Course Route Map

Module – 1

- Introduction to data and its management
- Introduction to DBMS
- Introduction to RDBMS
- EF Codd 12 Rules

Module 2:

- SQL Commands:
- Basic SQL Commands (create, insert, update).
- DDL Commands in detail(create, alter, delete, truncate)
- Built in Functions
- Grouping the Result of a Query
- Working with Integrity Constraints
- Querying Multiple Tables (Joins)
- Working with Sub Queries
- Working with DCL, TCL Commands
- VIEWS in Oracle

Module - 3

- PL-SQL (Procedure Language – SQL)
- Procedures in PL/SQL
- Functions in PL/SQL
- Packages in PL/SQL
- EXCEPTIONS in PL/SQL
- Database Triggers in PL/SQL
- SQL QUERIES FOR PRACTICE

CHAPTER – 1

Introduction to data and its management

Topics:

- What is Data?
- How many ways the data can be managed?
- What is File or Flat File System?
- Disadvantages of flat file system

What is Data?

Different ways of managing the Data in Computer:

1. Using Files
2. Using DBMS

What is File?

➔ It is a collection of information stored in bytes/char. format.
➔ It was first backend database.
➔ It is managed by file system

Disadvantages of Flat File System or File:

- Data retrieval
- Data redundancy
- Data integrity
- Data security
- Data indexing

Data retrieval:-

- To retrieve data from database, we are using sequel language where as if we want to retrieve data from flat files we must develop application programs using high level language.
- Whenever we are installing DBMS software automatically a user interface created and also automatically database is created in the disk that's why using this interface end users directly interacting with database and also application programs indirectly interacting with database.

Data redundancy:-

- Sometimes we maintain multiple copies of the same data in different locations. This type of duplication data is also called as redundant data.
- In flat files whenever a data is modified in one file, it is not affected to another file that's why flat file does not maintain consistent state.
- Where as in case of database, if the data is modified in one location then that update will reflects in all other locations

Data integrity:-

- If we want to insert proper data into databases we are defining set of rules according to client requirement .These set or rules are also called as business rules. In databases we are defining business rules using two methods.
    - constraints
    - triggers

Data security:-

- Flat files data cannot be secured because flat files does not provide security mechanism whereas databases provides role based security.

Data indexing:-

- Databases internally maintains Indexed mechanism in order to retrieve the data very fast.

Chapter – 2

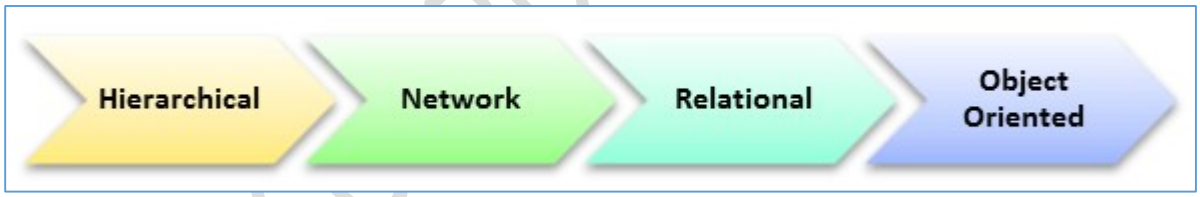
Introduction to DBMS

- What is DBMS?
- Different types of DBMS
- Applications of DBMS
- Flavors of DBMS

What is DBMS?

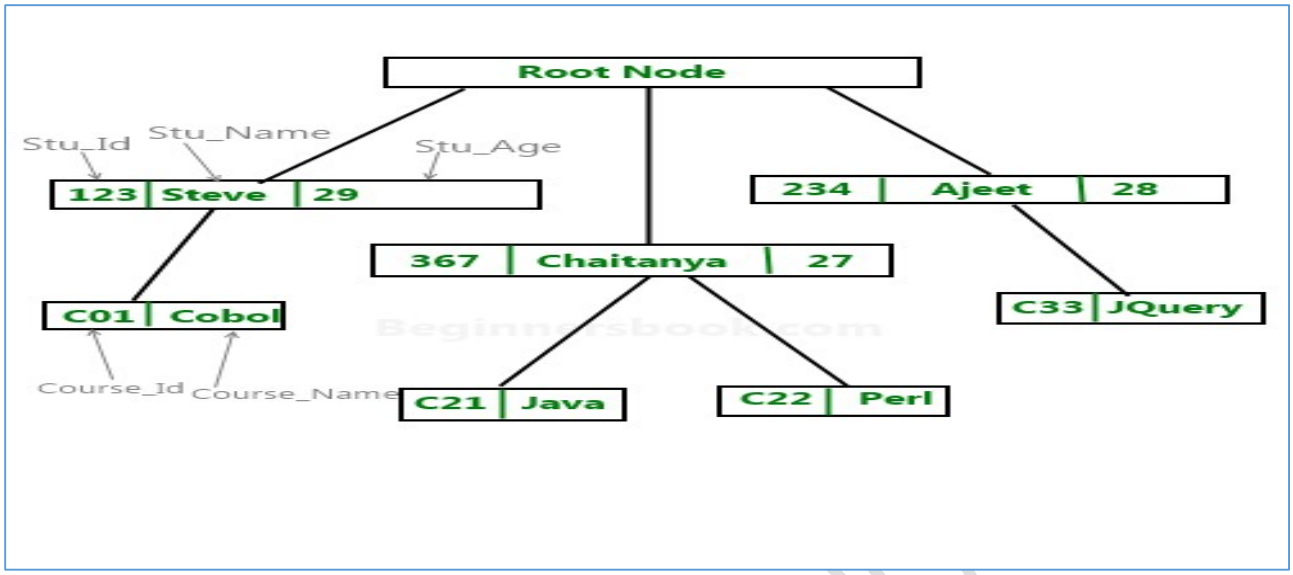
- Data Base Management System
- The DBMS provides a centralized view of data that can be accessed by multiple users, from multiple locations, in a controlled manner.
- A DBMS can limit what data the end user sees, as well as how that end user can view the data, providing many views of a single database schema.
- End users and software programs are free from having to understand where the data is physically located or on what type of storage media it resides because the DBMS handles all requests.

Types of DBMS



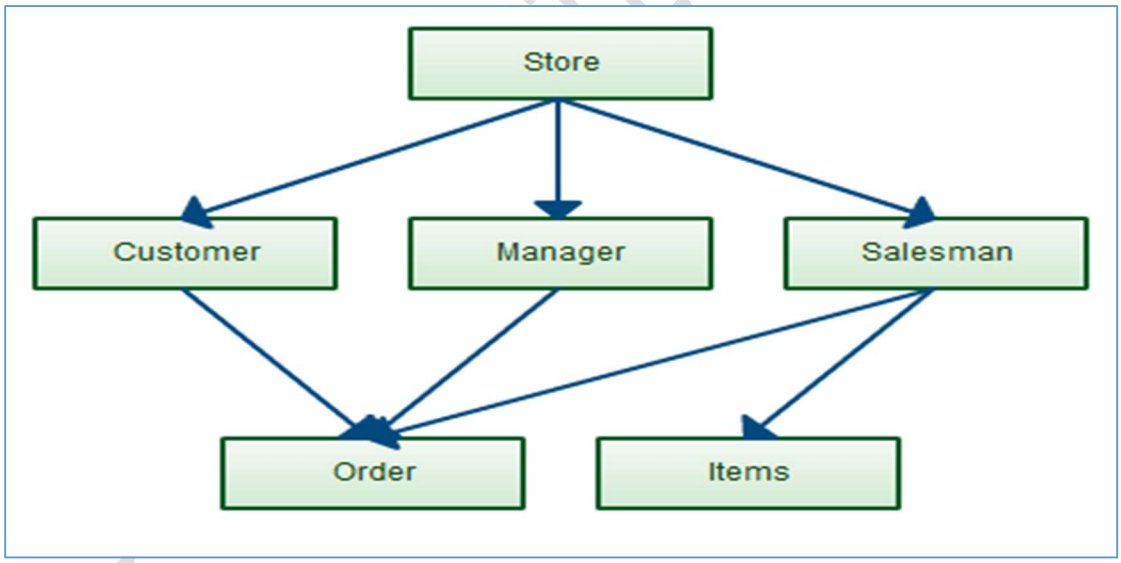
Hierarchical Database

- These DBMS employ parent and child relationships to store the data.
- They store data in a tree-like structure so that it is easy to find and use. Similarly, the configuration of the DBMS is present in the nodes of the tree.



2. Network DBMS

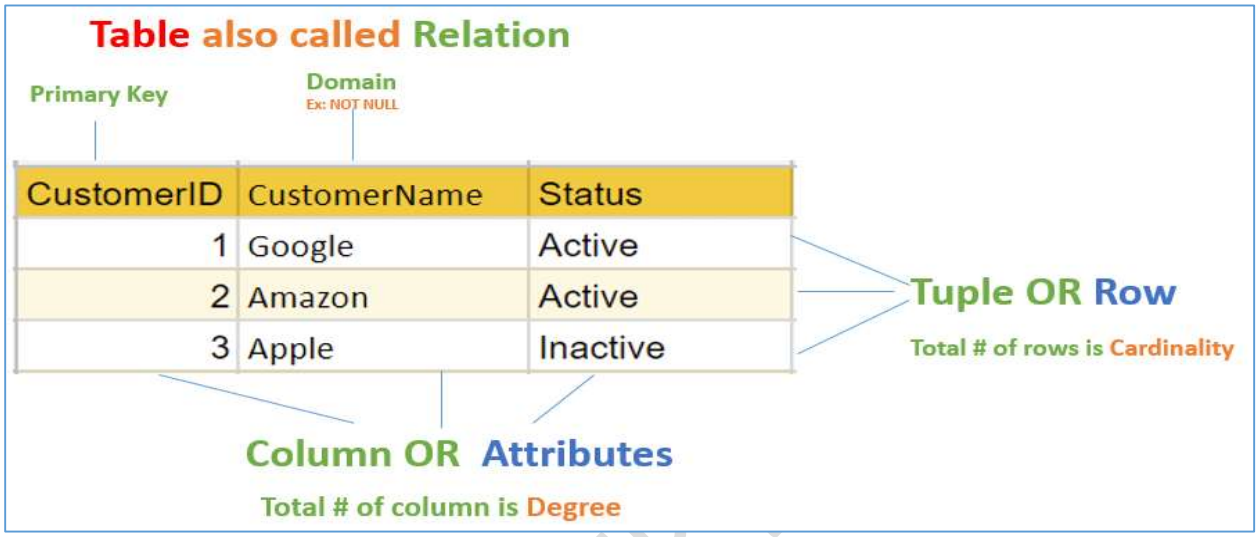
- Network DBMS supports many-to-many relations which results in complex database structures.
- RDM Server is a major example of the network DBMS.



3. Relational DBMS

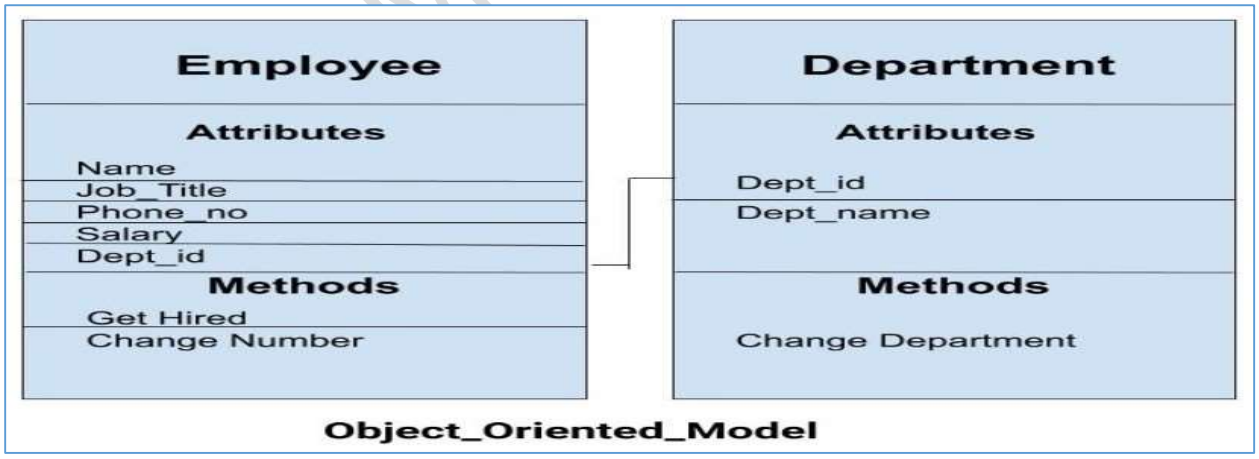
- Relational DBMS stores data using the database relationships in the form of tables, also known as relations or tuples.

- They do not support many to many relationships and have pre-defined data types that they can support. They are the most popular DBMS type in the industry.
- Example: Oracle, MySQL, MS SQL, etc.



4. Object-Oriented Relational DBMS

- This DBMS supports the storage of miscellaneous data types. They store data in the form of objects.
- The object has attributes (i.e., name, id, gender, etc.) and the logic for what needs to be done with the data. Example: PostgreSQL.



Applications of DBMS

- Most commercial airlines rely on a DBMS for data-intensive applications such as
  - scheduling flight plans
  - managing customer flight reservations.
- Applications include storing
  - customer information,
  - account information,
  - tracking account transactions
    - including withdrawals and deposits
  - tracking loan payments.
  - ATMs are a good example of a banking system that relies on a DBMS to track and manage that activity.
- Manufacturing and supply-chain management.
  - Manufacturing companies also rely on a DBMS to keep track of and manage inventory in warehouses.
  - A DBMS can also be used to manage data for supply chain management applications that track the flow of goods and services,
    - movement and storage of raw materials,
    - work-in-process inventory,
    - finished goods from the point of origin to the point of consumption.
- DBMS manage sales for any type of organization.
  - storing product,
  - customer and salesperson information,
  - recording the sale,
  - tracking fulfillment
  - maintaining sales history information.
- Human resources.
  - manage employee information
    - managing employee data such as addresses, phone numbers, salary details, payroll and paycheck generation.

Flavors of DBMS

Oracle

- Oracle Database is a commercial relational database management system. It utilizes enterprise-scale database technology with a robust set of features right out of the box. It can be stored in the cloud or on-premises.

MySQL

- MySQL is a relational database management system that is commonly used with open-source content management systems and large platforms like Facebook, Twitter, and Youtube.

SQL Server

- Developed by Microsoft, SQL Server is a relational database management system built on top of structured query language (SQL), a standardized programming language that allows database administrators to manage databases and query data.

CHAPTER- 3

Introduction to RDBMS

Topics :

- Feature of RDBMS
- Advantages of RDBMS over FMS ad DBMS
- The 12 rules (E.F Codd's Rules –RDBMS)
- Support of Normalization Process for Data Management
- Oracle Corporation Products
- Oracle Versions
- About SQL&SQL*PLUS

What is RDBMS?

- RDBMS is an acronym for Relational Database Management System and is a type of database management system that stores data in a structured format using rows and columns, making it easy to locate and access data in relation to another piece of data in the database.

Employee table

Empno (PK)	Ename	Job	Deptno (FK)
101	A	Salesman	10
102	B	Manager	10
103	c	Manager	20

Department table

Deptno (PK)	dname	loc
10	Sales	Chicago
20	Sales	Chicago
30	Finance	New York

Features of RDBMS:

- Provides data to be stored in tables
- Persists data in the form of rows and columns
- Provides facility called primary key, which is used to uniquely identify the rows
- Creates indexes for quicker data retrieval

- Provides a virtual table creation in which sensitive data can be stored and simplified query can be applied.(views)
- Sharing a common column in two or more tables(primary key and foreign key)
- Provides multi user accessibility that can be controlled by individual users.

POPULAR RDBMS VENDORS

- MySQL:
  - An open source RDBMS that relies on SQL for processing data in the database.
- PostgreSQL:
  - An open source database not controlled by any corporation. Typically used for web application development
- Oracle DB:
  - A multi-model database management system produced and marketed by Oracle Corporation
- SQL Server:
  - A RDBMS owned by Microsoft that is closed source
- SQLite:
  - A RDBMS contained in a C library. In contrast to many other database management systems, SQLite is not a client server database engine. Rather, it is embedded into the end program.

Difference between DBMS and RDBMS

	DBMS	RDBMS
User Availability	It only supports one user at a time.	RDBMS supports more than one user at a time.
Security	DBMS doesn't have any kind of security	There are multiple layers of security in RDBMS.
Usage	DBMS is used to handle small amounts of data.	RDBMS is used to handle huge sums of data.
Order	Data is stored in forms of files.	Data in RDBMS is stored in the form of tables.
Speed	DBMS takes time to load data.	RDBMS loads data faster.

Chapter – 4

E F Codd 12 Rules

Dr. Edgar F Codd Rules

- Dr E.F.Codd, also known to the world as the 'Father of Database Management Systems' had propounded 12 rules which are in-fact 13 in number.
- The rules are numbered from zero to twelve.
- According to him, a DBMS is fully relational if it abides by all his twelve rules. Till now, only few databases abide by all the eleven rules.

Rule 0 – Foundation rule

- Any relational database management system that is propounded to be RDBMS or advocated to be a RDBMS should be able to manage the stored data in its entirety through its relational capabilities.

Rule 1 – Rule of Information

- Relational Databases should store the data in the form of relations. Tables are relations in Relational Database Management Systems. Be it any user defined data or meta-data, it is important to store the value as an entity in the table cells.

Rule 2 – Rule of Guaranteed Access

- The use of pointers to access data logically is strictly forbidden. Every data entity which is atomic in nature should be accessed logically by using a right combination of the name of table, primary key represented by a specific row value and column name represented by attribute value.

Rule 3 – Rule of Systematic Null Value Support

- Null values are completely supported in relational databases. They should be uniformly considered as 'missing information'. Null values are independent of any data type. They should not be mistaken for blanks or zeroes or empty strings. Null values can also be interpreted as 'inapplicable data' or 'unknown information.'

Rule 4 – Rule of Active and online relational Catalog

- In the Database Management Systems lexicon, 'metadata' is the data about the database or the data about the data. The active online catalog that stores the metadata is called 'Data dictionary'. The so called data dictionary is accessible only by authored users who have the required privileges and the query languages used for accessing the database should be used for accessing the data of data dictionary.

Rule 5 – Rule of Comprehensive Data Sub-language

- A single robust language should be able to define integrity constraints, views, data manipulations, transactions and authorizations. If the database allows access to the aforementioned ones, it is violating this rule.

Rule 6 – Rule of Updating Views

- Views should reflect the updates of their respective base tables and vice versa. A view is a logical table which shows restricted data. Views generally make the data readable but not modifiable. Views help in data abstraction.

Rule 7 – Rule of Set level insertion, update and deletion

- A single operation should be sufficient to retrieve, insert, update and delete the data.

Rule 8 – Rule of Physical Data Independence

- Batch and end user operations are logically separated from physical storage and respective access methods.

Rule 9 – Rule of Logical Data Independence

- Batch and end users can change the database schema without having to recreate it or recreate the applications built upon it.

Rule 10 – Rule of Integrity Independence

- Integrity constraints should be available and stored as metadata in data dictionary and not in the application programs.

Rule 11 – Rule of Distribution Independence

- The Data Manipulation Language of the relational system should not be concerned about the physical data storage and no alterations should be required if the physical data is centralized or distributed.

Rule 12 – Rule of Non Subversion

- Any row should obey the security and integrity constraints imposed. No special privileges are applicable.

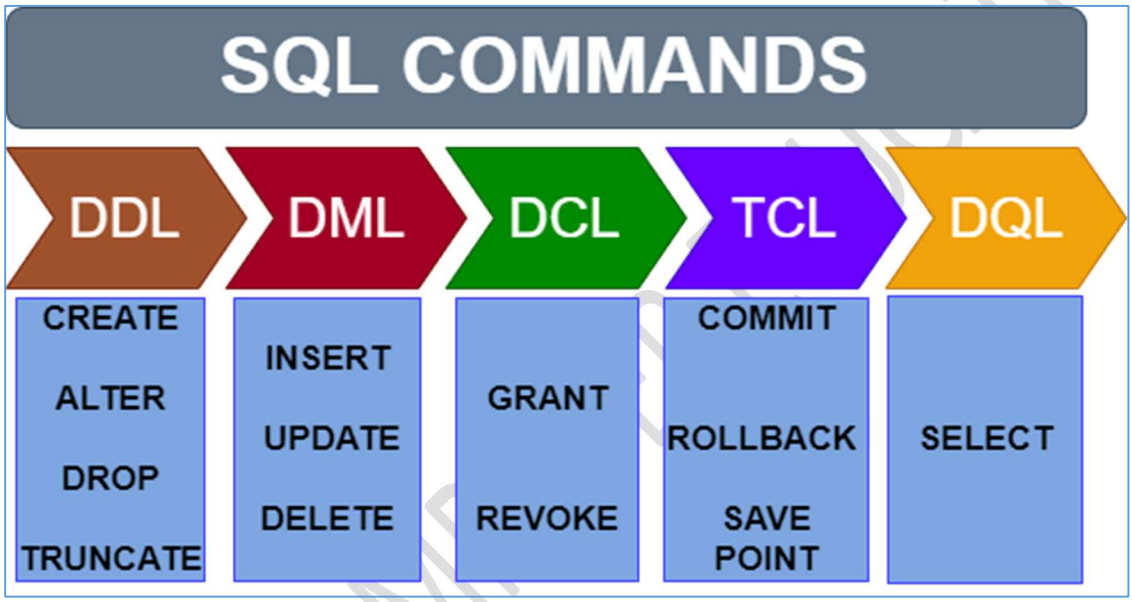
Oracle implements 11+ rules and so does Sybase. SQL Server also implements 11+ rules while FoxPro implements 7+ rules.

Chapter – 5

Types of SQL Queries

All SQL Queries are divided into 5 different categories:

- Data Definition Language (DDL)
- Data Retrieval Language (DRL)
- Data Manipulation Language (DML)
- Transaction Control Language (TCL)
- Database Security and Privileges (DCL)



DDL – Data Definition Language		
All these queries are used to create or modify the structure of the table		
1.	create table	used to define the structure of a table
2	alter table	used to modify the structure of the existed table
3	truncate	used to remove the data present in the table
4	drop	remove the data and structure of the table

DML – Data Manipulation Language		
All these queries are used to create or modify the data in a table		
1.	insert	used to add a new data to the table
2	update	used to modify the existed data
3	delete	used to remove the data from table

DRL – Data Retrieval Language		
1.	select	used to get or retrieve or fetch the required data from one or more tables.

TCL – Transaction Control Language		
All these queries are used to make the changes permanent or revert all the changes		
1.	commit	save the changes permanently in database
2	rollback	revert all the changes or don't save
3	savepoint	book mark the point

DCL – Data Security and Privileges Language		
All these queries are used to add or remove privileges to the user		
1.	grant	Add or provide the privileges to the user
2	revoke	Remove the added privileges to the user

Data Security and Privilege Language

Commands

- **Create User**
- **Grant**
- **Revoke**

How to create a new user in Oracle?

Query: CREATE USER

```
CREATE USER  <user-name> IDENTIFIED BY <password> [DEFAULT
TABLESPACE <tablespace>] [QUOTA {<size> | UNLIMITED} ON
<tablespace>] [PROFILE <profile>] [PASSWORD EXPIRE] [ACCOUNT
{LOCK | UNLOCK}];
```

Syntax:

Steps to create a new user:

Step 1: Login to SQL with Admin or super user credentials

Step 2: use create user command to create a new user

How to navigate from one user to another user?

Query: CONNECTION OR CONN

```
CONN user-name/password   --OR—

CONN user-name
```

Syntax:

Once the user is successfully created, we have to provide session privilege to the user. Otherwise, we will get the below error.

```
SQL> conn demo/demo;
ERROR:
ORA-01045: user DEMO lacks CREATE SESSION privilege; logon denied
```

How to provide session privilege to the created user?

Query: GRANT

Use the GRANT statement to give privileges to a specific user or role, or to all users, to perform actions on database objects.

The following types of privileges can be granted:

- Delete data from a specific table.
- Insert data into a specific table.
- Create a foreign key reference to the named table or to a subset of columns from a table.
- Select data from a table, view, or a subset of columns in a table.
- Create a trigger on a table.
- Update data in a table or in a subset of columns in a table.
- Run a specified function or procedure.
- Use a sequence generator or a user-defined type.

Syntax:

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

List of Privilege types:

ALL PRIVILEGES | privilege-list

- DELETE |
- INSERT |
- REFERENCES [column list] |
- SELECT [column list] |
- TRIGGER |
- UPDATE [column list]
    - column list
        - (column-identifier {, column-identifier}*)

Examples:

- How to grant the create session privilege and resource to the user?

- GRANT CREATE SESSION, RESOURCE to demo;

- How to grant the create tablespace privilege to the user?
  - GRANT CREATE TABLESPACE to demo
  - If the user doesn't have the above privilege, then he will get the below error while creating the table.

```
SQL> create table student(
  2  htno number(5));
create table student(
*
ERROR at line 1:
ORA-01950: no privileges on tablespace 'SYSTEM'
```

- How to grant the create table privilege to the user?
  - GRANT CREATE TABLE to demo;
  - If we try to create a table without the above privilege, then we will get the below error:

```
SQL> create table student(
  2  htno number(5));
create table student(
*
ERROR at line 1:
ORA-01031: insufficient privileges
```

  - Steps:
    - Login to Admin or Super user
    - Execute the above query

How to provide the access to the tables available in other users?

- To grant the SELECT privilege on table t to the authorization IDs maria and harry, use the following syntax:
  - GRANT SELECT ON TABLE t TO maria,harry

- To grant the UPDATE and TRIGGER privileges on table t to the authorization IDs anita and zhi, use the following syntax:
  - GRANT UPDATE, TRIGGER ON TABLE t TO anita,zhi

- To grant the SELECT privilege on table s.v to all users, use the following syntax:
  - GRANT SELECT ON TABLE s.v to PUBLIC

- To grant the EXECUTE privilege on procedure p to the authorization ID george, use the following syntax:
  - GRANT EXECUTE ON PROCEDURE p TO george

What is the Role?

- It is a collection of different object privileges on different database objects.
- it is created by dba.

How to create a Role?

Query: CREATE ROLE

```
CREATE ROLE role-name
```

Syntax:

Steps to create a Role?

Step 1: Login to super user or admin

Step 2: use create role to create the role

How to assign the privileges to the role?

Query: GRANT

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

Syntax:

- To grant the SELECT privilege on table t to the role purchases_reader_role, use the following syntax:
  - GRANT SELECT ON TABLE t TO purchases_reader_role
- To grant the role purchases_reader_role to the authorization IDs george and maria, use the following syntax:

- GRANT purchases_reader_role TO george,maria

- Some other examples:
  - Grant select, insert on scot.emp to my_role;
  - Grant delete on scott.dept to my_role;
  - Grant update,select on scott.salgrade to my_role;
  - Grant insert on scott.emp007 to my_role;
  - Grant my_role to user1;
  - Grant my_role to user2;

Revoke

- REVOKE statement to remove privileges from a specific user or role, or from all users, to perform actions on database objects.
- The following types of privileges can be revoked:
  - Delete data from a specific table.
  - Insert data into a specific table.
  - Create a foreign key reference to the named table or to a subset of columns from a table.
  - Select data from a table, view, or a subset of columns in a table.
  - Create a trigger on a table.
  - Update data in a table or in a subset of columns in a table.
  - Run a specified routine (function or procedure).
  - Use a sequence generator or a user-defined type.

Syntax:

```
REVOKE privilege-type ON [ TABLE ] { table-Name | view-Name } FROM grantees
```

Privilige-types

ALL PRIVILEGES | privilege-list

- DELETE |
- INSERT |
- REFERENCES [column list] |
- SELECT [column list] |
- TRIGGER |
- UPDATE [column list]

- column list
  - (column-identifier {, column-identifier}*)

Examples:

- To revoke the SELECT privilege on table t from the authorization IDs maria and harry, use the following syntax:
  - REVOKE SELECT ON TABLE t FROM maria,harry

- To revoke the UPDATE and TRIGGER privileges on table t from the authorization IDs anita and zhi, use the following syntax:
  - REVOKE UPDATE, TRIGGER ON TABLE t FROM anita,zhi

- To revoke the SELECT privilege on table s.v from all users, use the following syntax:
  - REVOKE SELECT ON TABLE s.v FROM PUBLIC

- To revoke the UPDATE privilege on columns c1 and c2 of table s.v from all users, use the following syntax:
  - REVOKE UPDATE (c1,c2) ON TABLE s.v FROM PUBLIC

- To revoke the EXECUTE privilege on procedure p from the authorization ID george, use the following syntax:
  - REVOKE EXECUTE ON PROCEDURE p FROM george RESTRICT

- To revoke the role purchases_reader_role from the authorization IDs george and maria, use the following syntax:
  - REVOKE purchases_reader_role FROM george,maria

- To revoke the SELECT privilege on table t from the role purchases_reader_role, use the following syntax:
  - REVOKE SELECT ON TABLE t FROM purchases_reader_role
- Other Example:
  - Revoke select on scott.emp from my_role;

How to change the user password?

- alter user <user-name> identified by <your new password>

What is the table name where all the usernames and passwords are available?

- dba_users

Query to retrieve username and password from dba_users table

Step 1: login to system/sys(root user or admin)

Step 2: use the select query on dba_users table

- select username,password from dba_users;

How to drop a role?

- DROP ROLE role-name;

How to drop the user?

- DROP USER user-name CASCADE;

Steps to Drop role or user:

Step 1: Login using admin or super user credentials

Step 2: use drop query to remove role or user

Chapter – 6
Basic Operations on Table

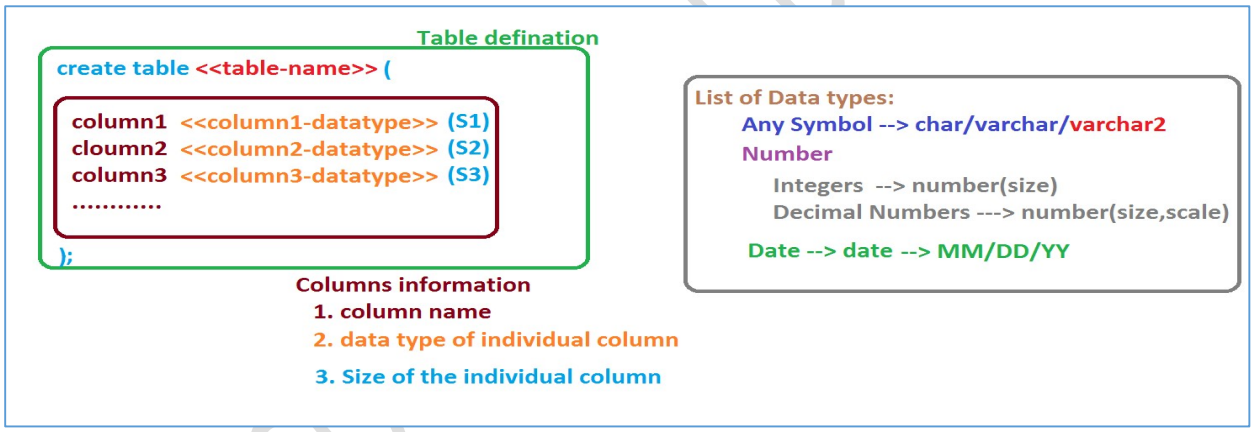
- Basic SQL Commands
  - Create
    - Oracle Pre Defined Data types
  - Insert
  - Select

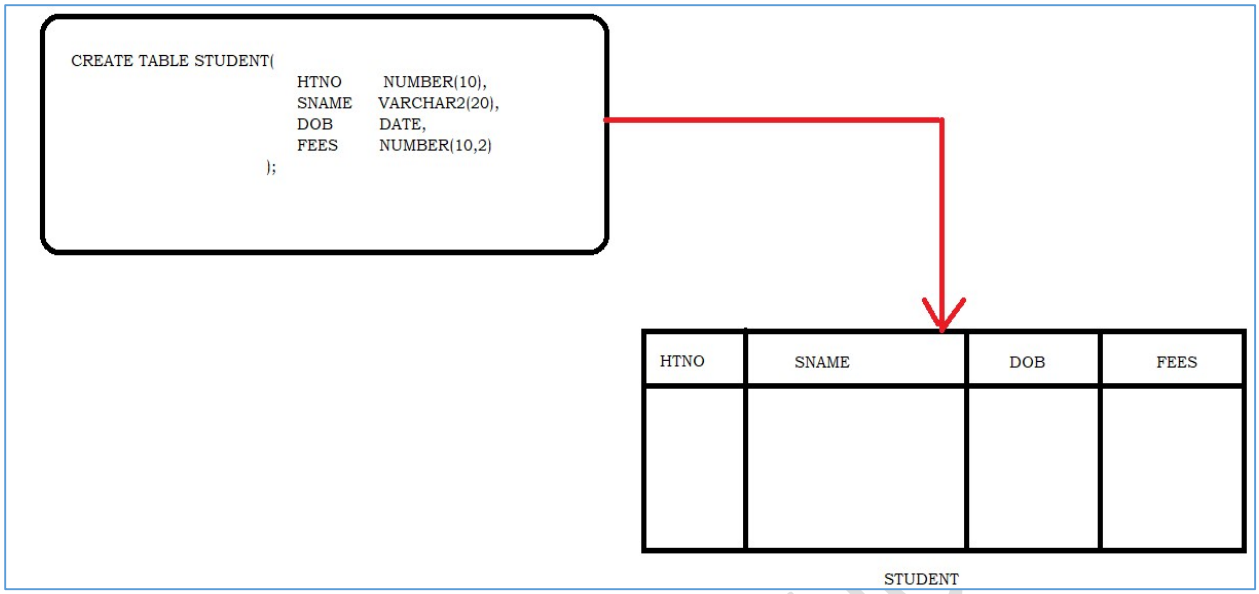
Basic Operation 1 – Table creation

How to create a table?

Command: create table

Syntax:





Basic Operation 2 – Insert the data into a table

How to save or create a new data in a table?

Command: insert

Syntax:

```
insert into <<table-name>> values( col1-value,
                                   col2-value,
                                   col3-value,
                                   ..............
);
```

Rules:

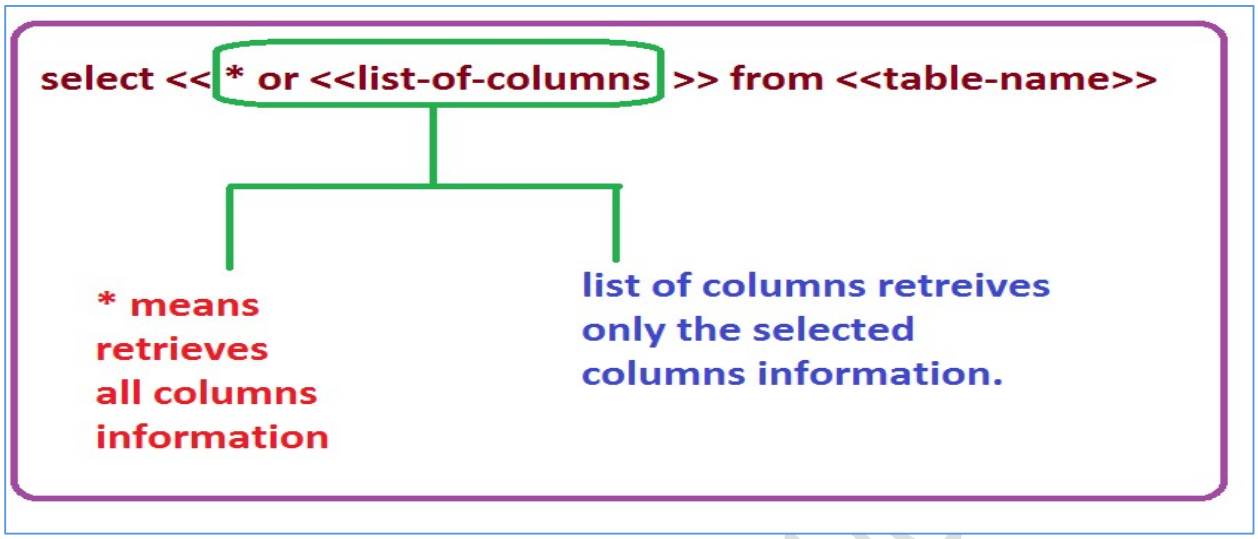
1. All date and char type of values should be enclosed in single quotes
2. the value type should match with the data type defined in table defination for the respective column.
3. Values should be in order of columns in table defination.

Basic Operation 3 – Retrieve the data from the table

How to fetch the data from the table?

Command: select

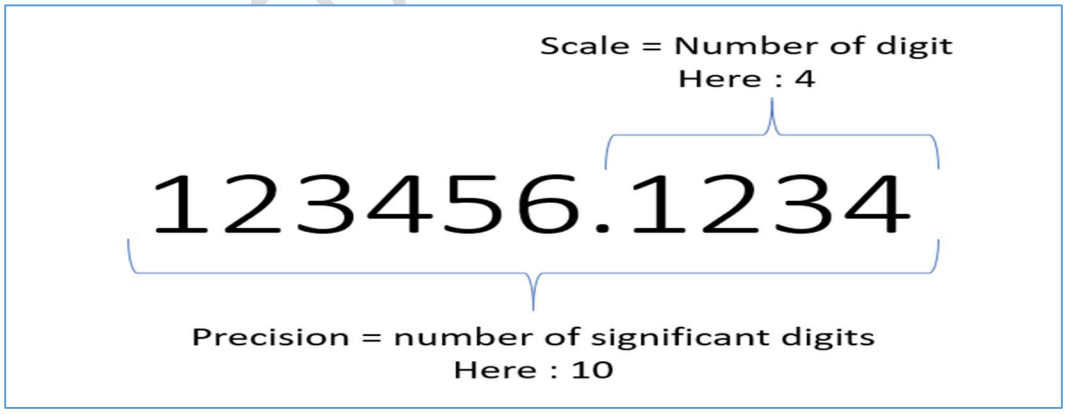
Syntax:

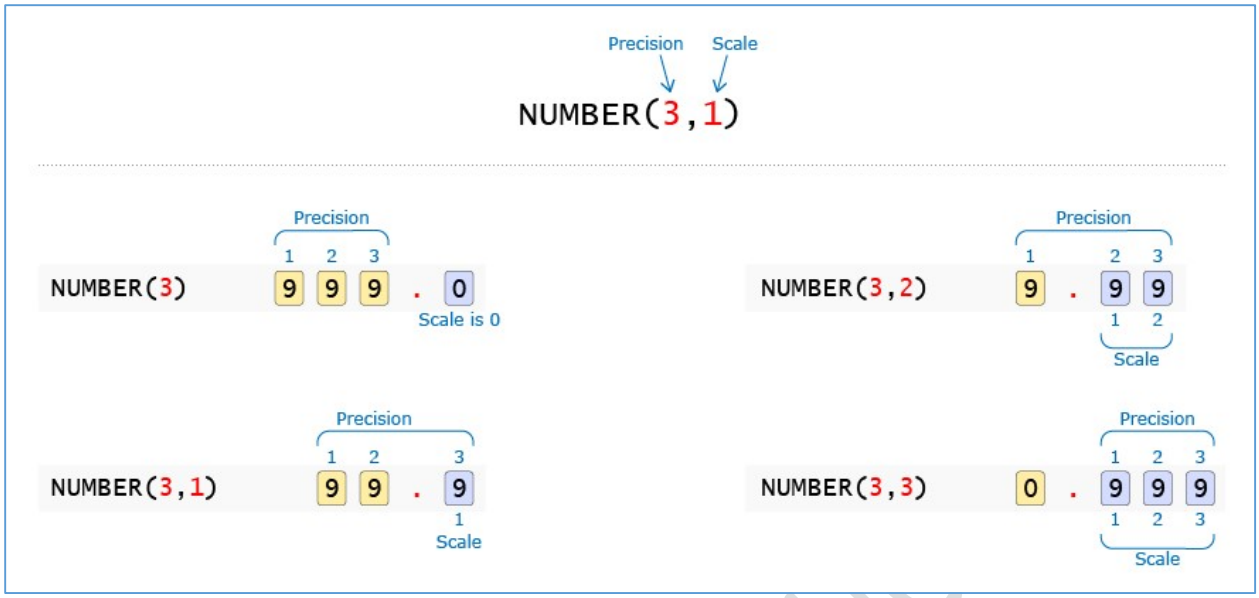


Chapter – 7

Data types in SQL

- **NUMBER:** This data type allows us to enter numeric values such as integer values or decimal values. In SQL these data types further divided into
  - NUMBER(SIZE)
  - **NUMBER(P,S) (P=Precision, S= Scale)**





Let us see some examples of how input data is changed before storage.

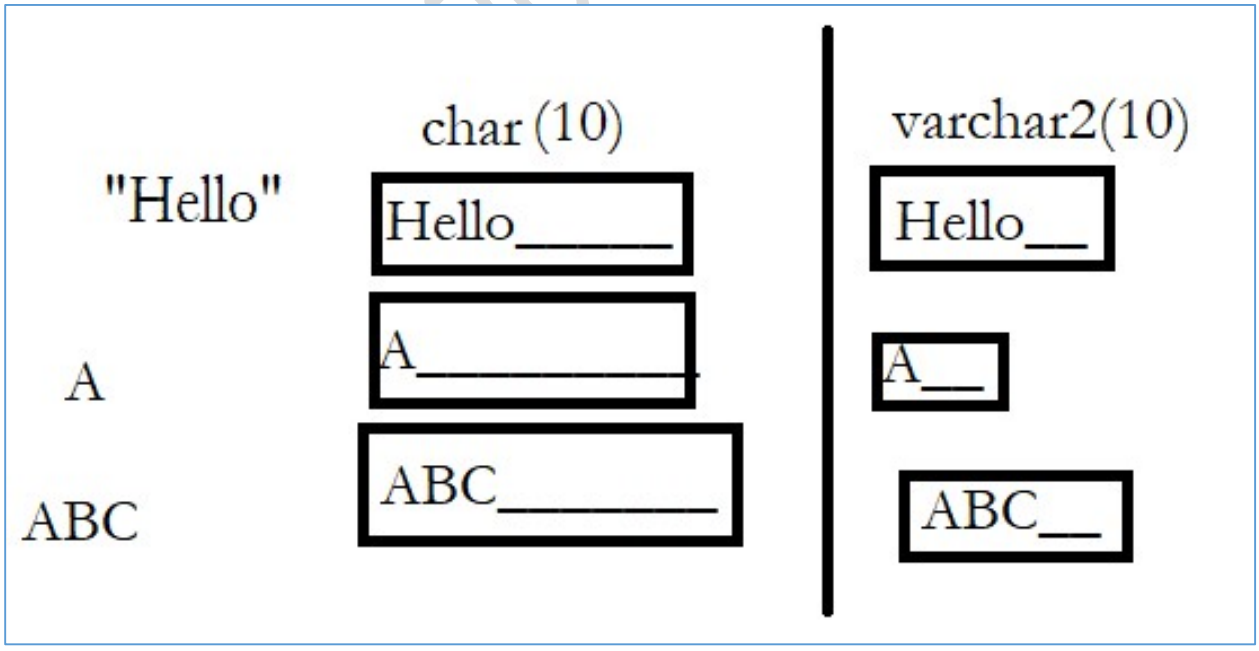
Input	Data Type	Stored Value	Explanation
121.79	NUMBER	121.79	Since no precision and scale are specified, this can store any decimal number with 38 significant digits. The number is stored without any changes.
121.79	NUMBER(3)	122	Scale defaults to zero if precision is specified but scale is not. Hence the number is rounded to zero decimal places.
121.79	NUMBER(5,2)	121.79	Here 5 - 2 = 3 digits are allowed before decimal point and 2 digits are allowed after decimal point. Since number conforms to these rule, it is stored without any changes.
121.79	NUMBER(6,1)	121.8	Only 1 digit is allowed after decimal point so the number is rounded to 1 decimal place.
121.79	NUMBER(4,2)	error	Only 4 - 2 = 2 digits are allowed before decimal point while the number has three digits. This is an error scenario as truncation will result in incorrect value getting stored.

- **Alphabets**: The data type allows us to enter character values such as student names employee names e.t.c., this data type is further divided into three types

    1. Char(Size)
    2. Varchar(Size)/Varchar2(Size)

	CHAR(n)	VARCHAR2(n)
Useful for	Storing characters having pre-determined length	Storing characters whose length vary a lot
Storage size	size for n characters	size for actual no. of characters + fixed size to store length
Storage Characteristic	Trailing spaces are applied if data to be stored has smaller length than n.	Trailing spaces are not applied.
Maximum size	2000 bytes	4000 bytes
Example	A CHAR(10) field will store "Hello" as 10 bytes by appending 5 trailing spaces.	A VARCHAR2(10) field will store "Hello" as 7 bytes (assuming 2 bytes to store length).
Alternate Name	CHARACTER(n)	CHARACTER VARYING(n)

Example:



DATE:

The data type allows us to enter date values such as student joining date employee hiring data e.t.c., the default date format is Oracle is DD/MM/.YYYY.

Ex: Jdate date

Timestamp:

The data type allows to enter both date and time values. The default format of timestamp is DD/MM/YYYY HH/MM/SS.

EX: Logintime timestamp

These data types are further divided into three types CLOB, BLOB, Bfile.

- **CLOB (Character Large Object):** This data type allows us to all types of characters and the minimum length of this data type is 1 to 4GB.
- Whenever we need to enter employee history student conduct e.t.c., then we need to use this data type.
- **BLOB (Binary large Object):** This data type allows us to enter any type of photos, graphical pictures clippings, sounds, multimedia messages e.t.c., the minimum length of this data type is 1 to 4 GB.
- **BFILE:** This data type also allows us to enter BLOB types of values and also binary data, XML data e.t.c., the minimum length of data type is 1 to 4 GB.

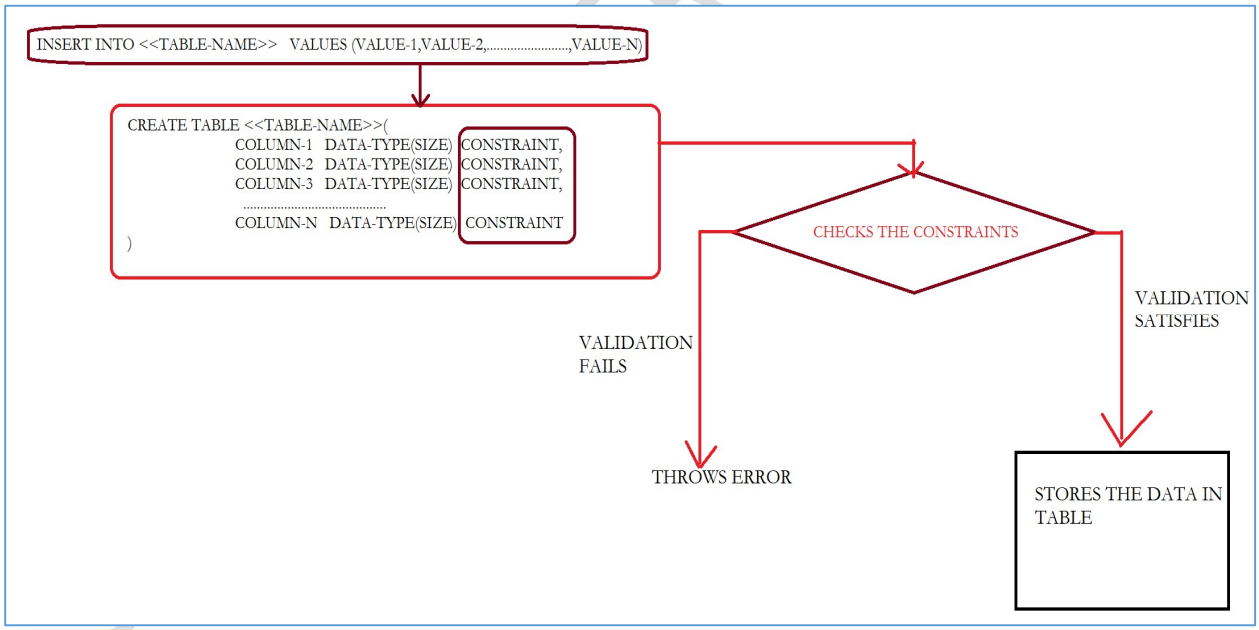
Chapter – 8

Create Table - Working with Integrity Constraints

Working with Integrity Constraints

- Importance of Data Integrity
- Support of Integrity Constraints for Relating Table in RDBMS
  - NOT NULL constraint
  - UNIQUE constraint
  - PRIMARY KEY constraint
  - FOREIGN KEY constraint
  - CHECK constraint
- Working with different types of integrity Constraints

Importance of Integrity Constraints



Syntax to define a constraints

```
create table <<table-name>> (
  <<column-name>> <<data-type>>(size)  <<constraint-type>>,
  .....
);
```

Column Level Constraint

```
create table <<table-name>> (
  <<column-name>> <<data-type>>(size),
  .....
  constraint <<constraint-name>>  <<constraint-type>>,
  .....(n number of constraints)
);
```

Table Level Constraints

Note:

- Constraint names should be unique across the database.
- It is recommended in the real time to provide a name to every constraint definition

Different types of Integrity Constraints

- Not Null
- Unique
- Primary Key
- Check
- Default
- Foreign Key

Not Null	Won't allow any null values
Unique	Won't allow duplicate values
Primary key	Combination of Not Null and Unique That means it won't allow null values as well as duplicate value
Check	Used to define the condition
Default	Used to define the default value to the column whenever the user skipped that column from the input values list

Note:

1. All the constraint names should be unique
2. We are not allowed to define more than one primary key in any table
3. All the values are case sensitive.

Not Null

- By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

```
CREATE TABLE Student (
    StudentId INTEGER CONSTRAINT Stud_SId_nn NOT NULL,
    FName VARCHAR2(10) NOT NULL,
    LName VARCHAR2(10));
```

Unique Constraint

- The UNIQUE constraint ensures that all values in a column are different.

```
CREATE TABLE Student (
    StudentId INTEGER,
    FName VARCHAR2(10),
    ContactNo NUMBER(10) CONSTRAINT Stud_cno_uk UNIQUE);
```

```
CREATE TABLE Student (
    StudentId INTEGER,
    FName VARCHAR2(10),
    ContactNo NUMBER(10),
    CONSTRAINT Stud_cno_uk UNIQUE (ContactNo));
```

Difference between Unique and Notnull

Not Null	Unique
Not null won't allows Null values	Allows Null values
Not null allows duplicate values	Won't allow duplicate values

Primary Key Constraint

- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Primary key = Not null + Unique

Ex:

htno is primary key in student table

productid is primary key in product table

……

```
CREATE TABLE Student (
    StudentId INTEGER CONSTRAINT stud_sid_pk PRIMARY KEY,
    FName VARCHAR2(10),
    ContactNo NUMBER(10));
```

```
CREATE TABLE Student (
    StudentId INTEGER,
    FName VARCHAR2(10),
    ContactNo NUMBER(10),
    CONSTRAINT stud_sid_pk PRIMARY KEY (StudentId));
```

Note:

- Every table should have only one primary key constraint.
- Oracle Triggers an error if we tried to define more than one primary key in a table.

Composite key:

- Another type of primary key called Composite key
- if we are combining more than one column to frame a primary key, then such a type of key is called as composite key.
- we have to take help of table level constraint to define a composite key

Example:

```
SQL> create table contactinfo(
  2  empid number(5),
  3  phno number(10),
  4  constraint pk_empid_phno primary key(empid,phno));
```

Candidate key →

Another primary key in a table is called candidate key

In the above example: phno is candidate key

Default Constraint

- The DEFAULT constraint is used to set a default value for a column.
- The default value will be added to all new records, if no other value is specified.

```
CREATE TABLE Student (
    StudentId INTEGER,
    FName VARCHAR2(10),
    DOJ DATE DEFAULT SYSDATE);
```

Check Constraint

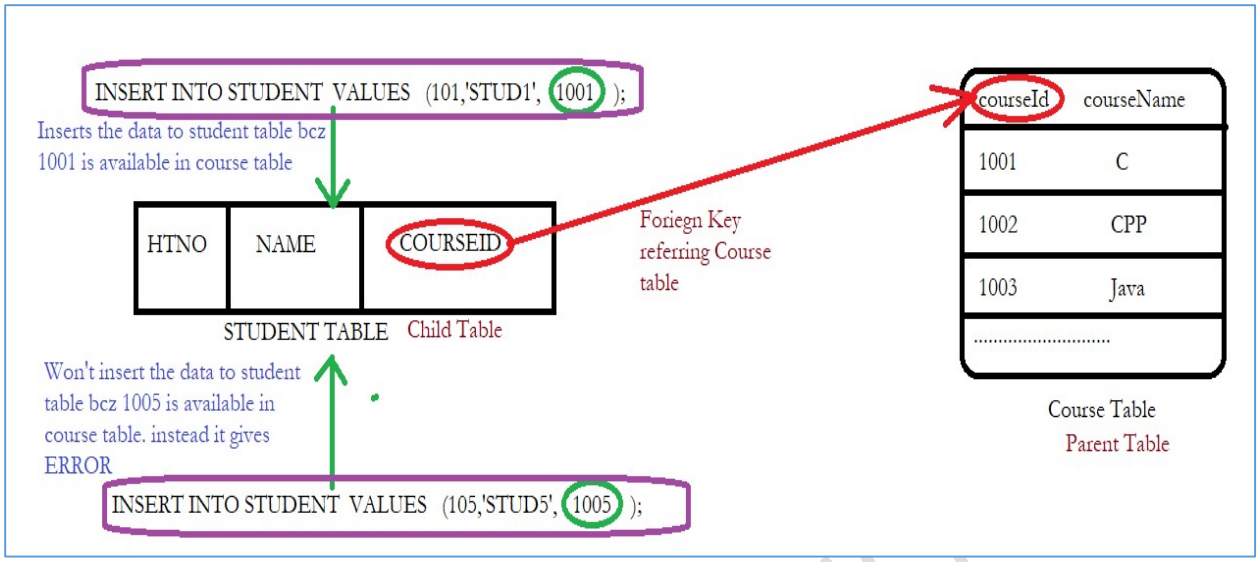
- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a column it will allow only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

```
CREATE TABLE Student (
    StudentId INTEGER,
    FName VARCHAR2(10),
    Gender CHAR(1) CONSTRAINT Stud_gender_ck1 CHECK(Gender IN('M', 'F')));
```

```
CREATE TABLE Student (
    StudentId INTEGER,
    FName VARCHAR2(10),
    Gender CHAR(1),
    CONSTRAINT Stud_gender_ck1 CHECK(Gender IN('M', 'F')));
```

Foreign Key Constraint

- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.
- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.




```
CREATE TABLE Marks(
    CourseId INTEGER,
    StudentId INTEGER CONSTRAINT marks_sid_fk REFERENCES Student(StudentId),
    MarksScored DECIMAL(5,2));
```

```
CREATE TABLE Marks(
    CourseId INTEGER,
    StudentId INTEGER,
    MarksScored DECIMAL(5,2),
    CONSTRAINT marks_sid_fk FOREIGN KEY (StudentId) REFERENCES Student(StudentId));
```

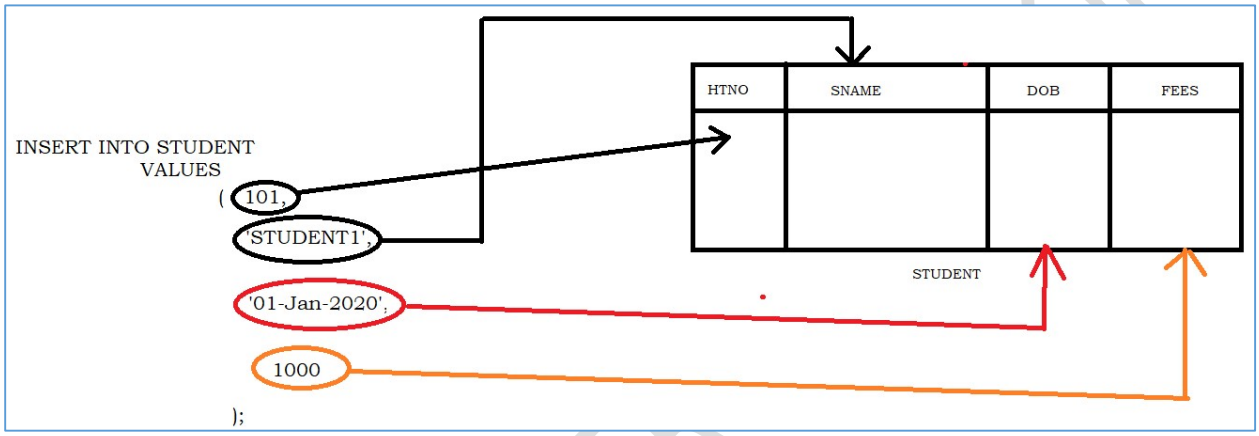
Chapter – 9

DML - Insert Query

How to save the data in a table

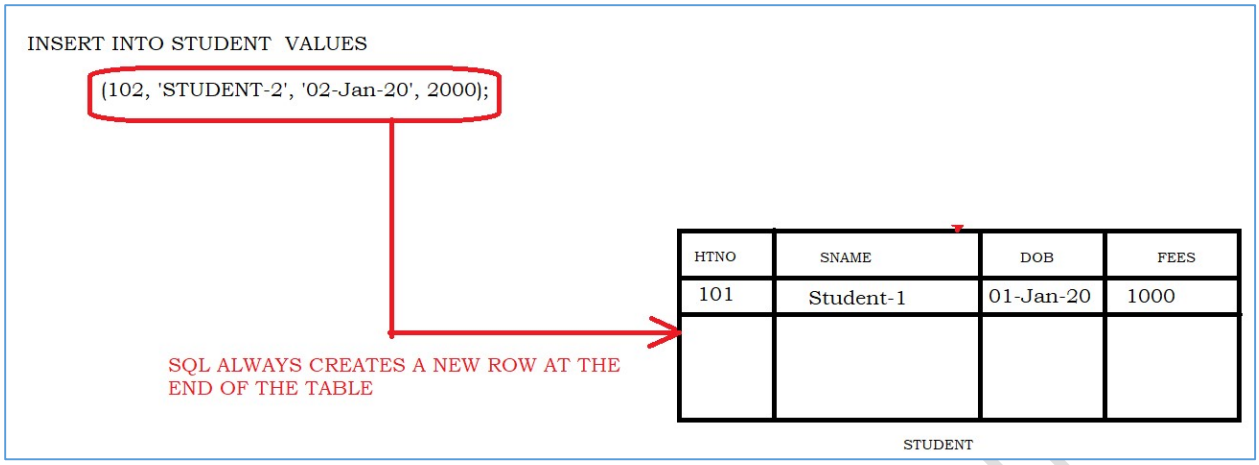
Syntax – 1 :

INSERT INTO <TABLE NAME> VALUES (Value-1, Value-2,------,Value-n)



HTNO	SNAME	DOB	FEES
101	Student-1	01-Jan-20	1000

STUDENT



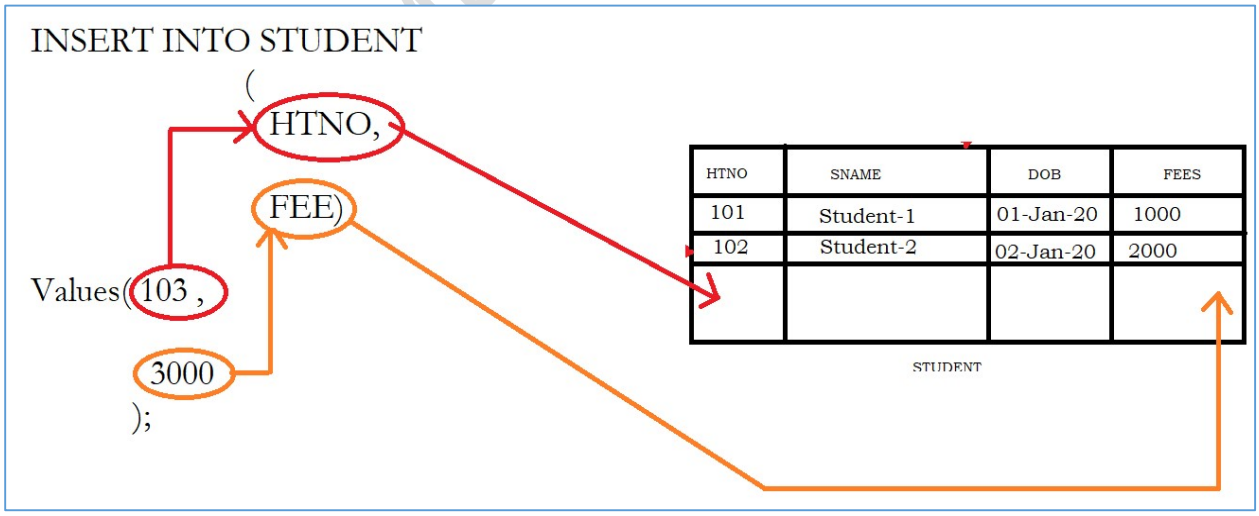
Note:

- The user must provide the values all the columns in table.
- If the number of values in insert query is not equal to number of columns in table, then sql triggers an exception saying the INSUFFICIENT VALUES.

Syntax – 2

INSERT INTO <TABLE NAME>(COLUMN-1, COLUMN-2,.....COLUMN-N)

VALUES (Value-1, Value-2,------,Value-n)



Chapter – 10

DML – Update Query

Update Table

Syntax:

UPDATE *tablename* SET *column_name* =*value* [WHERE *condition*]

Use case – 1

Update all the rows in the table

Syntax:

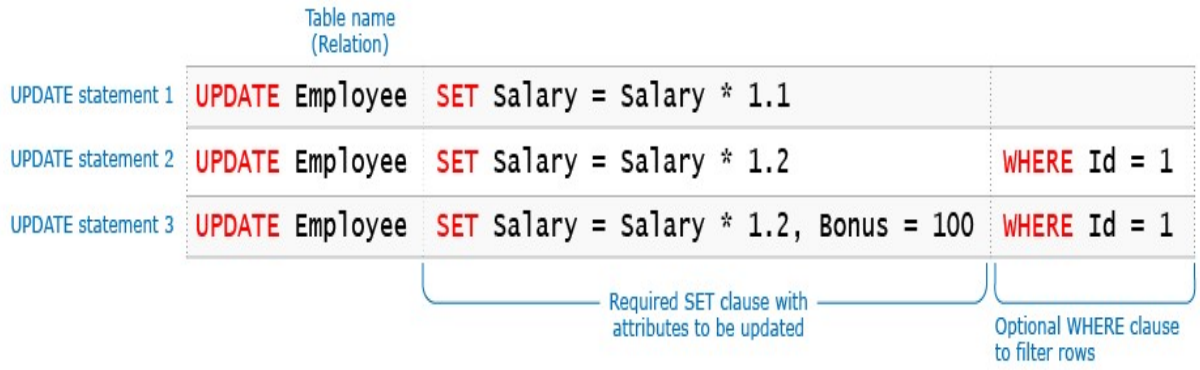
UPDATE *tablename* SET *column_name* =*value*

Use case – 2

Update selected rows in the table

Syntax:

UPDATE *tablename* SET *column_name* =*value* [WHERE *condition*]



Chapter – 11

DML – Delete Query

Delete

Syntax:

DELETE FROM table-name WHERE condition

Use case – 1

Remove all the rows from the table

Syntax:

DELETE FROM table-name

Use case – 2

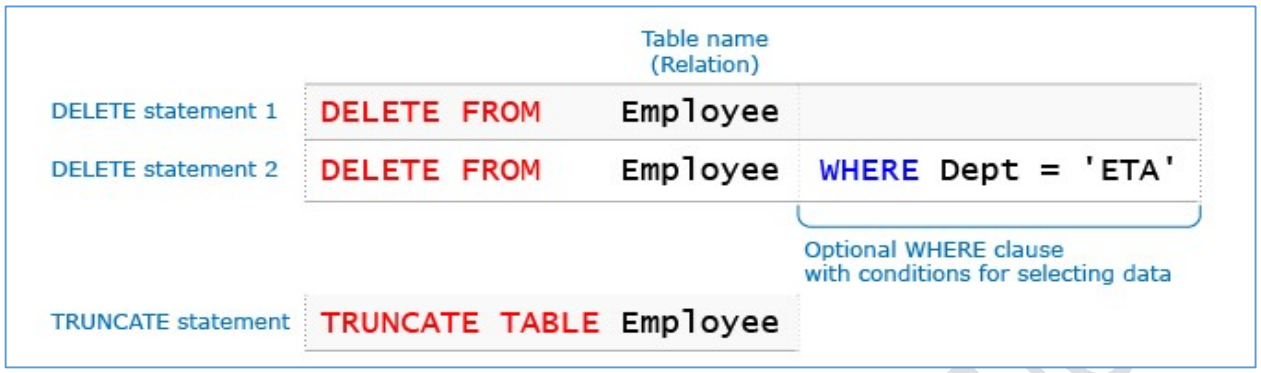
Remove the selected rows from the table

Syntax:

DELETE FROM table-name WHERE condition

What is the difference between Truncate and Delete?

DELETE	TRUNCATE
Data can be recovered	Data cannot be recovered.
DML statement	DDL statement
DELETE does not release the memory occupied by the records of the table	TRUNCATE releases the memory occupied by the records of the table
We can remove selected data from the table using "WHERE" clause	We don't have an option to chose the selected data to be removed



Chapter – 12

DDL – Truncate Query

Truncate

Syntax:

TRUNCATE TABLE table-name

Chapter – 13

DDL – Drop Query

Drop Table

Used to remove the table contents along with the structure of the table

Syntax:

Drop table <<table-name>>

Chapter – 14

DDL – Alter Query

How to modify the structure of the table?

- Alter Query is used to modify the structure of the table
- We can perform the below operation using alter command
  - Add or modify or delete or rename a column
  - Add or delete or disable a constraint

Syntax:

Adding a new column	ALTER TABLE <Table name> add <<column-name datatype(size)>>
Adding multiple columns	ALTER TABLE<table name> add (col1 datatype(size), col2 datatype(size) ----, coln datatype(size))
Modify the column definition	ALTER TABLE <Table name> MODIFY <Column name> data type(size)
	ALTER TABLE <Table name> MODIFY (col1 data type(size), col2 data type(size) ----, coln data type(size))
Delete a column	ALTER TABLE< table name> DROP COLUMN <<COLUMN NAME>>
Rename Column	ALTER TABLE<table name> rename column<old column name> to <new column name)

Constraints

Adding a new constraint	ALTER TABLE <Table name> add constraint <<constraint-defination>>
Delete a constraint	ALTER TABLE <Table name> drop constraint <<constraint-name>>

Example:

	Clauses
Alter statement 1	ALTER TABLE Student ADD Address VARCHAR2(20)
Alter statement 2	ALTER TABLE Student MODIFY Address VARCHAR2(50)
Alter statement 3	ALTER TABLE Student RENAME COLUMN Address TO ResidentialAddress
Alter statement 4	ALTER TABLE Student DROP (ResidentialAddress)
Alter statement 5	ALTER TABLE Student ADD CONSTRAINT stud_sid_pk PRIMARY KEY (StudentId)
Alter statement 6	ALTER TABLE Student DROP CONSTRAINT stud_sid_pk

Chapter – 15

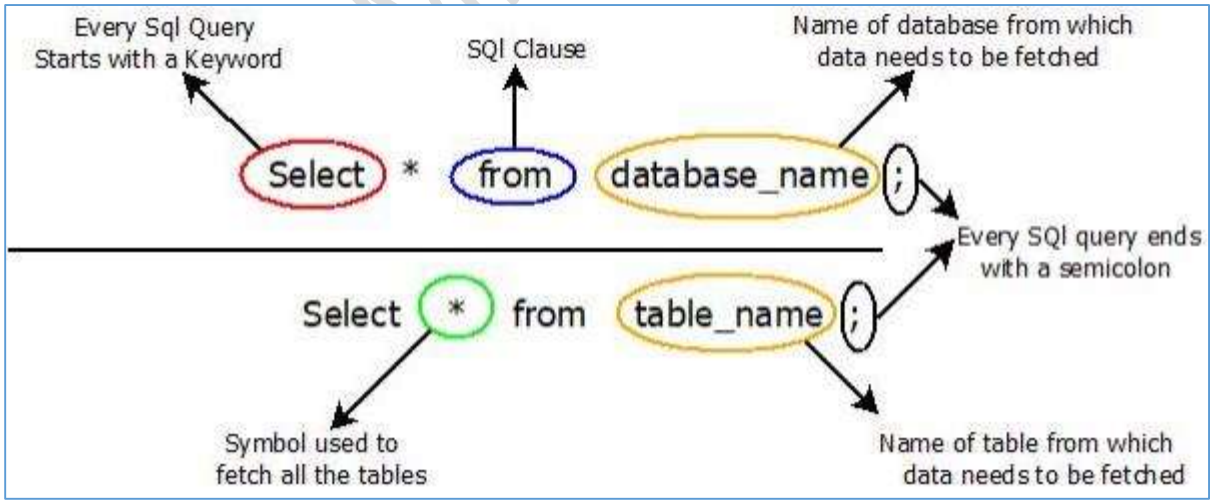
DRL – Select Query

How to Retrieve the Data from the table?

- SELECT command is used to retrieve the data from the existing table.
- Using this command
  - we can retrieve all records or
  - We can retrieve some specific records in the table(Using where clause).

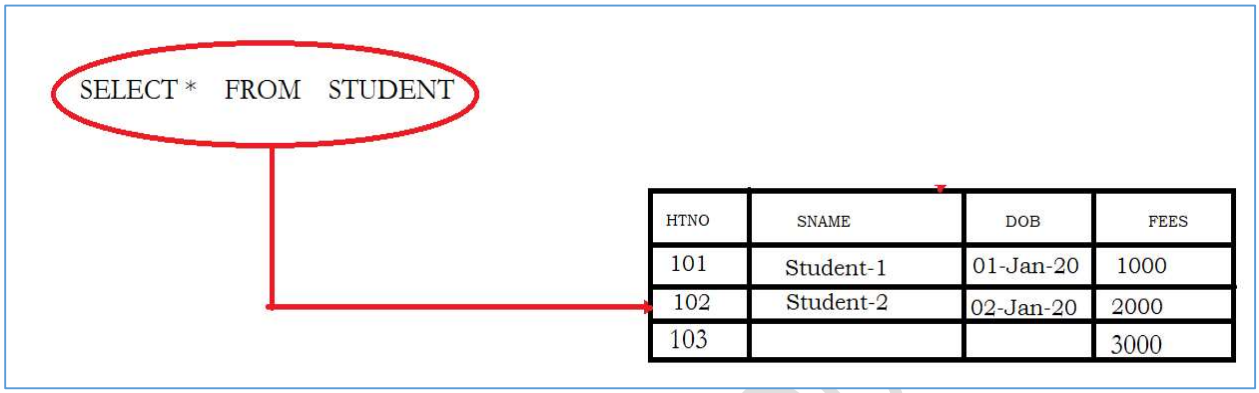
```
SELECT [ ALL / DISTINCT ] column-name1, column-name2, ------ FROM table-specification
                    [ WHERE search-condition ]
                  [ GROUP BY grouping column ]
                    [ HAVING search-condition ]
                  [ ORDER BY sort-specification ]
```

Basic Select Command Syntax:

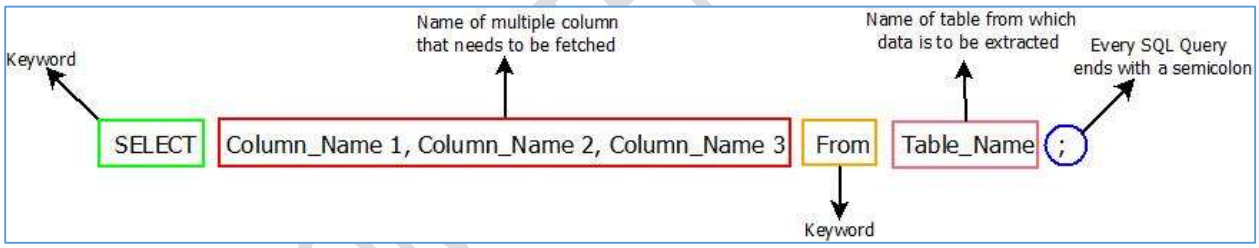


- Above Query is used to retrieve the complete information saved in a table.

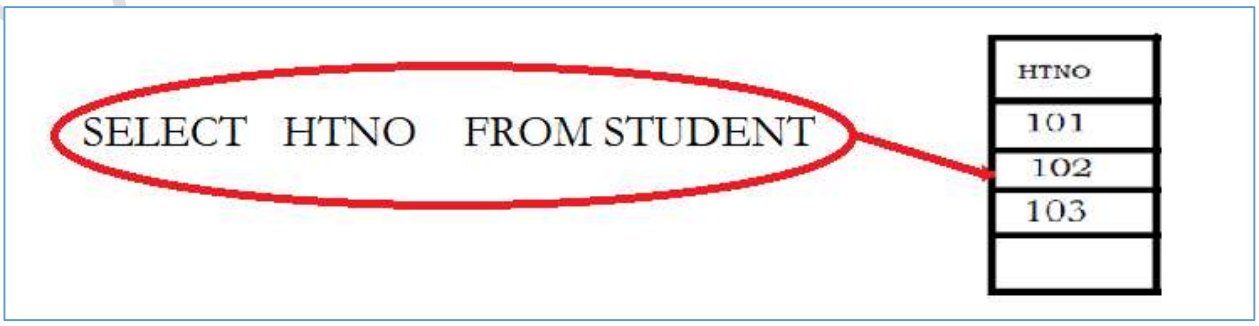
- Complete Information means
    - All rows in a table -→ All entitities
    - All columns in a table → All properties of an entity

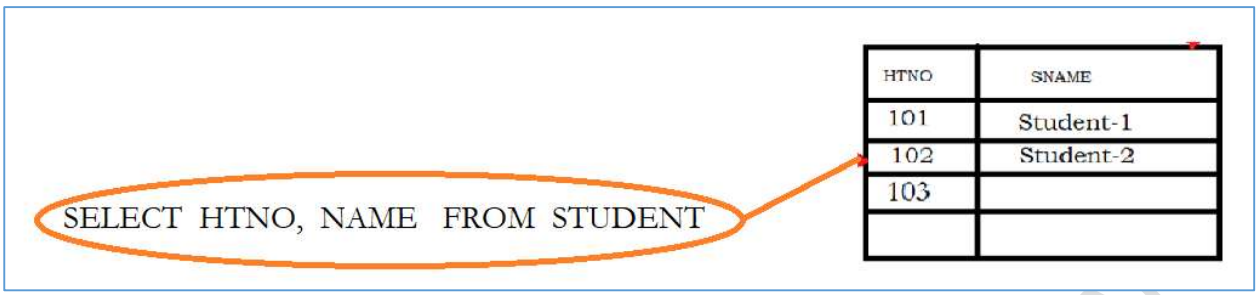


How to retrieve the selected columns from the table?



- Above Query is used to retrieve the all rows saved in a table.
- Result contains means
    - All rows in a table -→ All entitities
    - Listed columns in the query → All properties of an entity





Chapter – 16

DRL – Select Query using Where Clause

- DRL-SELECT Statements using WHERE clause
  - Comparison and Conditional Operators
  - Arithmetic and Logical Operators
  - Set Operators (UNION, UNION ALL, INTERSECT, MINUS)
  - Special Operators – IN (NOT IN), BETWEEN (NOT BETWEEN), LIKE (NOTLIKE),
  - IS NULL (IS NOT NULL)
  - Operators Support

Purpose of Where clause

- Where Clause in the select query is used to filter the rows present in the table
- SQL engines retrieves or fetches only those rows that satisfies the given criteria

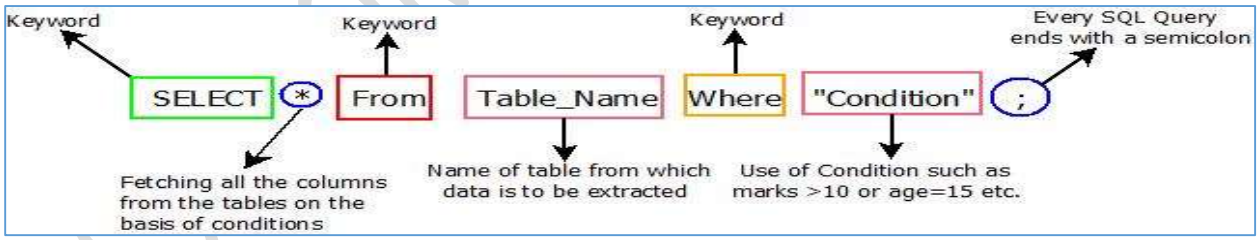
Use Case – 1

How to retrieve the selected rows from the table?

Syntax:

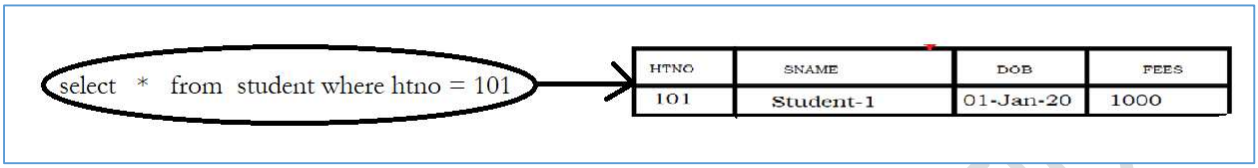
SELECT * from <<table-name>> where <<selection-criteria>>

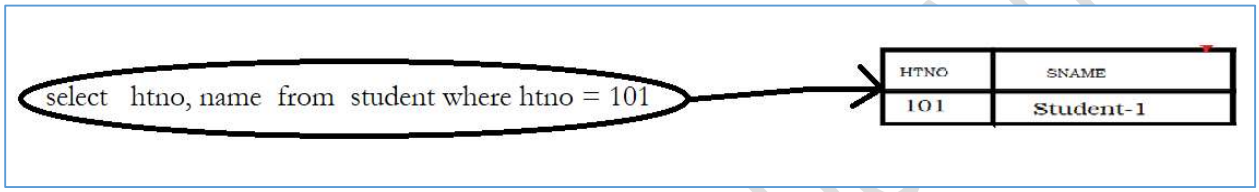
SELECT <<columns-list>> from <<table-name>> where <<selection-criteria>>



- Above Query is used to retrieve the required information from the table based on the criteria.
- SQL filters the data based on the condition and retrieves those data that satisfies the condition
- Complete Information means

- All rows in a table -→ List of rows that satisfies the given criteria
- All columns in a table → All properties of an entity





Operators in where clause

Arithmetic Operators

Operator	Symbol	Usage	Result
Addition	+	15 + 5	20
Subtraction	-	15 - 5	10
Multiplication	*	15 * 5	75
Division	/	15 / 5	3

Comparison Operators

Operator	Symbol	Usage	Result
Equal to	=	15 = 5	false
Not equal to	<>	15 <> 5	true
Greater than	>	15 > 5	true
Greater than equal to	>=	15 >= 5	true
Less than	<	15 < 5	false
Less than equal to	<=	15 <= 5	false

Logical Operators

Operator	Symbol	Usage	Example
And	AND	Returns TRUE if both conditions are true	Salary >= 30000 AND Dept = 'ETA'
Or	OR	Returns TRUE if any one of the condition is true	Salary > 75000 OR Dept = 'ICP'
Not	NOT	Returns TRUE if following condition is false	Id NOT IN (2,3)

Other Comparison Operators

Operator	Symbol	Usage	Example
Range	BETWEEN <*lower limit*> AND <*upper limit*>	Matches value between a range of values (Both inclusive)	Salary BETWEEN 2500 AND 3000
List	IN (List of values)	Matches any of a list of values	Dept IN ('IVS', 'ETA', 'ICP')
String pattern matching	LIKE	Matches a character pattern	SupplierId LIKE 'S%'
NULL Test	IS NULL	Is a null value	Bonus IS NULL

Pattern Matching:

- This technique is used to filter the rows from the table whenever the actual value is not known.
- We have to use 'Like' operator for comparing the pattern with the value present in the column.
- The pattern should be enclosed in single quotes.
- We can use this technique for all data types.

Use Cases:

- List of student names starts with 'A'
- List of students who got single digit marks

Wild Card Characters in Oracle:

- % → 0 or more characters
- _(Underscore) → Single Character

Chapter – 17

DRL – Select Query – Group By clause

- Grouping the Result of a Query
  - Using Group by and Having Clause of DRL Statement
  - Using Order by clause

- Group by clause is used to retrieve the consolidated information from the table based on some column(s).
- SQL virtually divides the complete table into different groups based on the column(s) specified in the group by clause
- we can perform the below operations on each individual group:
  - sum() → sum of all values present in the column
  - count() → count the number of not null values present in the column
  - max() → find the max value present in the column
  - min() → find the min value present in the column
  - avg() → to calculate the column average.
- Having clause in group by is used to filter the groups.

Syntax:

```
SELECT <<LIST-COLUMNS>> FROM <<TABLE-NAME>>

WHERE <<CONDITION>>

GROUP BY <<COLUMN-NAME(S)>> HAVING <<FILTER-THE-GROUPS>>

ORDER BY <<LIST-COLUMNS>> [ASC/DESC]
```

Use Cases of group by clause:

- Number of employees working in each department
- Number of students joined in each course
- Number of products purchased by each customer
- Number of accounts managed by a customer in bank
- And so on

Example:

```
create table employee(
empid number(5),
empname varchar2(20),
age number(5),
city varchar2(20),
salary number(10),
dob date);
```

Test Data:

```
insert into employee values (101,'Employee1',35,'Hyderabad',25000, '01-JAN-92');

insert into employee values(102,'Employee2',40,'Vijayawada',30000,'01-FEB-92');

insert into employee values
(103,'Employee3',35,'Hyderabad',25000,'01-JAN-93')

insert into employee values
(104,'Employee4',38,'Eluru',40000,'01-MAR-93');

insert into employee values
(105,'Employee5',35,'Rajamundry',25000,'01-JAN-92');

insert into employee values
(106,'Employee6',35,'Tirupathi',25000,01-JAN-95);

insert into employee values
```

(107,'Employee7',45,'Vijayawada','10-JAN-95');

insert into employee values

(108,'Employee8',45,'Vijayawada',50000,'10-JAN-95');

insert into employee values

(109,'Employee9',45,'Vijayawada',0,'10-JAN-95');

insert into employee values

(110,'Employee10',45,'Hyderabad',25000 '01-FEB-93');

Queries

SQL> select * from employee;

EMPID	EMPNAME	AGE	CITY	SALARY	DOB
101	Employee1	35	Hyderabad	25000	01-JAN-92
102	Employee2	40	Vijayawada	30000	01-FEB-92
103	Employee3	35	Hyderabad	25000	01-JAN-93
104	Employee4	38	Eluru	40000	01-MAR-93
105	Employee5	35	Rajamundry	25000	01-JAN-92
106	Employee6	35	Tirupathi	25000	01-JAN-95
107	Employee7	45	Vijayawada		10-JAN-95
108	Employee8	45	Vijayawada	50000	10-JAN-95
109	Employee9	45	Vijayawada	0	10-JAN-95
110	Employee1	45	Hyderabad	25000	01-FEB-93

10 rows selected.

SQL> select city,count(*) from employee group by city;

```
CITY        COUNT(*)
---------- ----------
Rajamundry          1
Eluru               1
Vijayawada          4
Hyderabad           3
Tirupathi           1
```

SQL> select city, sum(salary) from employee group by city;

```
CITY       SUM(SALARY)
---------- -----------
Rajamundry       25000
Eluru            40000
Vijayawada       80000
Hyderabad        75000
Tirupathi        25000
```

SQL> select city,max(salary) from employee group by city;

```
CITY       MAX(SALARY)
---------- -----------
Rajamundry       25000
```

Eluru 40000

Vijayawada 50000

Hyderabad 25000

Tirupathi 25000

SQL> select city,min(salary) from employee group by city;

CITY MIN(SALARY)

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

Rajamundry 25000

Eluru 40000

Vijayawada 0

Hyderabad 25000

Tirupathi 25000

SQL> select city,avg(salary) from employee group by city;

CITY AVG(SALARY)

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

Rajamundry 25000

Eluru 40000

Vijayawada 26666.6667

Hyderabad 25000

Tirupathi 25000

SQL>

Having clause

---

is applied on groups that each group contains related rows.

Having clause is used to filter the groups based on some criteria.

Q1. retrieve the list of cities contains more than two employees are working.

```
SQL> select city,count(*) from employee group by city;

CITY        COUNT(*)
---------- ----------
Rajamundry          1
Eluru               1
Vijayawada          4
Hyderabad           3
Tirupathi           1

SQL> select city,count(*) from employee group by city having count(*)>2;

CITY        COUNT(*)
---------- ----------
Vijayawada          4
Hyderabad           3

SQL>
```

Chapter – 18

DRL – Select Query – Order by clause

ORDER BY: used to sort the result-set in ascending or descending order

```
SELECT * FROM table_name ORDER BY column;
SELECT * FROM table_name ORDER BY column DESC;
SELECT * FROM table_name ORDER BY column1 ASC, column2 DESC;
```

```
SQL> select city,count(*) from employee group by city;

CITY        COUNT(*)
---------- ----------
Rajamundry          1
Eluru               1
Vijayawada          4
Hyderabad           3
Tirupathi           1

SQL> select city,count(*) from employee group by city order by city;

CITY        COUNT(*)
---------- ----------
Eluru               1
Hyderabad           3
Rajamundry          1
Tirupathi           1
Vijayawada          4
```

```
SQL> select city,count(*) from employee group by city order by city desc
  2  ;

CITY        COUNT(*)
---------- ----------
Vijayawada          4
Tirupathi           1
Rajamundry          1
Hyderabad           3
Eluru               1

SQL>


SQL> select city,count(*) from employee group by city order by  count(*) desc;

CITY        COUNT(*)
---------- ----------
Vijayawada          4
Hyderabad           3
Eluru               1
Tirupathi           1
Rajamundry          1

SQL> select city,count(*) from employee group by city order by  count(*)
desc,city;

CITY        COUNT(*)
```

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

Vijayawada 4

Hyderabad 3

Eluru 1

Rajamundry 1

Tirupathi 1

SQL> select city,count(*) from employee group by city order by count(*) desc,city desc;

CITY COUNT(*)

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

Vijayawada 4

Hyderabad 3

Tirupathi 1

Rajamundry 1

Eluru 1

SQL>

Chapter – 19

Joining two tables

- Querying Multiple Tables (Joins)
  - Equi Join/Inner Join/Simple Join
  - Cartesian Join
  - Non-Equi Join
  - Outer Joins
  - Self-Join

Joins

---

this mechanism is used to retrieve the required information by combining multiple tables.

Employee (empid,empname,designation,deptid,branchid)

department(deptid,deptname)

branch(branchid,branchname,address)

Types:

1. INNER JOIN OR EQUI JOIN
2. OUTER JOIN
   2.1 : LEFT OUTER JOIN
   2.2 : RIGHT OUTER JOIN
   2.3 : FULL JOIN
3. CROSS JOIN

INNER JOIN OR EQUI JOIN

EQUI - JOIN or INNER - JOIN is

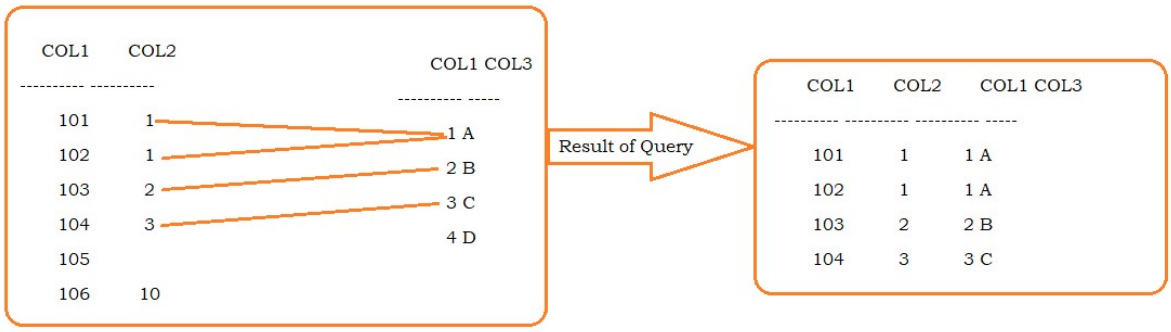
fetches only the matched rows from both the tables.

ignores the unmatched rows in both the tables.

Use Case 1 ---> no values are repeated in second table

```
create table join1Table(
col1 number(5),
col2 number(5));

insert into join1Table values(101,1);
insert into join1Table values(102,1);
insert into join1Table values(103,2);
insert into join1Table values(104,3);
insert into join1Table values(105,NULL);
insert into join1Table values(106,10);

SQL> select * from join1Table;

      COL1       COL2
---------- ----------
       101          1
       102          1
       103          2
       104          3
       105
       106         10

6 rows selected.
```

```
create table join2Table(
col1 number(5),
col3 varchar2(5));

insert into join2Table values(1,'A');
insert into join2Table values(2,'B');
insert into join2Table values(3,'C');
insert into join2Table values(4,'D');

SQL> select * from join2Table;

      COL1 COL3
---------- -----
         1 A
         2 B
         3 C
         4 D
```



```
SQL> select * from
                join1Table j1
                        inner join
                                join2Table j2
                                        on
                                           j1.col2=j2.col1;

      COL1       COL2       COL1 COL3
---------- ---------- ---------- -----
       101          1          1 A
       102          1          1 A
       103          2          2 B
       104          3          3 C

SQL>

SQL>
select col1,col2,col3
                        from
                          join1table j1
                                inner join
                                     join2table j2
                                             on j1.col2=j2.col1;
Error Message:
select col1,col2,col3 from join1table j1 inner join join2table j2 on
j1.col2=j2.col1
       *
ERROR at line 1:
```

ORA-00918: column ambiguously defined

Solution:

```
SQL>
select
        j1.col1, j1.col2, j2.col3
                from
                   join1table j1
                      inner join join2table j2
                              on j1.col2=j2.col1;

     COL1       COL2 COL3
---------- ---------- -----
       101          1 A
       102          1 A
       103          2 B
       104          3 C
```

Joins and Group By clause

```
SQL>
        select j2.col3,count(*)
                from
                   join1table j1 inner join join2table j2
                              on j1.col2=j2.col1
                                 group by j2.col3;
COL3    COUNT(*)
----- ----------
```

A 2

B 1

C 1

Using Join, where and group by clauses

```
SQL>
        select
           j2.col3,count(*)
              from join1table j1 inner join join2table j2
                    on j1.col2=j2.col1
                      where j2.col1>2
                            group by j2.col3;
```

COL3 COUNT(*)

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

C 1

SQL> select j2.col3,count(*) from join1table j1 inner join join2table j2 on j1.col2=j2.col1 where j2.col1>=2 group by j2.col3;

COL3 COUNT(*)

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

B 1

C 1

SQL>

Use Case – 2 --- Second table has duplicate values

```
create table join3Table(
```

col1 number(5),

col3 varchar2(5));

insert into join3Table values(1,'A');

insert into join3Table values(1,'A1');

insert into join3Table values(2,'B');

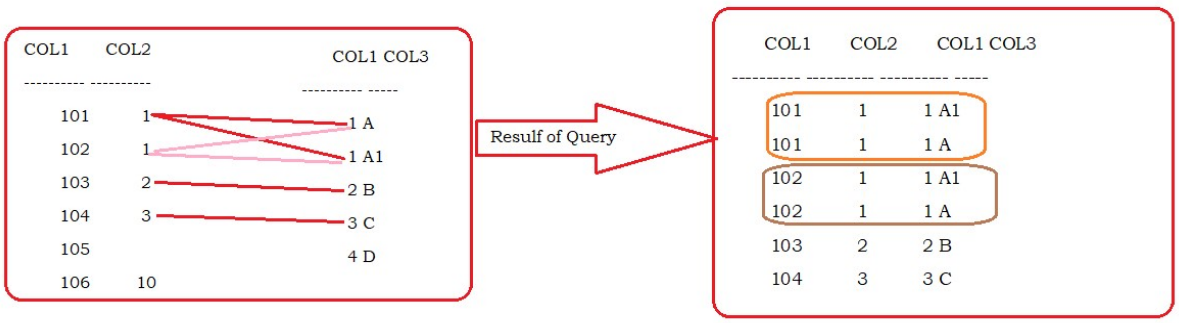
insert into join3Table values(3,'C');

insert into join3Table values(4,'D');

SQL> select * from join3Table;

```
      COL1 COL3
---------- -----
         1 A
         1 A1
         2 B
         3 C
         4 D
```

SQL>



SQL>

```
select * from
        join1Table j1 inner join join3Table j2
                on j1.col2=j2.col1;
```

COL1	COL2	COL1	COL3
101	1	1	A1
101	1	1	A
102	1	1	A1
102	1	1	A
103	2	2	B
104	3	3	C

6 rows selected.

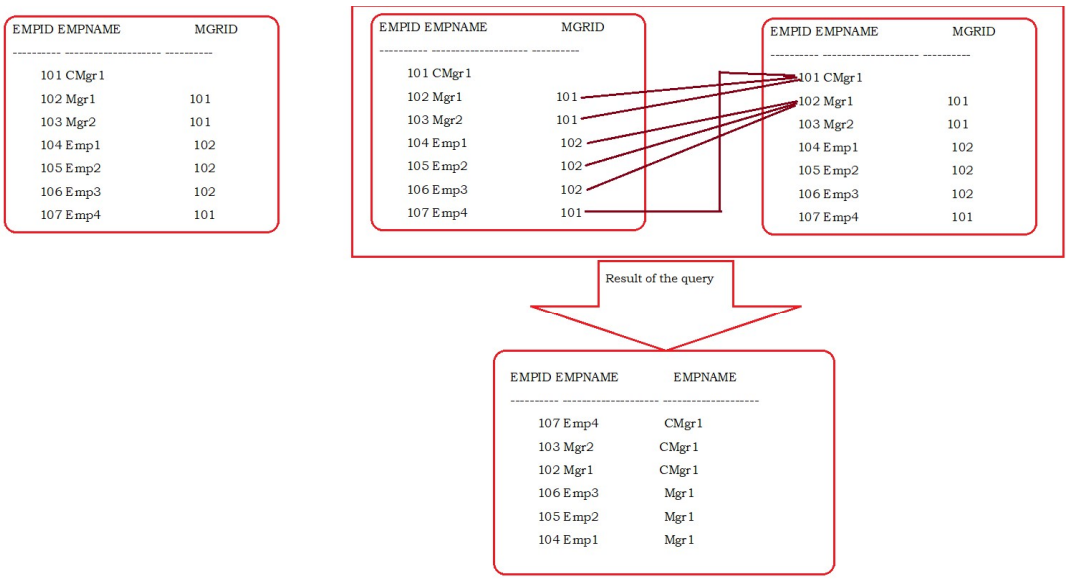
Use Case - 3 ---> Self join ---> Join one table

```
create table EmpMgrTable(
 empid number(5),
 empname varchar2(20),
 mgrid number(5));
```

```
insert into empmgrtable values(101,'CMgr1',NULL);
```

```
insert into empmgrtable values(102,'Mgr1',101);
insert into empmgrtable values(103,'Mgr2',101);
insert into empmgrtable values(104,'Emp1',102);
insert into empmgrtable values(105,'Emp2',102);
insert into empmgrtable values(106,'Emp3',102);
insert into empmgrtable values(107,'Emp4',101);
```

```
SQL> select * from empmgrtable;

     EMPID EMPNAME              MGRID
---------- -------------------- ----------
       101 CMgr1
       102 Mgr1                    101
       103 Mgr2                    101
       104 Emp1                    102
       105 Emp2                    102
       106 Emp3                    102
       107 Emp4                    101

7 rows selected.
```



SQL>

select

j1.empid, j1.empname, j2.empname

from empmgrtable j1 **inner join** empmgrtable j2

on j1.mgrid=j2.empid;

EMPID	EMPNAME	EMPNAME
107	Emp4	CMgr1
103	Mgr2	CMgr1
102	Mgr1	CMgr1
106	Emp3	Mgr1
105	Emp2	Mgr1
104	Emp1	Mgr1

6 rows selected.

SQL query to retrieve the empid,empname and manager name from the employee table.

```
SQL> select
        t1.empid,  t1.empname,  t2.empname "Manager Name"
        from empmgrtable t1 join empmgrtable t2
        on t1.mgrid=t2.empid;

    EMPID EMPNAME          Manager Name
---------- ------------------- --------------------
       107 Emp4               CMgr1
       103 Mgr2               CMgr1
       102 Mgr1               CMgr1
       106 Emp3               Mgr1
       105 Emp2               Mgr1
       104 Emp1               Mgr1

6 rows selected.

SQL>
```

Another Example:

Order(Id,OrderDate,OrderNumber,CustomerId,TotalAmount)

Customer(Id,FirstName,LastName,City,Country,Phone)

List all orders with customer information

```
SELECT OrderNumber, TotalAmount, FirstName, LastName, City, Country
        FROM Order INNER JOIN Customer C
                ON C.Id = O.CustomerId
```

```
SELECT *
        FROM
            emp e INNER JOIN dept d
                ON e.deptno = d.deptno
```

Table 1: Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	null	11/17/1981	5000	100	10
7698	BLAKE	MANAGER	7839	5/1/1981	2850	null	30
7782	CLARK	MANAGER	7839	5/1/1981	2450	100	10
7566	JONES	MANAGER	7839	5/1/1981	2975	120	20
7788	SCOTT	ANALYST	7566	4/19/1987	3000	120	20
7902	FORD	ANALYST	7566	12/3/1981	3000	120	20
7369	SMITH	CLERK	7902	12/17/1980	800	120	20
7499	ALLEN	SALESMAN	7698	9/28/1981	1600	300	30
7521	WARD	SALESMAN	7698	9/28/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	9/28/1981	1250	1400	30

Table 2: Dept

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
70	MARKETING	ATLANTA

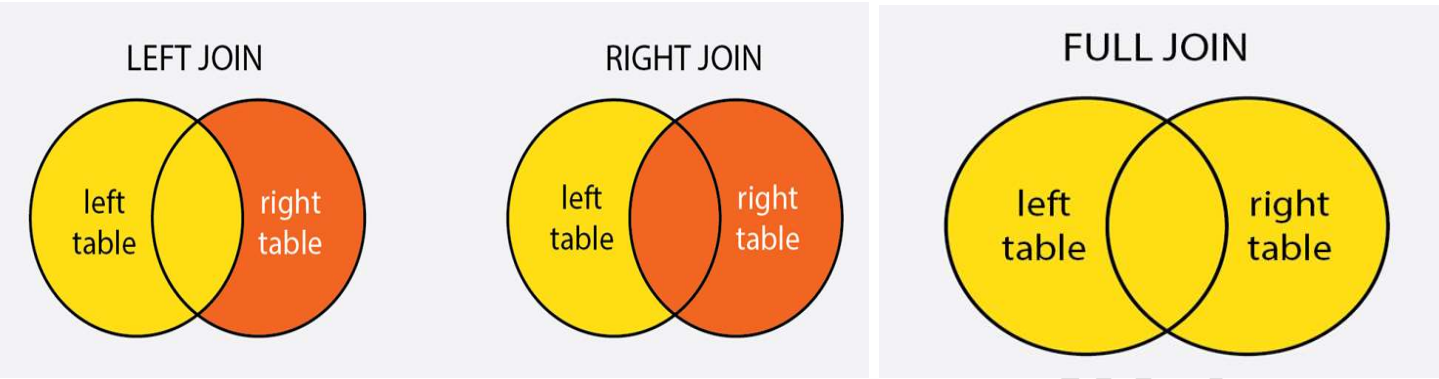
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7839	KING	PRESIDENT	null	########	5000	100	10	10	ACCOUNTING	NEW YORK
7782	CLARK	MANAGER	7839	6/9/1981	2450	100	10	10	ACCOUNTING	NEW YORK
7788	SCOTT	ANALYST	7566	########	3000	120	20	20	RESEARCH	DALLAS
7369	SMITH	CLERK	7902	########	800	120	20	20	RESEARCH	DALLAS
7902	FORD	ANALYST	7566	########	3000	120	20	20	RESEARCH	DALLAS
7566	JONES	MANAGER	7839	4/2/1981	2975	120	20	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	7698	########	1600	300	30	30	SALES	CHICAGO
7698	BLAKE	MANAGER	7839	5/1/1981	2850	null	30	30	SALES	CHICAGO

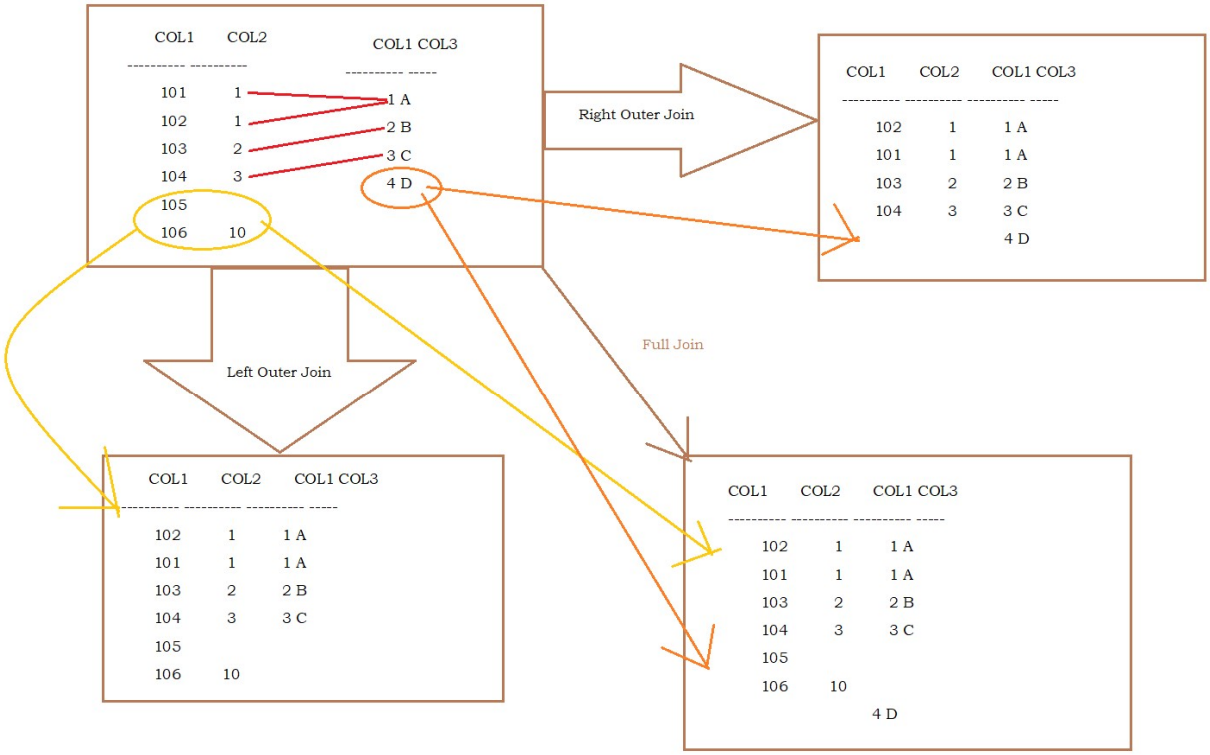
OUTER JOIN

- Fetches both matched rows and unmatched rows from which table means it depends on the type of join we used.

Three types of outer joins

- Left Outer Join or Left Join
- Right Outer Join or Right Join
- Full Join





LEFT JOIN

- Also called as LEFT OUTER JOIN
- fetches the matched rows from both the tables
- fetches the unmatched rows from LEFT HAND SIDE TABLE
- ignores the unmatched rows from RIGHT HAND SIDE TABLE
- RESULT CONTAINS ALL THE ROWS PRESENT IN THE LEFT HAND SIDE TABLE

SQL> select * from join1table;

```
      COL1       COL2
---------- ----------
       101          1
       102          1
       103          2
       104          3
       105
       106         10
```

6 rows selected.

SQL> select * from join2table;

```
      COL1 COL3
---------- -----
         1 A
         2 B
         3 C
         4 D
```

SQL> select * from join1table j1 left join join2table j2 on j1.col2=j2.col1;

```
      COL1       COL2       COL1 COL3
---------- ---------- ---------- -----
       102          1          1 A
       101          1          1 A
```

103	2	2 B
104	3	3 C
105		
106	10	

6 rows selected.

SQL> select * from join1table j1 left join join3table j2 on j1.col2=j2.col1;

COL1	COL2	COL1	COL3
102	1	1	A
101	1	1	A
102	1	1	A1
101	1	1	A1
103	2	2	B
104	3	3	C
105			
106	10		

8 rows selected.

SQL> select * from join1table j1 left outer join join3table j2 on j1.col2=j2.col1;

COL1	COL2	COL1	COL3
102	1	1	A
101	1	1	A

102	1	1 A1
101	1	1 A1
103	2	2 B
104	3	3 C
105		
106	10	

8 rows selected.

SQL>

SQL> select * from join1table;

COL1	COL2
101	1
102	1
103	2
104	3
105	
106	10

6 rows selected.

SQL> select * from join2table;

```
      COL1 COL3
---------- -----
         1 A
         2 B
         3 C
         4 D
```

SQL> select * from join1table j1 left join join2table j2 on j1.col2=j2.col1;

```
      COL1       COL2       COL1 COL3
---------- ---------- ---------- -----
       102          1          1 A
       101          1          1 A
       103          2          2 B
       104          3          3 C
       105
       106         10
```

6 rows selected.

SQL> select * from join1table j1 left join join3table j2 on j1.col2=j2.col1;

```
      COL1       COL2       COL1 COL3
---------- ---------- ---------- -----
       102          1          1 A
       101          1          1 A
```

```
       102          1          1 A1
       101          1          1 A1
       103          2          2 B
       104          3          3 C
       105
       106         10
```

8 rows selected.

SQL> select * from join1table j1 left outer join join3table j2 on j1.col2=j2.col1;

```
      COL1       COL2       COL1 COL3
---------- ---------- ---------- -----
       102          1          1 A
       101          1          1 A
       102          1          1 A1
       101          1          1 A1
       103          2          2 B
       104          3          3 C
       105
       106         10
```

8 rows selected.

SQL>

RIGHT JOIN

- Also called as RIGHT OUTER JOIN
- fetches the matched rows from both the tables

- fetches the unmatched rows from RIGHT HAND SIDE TABLE
- ignores the unmatched rows from LEFT HAND SIDE TABLE
- RESULT CONTAINS ALL THE ROWS PRESENT IN THE RIGHT HAND SIDE TABLE

SQL> select * from join1table;

```
      COL1       COL2
---------- ----------
       101          1
       102          1
       103          2
       104          3
       105
       106         10

6 rows selected.
```

SQL> select * from join2table;

```
      COL1 COL3
---------- -----
         1 A
         2 B
         3 C
         4 D
```

SQL> select * from join1table j1 right join join2table j2 on j1.col2=j2.col1;

COL1	COL2	COL1	COL3
102	1	1	A
101	1	1	A
103	2	2	B
104	3	3	C
		4	D

5 rows selected.

FULL JOIN

- Also called as FULL OUTER JOIN
- fetches the matched rows from both the tables
- fetches the unmatched rows from RIGHT HAND SIDE TABLE
- fetches the unmatched rows from LEFT HAND SIDE TABLE
- RESULT CONTAINS ALL THE ROWS PRESENT IN THE RIGHT HAND SIDE TABLE AND ALL THE ROWS PRESENT IN THE LEFT HAND SIDE TABLE.

SQL> select * from join1table j1 full join join2table j2 on j1.col2=j2.col1;

COL1	COL2	COL1	COL3
102	1	1	A
101	1	1	A
103	2	2	B
104	3	3	C
105			
106	10		

4 D

7 rows selected.

SQL> select * from join1table j1 full join join3table j2 on j1.col2=j2.col1;

COL1	COL2	COL1	COL3
102	1	1	A
101	1	1	A
102	1	1	A1
101	1	1	A1
103	2	2	B
104	3	3	C
105			
106	10		

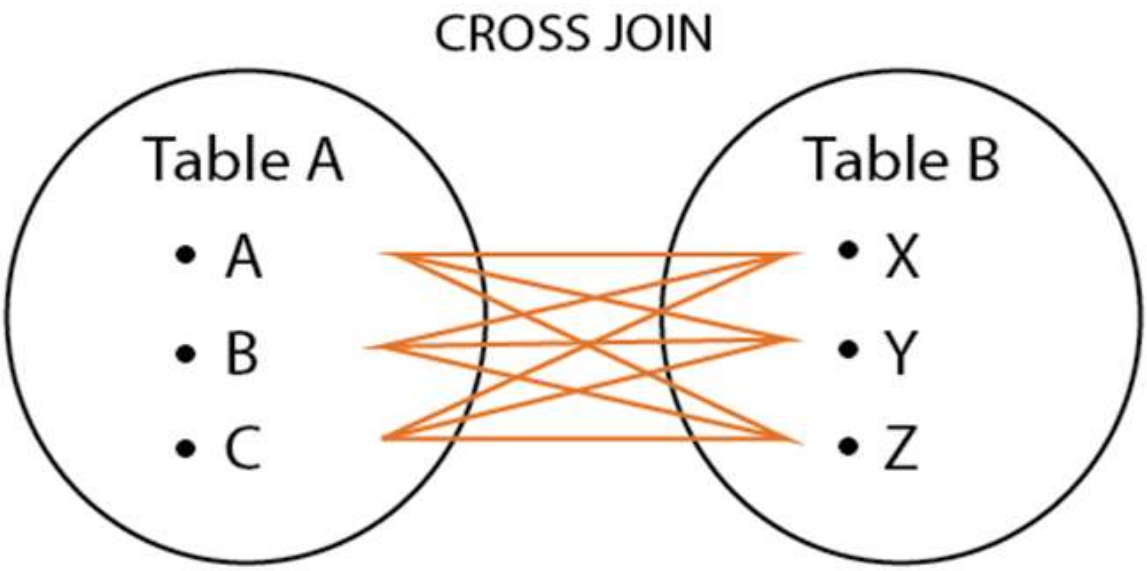
4 D

9 rows selected.

SQL>

CROSS- JOIN

- Every row in the first table maps to all the rows in the second table (1 to N).



SQL> select * from join1table cross join join2table;

COL1	COL2	COL1	COL3
101	1	1	A
102	1	1	A
103	2	1	A
104	3	1	A
105		1	A
106	10	1	A
101	1	2	B
102	1	2	B
103	2	2	B
104	3	2	B
105		2	B

COL1	COL2	COL1	COL3
106	10		2 B
101	1		3 C
102	1		3 C
103	2		3 C
104	3		3 C
105			3 C
106	10		3 C
101	1		4 D
102	1		4 D
103	2		4 D
104	3		4 D

COL1	COL2	COL1	COL3
105			4 D
106	10		4 D

24 rows selected.

SQL> select * from join1table;

COL1	COL2
101	1
102	1

103 2

104 3

105

106 10

6 rows selected.

SQL> select * from join2table;

COL1 COL3

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

1 A

2 B

3 C

4 D

SQL>

Important Points

- Inner joins output the matching rows from the join condition in both of the tables.
- Cross join returns the Cartesian product of both tables.
- Outer join returns the matched and unmatched rows depending upon the left, right and full keywords.
- SQL self-join joins a table to itself.
- You should always use a table alias while using the joins in the queries.
- Always use the two-part name [table alias].[column] name format for columns in queries.
- In the case of multiple SQL joins in a query, you should use the logical orders of the tables in such a way to satisfy your data requirement

and minimize the data flow between various operators of the execution plan.

- You can combine multiple joins such as inner join, outer join and self-join together. However, you should use the joins and their orders to get the required data.

Chapter – 20

Sub Queries and Nested Queries

Working with Sub Queries

- What is the purpose of a Sub Query?
- Type of Sub Queries
  - Single Row
  - Multiple Row
- Applying Group Functions in Sub Queries
- The impact of Having Clause in Sub Queries
- IN, ANY/SOME, ALL Operators in Sub Queries
- Correlated Sub Queries
- Handling Data Retrieval with EXISTS and NOT EXISTS Operators

Introduction to Nested Queries or Sub Queries

- A query contains another query is known as nested query or sub query.
- We can write query into a query upto 256 queries.
- In between the outer query and inner query,
  - inner query executed first then it will execute the outer query .
- Subquery is used when we know how to search for a value using a SELECT statement, but do not know the exact value.

When :

Whenever we have to execute more than one query in a particular sequence in order to retrieve the result from Database, then we have to go with the concept called NESTED QUERIES OR INNER QUERIES

Use Case – 1

Employee Table

empId	empName	designation	departmentId	ManagerId
101	Mgr1	Manager	1	-
102	clerk1	clerk	1	101
103	clerk2	clerk	2	104
104	Mgr2	Manager	2	-

Query:
retrieve the clerk1 manager's name

Step 1: find the managerId of the employee whose name is "clerk1'

select managerId from employee where empName like 'clerk1' ----> 101

Step 2 : find the name of the employee whose empId is 101

select empName from employee where empId=101

Output: Mgr1

Using Nested Query: select empName from employee where empId = (select managerId from employee where empName like 'clerk1')

Result : Mag1

Use Case – 2

Employee Table

empId	empName	designation	departmentId	ManagerId
101	Mgr1	Manager	1	-
102	clerk1	clerk	1	101
103	clerk2	clerk	2	104
104	Mgr2	Manager	2	-

department table

departmentId	deptName
1	accounts
2	HR
3	finance

Requirement : retrieve the list of employees who are working in accounts department

Step 1: retrieve the departmentId of "accounts" department from department table

select departmentId from department where deptName like 'accounts'----> 1

Step 2 : retrieve the list of employees from employees table who are working in department with departmentId 1

select * from employee where departmentId = 1

Inner Query : select * from employee where departmentId=(select departmentId from department where deptName like 'accounts'

Q) List the employees who got highest pay.

```
select * from emp where sal =  (select max(sal) from emp );
```

Q) List the employees whose pay is MORE than Average pay.

```
select * from emp where sal > (select avg(sal) from emp );
```

Q)List the employees whose pay is LESS than Average pay.

```
select * from emp where sal < (select avg(sal) from emp );
```

Q) Remove the employees who didnt got increments.

```
delete from emp where empno not in (select distinct empno from incr );
```

Q) Double the comm if they got increment

```
update emp set comm = comm * 2  where empno in
                                (select distinct empno from incr );
```

TYPES OF SUB QUERIES

1. Single row subqueries
2. Multi row subqueries
3. Correlated subqueries

Single row subqueries

- In single row subquery, it will return one value.

Example:

SQL> select * from emp where sal > (select sal from emp where empno = 7566);

Multi row subqueries

- In multi row subquery, it will return more than one value.
- In such cases we should include operators like IN, ANY, ALL, EXISTS or NOT IN between the comparison operator and the subquery.
- As sub-query is returning more than one value, we can't use any relational operator alone in where clause of outer sub-query.

Example:

```
select * from emp where sal >
                    all (select sal from emp where sal between 2500 and 4000);
```

ALL Verb Use Case

```
SELECT empno, sal  FROM   emp        WHERE     sal > ALL (2000, 3000, 4000);

   EMPNO       SAL
---------- ----------
    7839       5000

SQL>

-- Transformed to equivalent statement without ALL.

SELECT empno, sal  FROM   emp        WHERE  sal > 2000 AND sal > 3000 AND sal > 4000;

   EMPNO       SAL
---------- ----------
    7839       5000

SQL>
```

ANY Verb Use Case

```
SELECT empno, sal    FROM   emp      WHERE  sal > ANY (2000, 3000, 4000);

   EMPNO       SAL
---------- ----------
    7566       2975
    7698       2850
    7782       2450
    7788       3000
    7839       5000
    7902       3000

SQL>

-- Transformed to equivalent statement without ANY.

SELECT empno, sal       FROM   emp   WHERE  sal > 2000 OR sal > 3000 OR sal > 4000;

   EMPNO       SAL
---------- ----------
    7566       2975
    7698       2850
    7782       2450
    7788       3000
    7839       5000
    7902       3000

SQL>
```

Difference between IN and EXISTS

IN	EXISTS
Works on result set list	Works on Virtual tables
the JOIN clause returns rows from another table	The EXISTS operator returns TRUE or FALSE
Doesn't work on subqueries resulting in Virtual tables with multiple columns	Is used with co-related queries
Compares every value in the result list	Exists comparison when match is found
Performance is comparatively SLOW for larger result set of subquery	Performance is comparatively FAST for larger result set of subquery

Example:

Customer Table

customer_id	last_name	first_name
4000	Jackson	Joe
5000	Smith	Jane
6000	Ferguson	Samantha
7000	Reynolds	Allen
8000	Anderson	Paige

customer_id	last_name	first_name
9000	Johnson	Derek

Orders Table

order_id	customer_id	order_date
1	7000	2016/04/18
2	5000	2016/04/18
3	8000	2016/04/19
4	4000	2016/04/20

Example for Exists:

```
SELECT * FROM Customers WHERE EXISTS
      ( SELECT * FROM Orders
        WHERE Orders.CustomerID = customers.Customers.ID )
```

Example for IN:

```
SELECT * FROM Customers  WHERE ID IN
                  (SELECT CustomerID FROM Orders )
```

Result is:

customer_id	last_name	first_name
4000	Jackson	Joe

customer_id	last_name	first_name
5000	Smith	Jane
7000	Reynolds	Allen
8000	Anderson	Paige

What is sub-query?

- we are writing a another query in where clause

What is Nested Query?

- we are writing a query in between select and from clauses.

Example for Nested Query

```
select
        s.htno,
        s.sname,
        (select cname from course where cid=s.cid) "course Name “
 from student2 s;
```

Multi column sub query

Example:

```
select * from coursestudent where (cid,fid) IN
                (select cid,(select fid from faculty where fname='Faculty1')
                                from course01 where cname='Java');
```

Some Other Examples:

```
create table course01(
cid number(5) primary key,
cname varchar2(20) not null,
 fees number(10,2));
```

Table created.

```
insert into course01 values(1,'Java',1000);
insert into course01 values(2,'C',2000);
insert into course01 values(3,'Oracle',3000);
```

```
select * from course;
```

CID	CNAME	FEES
1	Java	1000
2	C	2000
3	Oracle	3000

```
create table faculty(
 fid number(10) primary key,
 fname varchar2(20));
```

```
insert into faculty values(101,'Faculty1');
insert into faculty values(102,'Faculty2');
```

```
insert into faculty values(103,'Faculty3');

select * from faculty;

       FID FNAME
---------- --------------------
       101 Faculty1
       102 Faculty2
       103 Faculty3
create table coursestudent(
 sid number(10) primary key,
 sname varchar2(20),
 cid number(5) references course01(cid),
 fid number(10) references faculty(fid));


create table coursestudent(
 sid number(10) primary key,
 sname varchar2(20),
 cid number(5) references course(cid),
 fid number(10) references faculty(fid));

Table created.

insert into coursestudent values(1001,'student1',1,101);
insert into coursestudent values(1002,'student2',1,101);
insert into coursestudent values(1003,'student3',2,101);
insert into coursestudent values(1004,'student4',3,103);
```

select * from coursestudent;

SID	SNAME	CID	FID
1001	student1	1	101
1002	student2	1	101
1003	student3	2	101
1004	student4	3	103

select * from coursestudent;

SID	SNAME	CID	FID
1001	student1	1	101
1002	student2	1	101
1003	student3	2	101
1004	student4	3	103

select * from course;

CID	CNAME	FEES
1	Java	1000
2	C	2000
3	Oracle	3000
4	JSP	4000
5	servlets	5000

select * from faculty;

```
       FID FNAME
---------- --------------------
       101 Faculty1
       102 Faculty2
       103 Faculty3
```

write a subquery to retrieve all the students who joined for Java course

```
select * from coursestudent where cid=(select cid from course where
cname='Java');

       SID SNAME                    CID       FID
---------- -------------------- ---------- ----------
      1001 student1                     1        101
      1002 student2                     1        101
```

retrieve the course mates of student2

```
select cid from coursestudent where sname='student2';

       CID
----------
1
```

```
select * from coursestudent where cid=1;
```

HTNO	SNAME	CID
1001	student1	1
1002	student2	1

select * from coursestudent where cid=(select cid from coursestudent where sname='student2');

HTNO	SNAME	CID
1001	student1	101
1002	student2	101

SQL query to fetch the list of students joined for either Java or oracle course.

select * from coursestudent where cid IN (select cid from course where cname='Java' OR cname='Oracle');

SID	SNAME	CID	FID
1001	student1	1	101
1002	student2	1	101
1004	student4	3	103

List of all students who paid the fee > fee of java and > fee of C

SQL> select cs.* from coursestudent cs LEFT JOIN course01 c ON c.cid=cs.cid where c.fees > ANY (select fees from course01 where cname IN ('Java',

'C'));

SID	SNAME	CID	FID
1003	student3	2	101
1004	student4	3	103

List of students who paid thier fee equal to fees of Java or C

SQL> select cs.* from coursestudent cs LEFT JOIN course01 c ON c.cid=cs.cid where c.fees IN (select fees from course01 where cname IN ('Java','C'

));

SID	SNAME	CID	FID
1001	student1	1	101
1002	student2	1	101
1003	student3	2	101

SQL>

SQL query to retrieve the students who joned for Java course and the faculty name is faculty1?

select * from coursestudent where (cid,fid) IN (select cid,(select fid from faculty where fname='Faculty1') from course where cname='Java');

SID	SNAME	CID	FID

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

1001 student1 1 101

1002 student2 1 101

select * from coursestudent where (cid,fid) IN (select cid,(select fid from faculty where fname='Faculty2')

no rows selected

select cid from course where cname='Java';

CID

1

SQL quert to fetch the course id and id?

select cid,(select fid from faculty where fname='Faculty1') from course where cname='Java';

CID (SELECTFIDFROMFACULTYWHEREFNAME='FACULTY1')

---

1 101

select fid,(select cid from course where cname='Java') from faculty where fname='Faculty1';

FID (SELECTCIDFROMCOURSEWHERECNAME='JAVA')

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

101 1

select fid,(select cid from course where cname='Java') cid from faculty where fname='Faculty1';

FID CID

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

101 1

Chapter – 21

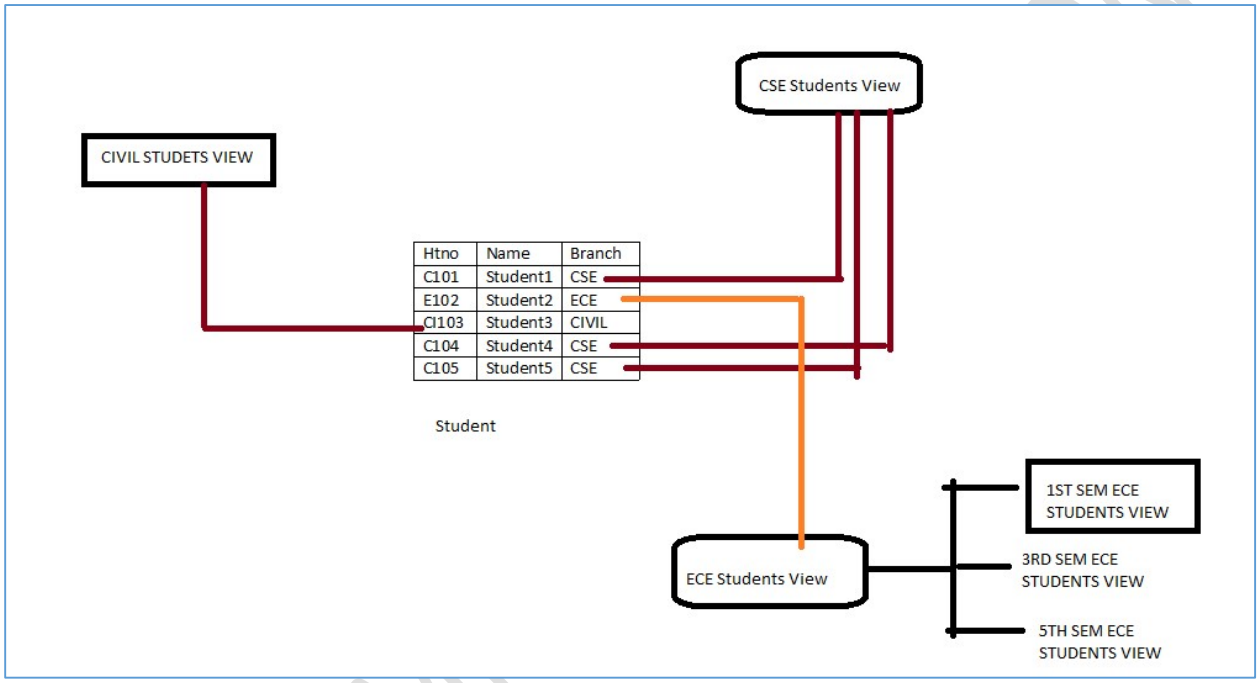
Working with DCL, TCL Commands

- Grant, Revoke
- Commit, Rollback, Savepoint

Chapter – 22

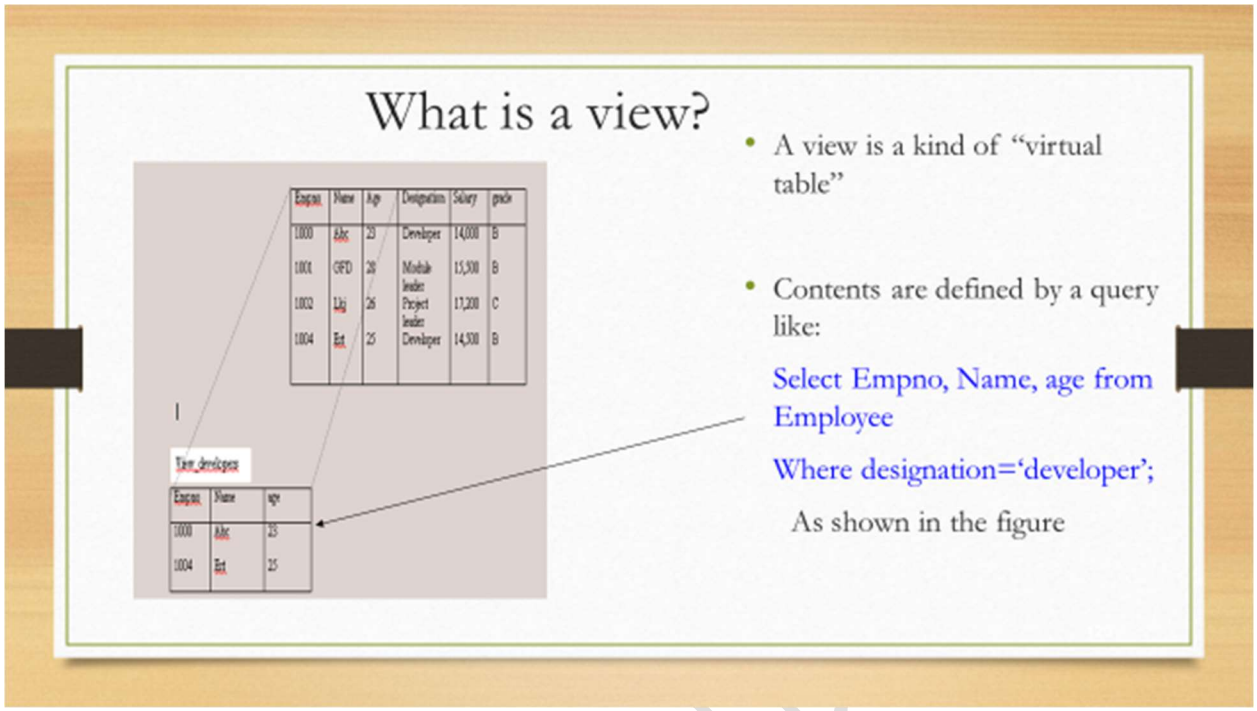
VIEWS in Oracle

- Understanding the Standards of VIEWS in Oracle
- Types of VIEWS
- Practical approach of SIMPLE VIEWS and COMPLEX VIEWS
- Using VIEWS for DML Operations
- About Materialized Views



What is a View?

- A view is a customized representation of data from one or more tables.
- The tables that the view is referencing are known as base tables.
- A view can be considered as a stored query or a virtual table.
- Only the query is stored in the Oracle data dictionary; the actual data is not copied any-where. This means that creating views does not take any storage space, other than the space in dictionary.
- View is a database object which contains logical representation of data.
- By using views we can hide some column data from the user.
- If we perform any type of operations on view the same operations automatically affected to the corresponding base table and vice – versa.
- We can create a view based on another view upto maximum 32 views.
- If we drop the base table the corresponding views will not be dropped and those become invalid.
- On invalid views we can't perform any type of operations.
- Views will become invalid in three cases.
    - When we drop the base table.
    - When we change the table name.
    - When we modify the structure (Rename column) of the base table.
- We can make an invalid view as a valid view.
- We can create a view based on one table and also based on the more than one table.



How to create a view?

```
CREATE VIEW view-name (column-name1, column-name2, ----------) AS query
```

Example:

```
CREATE VIEW ViewCustomerDetails AS
                    SELECT * FROM Customer_Details;
```

Types of Views:

- Views are divided into two types.
  1. Simple Views:
     - Creating the view based only one table is known as simple view.
  2. Complex View:
     - Creating the view based on more than one table is called Complex view.

Simple Views

- These views are created by using select statement doesn't contain Join conditions/arithmetic expressions and Group by or Having Clause.

Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

```
CREATE VIEW emp_view AS
SELECT DeptID, AVG(Salary)
FROM Employee
GROUP BY DeptID;
```

Create View of grouped records on Employee table

emp_view

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

Example 1:

Create or Replace view my_view as

select empno,ename,sal,deptno from emp;

Select * from my_view;

inserting data into my_view: Insert into my_view values(1,'kiran',5000,10);

Example2:

Create or replace view dept10 as

select empno,ename,sal,deptno from emp

where deptno=10;

Insert into dept10 values(2,'kishore',8000,20);

Insert into dept10 values(3,'kishore',8000,10);

View Constraint -- With check option:

- if inserted data satisfies view condition then the record is inserted into views and base table.
- if not satisifies view condition, it shows constraint voilated error messsage.

example:

```
Create or replace view dept10 as
        select * from emp where deptno=10
                with check option;
insert into dept10(empno,ename,sal,deptno) values(5,'eee',5000,20);  X
insert into dept10(empno,ename,sal,deptno) values(5,'eee',5000,10);
```

View Constraint -- Read only Views:

- these views are only for reading purpose.
- DML operations are not supported.
- only select & desc commands valid.

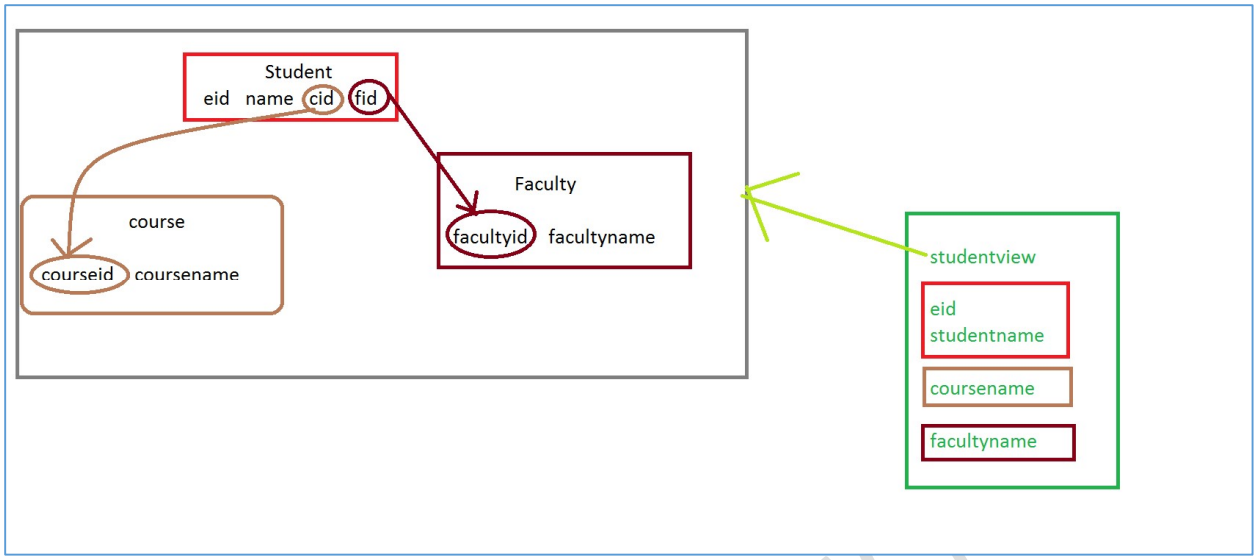
example:

```
Create or replace view read_view as
        select * from emp
        with read only;
```

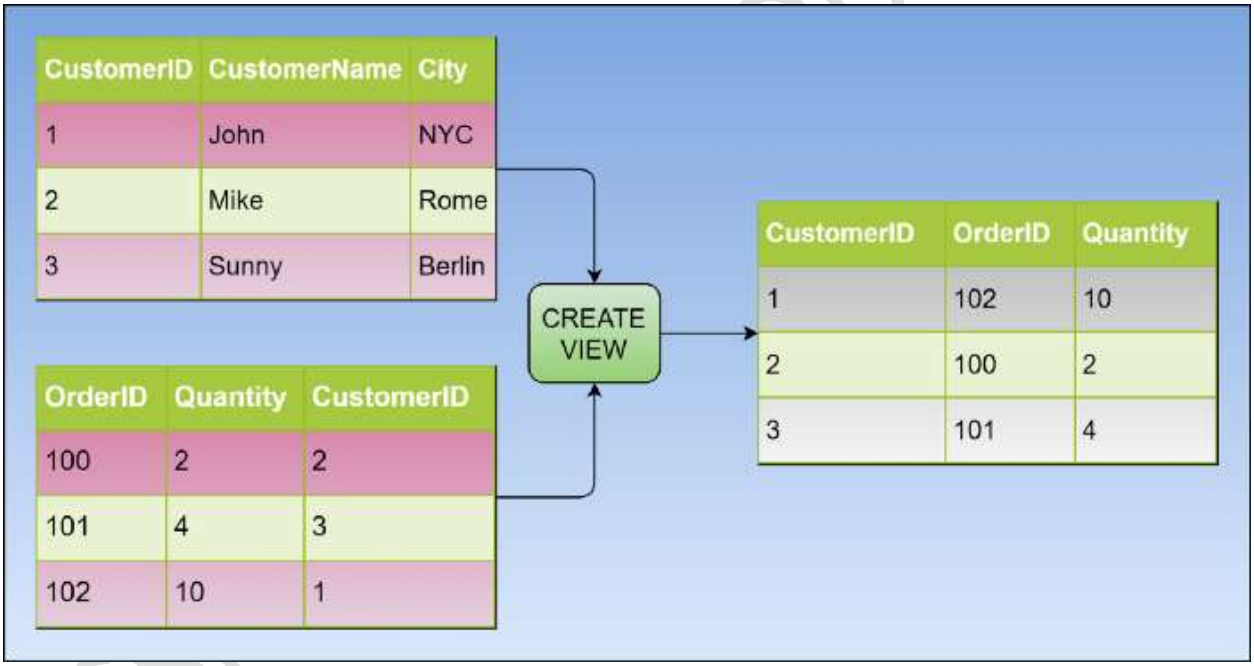
Complex View Example

- Creating the views based on more than one table is known as complex views.
- These views are created by using select statement contains join conditions/arithmetic expressions/group by clause.
- In Complex views, DML Operations are partial supported.
- Only select & Desc Commands are Valid.

Example – 1



Example – 2



- Syntax:

CREATE VIEW<View Name> AS
SELECT columns list FROM
<tab1>,<tab2>-----,<tab n>

- Example:

```
create view empdet1 as
        select empno,ename,dname,loc from
                emp, dept
                    where emp.deptno=dept.deptno;
```

Example – 2:

```
Create or Replace view Comp_view as
        select e.empno,e.ename,e.sal,e.deptno, d.dname,d.loc
                from Emp e, Dept d
                        where e.deptno=d.deptno;
insert into comp_view values(6,'fff',5000,10,'ACCOUNTING','NEW YORK');
```

Example - 3:

```
Create or replace view  comp_view1 as
        Select Deptno,Max(sal) as Maxsal,Min(sal) as Minsal,
        Sum(sal) as Sumsal from Emp
                group by Deptno;
```

Force views:

- These views are created by without having base table.
- These views are activated, whenever base table is created.

Example :

```
Create or replace Force view f_view as
        select * from emp890;
Select * from f_view;
```

To see the no.of views.

select view_name from user_views;

Dropping views:

Drop view my_view;

Drawbacks of Views:

- Read Only View:

It is specific type of the view. So we can't perform any type of operations of DML operations.

Another Example:

```
CREATE TABLE tblEmployee
(
  Id int Primary Key,
  Name nvarchar(30),
  Salary int,
  Gender nvarchar(10),
  DepartmentId int
)

CREATE TABLE tblDepartment
(
 DeptId int Primary Key,
 DeptName nvarchar(20)
)

Insert into tblDepartment values (1,'IT')
Insert into tblDepartment values (2,'Payroll')
```

```
Insert into tblDepartment values (3,'HR')
Insert into tblDepartment values (4,'Admin')

Insert into tblEmployee values (1,'John', 5000, 'Male', 3)
Insert into tblEmployee values (2,'Mike', 3400, 'Male', 2)
Insert into tblEmployee values (3,'Pam', 6000, 'Female', 1)
Insert into tblEmployee values (4,'Todd', 4800, 'Male', 4)
Insert into tblEmployee values (5,'Sara', 3200, 'Female', 1)
Insert into tblEmployee values (6,'Ben', 4800, 'Male', 3)

Create View vWEmployeesByDepartment as
    Select Id, Name, Salary, Gender, DeptName
        from tblEmployee  join tblDepartment
            on tblEmployee.DepartmentId = tblDepartment.DeptId

SELECT * from vWEmployeesByDepartment
```

Materialized views

- These are special type of views
- The main difference between normal view and materialized view is if we drop the base table then the normal view becomes invalid where as materialized view won't be invalid.
- Materialized view stores data.
- We can't perform DML Operations on Materialized view.

- Example:

```
create materialized view empmview as
    select * from emp;
```

Difference between View and Materialized view

View	Materialized Views
View is nothing but the logical structure of the table which will retrieve data from 1 or more table.	Materialized views are also logical structure but data is physically stored in database.
You need to have Create view privileges to create simple or complex view	You need to have create materialized view 's privileges to create Materialized views
Data access is not as fast as materialized views	Data retrieval is fast as compare to simple view because data is accessed from directly physical location
There are 2 types of views: • Simple View • Complex view	4.There are following types of Materialized views: • Refresh on Auto • Refresh on demand
In Application level views are used to restrict data from database	Materialized Views are used in Data Warehousing.

Scenario of Materialized Views

- Suppose there are 2 tables named Employee and Department.
- The Employee table contains 1 million records and department table contains 20 records.
- We need to fetch the Employees associated with that department.

Step 1:

To Perform above scenario we basically create view:

```
Create View V_Employee as
        Select E.Employee_num,E.Employee_name,D.Department_Name
            from Employee E , Department D where E.Dept_no=D.Dept_no;
```

Step 2:

Fetch the records from the View

Select * from V_Employee;

It will fetch 10 million records with associated department.

But to fetch that records check the time.

Let us consider it will take 2 Mins means 120 secs to fetch records

Step 3 :

Let us Create materialized view which will refresh automatically.

Create or Replace Materialized view MV_Employee as

Select E.Employee_num,E.Employee_name,D.Department_Name

from Employee E , Department D where E.Dept_no=D.Dept_no

Refresh auto on commit select * from Department;

We have created materialized views in sql for that.and lets check performance.

Select* from MV_Employee;

It will fetch 1 million records in 60 secs.

So performance is improved double when you use materialized view.

Chapter – 23

Built in Functions

- Arithmetic Functions, Character Functions, Date Functions, Conversion Functions
- Aggregate Functions, OLAP Functions & General Functions

- Absolute value is the measure of the magnitude of value.
- Absolute value is always a positive number.

Syntax: abs (*value*)

Ex:

```
SQL> select abs(5), abs(-5), abs(0), abs(null) from dual;

   ABS(5)    ABS(-5)     ABS(0) ABS(NULL)
---------- ---------- ---------- ------------
         5          5          0
```

b) SIGN

Sign gives the sign of a value.

Syntax: sign (*value*)

Ex:

```
SQL> select sign(5), sign(-5), sign(0), sign(null) from dual;

  SIGN(5)    SIGN(-5)    SIGN(0) SIGN(NULL)
---------- ---------- ---------- -------------
```

1 -1 0

c) SQRT

This will give the square root of the given value.

Syntax: sqrt (*value*) -- here value must be positive.

Ex:

SQL> select sqrt(4), sqrt(0), sqrt(null), sqrt(1) from dual;

SQRT(4)	SQRT(0)	SQRT(NULL)	SQRT(1)
2	0		1

d) MOD

This will give the remainder.

Syntax: mod (*value, divisor*)

Ex:

SQL> select mod(7,4), mod(1,5), mod(null,null), mod(0,0), mod(-7,4) from dual;

MOD(7,4)	MOD(1,5)	MOD(NULL,NULL)	MOD(0,0)	MOD(-7,4)
3	1		0	-3

e) NVL

This will substitutes the specified value in the place of null values.

Syntax: nvl (*null_col, replacement_value*)

Ex:

SQL> select * from student; -- here for 3rd row marks value is null

NO	NAME	MARKS
1	a	100
2	b	200
3	c	

SQL> select no, name, nvl(marks,300) from student;

NO	NAME	NVL(MARKS,300)
1	a	100
2	b	200
3	c	300

SQL> select nvl(1,2), nvl(2,3), nvl(4,3), nvl(5,4) from dual;

NVL(1,2)	NVL(2,3)	NVL(4,3)	NVL(5,4)
1	2	4	5

SQL> select nvl(0,0), nvl(1,1), nvl(null,null), nvl(4,4) from dual;

NVL(0,0)	NVL(1,1)	NVL(null,null)	NVL(4,4)
0	1		4

f) POWER

Power is the ability to raise a value to a given exponent.

Syntax: power (*value, exponent*)

Ex:

SQL> select power(2,5), power(0,0), power(1,1), power(null,null), power(2,-5) from dual;

POWER(2,5)	POWER(0,0)	POWER(1,1)	POWER(NULL,NULL)	POWER(2,-5)
32	1	1		.03125

g) EXP

This will raise e value to the give power.

Syntax: exp (*value*)

Ex:

```
SQL> select exp(1), exp(2), exp(0), exp(null), exp(-2) from dual;

    EXP(1)       EXP(2)       EXP(0)  EXP(NULL)    EXP(-2)
--------     ---------     --------  ------------  ----------
2.71828183  7.3890561          1                    .135335283
```

h) LN

This is based on natural or base e logarithm.

Syntax: ln (*value*) -- here value must be greater than zero which is positive only.

Ex:

```
SQL> select ln(1), ln(2), ln(null) from dual;

    LN(1)      LN(2)      LN(NULL)
-------      -------      ------------
       0      .693147181
```

Ln and Exp are reciprocal to each other.

EXP (3) = 20.0855369

LN (20.0855369) = 3

i) LOG

This is based on 10 based logarithm.

Syntax: log (10, *value*) -- here value must be greater than zero which is positive only.

Ex:

```
SQL> select log(10,100), log(10,2), log(10,1), log(10,null) from dual;

LOG(10,100)  LOG(10,2)  LOG(10,1) LOG(10,NULL)
-----------  ----------  ----------  -----------------
          2  .301029996           0
```

LN (value) = LOG (EXP(1), value)

```
SQL> select  ln(3), log(exp(1),3) from dual;

      LN(3)  LOG(EXP(1),3)
-----------  -----------------
```

1.09861229 1.09861229

j) CEIL

This will produce a whole number that is greater than or equal to the specified value.

Syntax: ceil (*value*)

Ex:

SQL> select ceil(5), ceil(5.1), ceil(-5), ceil(-5.1), ceil(0), ceil(null) from dual;

CEIL(5)	CEIL(5.1)	CEIL(-5)	CEIL(-5.1)	CEIL(0)	CEIL(NULL)
5	6	-5	-5	0	

k) FLOOR

This will produce a whole number that is less than or equal to the specified value.

Syntax: floor (*value*)

Ex:

SQL> select floor(5), floor(5.1), floor(-5), floor(-5.1), floor(0), floor(null) from dual;

FLOOR(5)	FLOOR(5.1)	FLOOR(-5)	FLOOR(-5.1)	FLOOR(0)	FLOOR(NULL)
5	5	-5	-6	0	

l) ROUND

This will rounds numbers to a given number of digits of precision.

Syntax: round (*value, precision*)

Ex:

SQL> select round(123.2345), round(123.2345,2), round(123.2354,2) from dual;

ROUND(123.2345)	ROUND(123.2345,0)	ROUND(123.2345,2)	ROUND(123.2354,2)

---

123 123 123.23 123.24

SQL> select round(123.2345,-1), round(123.2345,-2), round(123.2345,-3),
round(123.2345,-4) from dual;

ROUND(123.2345,-1) ROUND(123.2345,-2) ROUND(123.2345,-3) ROUND(123.2345,-4)

---

120 100 0 0

SQL> select round(123,0), round(123,1), round(123,2) from dual;

ROUND(123,0) ROUND(123,1) ROUND(123,2)

---

123 123 123

SQL> select round(-123,0), round(-123,1), round(-123,2) from dual;

ROUND(-123,0) ROUND(-123,1) ROUND(-123,2)

-123 -123 -123

SQL> select round(123,-1), round(123,-2), round(123,-3), round(-123,-1), round(-123,-

2), round(-123,-3) from dual;

ROUND(123,-1) ROUND(123,-2) ROUND(123,-3) ROUND(-123,-1) ROUND(-123,-2)

ROUND(-123,-3)

---

120 100 0 -120 -100 0

SQL> select round(null,null), round(0,0), round(1,1), round(-1,-1), round(-2,-2) from

dual;

ROUND(NULL,NULL) ROUND(0,0) ROUND(1,1) ROUND(-1,-1) ROUND(-2,-2)

---

0 1 0 0

m) TRUNC

This will truncates or chops off digits of precision from a number.

Syntax: trunc (*value, precision*)

Ex:

SQL> select trunc(123.2345), trunc(123.2345,2), trunc(123.2354,2) from dual;

TRUNC(123.2345)	TRUNC(123.2345,2)	TRUNC(123.2354,2)
123	123.23	123.23

SQL> select trunc(123.2345,-1), trunc(123.2345,-2), trunc(123.2345,-3), trunc(123.2345,-4) from dual;

TRUNC(123.2345,-1)	TRUNC(123.2345,-2)	TRUNC(123.2345,-3)	TRUNC(123.2345,-4)

120 100 0

0

SQL> select trunc(123,0), trunc(123,1), trunc(123,2) from dual;

TRUNC(123,0) TRUNC(123,1) TRUNC(123,2)

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

123 123 123

SQL> select trunc(-123,0), trunc(-123,1), trunc(-123,2) from dual;

TRUNC(-123,0) TRUNC(-123,1) TRUNC(-123,2)

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

-123 -123 -123

SQL> select trunc(123,-1), trunc(123,-2), trunc(123,-3), trunc(-123,-1), trunc(-123,2),

trunc(-123,-3) from dual;

TRUNC(123,-1) TRUNC(123,-2) TRUNC(123,-3) TRUNC(-123,-1) TRUNC(-123,2) TRUNC(-

123,-3)

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

120 100 0 -120 -123 0

SQL> select trunc(null,null), trunc(0,0), trunc(1,1), trunc(-1,-1), trunc(-2,-2) from dual;

TRUNC(NULL,NULL) TRUNC(0,0) TRUNC(1,1) TRUNC(-1,-1) TRUNC(-2,-2)

---

0 1 0 0

n) BITAND

This will perform bitwise and operation.

Syntax: bitand (*value1, value2*)

Ex:

SQL> select bitand(2,3), bitand(0,0), bitand(1,1), bitand(null,null), bitand(-2,-3) from

dual;

BITAND(2,3) BITAND(0,0) BITAND(1,1) BITAND(NULL,NULL) BITAND(-2,-3)

---

2 0 1

-4

o) GREATEST

This will give the greatest number.

Syntax: greatest (*value1, value2, value3 … valuen*)

Ex:

SQL> select greatest(1, 2, 3), greatest(-1, -2, -3) from dual;

GREATEST(1,2,3)	GREATEST(-1,-2,-3)
3	-1

- If all the values are zeros then it will display zero.
- If all the parameters are nulls then it will display nothing.
- If any of the parameters is null it will display nothing.

p) LEAST

This will give the least number.

Syntax: least (*value1, value2, value3 … valuen*)

Ex:

SQL> select least(1, 2, 3), least(-1, -2, -3) from dual;

LEAST(1,2,3)	LEAST(-1,-2,-3)
1	-3

- If all the values are zeros then it will display zero.
- If all the parameters are nulls then it will display nothing.
- If any of the parameters is null it will display nothing.

q) COALESCE

This will return first non-null value.

Syntax: coalesce (*value1, value2, value3 … valuen*)

Ex:

SQL> select coalesce(1,2,3), coalesce(null,2,null,5) from dual;

COALESCE(1,2,3) COALESCE(NULL,2,NULL,5)

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

1 2

STRING FUNCTIONS

- Initcap
- Upper
- Lower
- Length
- Rpad
- Lpad
- Ltrim
- Rtrim
- Trim
- Translate
- Replace
- Soundex
- Concat (' || ' Concatenation operator)
- Ascii
- Chr
- Substr
- Instr
- Decode
- Greatest
- Least
- Coalesce

a) INITCAP

This will capitalize the initial letter of the word.

Syntax: initcap (*string*)

Ex:

SQL> select initcap('computer') from dual;

INITCAP

Computer

b) UPPER

This will convert the string into uppercase.

Syntax: upper (*string*)

Ex:

SQL> select upper('computer') from dual;

UPPER

COMPUTER

c) LOWER

This will convert the string into lowercase.

Syntax: lower (*string*)

Ex:

SQL> select lower('COMPUTER') from dual;

LOWER
---
computer

d) LENGTH

This will give length of the string.

Syntax: length (*string*)

Ex:

SQL> select length('computer') from dual;

```
LENGTH
----------
       8
```

e) RPAD

This will allows you to pad the right side of a column with any set of characters.

Syntax: rpad (*string, length [, padding_char]*)

Ex:

SQL> select rpad('computer',15,'*'), rpad('computer',15,'*#') from dual;

```
RPAD('COMPUTER'  RPAD('COMPUTER'
---------------  ---------------
computer*******  computer*#*#*#*
```

-- Default padding character was blank space.

f) LPAD

This will allows you to pad the left side of a column with any set of characters.

Syntax: lpad (*string, length [, padding_char]*)

Ex:

```
SQL> select lpad('computer',15,'*'), lpad('computer',15,'*#') from dual;

LPAD('COMPUTER'  LPAD('COMPUTER'
---------------  ---------------
*******computer  *#*#*#*computer
```

-- Default padding character was blank space.

g) LTRIM

This will trim off unwanted characters from the left end of string.

Syntax: ltrim (*string [,unwanted_chars]*)

Ex:

SQL> select ltrim('computer','co'), ltrim('computer','com') from dual;

```
LTRIM(  LTRIM
--------  ---------
mputer    puter
```

SQL> select ltrim('computer','puter'), ltrim('computer','omputer') from dual;

```
LTRIM('C  LTRIM('C
----------  ----------
computer    computer
```

-- If you haven't specify any unwanted characters it will display entire string.

h) RTRIM

This will trim off unwanted characters from the right end of string.

Syntax: rtrim (*string [, unwanted_chars]*)

Ex:

SQL> select rtrim('computer','er'), rtrim('computer','ter') from dual;

```
RTRIM(  RTRIM
--------  ---------
comput    compu
```

SQL> select rtrim('computer','comput'), rtrim('computer','compute') from dual;

```
RTRIM('C  RTRIM('C
----------  ----------
computer    computer
```

-- If you haven't specify any unwanted characters it will display entire string.

i) TRIM

This will trim off unwanted characters from the both sides of string.

Syntax: trim (*unwanted_chars* from *string*)

j) TRANSLATE

This will replace the set of characters, character by character.

Syntax: translate (*string, old_chars, new_chars*)

Ex:

SQL> select translate('india','in','xy') from dual;

```
TRANS
--------
xydxa
```

k) REPLACE

This will replace the set of characters, string by string.

Syntax: replace (*string, old_chars [, new_chars]*)

Ex:

SQL> select replace('india','in','xy'), replace('india','in') from dual;

```
REPLACE    REPLACE
-----------  -----------
Xydia        dia
```

l) SOUNDEX

This will be used to find words that sound like other words, exclusively used in where clause.

Syntax: soundex (*string*)

Ex:

SQL> select * from emp where soundex(ename) = soundex('SMIT');

```
EMPNO ENAME      JOB              MGR HIREDATE         SAL
DEPTNO
--------  --------      -----           -----  ------------      ---------- ----------
 7369    SMITH      CLERK         7902   17-DEC-80        500         20
```

m) CONCAT

This will be used to combine two strings only.

Syntax: concat (*string1, string2*)

Ex:

```
SQL> select concat('computer',' operator') from dual;

CONCAT('COMPUTER'
-------------------------
computer operator
```

If you want to combine more than two strings you have to use concatenation operator (||).

```
SQL> select 'how' || ' are' || ' you' from dual;

'HOW'||'ARE
--------------
how are you
```

n) ASCII

This will return the decimal representation in the database character set of the first

character of the string.

Syntax: ascii (*string*)

Ex:

SQL> select ascii('a'), ascii('apple') from dual;

ASCII('A')	ASCII('APPLE')
97	97

o) CHR

This will return the character having the binary equivalent to the string in either the

database character set or the national character set.

Syntax: chr (*number*)

Ex:

SQL> select chr(97) from dual;

```
CHR
-----
  a
```

p) SUBSTR

This will be used to extract substrings.

Syntax: substr (*string, start_chr_count [, no_of_chars]*)

Ex:

SQL> select substr('computer',2), substr('computer',2,5), substr('computer',3,7) from dual;

```
SUBSTR(  SUBST  SUBSTR
---------- -------  --------
omputer  omput   mputer
```

- If *no_of_chars* parameter is negative then it will display nothing.
- If both parameters except *string* are null or zeros then it will display nothing.

- If *no_of_chars* parameter is greater than the length of the string then it ignores and calculates based on the orginal string length.
- If *start_chr_count* is negative then it will extract the substring from right end.

1	2	3	4	5	6	7	8
C	O	M	P	U	T	E	R
-8	-7	-6	-5	-4	-3	-2	-1

q) INSTR

This will allows you for searching through a string for set of characters.

Syntax: instr (*string, search_str [, start_chr_count [, occurrence]]*)

Ex:

SQL> select instr('information','o',4,1), instr('information','o',4,2) from dual;

INSTR('INFORMATION','O',4,1)	INSTR('INFORMATION','O',4,2)
4	10

- If you are not specifying *start_chr_count* and *occurrence* then it will start search from the beginning and finds first occurrence only.
- If both parameters *start_chr_count* and *occurrence* are null, it will display nothing.

r) DECODE

Decode will act as value by value substitution.

For every value of field, it will checks for a match in a series of if/then tests.

Syntax: decode (*value, if1, then1, if2, then2, ... else*);

Ex:

SQL> select sal, decode(sal,500,'Low',5000,'High','Medium') from emp;

SAL	DECODE
500	Low
2500	Medium
2000	Medium
3500	Medium
3000	Medium

5000	High
4000	Medium
5000	High
1800	Medium
1200	Medium
2000	Medium
2700	Medium
2200	Medium
3200	Medium

SQL> select decode(1,1,3), decode(1,2,3,4,4,6) from dual;

DECODE(1,1,3)	DECODE(1,2,3,4,4,6)
3	6

- If the number of parameters are odd and different then decode will display nothing.
- If the number of parameters are even and different then decode will display last value.
- If all the parameters are null then decode will display nothing.
- If all the parameters are zeros then decode will display zero.

s) GREATEST

This will give the greatest string.

Syntax: greatest (*strng1, string2, string3 … stringn*)

Ex:

SQL> select greatest('a', 'b', 'c'), greatest('satish','srinu','saketh') from dual;

GREAT	GREAT
c	srinu

- If all the parameters are nulls then it will display nothing.
- If any of the parameters is null it will display nothing.

t) LEAST

This will give the least string.

Syntax: greatest (*strng1, string2, string3 … stringn*)

Ex:

SQL> select least('a', 'b', 'c'), least('satish','srinu','saketh') from dual;

LEAST LEAST

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

a saketh

- If all the parameters are nulls then it will display nothing.
- If any of the parameters is null it will display nothing.

u) COALESCE

This will gives the first non-null string.

Syntax: coalesce (*strng1, string2, string3 … stringn*)

Ex:

SQL> select coalesce('a','b','c'), coalesce(null,'a',null,'b') from dual;

COALESCE COALESCE

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

a a

DATE FUNCTIONS

- Sysdate
- Current_date
- Current_timestamp
- Systimestamp
- Localtimestamp
- Dbtimezone
- Sessiontimezone
- To_char
- To_date
- Add_months
- Months_between
- Next_day
- Last_day
- Extract
- Greatest
- Least
- Round
- Trunc
- New_time
- Coalesce

Oracle default date format is DD-MON-YY.

We can change the default format to our desired format by using the following command.

SQL> alter session set nls_date_format = 'DD-MONTH-YYYY';

But this will expire once the session was closed.

a) SYSDATE

This will give the current date and time.

Ex:

SQL> select sysdate from dual;

SYSDATE

24-DEC-06

b) CURRENT_DATE

This will returns the current date in the session's timezone.

Ex:

SQL> select current_date from dual;

CURRENT_DATE

24-DEC-06

c) CURRENT_TIMESTAMP

This will returns the current timestamp with the active time zone information.

Ex:

SQL> select current_timestamp from dual;

CURRENT_TIMESTAMP

24-DEC-06 03.42.41.383369 AM +05:30

d) SYSTIMESTAMP

This will returns the system date, including fractional seconds and time zone of the

database.

Ex:

SQL> select systimestamp from dual;

SYSTIMESTAMP

24-DEC-06 03.49.31.830099 AM +05:30

e) LOCALTIMESTAMP

This will returns local timestamp in the active time zone information, with no time zone

information shown.

Ex:

```
SQL> select localtimestamp from dual;

LOCALTIMESTAMP
---------------------------------------------------------------------------
24-DEC-06 03.44.18.502874 AM
```

f) DBTIMEZONE

This will returns the current database time zone in UTC format. (Coordinated Universal Time)

Ex:

```
SQL> select dbtimezone from dual;
```

DBTIMEZONE

-07:00

g) SESSIONTIMEZONE

This will returns the value of the current session's time zone.

Ex:

SQL> select sessiontimezone from dual;

SESSIONTIMEZONE

+05:30

h) TO_CHAR

This will be used to extract various date formats.

The available date formats as follows.

Syntax: to_char (*date*, *format*)

DATE FORMATS

D	--	No of days in week
DD	--	No of days in month
DDD	--	No of days in year
MM	--	No of month
MON	--	Three letter abbreviation of month
MONTH	--	Fully spelled out month
RM	--	Roman numeral month
DY	--	Three letter abbreviated day
DAY	--	Fully spelled out day
Y	--	Last one digit of the year
YY	--	Last two digits of the year
YYY	--	Last three digits of the year
YYYY	--	Full four digit year
SYYYY	--	Signed year
I	--	One digit year from ISO standard
IY	--	Two digit year from ISO standard
IYY	--	Three digit year from ISO standard
IYYY	--	Four digit year from ISO standard
Y, YYY	--	Year with comma
YEAR	--	Fully spelled out year

CC -- Century

Q -- No of quarters

W -- No of weeks in month

WW -- No of weeks in year

IW -- No of weeks in year from ISO standard

HH -- Hours

MI -- Minutes

SS -- Seconds

FF -- Fractional seconds

AM or PM -- Displays AM or PM depending upon time of day

A.M or P.M -- Displays A.M or P.M depending upon time of day

AD or BC -- Displays AD or BC depending upon the date

A.D or B.C -- Displays AD or BC depending upon the date

FM -- Prefix to month or day, suppresses padding of month or day

TH -- Suffix to a number

SP -- suffix to a number to be spelled out

SPTH -- Suffix combination of TH and SP to be both spelled out

THSP -- same as SPTH

Ex:

```
SQL> select to_char(sysdate,'dd month yyyy hh:mi:ss am dy') from dual;
```

TO_CHAR(SYSDATE,'DD MONTH YYYYHH:MI

---

24 december 2006 02:03:23 pm sun

SQL> select to_char(sysdate,'dd month year') from dual;

TO_CHAR(SYSDATE,'DDMONTHYEAR')

---

24 december two thousand six

SQL> select to_char(sysdate,'dd fmmonth year') from dual;

TO_CHAR(SYSDATE,'DD FMMONTH YEAR')

---

24 december two thousand six

SQL> select to_char(sysdate,'ddth DDTH') from dual;

TO_CHAR(S

24th 24TH

SQL> select to_char(sysdate,'ddspth DDSPTH') from dual;

TO_CHAR(SYSDATE,'DDSPTHDDSPTH

---

twenty-fourth TWENTY-FOURTH

SQL> select to_char(sysdate,'ddsp Ddsp DDSP ') from dual;

TO_CHAR(SYSDATE,'DDSPDDSPDDSP')

---

twenty-four Twenty-Four TWENTY-FOUR

i) TO_DATE

This will be used to convert the string into date format.

Syntax: to_date (*date*)

Ex:

SQL> select to_char(to_date('24/dec/2006','dd/mon/yyyy'), 'dd * month * day') from

dual;

```
TO_CHAR(TO_DATE('24/DEC/20
--------------------------
24 * december  * Sunday
```

-- If you are not using to_char oracle will display output in default date format.

j) ADD_MONTHS

This will add the specified months to the given date.

Syntax: add_months (*date, no_of_months*)

Ex:

SQL> select add_months(to_date('11-jan-1990','dd-mon-yyyy'), 5) from dual;

ADD_MONTHS

11-JUN-90

SQL> select add_months(to_date('11-jan-1990','dd-mon-yyyy'), -5) from dual;

ADD_MONTH

11-AUG-89

- If *no_of_months* is zero then it will display the same date.
- If *no_of_months* is null then it will display nothing.

k) MONTHS_BETWEEN

This will give difference of months between two dates.

Syntax: months_between (*date1, date2*)

Ex:

SQL> select months_between(to_date('11-aug-1990','dd-mon-yyyy'), to_date('11-jan-

1990','dd-mon-yyyy')) from dual;

MONTHS_BETWEEN(TO_DATE('11-AUG-1990','DD-MON-YYYY'),TO_DATE('11-JAN-1990','DD-MON-YYYY'))

7

SQL> select months_between(to_date('11-jan-1990','dd-mon-yyyy'), to_date('11-aug-1990','dd-mon-yyyy')) from dual;

MONTHS_BETWEEN(TO_DATE('11-JAN-1990','DD-MON-YYYY'),TO_DATE('11-AUG-1990','DD-MON-YYYY'))

-7

l) NEXT_DAY

This will produce next day of the given day from the specified date.

Syntax: next_day (*date, day*)

Ex:

SQL> select next_day(to_date('24-dec-2006','dd-mon-yyyy'),'sun') from dual;

NEXT_DAY(

31-DEC-06

-- If the day parameter is null then it will display nothing.

m) LAST_DAY

This will produce last day of the given date.

Syntax: last_day (*date*)

Ex:

SQL> select last_day(to_date('24-dec-2006','dd-mon-yyyy')) from dual;

LAST_DAY(

31-DEC-06

n) EXTRACT

This is used to extract a portion of the date value.

Syntax: extract ((year | month | day | hour | minute | second), *date*)

Ex:

SQL> select extract(year from sysdate) from dual;

EXTRACT(YEARFROMSYSDATE)

2006

-- You can extract only one value at a time.

o) GREATEST

This will give the greatest date.

Syntax: greatest (*date1, date2, date3 … daten*)

Ex:

SQL> select greatest(to_date('11-jan-90','dd-mon-yy'),to_date('11-mar-90','dd-mon-

yy'),to_date('11-apr-90','dd-mon-yy')) from dual;

GREATEST(

11-APR-90

p) LEAST

This will give the least date.

Syntax: least (*date1, date2, date3 … daten*)

Ex:

SQL> select least(to_date('11-jan-90','dd-mon-yy'),to_date('11-mar-90','dd-mon-

yy'),to_date('11-apr-90','dd-mon-yy')) from dual;

LEAST(

11-JAN-90

q) ROUND

Round will rounds the date to which it was equal to or greater than the given date.

Syntax: round (*date, (*day | month | year*)*)

If the second parameter was *year* then round will checks the month of the given date in the

following ranges.

JAN -- JUN

JUL -- DEC

If the month falls between JAN and JUN then it returns the first day of the current year.

If the month falls between JUL and DEC then it returns the first day of the next year.

If the second parameter was *month* then round will checks the day of the given date in the

following ranges.

1 -- 15

16 -- 31

If the day falls between 1 and 15 then it returns the first day of the current month.

If the day falls between 16 and 31 then it returns the first day of the next month.

If the second parameter was *day* then round will checks the week day of the given date in

the following ranges.

SUN -- WED

THU -- SUN

If the week day falls between SUN and WED then it returns the previous sunday.

If the weekday falls between THU and SUN then it returns the next sunday.

- If the second parameter was null then it returns nothing.
- If the you are not specifying the second parameter then round will resets the time to the
  begining of the current day in case of user specified date.

- If the you are not specifying the second parameter then round will resets the time to the
  begining of the next day in case of sysdate.

Ex:

```
SQL> select round(to_date('24-dec-04','dd-mon-yy'),'year'),
round(to_date('11-mar-
        06','dd-mon-yy'),'year') from dual;

ROUND(TO_ ROUND(TO_
--------- ---------
01-JAN-05 01-JAN-06

SQL> select round(to_date('11-jan-04','dd-mon-yy'),'month'),
round(to_date('18-jan-
        04','dd-mon-yy'),'month') from dual;

ROUND(TO_ ROUND(TO_
--------- ---------
01-JAN-04 01-FEB-04

SQL> select round(to_date('26-dec-06','dd-mon-yy'),'day'),
round(to_date('29-dec-
        06','dd-mon-yy'),'day') from dual;
```

ROUND(TO_ ROUND(TO_

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

24-DEC-06 31-DEC-06

SQL> select to_char(round(to_date('24-dec-06','dd-mon-yy')), 'dd mon yyyy hh:mi:ss am')

from dual;

TO_CHAR(ROUND(TO_DATE('

---

24 dec 2006 12:00:00 am

r) TRUNC

Trunc will chops off the date to which it was equal to or less than the given date.

Syntax: trunc (*date, (*day | month | year*)*)

- If the second parameter was *year* then it always returns the first day of the current year.
- If the second parameter was *month* then it always returns the first day of the current month.

- If the second parameter was *day* then it always returns the previous sunday.
- If the second parameter was null then it returns nothing.
- If the you are not specifying the second parameter then trunk will resets the time to the
  begining of the current day.

Ex:

```
SQL> select trunc(to_date('24-dec-04','dd-mon-yy'),'year'),
trunc(to_date('11-mar-
06','dd-mon-yy'),'year') from dual;

TRUNC(TO_ TRUNC(TO_
--------- ---------
01-JAN-04 01-JAN-06

SQL> select trunc(to_date('11-jan-04','dd-mon-yy'),'month'),
trunc(to_date('18-jan-
04','dd-mon-yy'),'month') from dual;

TRUNC(TO_ TRUNC(TO_
--------- ---------
01-JAN-04 01-JAN-04
```

```
SQL> select trunc(to_date('26-dec-06','dd-mon-yy'),'day'),
trunc(to_date('29-dec-06','dd-
mon-yy'),'day') from dual;

TRUNC(TO_ TRUNC(TO_
---------- ----------
24-DEC-06 24-DEC-06
```

```
SQL> select to_char(trunc(to_date('24-dec-06','dd-mon-yy')), 'dd mon
yyyy hh:mi:ss am')
from dual;

TO_CHAR(TRUNC(TO_DATE('
------------------------------
24 dec 2006 12:00:00 am
```

s) NEW_TIME

This will give the desired timezone's date and time.

Syntax: new_time (*date, current_timezone, desired_timezone*)

Available timezones are as follows.

TIMEZONES

AST/ADT	--	Atlantic standard/day light time
BST/BDT	--	Bering standard/day light time
CST/CDT	--	Central standard/day light time
EST/EDT	--	Eastern standard/day light time
GMT	--	Greenwich mean time
HST/HDT	--	Alaska-Hawaii standard/day light time
MST/MDT	--	Mountain standard/day light time
NST	--	Newfoundland standard time
PST/PDT	--	Pacific standard/day light time
YST/YDT	--	Yukon standard/day light time

Ex:

SQL> select to_char(new_time(sysdate,'gmt','yst'),'dd mon yyyy hh:mi:ss am') from dual;

TO_CHAR(NEW_TIME(SYSDAT

---

24 dec 2006 02:51:20 pm

SQL> select to_char(new_time(sysdate,'gmt','est'),'dd mon yyyy hh:mi:ss am') from dual;

TO_CHAR(NEW_TIME(SYSDAT

---

24 dec 2006 06:51:26 pm

t) COALESCE

This will give the first non-null date.

Syntax: coalesce (*date1, date2, date3 … daten*)

Ex:

SQL> select coalesce('12-jan-90','13-jan-99'), coalesce(null,'12-jan-90','23-mar-98',null)

from dual;

COALESCE(COALESCE(

--- ---

12-jan-90 12-jan-90

MISCELLANEOUS FUNCTIONS

- Uid
- User
- Vsize
- Rank
- Dense_rank

a) UID

This will returns the integer value corresponding to the user currently logged in.

Ex:

```
SQL> select uid from dual;

       UID
----------
       319
```

b) USER

This will returns the login's user name.

Ex:

```
SQL> select user from dual;
```

USER

SAKETH

c) VSIZE

This will returns the number of bytes in the expression.

Ex:

SQL> select vsize(123), vsize('computer'), vsize('12-jan-90') from dual;

VSIZE(123)	VSIZE('COMPUTER')	VSIZE('12-JAN-90')
3	8	9

d) RANK

This will give the non-sequential ranking.

Ex:

SQL> select rownum,sal from (select sal from emp order by sal desc);

ROWNUM	SAL
1	5000
2	3000
3	3000
4	2975
5	2850
6	2450
7	1600
8	1500
9	1300
10	1250
11	1250
12	1100
13	1000
14	950
15	800

SQL> select rank(2975) within group(order by sal desc) from emp;

RANK(2975)WITHINGROUP(ORDERBYSALDESC)

4

d) DENSE_RANK

This will give the sequential ranking.

Ex:

SQL> select dense_rank(2975) within group(order by sal desc) from emp;

DENSE_RANK(2975)WITHINGROUP(ORDERBYSALDESC)

3

CONVERSION FUNCTIONS

- Bin_to_num
- Chartorowid
- Rowidtochar
- To_number
- To_char
- To_date

a) BIN_TO_NUM

This will convert the binary value to its numerical equivalent.

Syntax: bin_to_num(*binary_bits*)

Ex:

SQL> select bin_to_num(1,1,0) from dual;

BIN_TO_NUM(1,1,0)

6

- If all the bits are zero then it produces zero.
- If all the bits are null then it produces an error.

b) CHARTOROWID

This will convert a character string to act like an internal oracle row identifier or rowid.

c) ROWIDTOCHAR

This will convert an internal oracle row identifier or rowid to character string.

d) TO_NUMBER

This will convert a char or varchar to number.

e) TO_CHAR

This will convert a number or date to character string.

f) TO_DATE

This will convert a number, char or varchar to a date.

GROUP FUNCTIONS

- Sum
- Avg
- Max
- Min
- Count

Group functions will be applied on all the rows but produces single output.

a) SUM

This will give the sum of the values of the specified column.

Syntax: sum (*column*)

Ex:

```
SQL> select sum(sal) from emp;

   SUM(SAL)
----------
      38600
```

b) AVG

This will give the average of the values of the specified column.

Syntax: avg (*column*)

Ex:

```
SQL> select avg(sal) from emp;

   AVG(SAL)
---------------
```

2757.14286

c) MAX

This will give the maximum of the values of the specified column.

Syntax: max (*column*)

Ex:

SQL> select max(sal) from emp;

MAX(SAL)

5000

d) MIN

This will give the minimum of the values of the specified column.

Syntax: min (*column*)

Ex:

SQL> select min(sal) from emp;

MIN(SAL)

500

e) COUNT

This will give the count of the values of the specified column.

Syntax: count (*column*)

Ex:

SQL> select count(sal),count(*) from emp;

COUNT(SAL) COUNT(*)

---

14 14