * from base;
SNO NAME
---
1 A
2 B
3 C
4 D
SQL> create materialized view log on base;
SQL> create materialized view mk1;
refresh fast
as
select * from base;
SQL> select rowid,sno,name from mk1;
SQL> update base set name='xy' where sno=1;
SQL> exec dbms_mview.refresh('mk1');
SQL> select rowid, sno, name from mk1;
[Here rowid's are not changed only data is affected]

On demand/ On commit:
In Oracle, we can refresh materialized views in 2 ways.
1. Manually
2. Automatically

1. **Manually:** In manual, method we are refreshing materialized view using "dbms_mview" package. This method is also called as "On Demand" method. By default method is "On demand".
2. **Automatically:** We can also refresh materialized view without using "dbms_mview" package. This method is also called as "On Commit" method.

Syntax:
create materialized view viewname
refresh complete/refresh fast on demand/on commit
as
select statement;

Eg:
SQL> create materialized view mk2
refresh fast on commit
as
select * from base;
SQL> select * from mk2;
SNO NAME

```
    1                   xy
    2                   B
    3                   C
    4                   D
SQL> update base set name='zz' where sno=2;
SQL> select * from base;
   SNO                  NAME
   --------------------------
    1                   xy
    2                   zz
    3                   C
    4                   D

SQL> select * from mk2;
   SNO                  NAME
   --------------------------
    1                   xy
    2                   B
    3                   C
    4                   D

SQL> commit;
SQL> select * from mk2;
   SNO                  NAME
   --------------------------
    1                   xy
    2                   zz
    3                   C
    4                   D
```


In all database systems, if we want to restrict table columns from on user into another user then also we are using "Views".

DATA CONTROL LANGUAGE(DCL):

1. Grant -> giving permissions
2. Revoke -> cancel permissions

Creating a User:

```
SQL> conn sys as sysdba
Enter Password: sys
OR
SQL> conn system/manager
```

Syntax:

1. Create user username identified by password;
2. Grant connect, resource (or) dba to username;

SQL> conn username/password;

Process for creating a User:

SQL> conn sys as sysdba
Enter Password: sys
SQL> create user murali identified by murali;
SQL> grant connect, resource to murali;
SQL> conn murali/murali;
SQL> select * from emp;
Error: table (or) view does not exists;
SQL> conn scott/tiger;
SQL> grant all on emp to murali;
SQL> conn murali/murali;
SQL> select * from emp;
Error: table (or) view does not exists.
SQL> select * from scott.emp;
SQL> create synonym ab for scott.emp;
SQL> select * from ab;

Privilege: Privilege is a right or permission which is used to given to the another "user".

All database systems having two types of privileges. They are:

1. System Privilege
2. Object Privilege

These Privileges are used to provide security for database objects.

Date : 29/4/15

1. System Privileges:

System Privileges are given by database administrator. These privileges are create table, create any view, create any materialized view, create session, create procedure, create any index,….;

Oracle having more than 80 system privileges. These privileges are used to users allowed to create database object, auditor database objects.

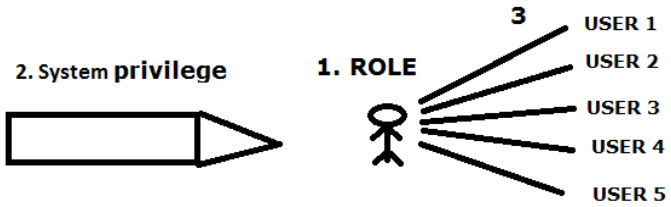
Syntax:

grant system privileges to username1, username2,……;

Eg: SQL> conn sys as sysdba
Enter password: sys
SQL> grant create procedure, create any materialized view, create any index to scott, murali;

ROLES:

Role is nothing but set of system privileges or set of object privilege. User defined rows are created by database administrator. In a multi user environment, number of users works on same project. In this case, sometime number of users requires common set of privileges. In this case only database administrator creating a user defined role and then assigns set of privileges into role and then only that role given to the number of users.



Creating User Defined Role:
Step 1: Create a role
Step 2: Assign system privileges to role
Step 3: Assign rolename to number of users.

STEP 1: Create a Role:
Syntax: create role rolename;
STEP 2: Assign System privileges to Role
Syntax: grant system privilege to rolename;
STEP 3: Assign Rolename to number of users.
Syntax: grant rolename to username1, username2,….;
Eg:
SQL> conn sys as sysdba
Enter password: sys
SQL> create role r1;
Role created
SQL> grant create any materialized view, create trigger, create procedure to r1;
SQL> grant r1 to scott, murali, dinesh;
In Oracle, if we want to view all the system privileges related to role then we are using "role_sys_privs" data dictionary.
Eg: SQL> desc role_sys_privs;
SQL> select role, privilege from role_sys_privs where role='R1';

ROLE	PRIVILEGE
R1	CREATE TRIGGER
R1	CREATE ANY MATERIALIZED VIEW
R1	CREATE PROCEDURE

Predefined Roles:
Whenever we are installing oracle server then automatically '3' predefined roles are created. They are:
1. Connect -> end user purpose
2. Resource -> developer purpose

3. DBA -> database administrator purpose

Eg: SQL> desc role_sys_privs;
SQL> select role, privilege from role_sys_privs where role in ('connect','resource');
NOTE: In Oracle, "connect role" internally having "create session" system privilege. This privilege is used to users allowed to connect to the "Oracle Server".
SQL> select role, privilege from role_sys_privs where role in('DBA');

4. Object Privilege:
Object Privileges are given by either database developers or database administrator. These privileges are used to "Users" are allowed to perform some operations on the object. Oracle having insert, update, delete, select object privileges related to table. These '4' object privileges also represented by using "all" keyword.
Oracle also supports execute object privilege for PL/SQL objects and also supports read, write object privileges for "utl_file" package.
Syntax:
grant object privilege on objectname to usernames/rolename/public;
Date: 30/4/15

SQL> conn scott/tiger;
SQL> grant all on emp to murali;
SQL> grant all on emp to r1;
SQL> grant all on emp to public;
NOTE: In Oracle, we can also give object privileges to particular columns. In this case we must specify number of columns. In this case we must specify number of columns within parenthesis "()".
Eg: SQL> grant update(ename) on emp to murali;

➔ **WITH GRANT OPTION:**
Who receives "With Grant Option" clause those users allowed to give same object privileges to another users.
Syntax:
grant object privileges on object name to usernames/public with grant option;

Eg: SQL> grant all on emp to murali with grant option;
NOTE: In all database systems "With Grant Option" clause does not work with roles.
Eg: SQL> grant all on emp to r1 with grant option;
Error: cannot GRANT to a role with GRANT OPTION.
In Oracle, all object privileges are stored under "user_tab_privs" data dictionary.
Eg: SQL> desc user_tab_privs;

REVOKE:
This command is used to "CANCEL" system privileges, object privileges from users.
Syntax:

1. Revoke system privileges from user1,user2,….;
2. Revoke object privileges on object name from username/public;

Eg: SQL> conn sys as sysdba;
Enter password: sys
SQL> revoke create any materialized view from scott;
Revoke succeeded
Generally, if we want to restrict table columns from one user into another user then we are creating views with required columns and then only that view given to the number of users.
Syntax: grant all on viewname to user1,user2,….;
SQL> conn scott/tiger
SQL> create or replace view vv1
as
select empno, ename, sal, deptno from emp;
SQL> select * from v1;
SQL> grant all on v1 to murali;
SQL> conn murali/murali;
SQL> select * from scott.v1;

Force View (or) Forced View:
In Oracle, we can also create views without using base tables. These type of views are also called as " Force Views".
Syntax:
create or replace force view viewname
as
select * from welcome;
Eg: SQL> create or replace force view v2
as
select * from welcome;
Warning: view created with compilation errors.
SQL> create table welcome (sno number(10));
SQL> alter view v2 compile;
SQL> desc v2;

Read Only views:
In Oracle, we can also create read only views using "with read only" option clause. In Read only views we cannot perform DML operations. These views are used for reporting purpose.
Syntax:
Create or replace view viewname
as
select * from tablename where condition with read only;
Eg: SQL> create or replace view v3

as
select * from emp with read only;
SQL> delete from v3 where deptno=10;
Error: cannot delete from view.

With Check View:

If we want to provide constraint type mechanism on views then we are using "With Check Option" clause on views. When a view contains "with check option" clause, then we are not allowed to insert other than where condition values through view2 base table.

Syntax:

create or replace view viewname
as
select * from tablename where condition with check option;
Eg: SQL> create or replace view v4
as
select * from emp where deptno=10 with check option;
SQL> select * from v4;

Testing:

SQL> insert into v4(empno,ename,deptno) values(1,'abc',30);
Error: view with check option where clause violated.
SQL> insert into v4(empno,ename,deptno) values(1.'abc,'10);
1 Row created
SQL> select * from emp;
We can also drop view using "drop view viewname".

NOTE: In all database systems through the views we can achieve logical data independence i.e., logical data independence refers whenever we are adding a new entity in "conceptual level" it is not effected in external level. i.e., whenever we are adding a new column in a table it is never effected in a view because of Logical Data Independence.
Eg: SQL> alter table emp add address varchar2(10);
SQL> select * from emp;
SQL> select * from v4;

Date : 1/5/15

INDEXES:

Index is an database object which is used to retrieve data very fast from the database. This process automatically improves performance of query. In all database systems creating Indexes in 2 ways:
1 Automatically
2 Manually

1. Automatically:

Whenever we are creating "Primary Key" or "Unique Key" then only database servers automatically creates b-tree indexes on those columns.

2. Manually:

We can also create "Indexes" explicitly by using "create index" command.

In Oracle, we are creating index explicitly by using following syntax:

Syntax:

Create index indexname on tablename(columnname);

Generally, indexes are created on "table columns" and also indexes are created by "database administrator" only.

In Oracle, whenever we are requesting data using "where" clause or "Order by" clause then only oracle server searching for "Indexes" in appropriate "Data Dictionary" within Oracle database. If "where" clause columns or "Order by" clause columns having index then Oracle server uses "Index Scan" method for retrieving data from database. This process automatically improves performance of the query.

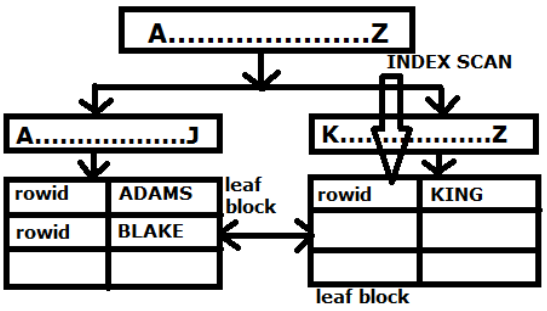
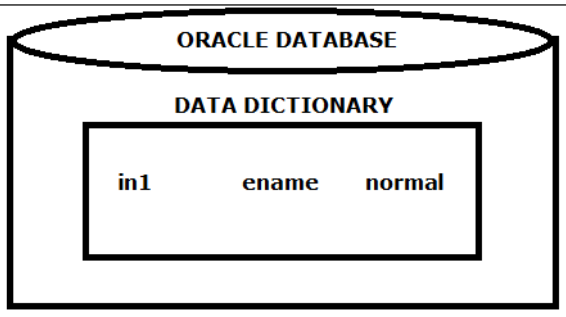
NOTE: In Oracle, whenever "where" clause contains "not equal to" (!=, <>), "is null", "is not null" operators then oracle server does not search for indexes. If those columns already having indexes also.

Oracle having 2 types of Indexes:

1. B-trees Indexes
2. Bit Map Indexes

1. **B-Tree Indexes:** In Oracle, by default indexes are b-tree indexes. Whenever we are requesting data using "where" or "Order by" columns, also those columns having b-tree indexes then Oracle server automatically creates "b-tree structure" based on indexed columns. In this "b-tree structure" always "Leaf blocks" stores actual data along with "Rowid's". Based on the "Where" clause columns Oracle server retrieves data from these "Leaf blocks" using "Index Scan Method". This process very fastly retrieve data from database.

Syntax:

DEVELOPERs	Data Base Administrators
SQL> select * from emp where ename='KING';	SQL> create index in1 on emp(ename);
	

SQL> create index in1 on emp(ename);

In Oracle, all indexes information stored under "user_indexes" data dictionary.

Eg: SQL> desc user_indexes;

SQL> select index_name, index_type from user_indexes where table_name='EMP';

Output:

```
INDEX_NAME                     INDEX_TYPE
-----------------------------------------
PK_EMP                         NORMAL
```

NOTE: In Oracle, if you want to view column names along with Index names then we are using "user_ind_columns" data dictionary.

Eg: SQL> desc user_ind_columns;
SQL> select index_name, column_name from user_ind_columns where table_name='EMP';

Output:

INDEX_NAME	COLUMN_TYPE
PK_EMP	EMPNO
IN1	ENAME

NOTE: In all database systems using indexes we can achieve "Physical Data Independence" i.e., whenever we are adding indexes(Internal Level) it is not effected in table(Conceptual Level) But performance is effected.

Date: 2/5/15

In Oracle, if you want to view query performance then database administrator viewing plan tables. These tables are automatically created by "Oracle server". Before we are displaying plan table then we must use "EXPLAIN PLAN FOR" clause in front of the query and then only we can display plan table using display function from "dbms_xplan" package.
Step 1:
Syntax: SQL> explain plan for select statement;
Step 2: (display plan table)
SQL> select * from table(dbms_xplan.display);
Eg: (without using Indexes)
SQL> select * from emp where ename='KING';

Testing Performance
SQL> explain plan for select * from emp where ename='KING';
SQL> select * from table(dbms_xplan.display());

With Using Indexes:
SQL>create index in1 on emp(ename);
SQL> select * from emp where ename='KING';

Testing Performance
SQL> explain plan for select * from emp where ename='KING';
SQL> slelect * from table(dbms_xplan.display());

Function Based Indexes:
Oracle 8i introduced "Function Based Indexes" by default "function based indexes" are b-tree indexes.

Generally, whenever we are requesting data using functions (or) expressions then oracle server does not search for Indexes if those columns already having "Indexes" also. To overcome this problem Oracle 8i introduced extension of the b-tree indexes called "Function Based Indexes" which is used to create Indexes on columns along with functions (or) expressions. Then only Oracle server searching for indexes if we are requesting data using functions (or) expressions also.

Syntax:
create index indexname on tablename(function name(column name) or expression);

Without using functions based Indexes:
SQL> select * from emp where upper(ename)='KING';

Testing Performance
SQL> explain plan for select * from emp where upper(ename)='KING';
SQL> select * from table(dbms_xplan.display());
[here oracle server does not search for indexes]

With using function based Indexes:
SQL> create index in2 on emp(upper(ename));
SQL> select * from emp where upper(ename)='KING';

Testing Performance
SQL> explain plan for select * from where upper(ename)='KING';
SQL> select * from table(dbms_xplan.display());

SQL> desc **user_indexes;**
SQL> select index_name, index_type from user_indexes where table_name='EMP';

```
INDEX_NAME                INDEX_TYPE
-----------------------------------------
IN2                       FUNCTION BASED NORMAL
```


VIRTUAL COLUMNS
Oracle 11g introduced "Virtual Columns" in a table which stores "Stored Expression" directly in Oracle database prior (mean before) to 11g Database Administrators only stores "Stored Expressions" indirectly using "VIEWS" , "Function Based Indexes". Where as when we are using "Virtual Columns" we can store "Stored Expressions" directly in database using "Generated Always As" clause.

Syntax:
columnname datatype(size) generated always as(stored expression)[virtual]
Eg:
SQL> create table test(a number(10), b number(10), c number(10) generated always as (a+b) virtual);

SQL> insert into test(a,b) values(20,40);
SQL> select * from test;

A	B	C
20	40	60

If we want to view "Virtual Column" expression then we are using "data_default" property from "user_tab_columns" data dictionary.

SQL> desc user_tab_columns;
SQL> select column_name, data_default from user_tab_columns where table_name='TEST';

COLUMN_NAME	DATA_DEFAULT
C	A+B

Oracle having two types of b-tree indexes.

1. Non unique b-tree Indexes
2. Unique b-tree Indexes

By default, all automatic Indexes are "Unique b-tree Indexes" and also all explicit Indexes are "Non unique b-tree Indexes". By default, "Unique b-tree Indexes" performance is very high compare to "Non unique b-tree Indexes". We can also create "Unique b-tree Indexes" explicitly by using following syntax.

Syntax:
Create unique index indexname on tablename(columnname);
Eg: SQL> create unique index in3 on emp(ename);
Index created
NOTE: We are not allowed to create unique indexes on duplicate value columns.
SQL> create unique index in4 on emp(job);
ERROR: cannot CREATE UNIQUE INDEX: duplicate key found.

2. BIT MAP INDEX:
Oracle 7.3, introduced Bit Map Indexes. Bit Map Indexes are used in Data Ware housing applications. Generally, Bit Map Indexes are created on "Low Cardinality Columns".
Syntax:
create bit map index indexname on tablename(columnname);

Date: 4/5/15

$$\text{CARDINALITY OF A COLUMN} = \frac{\text{NUMBER OF DISTINCT VALUES}}{\text{TOTAL NUMBER OF VALUES}}$$

Eg 1: Cardinality of empno = 14/14 = 1 (high cardinality) -> b-tree
Eg 2: Cardinality of job = 5/14 = 0.367…. (low cardinality) -> bit map indexes

Whenever we are creating a "Bit Map Index" then oracle server automatically creates a "Bit Map Table" based the column values in a table. Whenever user requesting data using "Where" clause then oracle server automatically uses "Index Scan" internally on "Bit Map Table" also perform operations on "Bits" within "Bit Map Table". When we are using logical operators in "Where" clause then oracle server automatically converts the resultant Bit Map into "ROWID" using a "Internal Bit Map" function.

JOB	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CLERK	1	0	0	0	0	0	0	0	0	0	1	1	0	1
SALESMAN	0	1	1	0	1	0	0	0	0	1	0	0	0	0
MANAGER	0	0	0	1	0	1	1	0	0	0	0	0	0	0
ANALYST	0	0	0	0	0	0	0	1	0	0	0	0	1	0
PRESIDENT	0	0	0	0	0	0	0	0	1	0	0	0	0	0

SQL> create bitmap index abc on emp(job);
SQL> desc user_indexes;
SQL> select index_type, index_name from user_indexes where table_name='EMP';
Output:

```
INDEX_TYPE          INDEX_NAME
-------------------------------------
NORMAL              PK_EMP
BITMAP              ABC
```

In Oracle, we can also drop Index using
"drop index indexname";

SYNONYMS

Synonyms is a database object which hides another schema user name, object name. Synonyms also provides security for the original object. It is an "Alias name" or "Reference Name" for the original objects. Synonyms are created by database administrators.
All database systems having 2 types of synonyms

1. Public Synonyms
2. Private Synonyms

In all database systems by default Synonyms are "PRIVATE Synonyms".
Syntax:
create synonym synonymname for
username. Objectname@database link;

In Oracle, if you want to create Public Synonyms then we are using "CREATE PUBLIC SYNONYM" privilege to user otherwise oracle server returns as "Insufficient Privilege Error".
Syntax: grant create public synonym to username;

Syntax:
create public synonym synonymname for
username. Objectname@database link;

Scott/tiger	SQL> conn murali/murali	SQL> conn a1/a1
SQL> grant all on emp to murali, a1;	SQL> select * from scott.emp; SQL>create synonym abc for scott.emp; SQL> select * from abc; SQL> create public synonym xy for scott.emp; Error: Insufficient Privilege SQL> conn sys as sysdba; Enter password: sys SQL> grant create public synonym to murali; SQL> conn murali/murali; SQL> create public synonym xy for scott.emp; SQL> select * from xy;	SQL> select * from scott.emp; SQL> select * from abc; SQL> select * from xy;

All Synonyms information stored under " user_synonyms" data dictionary.
SQL> desc user_synonyms;
We can also drop synonyms by using
" drop synonym synonymname"

LOCKS

Locking is a mechanism which prevents unauthorized access for our resource. Whole database systems having 2 types of locks.

1. Row Level Locks.
2. Table Level Locks

1. **Row Level Locks:** Oracle 6.0 introduced "Row Level Locks". If you want to establishes "Row Level Locks" then we are using " for update" clause in all databases. This clause used only in "Select" statement. In this method we are locking set of rows.

Syntax: select * from tablename where condition for update[no wait];
Whenever we are performing locks then another user query the data query but they cannot perform DML operation whenever we are using "Commit" and "Rollback" locks are released.

Date: 5/5/15

Scott/tiger	Murali/murali
SQL> select * from emp where deptno=10 for update; SQL> commit;[for releasing lock]	SQL> update scott.emp set sal=sal+100 where deptno=10; [we cannot perform DML operations]

NO WAIT: "No wait" is an optional clause used along with "for update" clause. When we are using "no wait" clause oracle server automatically returns control to the current session if another user not releasing locks also. In this case oracle server returns an error.
ORA-0054: resource busy

Scott/tiger	Murali/murali
SQL> select * from emp where deptno=10 for update no wait; ORA-0054: resource busy SQL>_	SQL> select * from scott.emp where deptno=10 for update;

DEFAULT LOCKS: In all database systems whenever we are using DML operations then automatically database servers internally uses "Exclusive Locks". These are also called as "Default Locks".

Eg:

Scott/tiger	Murali/murali
SQL> update emp set sal=sal+100 where deptno=10; 2 rows updated 3 SQL> commit;[for releasing locks]	SQL> update scott.emp set sal=sal+100 where deptno=10; [we could not perform DML operations because of "Default Locks"]

DEAD LOCKS:

When we try to perform same DML operations on same resource at a time through different sessions for a "single user" or through different users then oracle server automatically returns an error.
ORA-0060: dead lock detected

Scott/tiger	Scott/tiger
SQL> update emp set sal=sal+100 where deptno=100; 3 rows updated 4 SQL> update emp set sal=sal+100 5 where deptno=20; ORA-0060: dead lock detected SQL> commit; [for releasing locks]	SQL> update emp set sal=sal+100 where depnto=20; rows updated

TABLE LEVEL LOCKS:

In this method we are locking a table. Oracle having 2 types of table level locks. "Table Level Locks" are handled by Database Administrators.

1. Share lock
2. Exclusive lock

1. **Share lock:** When we are using this lock another user query the data but they cannot perform DML operations and also at a time number of users can lock the resource.

Syntax: lock table tablename in share mode;

Here, also when ever we are using "Commit" or "Rollback" then only automatically locks are released.

Eg:

Scott/tiger	Murali/murali
SQL> lock table emp in share mode; SQL> commit; [for releasing lock]	SQL> select * from scott.emp; SQL> lock table scott.emp in share mode; SQL> update scott.emp set sal=sal+100; [we cannot perform DML operations]

2. **Exclusive lock:** When we are using "exclusive locks" then another user query the data but they cannot perform DML operations and also at a time only one user lock the resource.

Syntax: lock table tablename in exclusive mode;

Eg:

Scott/tiger	Murali/muarli
SQL> lock table emp in exclusive mode; SQL> commit;[for releasing locks]	SQL> select * from scott.emp; SQL> lock table scott.emp in share mode;

NOTE: in all database systems whenever we are using cursor locking mechanism then automatically database servers internally uses "Exclusive mode".

SEQUENCE

Sequence is an database object which is used to generate "Sequence numbers" automatically. Generally, sequences are used to generates "Primary key values" automatically. Generally, sequences are created by database administrator. Sequence is an "Independent database object". Once a sequence has been created then number of users simultaneously accepts that "sequence".

Syntax:

```
create sequence sequencename
start with n
increment by n
minvalue n
maxvalue n
cycle/nocycle
cache/nocache;
```

In Oracle, if we want to access sequence values then we are using 2 pseudo columns.

1. Currval
2. Nextval

Syntax (currval): sequencename.currval;
Syntax (nextval): sequencename.nextval;
These Pseudo columns are used in Insert, Update, Delete, Select statements.
In Oracle, if we want to generate sequence values using "SELECT statement" then we must use "dual" table.
Syntax: select sequencename.currval from dual;
Syntax: select sequencename.nextval from dual;

Currval pseudo column returns current value of the sequence if sequence session already having a value. Where as nextval pseudo column returns next values for sequence update.
In Oracle, if we want to generate first sequence number then we must use nextval pseudo column. Because currval pseudo column returns value if sequence session already having a value.
In all database systems, we are using "Currval" pseudo column after using "Nextval" pseudo columns only.
Eg: SQL> create sequence s1
start with 5
increment by 2
maxvalue 100;

Sequence created
SQL> select s1 currval from dual;
Error: sequence s1.currval is not yet defined in this session
SQL> select s1.nextval from dual;
5
SQL> select s1.nextval from dual;
7
SQL> select s1.nextval from dual;
9
SQL> select s1.nextval from dual;
11
SQL> select s1.currval from dual;
11

Date: 6/5/15

NOTE: In sequence we can also change "Sequence parameter values" by using "ALTER" command.
Syntax:
alter sequence sequencename parametername newvalue;
Eg: alter sequence s1 increment by -1;
Sequence altered
In all database systems we can modify all sequence parameter values except "Start with" clause.

Eg: SQL> alter sequence s1 start with 3;
Error: cannot alter starting sequence number

NOTE: Generally, start with values cannot be less than "min value"
Eg: SQL> create sequence s1
start with 4
increment by 1
minvalue 5
max value 100;
Error: START WITH cannot be less than MIN VALUE

CACHE: Cache is an "Optional" clause defined in "Sequence" object which is used to access "Sequence values" very fastly. This process automatically improves performance of the application. Generally, cache is an memory area which is used to store set of "Sequence Numbers".
Generally, whenever we are creating a "Sequence" those sequences are created in hard disk. Whenever we are requesting sequences using a client tool, then server process checks requested sequence is available in cache memory area. If it is not available then control goes to hard disk and returns "Sequence value" from the disk and storing into "Cache memory area". Then only that value return to client tool. This default process retrieves "Sequence Values" slowly from the database because each and every time server process interacting with disk. To overcome this problem to access sequence values very fastly then database administrator defines "Cache parameter" in sequence object. Based on the parameter value cache memory pre-allocates set of "Sequence numbers". Whenever we are requesting sequences then server process, directly access sequence values from the "Cache memory" are not from the "Hard disk". This process automatically improves performance of the application because this process reduces Disk I/O(input/output). Whenever system crashes (or) power failure then automatically cache values are lost.
In Oracle, cache default value is "20" and also cache minvalue is "2".
Eg: SQL> create sequence s1
start with 1
cache 1;
Error: The number of values to CACHE must be greater than 1.
Generally, sequences are used to generate "Primary key" values automatically.
Eg: SQL> create sequence s1
start with 1;
SQL> create table test(sno number(10) primary key, name varchar2(10));
SQL> insert into test(sno,name) values(s1.nextval, '&name');
Enter value for name: murali
SQL> /
Enter value for name: abc
SQL> /
Enter value for name: xyz
SQL> select * from test;

Output:

SNO NAME
---
1 MURALI
2 ABC
3 XYZ

In Oracle, all sequences information stored under "user_sequences" data dictionary.
Eg: SQL> desc user_sequences;
We can also drop a sequence using "drop sequence sequencename".
Eg: SQL> drop sequence s1;

MERGE

Oracle 9i introduced "Merge" statement. Merge is an DML command which is used to transfer data from "source table" into "target table" when table structure are same.
Generally, merge statement used in data warehousing applications. In merge statement we are using update, insert commands. This command is also called as "UPSERT" command (UP-> update, SERT->insert).

Syntax:

```
Merge into targettablename
using sourcetablename
on(join condition)
when matched then
update set targettablecol1= sourcetablecol1,…….
when not matched then
insert(targettablecolumnnames) values(sourcetablecolumnnames);
```

Date: 8/5/15

Eg: SQL> select * from dept; (target table)
SQL> create table depts as select * from dept;
SQL> insert into depts values(1,'a','b');
SQL> select * from depts;
SQL> merge into dept d
using depts s
on(d.deptno=s.deptno)
when matched then
update set d.dname=s.dname, d.loc=s.loc
when not matched then
insert (d.deptno,d.dname,d.loc) values(s.deptno, s.dname, s.loc)
5 rows merged
SQL> select * from dept;
Through "Merge" statement we cannot update "ON" clause columns.

SET OPERATORS:

Set operators are used to retrieve data from single (or) multiple tables. These operators are also called as "Vertical Joins".

1. **Union** -> It returns unique values and also automatically sorting data.
2. **Union all** -> It returns unique + duplicate data. (no automatic sorting)
3. **Intersect** -> It returns common values
4. **Minus** -> Values are in first query those values are not in second query.

Eg:

```
SQL> select job from emp where deptno=10
union
select job from emp where deptno=20;
```

NOTE: Whenever we are using set operators, corresponding expressions must belongs to same data type.

Eg:

```
SQL> select deptno from emp
union
select dname from dept;
```

Error: expression must have same data type as corresponding expression

NOTE: Always set operators returns first query column names (or) alias names as "Column headings".

Eg:

```
SQL> select dname from dept
union
select ename from emp;
```

Output:

```
DNAME
--------
Accounting
ADAMS
…….
```

NOTE: In Oracle, in corresponding expressions not belongs to same data type also then we are retrieving data from multiple query's using "Set Operators". In this case we must use appropriate "Type conversion" function.

Eg:

```
SQL> select deptno from emp
union
select dname from dept;
```

Error: expression must have same data type as corresponding expression

SOLUTION:

```
SQL> select deptno "deptno", to_char(null) "dept name" from emp
union
select to_number(null),dname from dept;
```

Output:

```
DEPTNO      DNAME
```

10
20
30

ACCOUNTING
SALESMAN
……

CONVERSIONS:

Converting one data type into another data type is called "Conversions".
Oracle having 2 types of conversions.

1. Implicit conversions
2. Explicit conversions

1. Implicit conversions:

A. In Orcale, when an expression contains string representing pure number (eg: '1234') then oracle server automatically converts "String type" in "Number type".

Eg: SQL> select sal+'100' from emp;
It works

B. In Oracle, whenever we are passing number into character functions then oracle server automatically converts number type into character type.

Eg: SQL> select length(1234) from dual;
Output: 4

C. In Oracle, whenever we are passing date string into date functions then oracle server automatically converts "Data String" into "Date Type". But here, passed parameter must be in "Default Date Format".

Eg: SQL> select last_day('12-aug-05') from dual;
Output: 31-aug-05.

Date: 9/5/15

IMPLICIT CONVERSION TABLE			
FROM	TO	ASSIGNMENT	EXPRESSION EVALUATION
Varchar2 (or) char	Number	Yes	Yes
Varchar2 (or) char	Date	Yes	Yes
Number	Varchar2 (or) char	Yes	No
Date	Varchar2 (or) char	Yes	No

EXPLICIT CONVERSIONS:

Using "Explicit conversions" functions we can also convert one data type into another data type explicitly.

DECODE():

Decode() is an conversion function which is used to decoding the values.
Decode() is also same as "if…. then … else if" constructor in PL/SQL. Decode() internally used equality operator (=).

Syntax:
Decode (columnname, value1, statement1,value2, statement2, statement);
Eg: SQL> select ename, sal, deptno, decode(deptno, 10,'ten', 20, 'twenty', others) from emp;
Output:

ENAME	SAL	DEPTNO	DECODE
		10	TEN
		20	TWENTY
		30	OTHERS

Using "decode()" we are decoding "number" into string (or) "Single character" into string.
NOTE: If we want to modify data conditionally within a table using " SQL" then we must use "Decode()" function.
Eg:
Q) Write a Query to update "comm." of the emp's in emp table based on following condition:
1. If job= 'CLERK' then update comm into 10% of sal.
2. If job= 'SALESMAN' then update comm into 20% of sal.
3. If job= 'ANALYST' then update comm is 30% of sal
4. Else comm is 40% of sal.
Ans: SQL> update emp set comm= (decode(job, 'CLERK', sal*0.1, 'SALESMAN', sal*0.2, 'ANALYST', sal*0.3, sal*0.4));
In Oracle, if we want to display aggregate functional values in tabular format and also if we want to display those values conditionally using PIVOTING(rows are converted into columns). Then we must use "Decode()" with in "Group By" clause.
Eg: SQL> select job, sum(decode(deptno, 10, sal)) "deptno 10",
sum (decode(deptno,20,sal)) "deptno 20",
sum (decode(deptno,30,sal)) "depnto 30" from emp group by job;
Output:

JOB	DEPTNO 10	DEPTNO 20	DEPTNO 30
ANALYST		6400	
CLERK	1700	2700	1350
MANAGER	2650	3175	3050
PRESIDENT	5200		
SALESMAN			5800

Eg: SQL> select dname, sum(decode(job, 'CLERK', 1,0)) "CLERKS"
sum (decode(job, 'SALESMAN',1,0)) "salesman"
sum (decode(job, 'ANALYST', 1,0)) "analyst"
from emp e, dept d
where e.deptno=d.deptno
group by dname;
Output:

DNAME	CLERKS	SALESMAN	ANALYST
ACCOUTING	1	0	0

RESEARCH	2	0	2
SALES	1	4	0

Case Statement:

Case statement also used to "Decoding" the values. "Case Statement" performance is very high compare to decode conversion function. Oracle 8.0, introduce "Case statement" and also "Oracle 8i" introduced case conditional statement. This is also called as " Searched Case Statement".

NOTE: Decode internally used "Equality operator" (=), where as in "Case statement" we can also use all SQL operator (<,>,<=,>=,<>,Like, is ,etc……) explicitly.

METHOD 1:

Syntax:

```
case columnname
when value1 then statement1
when value2 then statement2
--------------------------------
……..
else statements end;
```

Eg: SQL> select ename, sal, deptno, case deptno
when 10 then 'ten'
when 20 then 'twenty'
else 'others' end
from emp;

OUTPUT:

ENAME SAL DEPTNO DEPTNO DECODE

METHOD 2: (Case Coditional Statement (8i))

Syntax:

```
case
when columncondition1 then statement1
when columncondition2 then statement2
……
……
else statements end;
```

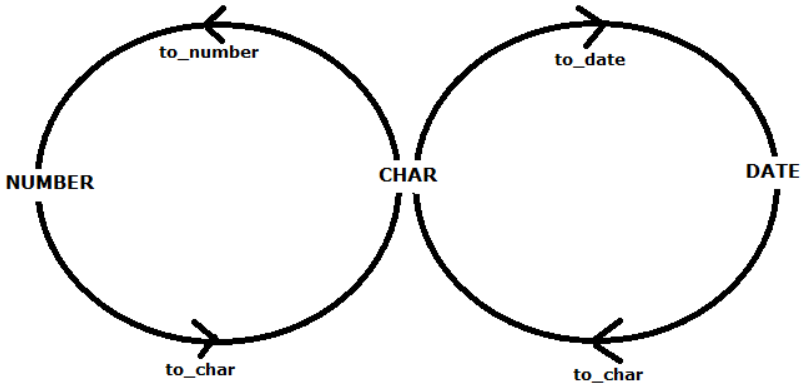
Eg: SQL> select ename, sal,
Case
when sal<1000 then 'Low Salary'
when sal between 1000 and 3000 then 'Medium Salary'
when sal in(3100,3200,3500,3800) then 'Special Salary'
else 'Other Salary' end
from emp;

Date: 11/5/15

Oracle also having following explicit conversion functions.

1. to_number()

2. to_char()
3. to_date()



1. **to_number():** It is used to converting a "String" representing numeric value with format into numeric value without format.

Eg: SQL> select '$45.6'+3 from dual;
Error: invalid number
SQL> select '45.6'+3 from dual;
O/P: 48.6
SQL> select to_number('$45.6')+3 from dual;
Error: Invalid number

NOTE: When ever we are using "to_number()" and also if we try to convert "String" representing numeric value with format into numeric value without format then we must use "second parameter" as same as "first parameter" format. Then only oracle server automatically converts "String type" to "Number type".
Solution:
SQL> select to_number('$45.6', '$99.9')+3 from dual;
O/P: 48.6

Not for other character:
Eg: SQL> select to_number('a45.6', 'a99.9') +3 from dual;
Error: Invalid number format model

2. **to_char():** to_char() is an "Over Loading" function which is used to convert "Number type" into "Character type" and also used to convert "Date type" into "Character type" or "Date String ".

Converting "Number type" into "Character type":
Syntax: to_char(number, 'character format')

FORMAT ELEMENTS:
9 -> representing Number
g -> Group separator (,)
d -> Decimal Indicator (.)

L -> Local Currency
$ -> Dollar ($)

Group Separator (,) and Decimal Indicator (.):
Eg: SQL> select to_char(1234567, '99g99g999') from dual;
O/P: 12,34,567
OR
SQL> select to_char(1234567, '99,99,999') from dual;
O/P: 12,34,567

Combination of both g & d:
SQL> select to_char(1234567, '99g99g999d99') from dual;
O/P: 12,34,567.00
OR
SQL> select to_char(1234567, '99,99,999.99') from dual;
O/P: 12,34,567.00

Leading Zero:
Eg: SQL> select to_char(123, '0999') from dual;
O/P: 0123
SQL> select to_char(123, '9990)) from dual;
O/P: 123
SQL> select to_char(123, '$999') from dual;
O/P: $123

Local Currency:
Eg: SQL> select to_char(123, 'L999') from dual;
O/P: $123

NOTE: If we want to display our own currency then we are using "nls_currency" parameter in third parameter of to "to_char()" function. In this case this parameter must be specified within single quotes. Before we are using this currency then we must use "Local Currency(L)" in the format model of the second parameter.
Eg: SQL> select ename, to_char(sal, 'L99g99g999d99', 'nls_currency = IndRs') from emp;
Output:

ENAME	SAL
SMITH	INDRS 800.00
ALLEN	INDRS 1,400.00

Eg: SQL> select ename, nvl(mgr, 'no manager') from emp;
Error: Invalid manager
Solution:
SQL> select ename, nvl(to_char(mgr), 'no manager') from emp;

Output:

```
ENAME      TO_CHAR(MGR), 'NO MANAGER'
-----------------------------------------------
KING       NO MANAGER
```

Eg: SQL> select to_char(sysdate, 'DD/MM/YYYY') from dual;
O/P: 11/05/2015

3. **to_date():** It is used to convert "Date String" into "Date Type".

Eg: SQL> select '15/AUG/05'+3 from dual;
Error: Invalid Number
Solution:
SQL> select to_date('15/AUG/05')+3 from dual;
Output: 18/AUG/05

NORMALIZATION

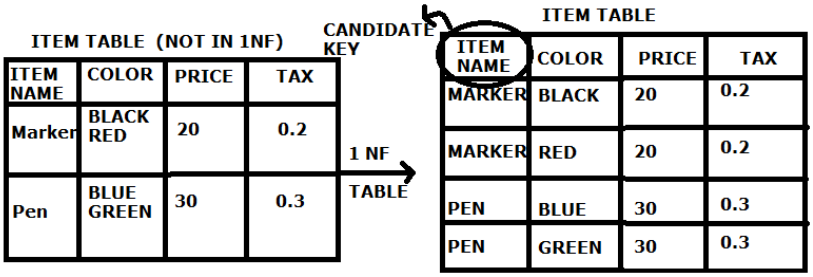
Normalization is an "Scientific Process" which is used to decomposing a table into number of tables. Normalization process automatically avoids insertion, updation, deletion problems and also this process reduces duplicate data.
In design phase of the SDLC, database designers designs "Logical Model" of the database. In this logical model only they are using "Normalization Process" through "Normal Forms".
In 1970's, E.F.CODD written a paper "Relational Model Data For Large Shared Data Banks". In this paper only E.F.CODD introduced first three "NORMAL FORMS".

NORMAL FORMS:
1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF
5. Fourth Normal Form
6. Fifth Normal Form

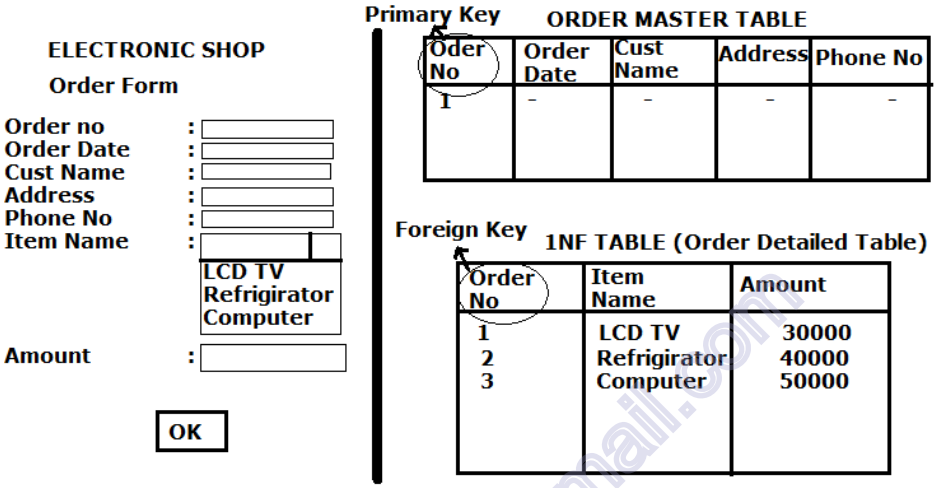
Date: 12/5/15

1. FIRST NORMAL FORM(1NF):

If a table is in 1NF, in that table each column contain a "Atomic value" and also identifying record "Uniquely" using a "Key".



Process: Identifying a "Repeating Groups" and put it into separate table in more "Atomic form". By default, 1NF process a table is an "Child table". Because in this table one column having duplicate data.

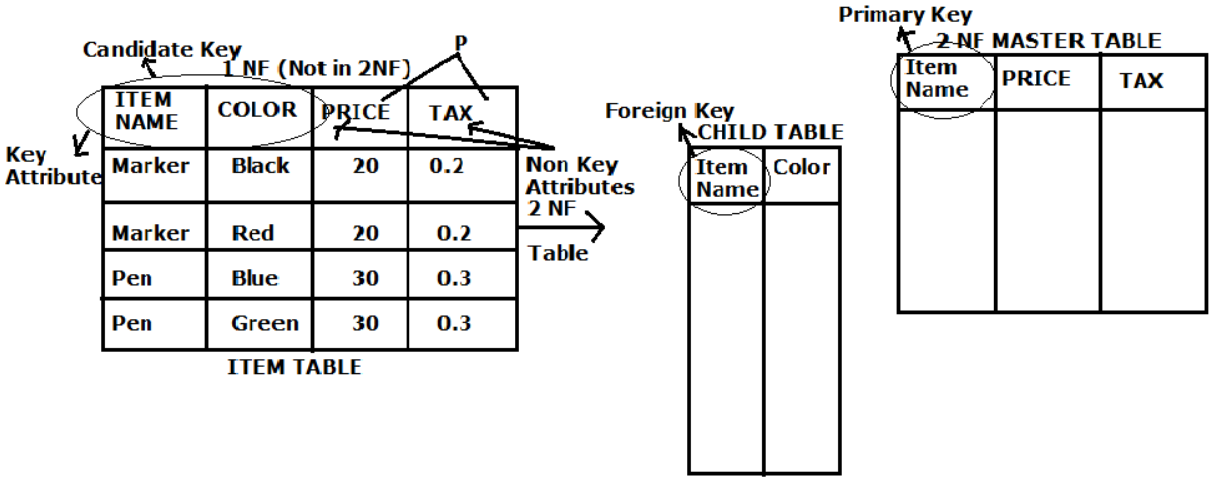


2. SECOND NORMAL FORM(2NF):

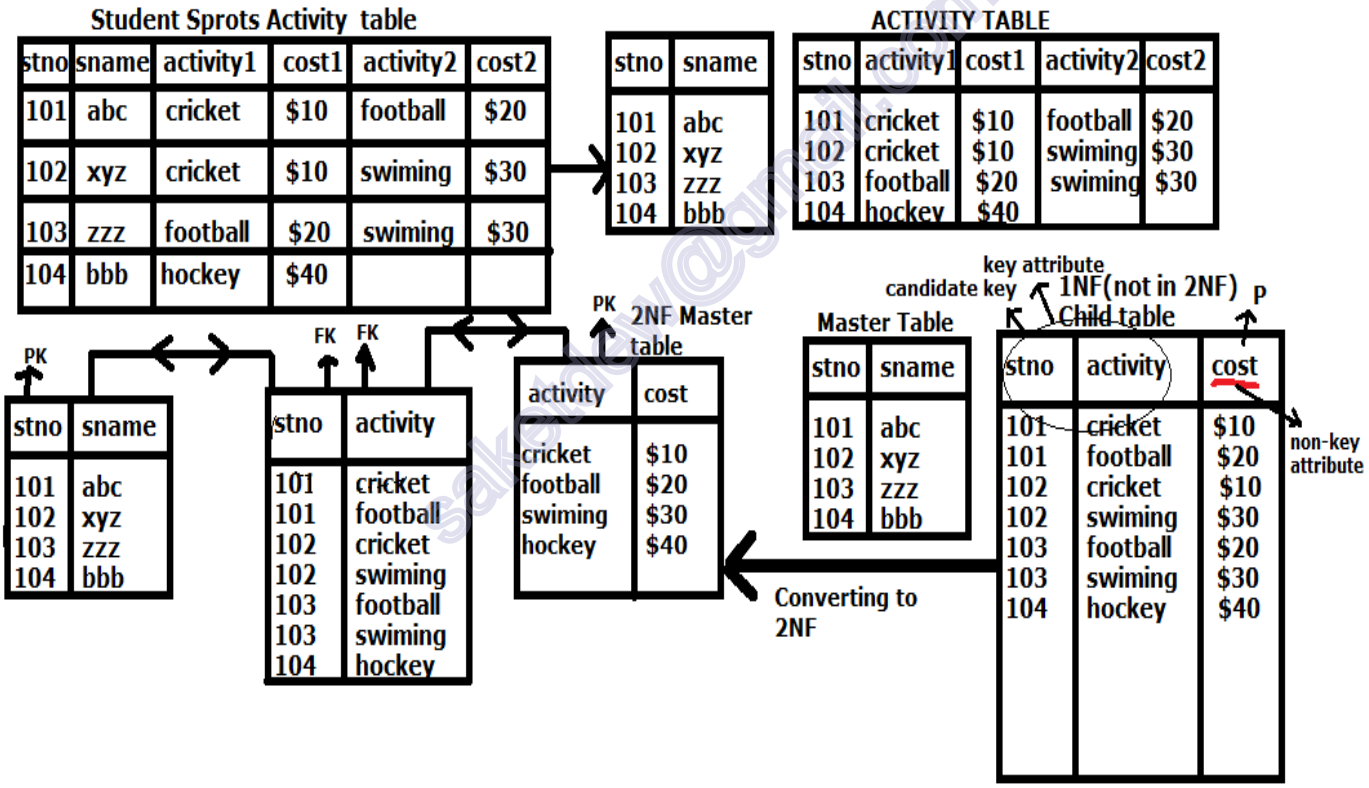
If a table is in 1NF and also all " Non key attributes" are fully functionally dependent on "Total Candidate Key". Always 1NF deals with "Atomicity" where as "2NF" deals with relationship between "Key and Non Key attributes".

If a table is in 1NF and also that table contains partial "Non Key attributes" then that table not in "2NF".

Process: Identifying partial non key attributes which depends on "Partial Key attributes" put into separate table. This table is called "2NF" table. By default this table is an "Master table". In this table only all "Non Key attributes" are "Fully Functionally dependent on total candidate key".


Candidate Key
1 NF (Not in 2NF)
P
ITEM NAME
COLOR
PRICE
TAX
Key Attribute
Marker
Black
20
0.2
Marker
Red
20
0.2
Pen
Blue
30
0.3
Pen
Green
30
0.3
ITEM TABLE
Non Key Attributes
2 NF
Table
Foreign Key
CHILD TABLE
Item Name
Color
Primary Key
2 NF MASTER TABLE
Item Name
PRICE
TAX

Date: 13/5/15


Student Sprots Activity table
stno
sname
activity1
cost1
activity2
cost2
101
abc
cricket
$10
football
$20
102
xyz
cricket
$10
swiming
$30
103
zzz
football
$20
swiming
$30
104
bbb
hockey
$40
ACTIVITY TABLE
key attribute
candidate key
1NF(not in 2NF)
Child table
P
non-key attribute
PK
FK
FK
2NF Master table
Master Table
Converting to 2NF

Ecode	Projcode	Dept	Depthead	Hours
E101	P1	Systems	D1	4
E102	P1	Sales	D2	5
E103	P1	Admin	D3	7
E101	P2	Systems	D1	9
E102	P2	Sales	D2	10
E104	P2	Researcl	D4	12
E102	P3	Sales	D2	8
E104	P3	Researcl	D4	6

Phase 1:

1. Ecode ---(<---FD----)---→ Hours(Worng) E101--→ 4,9
2. Projcode----(<---FD---)-→ Hours(Wrong) P1-→ 4,5,7
3. Ecode+Projcode ---→ Hours(Correct)

F -> Fully Functional Dependent

Phase 2:

1. Ecode----(<---FD---)--→ Dept(Correct) E101--→ Systems
2. Projcode-----(<---FD---)-→ Dept(Wrong) P1--→ systems, sales, admin

Once partial dependencies is satisfied we want to go for "Fully Functional Dependent".

Phase 3:

1. Ecode----(<---FD---)--→ Depthead(Correct)
2. Projcode--→ Depthead(Wrong)

Primary key

2NF Table(Master Table)

Ecode	Dept	Depthead
E101	Systems	D1
E102	Sales	D2
E103	Admin	D3
E104	Research	D4

CHILD TABLE

Ecode	Projcode	Hours
E101	P1	4
E102	P1	5
E103	P1	7
E101	P2	9
E102	P2	10
E104	P2	12
E102	P3	8
E104	P3	6

Date: 14/5/15

Before Normalization process in the above research having Insertion, Updation, Deletion problems.

Insertion Problem: In the above resource table when we are trying to insert particular department employee then we must assign project code. If project code is not available then we need to supply null values for the project code field. This is called "Insertion Problem".

Updation Problem: In the above resouce table ecode, dept, depthead attribute values are repeated. Whenever resource table having large amount of data and also when we are modifying data then we must change these three values correctly. Otherwise, Inconsistency Problem occurs. This is called " Updation Problem".

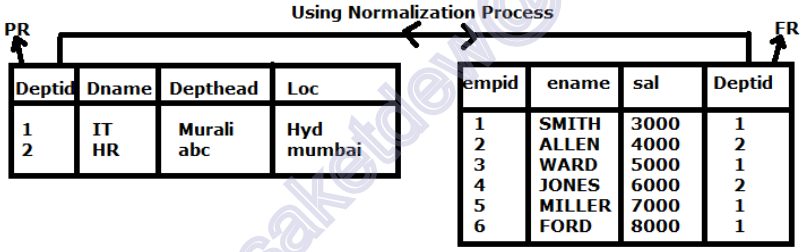
Deletion Problem: In the above resource table when we try to delete a employee record then automatically deparment details also deleted. This is called "Deletion Problem". Whenever we are using "Normalization Process" then automatically Insertion, Updation, Deletion problems avoided.

Without using Normalization Process

ENAME	SAL	DNAME	DEPTHEAD	LOC
SMITH	3000	IT	Murali	Hyd
ALLEN	4000	HR	abc	mumbai
WARD	5000	IT	Murali	Hyd
JONES	6000	HR	abc	mumbai
MILLER	7000	IT	Murali	Hyd
FORD	8000	IT	Murali	Hyd

Data Redundancy Problems

1. Disc space wasted
2. Data Inconsistency
3. DML Operations become slow



A Table is in "1NF":

1. Data in each column should be atomic:

Is there is no multiple values saperated by column (,).

2. Table does not contain repeating column groups.
3. Identifying a record uniquely using a "Primary Key".

1.

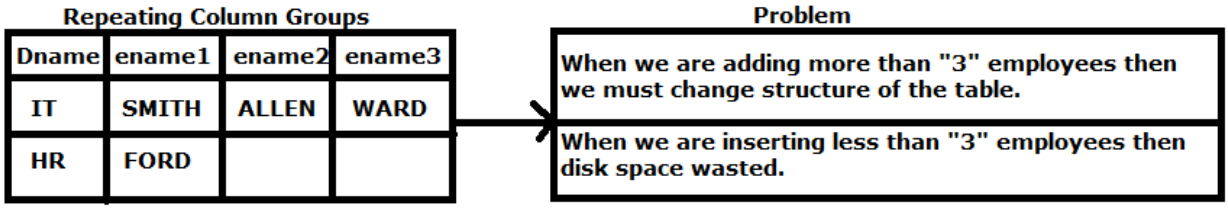
Non- Atom columns(emp)

DName	Ename
IT	SMITH, ALLEN, WARD
HR	FORD

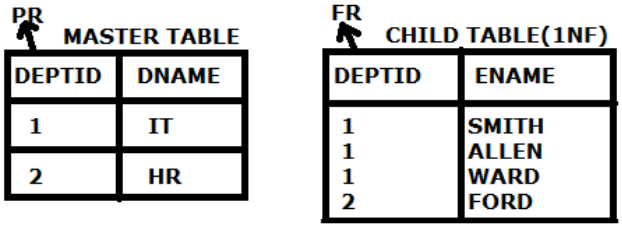
Problem

It is not possible to select, insert, update, delete a Single employee

2.



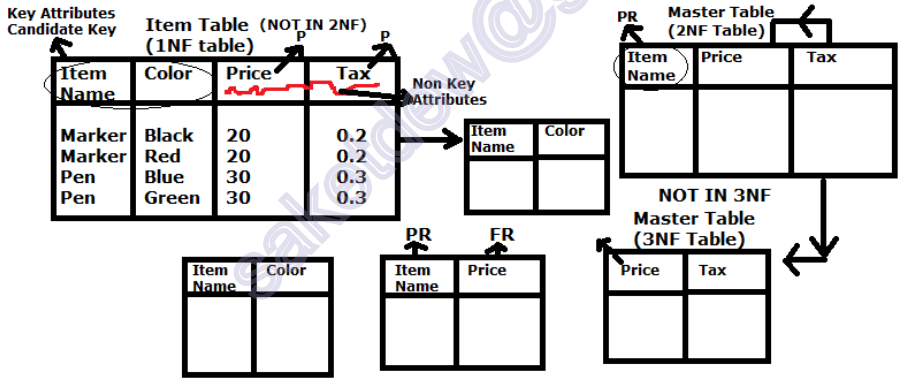
3. **SOLUTION:**

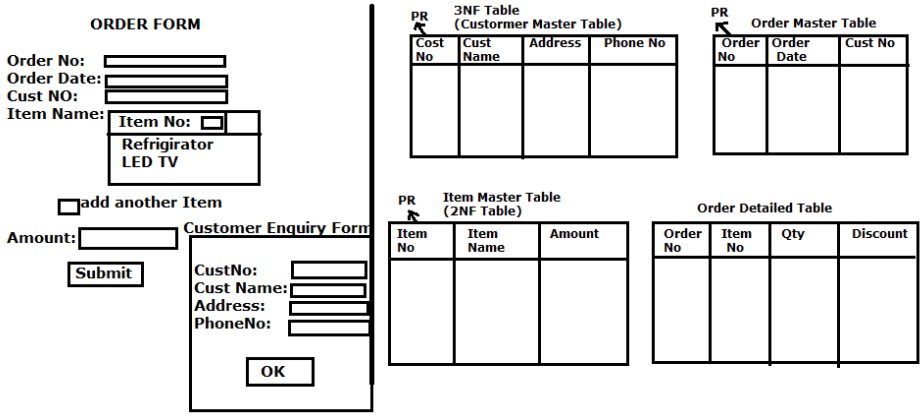


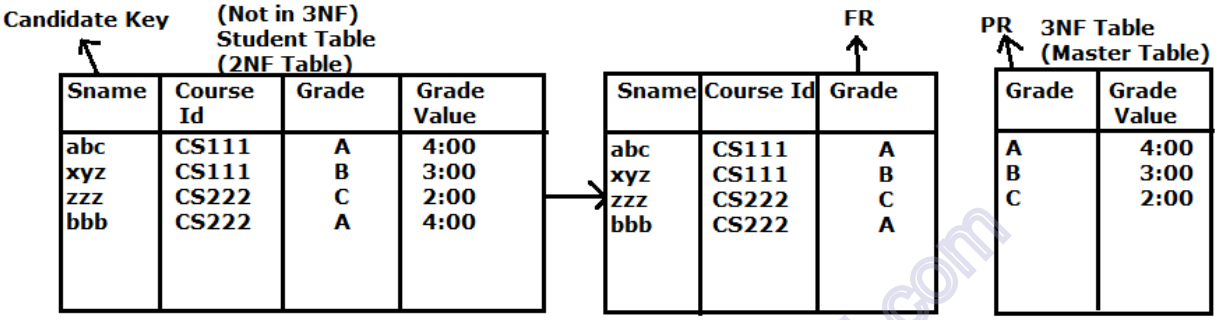
Date: 15/5/15

THIRD NORMAL FORM(3NF):

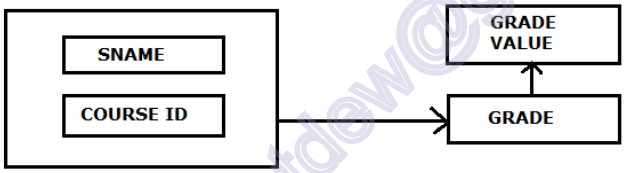
If a table is an 2NF and also all non key attributes are only dependent on "Candidate key".
If a table is in 2NF and also "Non Key Attributes" which dependents on another "Non Key Attributes" then that table not in "3NF".
Process: Identifying a "Non Key attributes" which depends on another "Non Key attributes" but into saperate table. This table is called "3NF" table. By default this table is an "Master table". In this table only all "Non key attributes" are only dependent on "Candidate Key".







LOGICAL DIAGRAM:



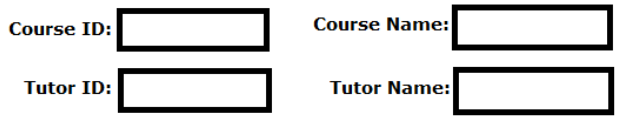
DATE: 16/5/15

1NF -> Remove Repeating Groups
2NF -> Remove Pratial Attributes
3NF -> Remove Attributes which are not dependent on Candidate Key.
(OR)
Remove Non-Key Attribute which are dependent on another Non-Key Attribute.

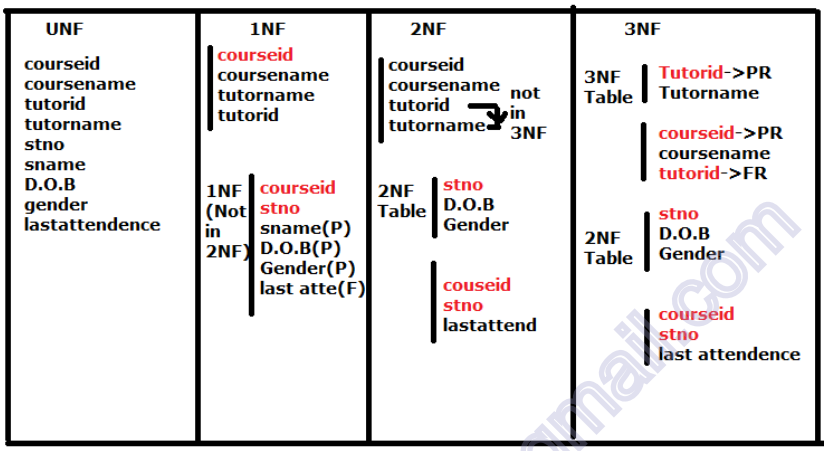
CLUSTERS

Cluster is an "Database Object" which contains group of tables together and it will "Shares same Data Blocks".
Generally, clusters are used to improves performance only.
In all database clusters tables must have a "Common Column Name". This common columns is also called as "Cluster Key".

In all databases, whenever we are submitting "Inner Join" (or) "Outer Join" then database servers checks "FROM" clause tables are available in clusters (or) not.
If those tables are available in cluster then database servers retrieve data very fastly from the "Cluster Tables".



stno	sname	D.O.B	Gender	Last Attendence



Creating Clusters:
Step 1: Create a Cluster
Step 2: Create an Index on Cluster
Step 3: Create Cluster Tables.

Step 1: Create a Cluster:
Clusters are created based on common column. This common column is also called as "Cluster Key".
Syntax: create cluster clustername(common colname datatype(size));

Step 2: Create an Index on Cluster:
Syntax: create index indexname on cluster clustername;

Step 3: Create Cluster Tables:
Syntax:
create tablename(commoncolname datatype(size), col1 datatype(size),….)
cluster
clustername(commoncolname);

Eg:
Sql> create cluster emp_dept(deptno number(10));

Sql> create index in1 on cluster emp_dept;
Sql> create table emp1(empno number(10), ename varchar2(10), sal number(10), deptno number(10)) cluster emp_dept(deptno);
Sql> create table dept1(deptno number(10), dname varchar2(10), loc varchar2(10)) cluster emp_dept(deptno);
Sql> desc emp1;
Sql> desc dept1;
Sql> insert into emp1 values(1, 'abc', 2000, 10);
Sql> insert into dept1 values(10, 'xyz', 'hyd');
Sql> select * from emp1;
Sql> select * from dept1;
NOTE: In all database system cluster tables having some "Rowid's".
Eg: sql> select rowid from emp;
Sql> select rowid from dep1;
Generally, we cannot drop cluster if cluster having tables. To overcome this problem Oracle 8i introduced to drop cluster along with tables using "Inclusing tables" clause.
Syntax:
Sql> drop cluster clustername including tables;
Eg: sql> drop emp_dept including tables;
Cluster dropped;

Date: 18/5/15

In Oracle all Clusters information stord under "user_clusters" data dictionary.
In all relational databases using Clusters we can achieve physical data independence, because whenever we are adding clusters table structure does not effects but only performance is effected when we are using "joins".

Candidate Keys:
1. Empno+skill
2. Ename+skill
3. Voterid+skill

empno	ename	skill id	skill	voter id
1	murali	1	Oracle	v1
1	murali	2	Sysbase	v1
1	murali	3	Ingress	v1
2	abc	4	DB2	v2
2	abc	1	Oracle	v2
3	xyz	5	Informix	v3
4	zzz			v4
5	bbb	6	SQL server	v5
5	bbb	4	DB2	v5

DATABASE DESIGNERS

Superkeys:
1. empno+skill
2. empno+ename+skill
3. empno+voterid+skill
4. empno+skill
5. ename+voterid+skill
6. voterid+skill
7. empno+ename+voterid+skill

Not a Super Key:
Eg:
1. empno+ename
2. empno+voterid
3. ename+voterid

SUPER KEY: A column (or) combination of columns which uniquely indentifying a record in a table is called "Super Key".
CANDIDATE KEY: A minimal "Super Key" which uniquely identifying a record in a table is called "Candidate Key"

(or)
A "Super Key" which does not contains another "Super Key" is also called as "Candidate Key".

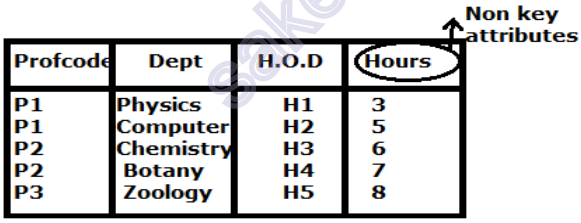
FUNCTIONAL DEPENDENCY(FD):
If any given two tuples in a relation "r" then if X(an attribute set) agrees then Y(another attribute set) cannot disagree then only X->Y is called "Functional Dependency".
X->Y here Y is Functionally Dependent on X
X functionally determines Y. (here X is also called a "Determinent" and Y is also called as "Dependent")

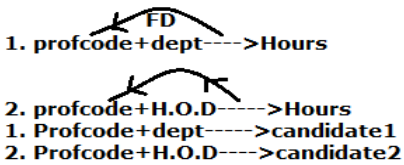
BCNF: When a table is in BCNF then every determinant is a "Candidate Key".
When a table contains multiple composite candidate keys and also those candidate keys are overlapped and also one candidate key non key attribuite which depends on another candidate key non key attribute then only we are using BCNF process.

Generally, when a table contains multiple composite candidate keys then deletion problem occurred to overcome this problem database designers uses "Boyce Code Normal Form" in database design.

Generally, second and third normal forms deals with relationship between key and non key attributes. Where as BCNF deals with relationship between non key attributes within composite candidate keys itself.

PROCESS: Identify one composite candidate key non key attributes which depends on another composite candidate key non key attributes put into saperate table. This table is also called as "BCNF table". In this table only every determinant behaves like a candidate key and also in this BCNF table. There like a candidate key and also in this BCNF table. There is no non key attributes.



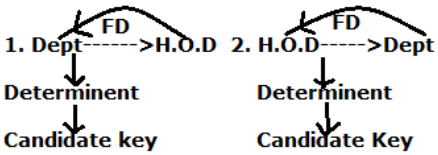


BCN TABLE

Dept	H.O.D
Physics	H1
Computer	H2
Chemistry	H3
Botany	H4
Zoology	H5

TABLE

Profcode	Dept	Hours
P1	Physics	3
P1	Computer	5
P2	Chemistry	6
P2	Botany	7
P3	Zoology	8



NOTE: If a table contains single composite candidate key then BCNF is also same as 3NF.

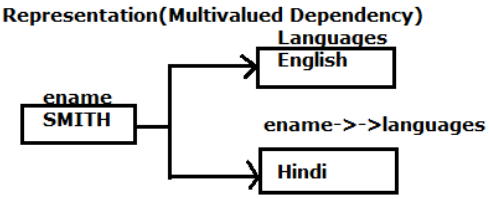
Date: 19/5/15

MULTIVALUED DEPENDENCY:

In a table, if a determinant determines multiple values in another column within same table is called "Multivalued dependency". Multivalued dependency represented by using Double Arrow marks ("->->").

➔ Single Value Multiple Columns

ename	Languages
SMITH	English
SMITH	Hindi



FOURTH NORMAL FORM(4NF): If a table is in 4NF then that table doesnot contain more than one independent multivalued attributes.

If a table contain minimum three(3) attributes and also using combination all these attributes uniquely identification of the record and also one attribute set of values which depends on another attribute set of values and also same attribute set of values which depends on another attribute set values and also some attributes are not logically related then only we are using 4NF process.

PROCESS: Identify independent multivalued attributes put it into saperate tables. These tables are called 4NF tables these tables does not contain more than one independent multivalued attributes.

Movie Table

Movie	Star	Producer
M1	S1	P1
M1	S2	P1
M1	S3	P1
M1	S4	P1
M1	S5	P2
M1	S6	P2
M2	S7	P3
M2	S2	P3
M2	S3	P3

4NF table

Movie	Star

4NF Table

Movie	Producer

Generally, when a table contains more than one independent multivalued attribute. Then those tables having more duplicate data to overcome this problem for reducing duplicate data then database designers uses 4NF process in this process one table does not contain more than one independent multivalued attribute.

ASSUMPTION: Each employee has multiple projects and also each employee having multiple skills.

EMPTable

empid	ProjectID	Skill
1111	RailwayResv.	JSP
1111	RailwayResv.	Oracle
1111	RailwayResv.	.net
1111	LibraryMgmt.	PHP
1111	LibraryMgmt.	Sql Server

4NF Table

empid	ProjectID
1111	RailwayResv.
1111	LibraryMgmt.

4NF Table

empid	Skill
1111	JSP
1111	Oracle
1111	.net
1111	PHP
1111	Sql Server

In the above resource table whenever an employee got a new project then we have to supply null values for the skill attribute and also whenever an employee got new skill then we have to supply null values for the 'project id' attribute. These null value problems are automatically avoided when we are using 4NF process.

FIFTH NORMAL FORM(5NF):

If a relation cannot be decomposed into further relations is called 5NF i.e., if a table is in 5NF the table doesnot contains cyclic dependencies.
Generally, if a table contains more than two attributes and also identifying a record using combination of all these attributes and also all these attributes set of values related to one another then only we are using 5NF.

NOTE: When a table contains multivalued attributes and also some attributes are not logically related when only we are using 4th NF process where as when a table contains multivalued attribute and also all attributes are logically related then only we are using 5th NF process.

5TH Normal Form is also called as "Projection Joined Normal Form" because in 5th NF if possible then we can decomposing a table into number of tables but when we are joining those tables then resultant record must be available in resource table.

Eg:

Product Table

Buyer	Product	Company
B1	Shirt	Levi's
B1	Jeans	Arrow
B1	Shirt	Pepe

5th Normal Form Tables

Buyer	Product

Product	Company

Buyer	Company

ORACLE SERVER ARCHITECTURE

Oracle server mainly consist of two parts:
1. Storage Area
2. Instance

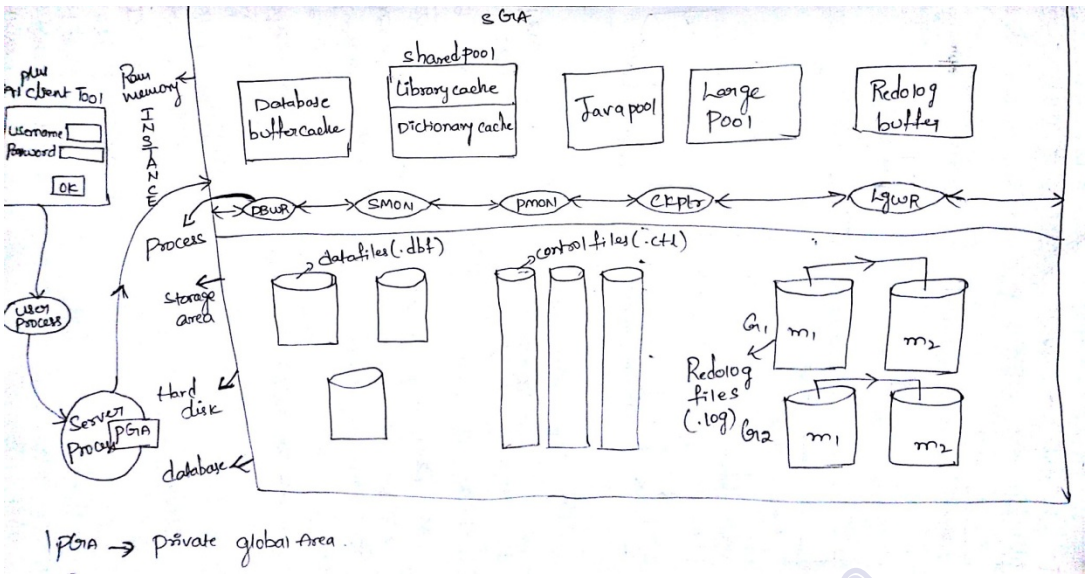
1. Storage Area: Whenever we are installing oracle server automatically three physical files are created in harddisk these files are called as Storage Area (or) Physical database.
1. Data files(.DBF)
2. Control files(.ctl)
3. Redolog files(.log)

All databases having two types of structures
1. Logical Structure
2. Physical Structure

LOGICAL STRUCTURE: A structure which is not visible in operating system is called "Logical Structure".
-> Logical structure contains database objects like table, views, synonyms, clusters, sequences.
-> Logical structure is handled by either developers (or) database administrators.

PHYSICAL STRUCTURE: A structure which is visible in operating system is called Physical structure. Physical structure contains datafiels, control files, redolog files.
-> Physical structure is handled by database administrator only.

Date: 20/5/15



Data Files: Data files stores database objects physically that is these files stores tables, views, sequences, indexes permanently into hard disk these files extension is ".dbf".
-> In Oracle all datafiles information stored under "dba_data_files" data dictionary.
Eg: sql> conn sys as dba
Enter password: sys
Sql> desc dba_data_files;
Sql> select file_name from dba_data_files;
CONTROL FILES: Control files controls all other files within storage area these files extension is ".ctl".
-> Control files also stroes physical database information these fiels are used by database administrator in backup and recovery process.
-> In Oracle, all control files information stored under "v$control file" data dictionary.
Eg: sql> desc v$ control file;
Sql> select name from v$control file;

REDOLOG FILES: Redolog files stores commited data from the redolog buffer. Redolog files also used by database administrator in backup and recovery process.
-> In Oracle, all log files information stored under "v$logfile" data dictionary.
Eg: sql> desc v$logfile;
Sql> select member from v$logfile;

INSTANCE: Whenever we are performing any operations within database those all operations also first performed in Physical memory area. This Physical Memory area is also called as "Instance". This Instance is available in RAM memory area this instance having two parts.
1. SGA
2. Processes

SGA: SGA is also called as System Global Area (or) Shared Global Area. SGA memory area consist of set of buffers these are

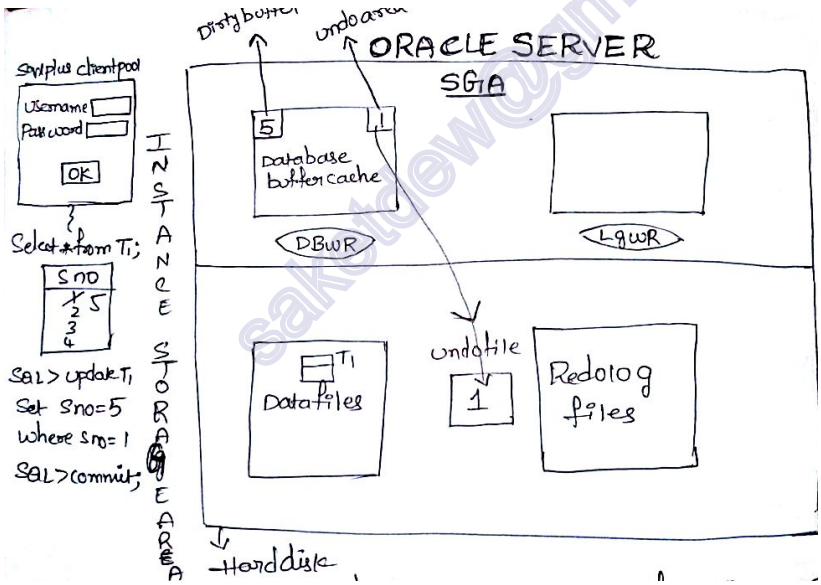
1. Database buffer cache
2. Shared pool
3. Java pool
4. Large pool
5. Redolog buffer

-> Whenever we are submitting sql, pl/sql code that code is stored in library cache within shared pool Library code always reduces parsing. Library cache also stores cache values defined in Sequence database object.

-> Shared Pool also contains dictionary cache which executes "dcl" related objects and also checks username and passwords given by the client connect to the server.

-> Whenever we are requesting a table using a client tool then server process always checks requested table is available in database buffer cahce. If requested table is not available in buffer cache then dbwriter process checks requested table is available in data files. If requested table is available in data files then copy of the table transferred into database buffer cache. Then only server process fetching that table information from database buffer cache into client tool.

-> Java Pool executes java related objects and also large pool performs large number of operations and also redolog buffer stored new information for the transaction.



UNDO: In Oracle, when we are performing dml transactions then new values for the transaction is automatically stored in dirty buffer within database buffer cache and also old value for the transaction is automatically stored in undo area within buffer cahce. Whenever user using commit. Then automatically new value for the transaction is stored in data files within storage area and also old values for the transaction is automatically stored in UNDO file within storage area.

-> In Oracle 9i, onwards we can also retrieve accidental data after commiting the transaction using flashback queries these flashback queries internally uses UNDO file.

Date: 21/5/15

REDO: Whenever we are performing transactions new value for the transaction is automatically storing "Redolog" buffer. Whenever "User" using "Commit (or) 1/3 fill of Redolog buffer" the Redolog buffer data automatically transferred into Redolog files. These Redolog files used by database administrator in backups or recovery process.

PROCESSESS: Oracle Instance internally having following background processes. These are:

1. Dbwr(database writer)
2. Lgwr(log writer)
3. Ckptr(check pointer)
4. Pmon(process monitor)
5. Smon(system monitor)

1. **DBWR:** Dbwr process fetches data from data base buffer cache and store that data in data files within storage area.
2. **LGWR:** Lgwr process fetches data from Redolog buffer and that data stored into "Redolog files".
3. **CKPTR:** Whenever data base writer perform some task then check pointer automatically generates an unique identification number for every transaction. This number is automatically stored in data files, control files, Redolog files. This number is also called as System Change Number(SCN). This number is used by data base administrator in "Flash Back Querys". Flash Back querys retrieves accidental data based on a specific point of time. After commiting the trasactions only in "Flash back queries" we are using as of clauses.

   Syntax:

   Sql> select * from tablename as of scn system change number;

   We can also view system change number using "current_scn" property from "v$database" data dictionary in sys as dba user only.

   Eg:

   Sql> conn sys as sysdba
   Enter the Password: sys
   Sql> create table b1(sno number(10);
   Sql> desc v$database;
   Sql> select current_scn from v$database;

   Output:

   CURRENT_SCN

   1012963
   Sql> insert into b1 values(50);
   Sql> commit;
   Sql> select count(*) from b1;

Output: 1

Sql> select count(*) from b1 as of scn 1012963;

Output: 0

4. **PMON:** Process monitor cleans up user processes if users does not disconnect properly from the server.
5. **SMON:** System monitor process recovery all processes. When ever problems are occurred in an process. That's why system monitor is also called as "Instance Recovery".

PGA: (Private Global Area)

In Oracle, server process contains a memory area which uniquely indentified by each client connect to the server. This memory area is also called as "Private Global Area". In pl/sql all connections are executed within PGA memory area.

Hierarchical Query's:

In Relational Databases we can also stores hierarchical data. If you want to store hierarchical data then minimum three columns required in a table. And also in those three columns tow columns must belongs to same data type. And also having some relation these two columns are also called as "Hierarchical Columns".

If you want to retrieve hierarchical data from a relational table then we are using following clauses.

1. Level
2. Start with
3. Connect by

1. **LEVEL:** Level is a Pseudo column which automatically assigns numbers to level in a tree structure. Levels are also starting with Level number "1" (one).
2. **START WITH:** Using "Start with" clause specifies starting condition with in tree structure.

Date: 22/5/15

Syntax: start with condition

3. **CONNECT BY:** Using "Connect BY" clause we are specifying relationship between hierarchical columns using "Prior" operator.

Syntax: connect by prior parent columnname=childcolumnname;

Syntax:

select level, columnname from tablename
where condition
start with condition
connect by prior parentcolname=childcolname;

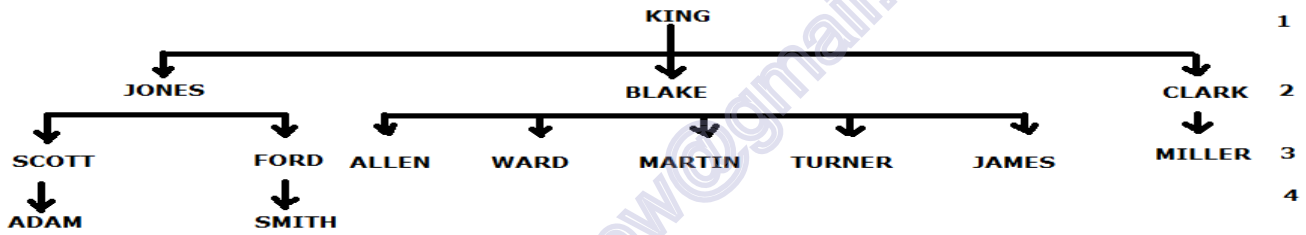
Q) Write an hierarchical query to display the employees who are working under other employees route node onwards from emp table?

ANS:

sql> select level, ename from emp
Start with mgr is null
Connect by prior empno=mgr;

OUTPUT:

LEVEL ENAME
------ ----------
1 KING
2 JONES
3 SCOTT
4 ADAMS
3 FORD
4 SMITH
2 BLAKE
3 ALLEN
3 WARD
3 MARTIN
3 TURNER
3 JAMES
2 CLARK
3 MILLER



NOTE: Oracle 9i, introduced "sys_connect_by_path()" function which returns path of the hierarchy in tree structure. This function accepts two parameters.

Syntax:

Sys_connect_by_path(colname,'delimiter name');

Eg:

```
sql> select level, sys_connect_by_path(ename,'->')
from emp
start with mgr is null
connect by prior empno=mgr;
```

Q) Write a hierarchical query to display the employees who are working under "BLAKE" from emp table?

ANS:

```
sql> select level, sys_connect_by_path(ename,'->')
from emp
start with ename='BLAKE'
connect by prior empno=mgr;
```

OUTPUT:

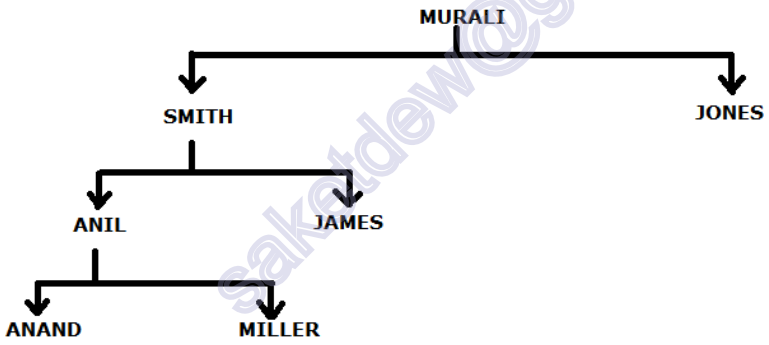
LEVEL ENAME

1->BLAKE
2->BLAKE->ALLEN
2->BLAKE->WARD

PRIOR: Prior is an "Unary Operator" used along with "Connect By" clause. When we are using prior operator infront of the parent column (mgr) then oracle server uses bottom to up search within tree structure where as when we are using prior operator infront of the child column then oracle server uses top to bottom search within tree structure.
EXECUTION: Based on the "Start with" condition oracle server take a "Prior" operator column value and that value is searching in another hierarchical column. Then only corresponding column data retrieve from the table.

Q) Create a hierarchical table based on the following tree structure?
ANS:
Sql>create table test(ename varchar2(10), empno number(10), mgr number(10));
Sql> insert into test values('anand',6000,8000);
Sql> select * from test;
OUTPUT:

ENAME	EMPNO	MGR
ANAND	6000	8000
MILLER	6001	8000
ANIL	8000	9000
JAMES	8001	9000
SMITH	9000	10000
JONES	9001	10000
MURALI	10000	NULL



Mainly, three conditions to build hierarchical table:
1. Table must contain 3 columns
2. One column must have a "NULL" value(for root node)
3. Any two columns must be same data type and interrelated (i.e., one column all values must be in another column for example "mgr" column is parent column and "empno" column is child column because parent column consist of "NULL" value)

NOTE: Whenever we are using "Order By" clause in hierarchical query's database servers sorting the data but changes the hierarchical structure within tree structure. To overcome this problem Oracle introduced "Order Siblings By" clause within hierarchical query's.
Syntax:

order siblings by columnname[asc/desc]
Eg:
Sql> select level, ename
from emp
start with mgr is null
connect by prior empno=mgr
order siblings by ename;

NOTE: Oracle 10g, introduced "connect_by_root" operator which returns root node in the hierarchy.
Syntax: connect_by_root columnname;
Eg:
Sql>select level, ename, connect_by_root ename
from emp
start with mgr is null
connect by prior empno=mgr
order siblings by ename;

Date: 23/5/15

TRANSACTION CONTROL LANGUAGE(TCL):
Transction: A logical unit of work in between two points is called "Transaction".
All database systems having 2 transactional command.
1. Commit
2. Savepoint

1. COMMIT: This command is used to save the transaction permanently to database.
Syntax: commit;
2. SAVEPOINT: A logical mark between transactions is called "Savepoint".
Syntax: savepoint savepointname;
3. ROLLBACK: This command is used to "UNDO" the transactions from memory.
Syntax: rollback;
ROLLBACK TO PERTICULAR SAVEPOINT:
Syntax: rollback to savepointname;
Eg: Sql> insert into emp(empno) values(1);
Sql> update emp set sal=sal+100 where ename='SMITH';
Sql> savepoint s1;
Sql> insert into emp(empno) values(2);
Sql> update emp set sal=sal+100 where ename='KING';
Sql> savepoint s2;
Sql> delete from emp where empno=2;
Sql> rollback to s1;
Sql> commit;
Sql> select * from emp;

NESTED TABLE:

Oracle 8.0, introduced nested table. A table within another table is also called as "Nested Table". Basically, nested table is an user defined datatype which is used to store number of data items in a single unit.

Generally, if we want to store number of data items in a single unit then we are using Index by table in PL/SQL. But these tables are not allowed to store permanently into database. To overcome this problem Oracle 8.0 introduced extension of the Index by tables called "Nested Table" which stores number of data items and also these tables are allowed to store permanently into oracle database using SQL.

CREATING NESTED TABLE:

Step 1: create an Object type.
Step 2: create an Nested table type
Step 3: create an Actual table.

Step 1: Create an Object Type:

Object Type: Object type is an user defined type which stores different data types into single unit. It is also same as "Structures" in "C language".

Syntax:

create or replace type typename
as object (attribute1 datatype(size), attribute2 datatype(size),…….);

Step 2: Create an Nested Table Type:

Create an Nested table type by using Object type.

Syntax:

Create or replace type typename
as table of objecttypename;

Step 3: Create an Actual Table:

Syntax:

create table tablename(col1 datatype(size),….,coln nestedtabletype)
nested table coln store as anyname;

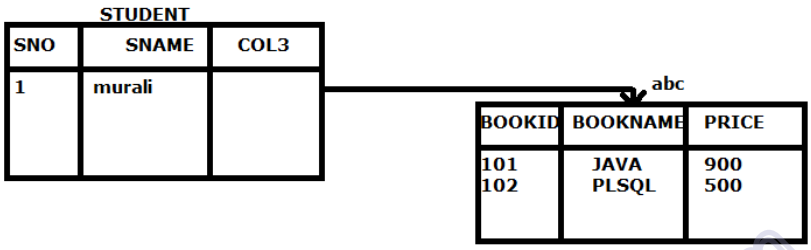
Eg: sql> create or replace type obj1
as object(bookid number(10), bookname varchar2(10), price number(10));
/
Sql> create or replace type
abc as table of obj;
/
Sql> create table student(sno number(10), sname varchar2(10), col3 abc) nested table col3 store as zzzz;
Table Created.
Sql> desc student;
NAME TYPE

SNO NUMBER(10)
SNAME VARCHAR2(10)
COL3 abc

If we want to store data into nested table type then we must use constructor. Here constructor name is also same as type name.

INSERTION:

Sql> insert into student values(1,'murali',abc(obj1(101,'java',900), obj1(102,'plsql',500)));
1 row created.
Sql> select * from student;



We can also retrieve modify delete nested table data saperately by using table operator i.e., in place of table name we are using following syntax in select, update, delete statements.

Syntax:

table (select nested tablecolname from relational tablename);

Eg: sql> select * from table(select col3 from student);

OUTPUT:

```
BOOKID      BOOKNAME  PRICE
-------------------------------------
101         JAVA      900
102         PLSQL     500.
```

Eg: sql> update table(select col3 from student) set price=1000 where bookid=101;

OUPUT:

```
BOOKID      BOOKNAME        PRICE
-----------------------------------------
101         JAVA            1000
```

Date: 25/5/15

PARTITIONS

Partitions: At the time of table creation a table can be decomposed into number of partions is called "Partition Table". Partition Tables are used in dataware housing applications. Partition Tables are created by database administrator and used in back up and recovery process.

Partition Tables are used to improve performance of the applications in backup and recovery process. Partition Tables are created on very large data bases. Partition Tables are created based on key column. This is called "Partition Key".
Oracle having 3 types of partitions.

1. Range partition
2. List partition
3. Hash partition

1. **Range Partition:** In this partition we are partitioning tables based on range of values.

Syntax:

create table tablename(col1 datatype(size), col2 datatype(size),….)
partition by range(key colname)(partition partitionname values less than(value),…..,partition partitionname values less than(max value));

If we want to view particular portion then we are using following syntax:

Syntax:

select * from tablename partition(partitionname1, partitionname2,……);

Eg:

sql> create table test(sno number(10), name varchar2(10), sal number(10))
partition by range(sal)(partition p1 values less than(1000), partition p2 values less than(2000), partition p3 values less than(max value));
Table created
Sql> insert into test values(&no, '&name', &sal);
Sql> select * from test partition(p1);

OUTPUT:

SNO	NAME	SAL
1	ABC	500

2. **List Partition:** Oracle 9i, introduced list partition in this partition we are partitioning tables based on list of values.

Syntax:

create table tablename(col1 datatype(size), col2 datatype(size),…..)
partition by list(key colname)
(partition partitionname values(value1, value2,…),…….. partition partitionname values(default));

NOTE: Generally, if we want to create partition based on character datatype column then we are using "List Partition".

Eg:

Sql> create table test(sno number(10), name varchar2(10)) partition by list(name)
(partition p1 values('India','Nepal'), partition p2 values('us','uk','canada')
partition p3 values(default));
sql> insert into test values(&sno,'&name');

sql> select * from test;

OUTPUT:

SNO NAME

---

1 AUS

2 JAPAN

3 INDIA

Sql> select * from test partition(p1);

OUTPUT:

SNO NAME

---

1 AUS

2 JAPAN

3. HASH PARTITION:

In "Hash Partition" oracle server only creates partitions by using "Hash Algorithm".

Syntax:

create table tablename(col1 datatype(size), col2 datatype(size),……)
partition by hash(keycolname) partition anynumber;

Eg:

Sql> create table test(sno number(10), sal number(10)) partition by hash(sal) partition 5;
Table created
Sql> insert into values test(&sno, &sal);
If we want to view partitions then we are using "user_tab_partitions" data dictionary.

Eg:

Sql> desc user_tab_partitions;
Sql> select partition_name from user_tab_partitions where table_name='TEST';

END OF SQL(STRUCTURED QUERY LANGUAGE(SEQUEL)**)**