

UNIT 2 : Relational Model And SQL

The relational Model – The catalog - Relational Algebra – Domain Relational Calculus – Tuple Relational Calculus - Fundamental operations – Additional Operations-

SQL-SQL fundamentals, (DRL)Select Command, Logical Operator, Range Searching, Pattern Matching, Oracle Function, Grouping data from Tables in SQL,

Joins-Joining Multiple Tables (Equi Joins), Joining a Table to itself (self Joins), Sub queries Union, intersect & Minus Clause,

Manipulation Data in SQL.DML.

DDL-Oracle data types, Data Constraints, Types– Keys - Column level & table Level Constraints, working with Tables. Defining different constraints on the table, Defining Integrity Constraints in the ALTER TABLE Command, creating view, Renaming the Column of a view,

TCL: Granting Permissions, - Updating, Selection, destroying view Creating Indexes,

DCL: Creating and managing User.

Adv Features: Integrity – Triggers - Security – Advanced SQL features –Embedded SQL– Dynamic SQL- Missing Information– Views – Introduction to Distributed Databases and Client/Server Databases.

CHAPTER-1

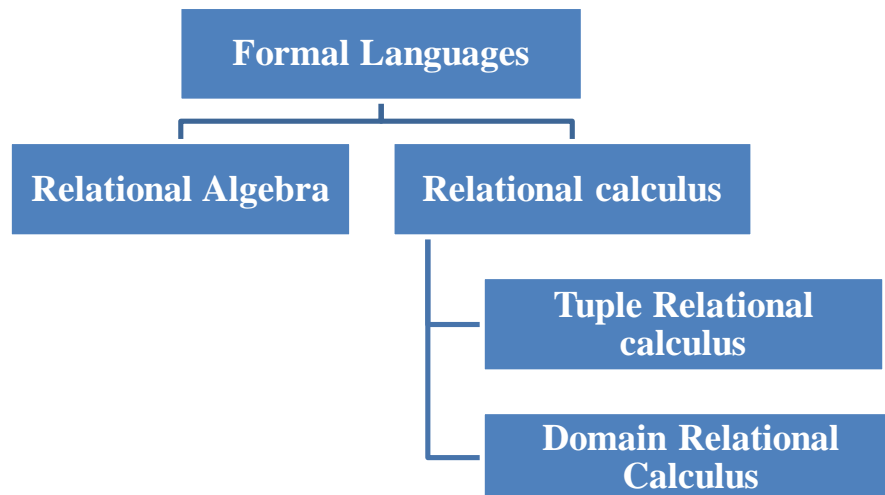
RELATIONAL MODEL AND RELATIONAL ALGEBRA

The Relational Model

Relational Model was proposed by E.F. Codd to model data in the form of relations or tables. After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDMBS languages like Oracle SQL, MySQL

The Catalogue: The system catalogue is a collection of tables and views that contain important information about a database. It is the place where a relational database management system stores schema metadata, such as information about tables and columns, and internal bookkeeping information. A system catalogue is available for each database. Information in the system catalogue defines the structure of the database. For example, the DDL (data dictionary language) for all tables in the database is stored in the system catalogue. Most system catalogues are copied from the template database during database creation, and are thereafter database-specific. A few catalogues are physically shared across all databases in an installation; these are marked in the descriptions of the individual catalogues. The system catalogue for a database is actually part of the database. Within the database are objects, such as tables, indexes, and views. The system catalogue is basically a group of objects that contain information that defines other objects in the database, the structure of the database itself, and various other significant information.

Formal Languages: Relational algebra and relational calculus are formal query language

**Relational Algebra:**

Relational algebra is a procedural query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data. When I say that relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.

On the other hand relational calculus is a non-procedural query language, which means it tells what data to be retrieved but doesn't tell how to retrieve it.

Types of operations in relational algebra:

We have divided these operations in two categories:

1. Basic Operations
2. Derived Operations

Basic/Fundamental Operations:

1. Select (σ)
2. Project (Π)
3. Union (\cup)
4. Set Difference ($-$)
5. Cartesian product (\times)
6. Rename (ρ)

Derived Operations:

1. Natural Join (\bowtie)
2. Left, Right, Full outer join (\Join , \Join , \Join)

3. Intersection (\cap)

4. Division (\div)

Basic/Fundamental Operations:

1. Select Operator (σ)

Select Operator is denoted by sigma (σ) and it is used to find the tuples (or rows) in a relation (or table) which satisfy the given condition. If you understand little bit of SQL then you can think of it as a where clause in SQL, which is used for the same purpose.

Syntax of Select Operator (σ):

σ Condition/Predicate (Relation/Table name)/ $\sigma_{condition} relation$

Example 1:

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

$\sigma_{Customer_City="Agra"} (CUSTOMER)$

Output:

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra

Example 2 : Person

$\sigma_{Hobby='stamps'}(Person)$

<i>Id</i>	<i>Name</i>	<i>Address</i>	<i>Hobby</i>
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

<i>Id</i>	<i>Name</i>	<i>Address</i>	<i>Hobby</i>
1123	John	123 Main	stamps
9876	Bart	5 Pine St	stamps

Selection Operators:

- ✓ Operators: <, >=, <=, >, =, ≠
- ✓ Simple selection condition:
- ✓ <attribute> operator <constant>
- ✓ <attribute> operator <attribute>
- ✓ <condition> AND <condition>
- ✓ <condition> OR <condition>
- ✓ NOT <condition>

Examples on Selection:

- ✓ $\sigma_{Id > 3000 \text{ Or } Hobby = 'hiking'}(Person)$
- ✓ $\sigma_{Id > 3000 \text{ AND } Id < 3999}(Person)$
- ✓ $\sigma_{NOT(Hobby = 'hiking')}(Person)$
- ✓ $\sigma_{Hobby \neq 'hiking'}(Person)$

Project Operator (Π)

Project operator is denoted by Π symbol and it is used to select desired columns (or attributes) from a table (or relation). Project operator in relational algebra is similar to the Select statement in SQL

Syntax of Project Operator (Π)

$\Pi \text{ column_name1, column_name2, ..., column_nameN}(\text{table_name}) / \Pi_{\text{attribute list}}(\text{relation})$

Project Operator (Π) Example

In this example, we have a table CUSTOMER with three columns, we want to fetch only two columns of the table, which we can do with the help of Project Operator Π .

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Bangalore

Query:

Π Customer_Name, Customer_City (CUSTOMER)

Output:

Customer_Name	Customer_City
Steve	Agra
Raghu	Agra
Chaitanya	Noida
Ajeet	Delhi
Carl	Bangalore

Example 2:**Person** **$\Pi_{\text{Name, Hobby}}(\text{Person})$**

Id	Name	Address	Hobby
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Name	Hobby
John	stamps
John	coins
Mary	hiking
Bart	stamps

Example 3: Example on Expression

$$\Pi_{Id, Name} (\sigma_{Hobby='stamps' \text{ OR } Hobby='coins'} (\text{Person}))$$

Person

<i>Id</i>	<i>Name</i>	<i>Address</i>	<i>Hobby</i>
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Result

<i>Id</i>	<i>Name</i>
1123	John
9876	Bart

Set Operators:

- ✓ Relation is a set of tuples => set operations should apply
- ✓ Result of combining two relations with a set operator is a relation => all its elements must be tuples having same structure
- ✓ Hence, scope of set operations limited to union compatible relations.

Union Compatible Relation:

- ✓ Two relations are *union compatible* if
 - Both have same number of columns
 - Names of attributes are the same in both
 - Attributes with the same name in both relations have the same domain
- ✓ Union compatible relations can be combined using *union*, *intersection*, and *set difference*

1. Union Operation (U)

- ✓ It performs binary union between two given relations and is defined as –
- ✓ $r \cup s = \{ t \mid t \in r \text{ or } t \in s \}$ Notation – $r \cup s$
- ✓ Where r and s are either database relations or relation result set .
- ✓ For a union operation to be valid, the following conditions must hold –
- ✓ r , and s must have the same number of attributes.
- ✓ Attribute domains must be compatible.

- ✓ Duplicate tuples are automatically eliminated.
- ✓ $\Pi \text{ author (Books)} \cup \Pi \text{ author (Articles)}$

Example:

Tables: Person (SSN, Name, Address, Hobby) and Professor (Id, Name, Office, Phone)
are not union compatible.

$\Pi_{Name}(\text{Person})$ and $\Pi_{Name}(\text{Professor})$
are union compatible and
 $\Pi_{Name}(\text{Person}) - \Pi_{Name}(\text{Professor})$

Set Difference (-):

The result of set difference operation is tuples, which are present in one relation but are not in the second relation. Notation – $r - s$.

Example: Finds all the tuples that are present in r but not in s.

$\Pi \text{ author (Books)} - \Pi \text{ author (Articles)}$

Output – Provides the name of authors who have written books but not articles.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho ρ .

Notation – $\rho_x(E)$, Where the result of expression E is saved with name of x.

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.

Rename (ρ) Syntax:

$\rho(\text{new_relation_name}, \text{old_relation_name})$

Rename (ρ) Example

Lets say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

$$\rho \text{ (CUST_NAMES, } \Pi \text{ (Customer_Name)(CUSTOMER))}$$
Output: CUST_NAMES

Steve

Raghu

Chaitanya

Ajeet

Carl

Derived Operation: Join

The expression : $\sigma_{\text{join-condition}} (R \times S)$

where join-condition' is a conjunction of terms:

$$A_i \text{ oper } B_i$$

in which A_i is an attribute of R , B_i is an attribute of S , and oper is one of $=, <, >, \geq, \neq, \leq$, is referred to as the (theta) join of R and S and denoted:

$$R \bowtie_{\text{join-condition}} S$$
Join operations in Relational Algebra:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by \bowtie .

Example: EMPLOYEE

EMP_CODE	EMP_NAME
----------	----------

101	Stephan
102	Jack
103	Harry

SALARY :

EMP_CODE	SALARY
101	50000
102	30000
103	25000

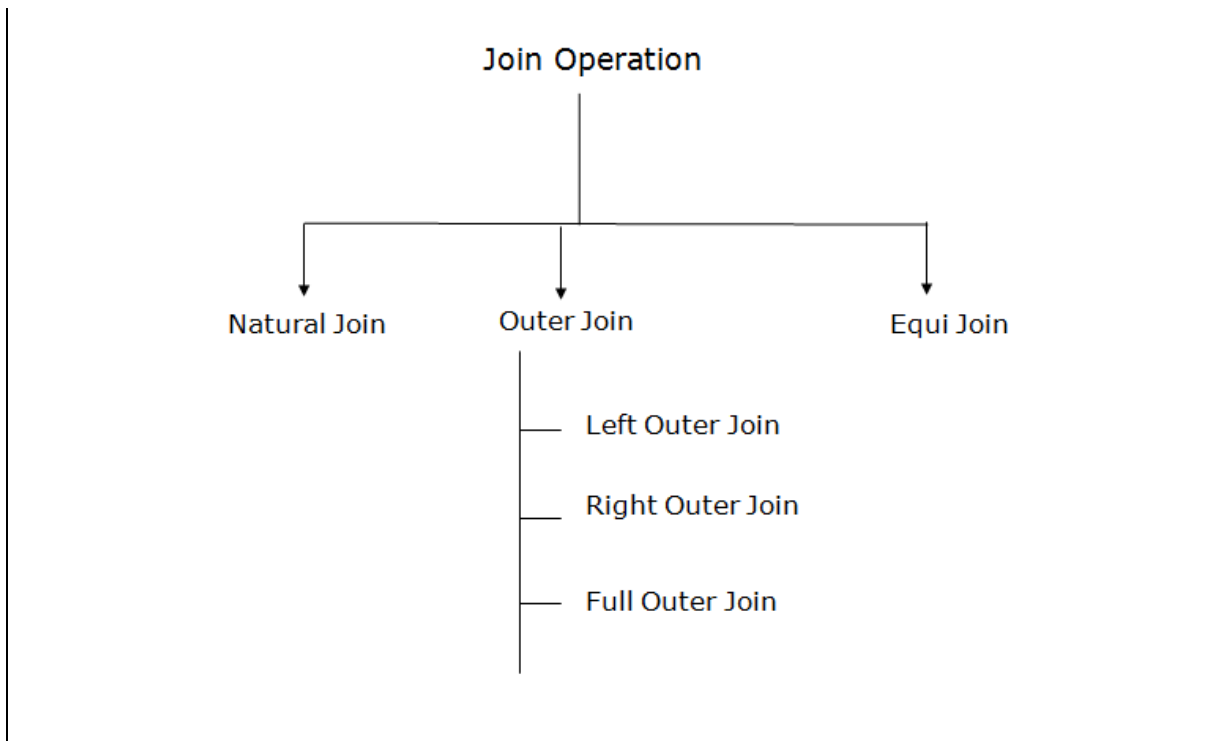
Operation: (EMPLOYEE ⋈ SALARY)

Result:

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000
102	Jack	30000
103	Harry	25000

Types of Join operations:

--



1. Natural Join:

- ✓ A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- ✓ It is denoted by \bowtie .

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

1. Π EMP_NAME, SALARY (EMPLOYEE \bowtie SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000

Harry	25000
-------	-------

2. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000

Hari	TCS	50000
------	-----	-------

Input:(EMPLOYEE \bowtie FACT_WORKERS)**Output:**

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

An outer join is basically of three types:

- i. Left outer join
- ii. Right outer join
- iii. Full outer join

i. Left outer join:

- ✓ Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- ✓ In the left outer join, tuples in R have no matching tuples in S.
- ✓ It is denoted by $\bowtie\leftarrow$.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

EMPLOYEE \bowtie FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

ii. Right outer join:

- ✓ Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- ✓ In right outer join, tuples in S have no matching tuples in R.
- ✓ It is denoted by $\bowtie\leftarrow$.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input: EMPLOYEE $\bowtie\leftarrow$ FACT_WORKERS

Output:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad

Kuber	HCL	30000	NULL	NULL
-------	-----	-------	------	------

iii. Full outer join:

- ✓ Full outer join is like a left or right join except that it contains all rows from both tables.
- ✓ In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- ✓ It is denoted by \bowtie .

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

EMPLOYEE \bowtie FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator (=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Input:

CUSTOMER ⋈ PRODUCT

Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai

3	Harry	3	Noida
---	-------	---	-------

EXAMPLE ON JOINS:

Find Ids of all professors who taught at least two courses in the same semester:

```
SELECT T1.ProfId
FROM Teaching T1, Teaching T2
WHERE T1.ProfId = T2.ProfId
      AND T1.Semester = T2.Semester
      AND T1.CrsCode <> T2.CrsCode
```

Equivalent to:

$$\pi_{\text{ProfId}} (\sigma_{\text{T1.CrsCode} \neq \text{T2.CrsCode and (Teaching T1} \\ \text{T1.Semester=T2.semester and T1.profId=T2.profId Teaching T2)})$$
Relational Calculus:

Contrary to Relational Algebra which is a procedural query language to fetch data and which also explains how it is done, Relational Calculus is a non-procedural query language and has no description about how the query will work or the data will be fetched. It only focusses on what to do, and not on how to do it.

Relational Calculus exists in two forms:

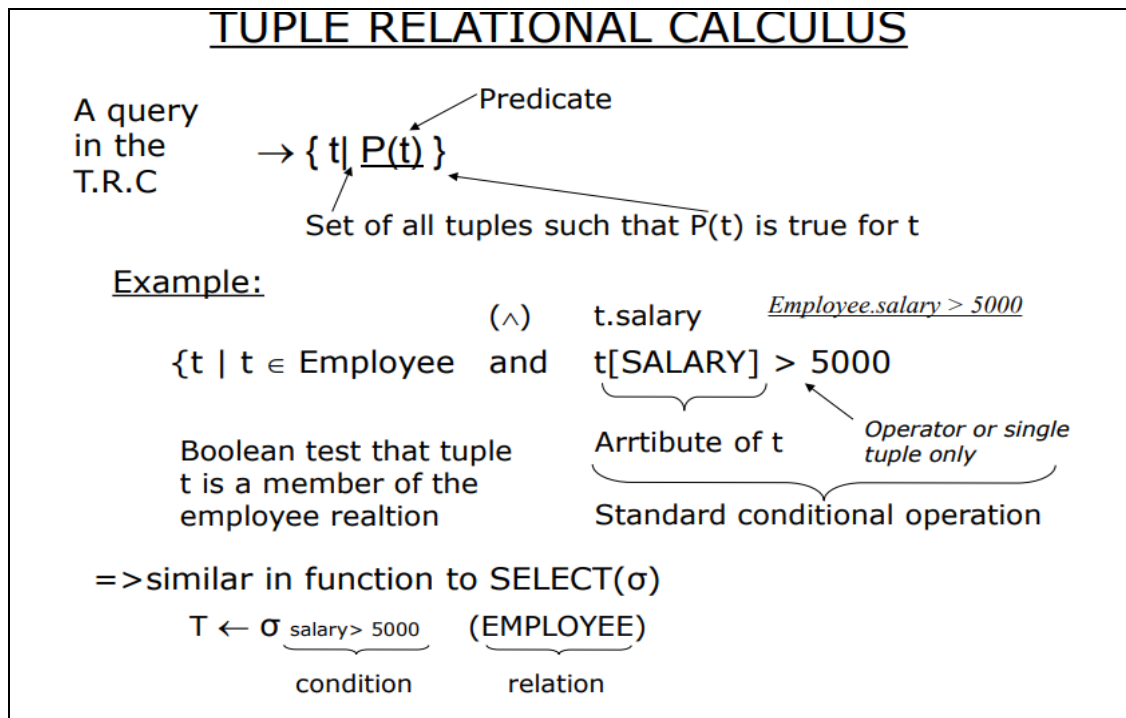
1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

1. TUPLE RELATIONAL CALCULUS

✓ Relational calculus is a query language which is non-procedural, and instead of algebra, it uses mathematical predicate calculus.

✓ When applied to databases, it is found in two TRC, DRC.

Syntax and Example of Tuple Relational calculus:



Example: Let us define tuple variable (there exist t) $\exists t$.

First Name	Last Name	Marks
James	Jhonson	90
Jack	Jill	20
Jones	Jim	30

$\{ t \mid \text{student}(t) \text{ AND } t.\text{marks} > 50 \}$ It display all with the condition $\text{marks} > 50$

$\{ t.\text{fname } t.\text{lname} \mid \text{student}(t) \text{ AND } t.\text{marks} > 50 \}$ Display two columns having condition $\text{marks} > 50$.

Let's see the example, to specify the range of a tuple variable S as the Staff relation, we write:

Staff(S)

And to express the query 'Find the set of all tuples S such that $F(S)$ is true,' we can write:

$\{ S \mid F(S) \}$, here, F is called a formula (well-formed formula, or wff in mathematical logic).

For example, to express the query to 'Find the staffNo, fName, lName, position, sex, DOB, salary,

and branchNo of all staff earning more than £10,000',

we can write: $\{S \mid \text{Staff}(S) \wedge S.\text{salary} > 10000\}$

Example:

$t \mid \text{TEACHER}(t) \text{ and } t.\text{SALARY} > 20000\}$, - It implies that it selects the tuples from the TEACHER in such a way that the resulting teacher tuples will have the salary greater than 20000. This is an example of selecting a range of values.

Example:

$\{t \mid \text{TEACHER}(t) \text{ AND } t.\text{DEPT_ID} = 8\}$, - T select all the tuples of teachers name who work under Department 8.

Domain Relational Calculus:

In domain relational calculus, filtering is done based on the domain of the attributes and not based on the tuple values. $\{x_1, x_2, x_3, \dots \mid \text{condition}(x_1, x_2, \dots)\}$

Example:

emp(empno, ename, job, sal), let us take domain variable as a, b, c, d

dept(deptno, dname, loc). let us take domain variable x, y, z

Query:

List the name and job of the employee whose name of SCOTT and JOB CLERK

a, b, c, d domain variables

$\{a, c \mid \text{there exist } b, d(\text{emp}(abcd) \text{ and } (a = \text{'SCOTT'}) \text{ and } (c = \text{'CLERK'}))\}$

Example:

Loan_no	branch	amount
100	BLORE	1000
200	MYS	1500
300	HBL	1230

Query: List all the loan number of all the loans with an amount greater than 1000.

$\{l \mid b, a \text{ there exists}(\text{loan}(l, b, a) \text{ and } a > 1000)\}$.

Query: List all the branch where loan amount less than 1500

$\{b \mid b, a \text{ there exists } (\text{loan}(a, b, a) \text{ and } a > 1000)\}$

Query: List the employees work for dname Accounting.

$\{t.ename \mid emp(t)$
 $AND(d)dept(d) \wedge dname='SALES'$
 $AND emp.deptno=dept.deptno\}$

Query: Obtain the names of courses enrolled by student named Mahesh

$\{c.name \mid course(c)$
 $\wedge (\exists s) (\exists e) (student(s)$
 $\wedge enrollment(e)$
 $\wedge s.name = "Mahesh"$
 $\wedge s.rollNo = e.rollNo$
 $\wedge c.courseId = e.courseId)$

Query: Get the names of students who have scored 'S' in all subjects they have enrolled.

Assume that every student is enrolled in at least one course.

$\{s.name \mid student(s)$
 $\wedge (\forall e)((enrollment(e)$
 $\wedge e.rollNo = s.rollNo)$
 $\wedge e.grade = 'S')\}$

Query: Determine the departments that do not have any girl students.

$\{d.dname \mid department(d)$
 $\wedge \neg(\exists s)(student(s)$
 $\wedge s.sex = 'F'$
 $\wedge s.deptNo = d.deptId)$

CHAPTER-2

STRUCTURED QUERY LANGUAGE

SQL stands for Structured Query Language, which is a standardised language for interacting with RDBMS (Relational Database Management System). Some of the popular relational database example are: MySQL, Oracle, mariaDB, PostgreSQL etc. SQL is used to perform C.R.U.D (Create, Retrieve, Update & Delete) operations on relational databases, it can also perform administrative tasks on database such as database security, backup, user management etc. We can create databases and tables inside database using SQL.

Sub Languages in SQL:

Language	Statement	Description
Data Retrieval Language (DRL)	SELECT	Retrieves data from the database
Data Manipulation Language (DML)	INSERT UPDATE DELETE MERGE	Enters new rows, changes existing rows, delete unwanted rows from the database.
Data Definition Language (DDL)	CREATE ALTER RENAME DROP TRUNCATE	Setup, change and remove data structure from tables
Transaction Control Language (TCL)	COMMIT ROLLBACK SAVEPOINT	Manage the changes made by data manipulation language. Changes to the data can be grouped together into logical transaction.
Data Control Language (DCL)	GRANT REVOKE	Gives and removes access right to both oracle database and structure within it.

Tool Used to execute SQL:

- ✓ **Oracle 11g** tool is used in this course.

ORACLE 11g Installation:

Install oracle 11g console window. **During installation it prompts to enter user name and password**, carefully enter **sys as user name and sys as password**. After the installation user can use many built in schemas, and in this course **SCOTT SCHEMA** is used for the complete practice till DDL is used in the course. To use the **SCOTT SCHEMA**, follow the below import SCOTT schema steps.

Importing SCOTT schema:

- ✓ After the installation of ORCALE 11g, SCOTT schema would not be used until user imports.
- ✓ So, Import SCOTT schema using below statement in the SQL import.
- ✓ **SQL>@%ORACLE_HOME%\RDBMS\ADMIN\SCOTT.sql**
- ✓ Then try connecting SCOTT schema using the statement

SQL> connect scott/tiger hit enter then enter password as tiger

or **SQL> conn scott** hit enter then enter password as tiger

, if it gives any error then write the below statement in the SQL prompt

SQL>conn sys as sysdba

Password: sys

SQL>ALTER USER scott

IDENTIFIED BY tiger (nothing is case sensitive)

You will receive a feedback saying user altered.

Then try reconnecting to scott by following above step **SQL>conn scott/tiger**

Table used in the SCOTT SCHEMA :**1. EMP**

- ✓ EMPNO: Primary key
- ✓ ENAME
- ✓ MGR
- ✓ HIREDATE
- ✓ JOB
- ✓ SAL
- ✓ COMM
- ✓ DEPTNO: Foreign key referring to DEPTNO of DEPT table

2. DEPT

- ✓ DEPTNO: Primary key
- ✓ DNAME
- ✓ LOC

3. SALGRADE

- ✓ GRADE
- ✓ LOSAL
- ✓ HISAL

In the above entities:

- EMPNO of EMP table is a PRIMARY KEY.
- DEPTNO of DEPT table is a PRIMARY KEY.
- DEPTNO of EMP table is a FOREIGN KEY referring to DEPTNO of DEPT table.
- SALGRADE IS A WEAK ENTITY.

DUAL: It is the table owned by SYS and can be accessed by all the users.

DATA RETRIEVAL LANGUAGE (DRL):

DRL is a language use to fetch the data from the tables.

SELECT is a keyword. It is used to retrieve information from the database.

CAPABILITIES OF SELECT STATEMENT:

- ✓ **PROJECTION:** This capability is used to choose columns of a table.
- ✓ **SELECTION:** This capability is used to choose rows of a table.
- ✓ **JOINING:** This capability is used to join/link two or more tables together.

SYNTAX:

SELECT * | { [DISTINCT] column | expression [alias], ...}

From table;

Here,

SELECT	is a list of one or more columns to be displayed
*	select all columns
DISTINCT	suppresses duplicates
column expression	select the named column or the expression alias
	gives selected columns different heading
FROM table	specifies the table containing column

Query 1: Query to selecting all columns.

SQL>SELECT * FROM EMP;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30

Query 2: Query to selecting specific columns.

SQL>SELECT EMPNO, ENAME FROM EMP;

EMPNO	ENAME
7369	SMITH
7499	ALLEN
7521	WARD

COLUMN ALIAS

A column alias renames the column heading. If the alias contains a space or special characters or it is case sensitive, then alias must be written in double quotations (“...”). The keyword AS can be used in between the expression or column name and alias. The advantage of using alias is that it becomes helpful in performing calculations.

Query 3: Query to display ENAME and giving the alias as EMPLOYEE NAME.

```
SELECT ENAME AS “EMPLOYEE NAME” FROM EMP;
```

EMPLOYEE NAME
SMITH
ALLEN
WARD
JONES

DISTINCT KEYWORD

Distinct keyword is used to eliminate duplicate tuples. It is to be included in SELECT statement.

Query 4: Query to display DEPTNO from the EMP table.

```
SELECT DISTINCT DEPTNO FROM EMP;
```

DEPTNO
30
20
10

ARITHMETIC OPERATORS

These operators can be used in any clause except in the FROM clause. The operators are:

Operators	Description
+	Add
-	Subtract
*	Multiply

/	Divide
()	Parentheses

Query 5: Query to display ENAME, SAL, COMM and sum of SAL+COMM.

SELECT ENAME, SAL, COMM, SAL+COMM FROM EMP;

ENAME	SAL	COMM	SAL+COMM
SMITH	800		
ALLEN	1600	300	1900
WARD	1250	500	1750

Query 6: Query to display ENAME, SAL, COMM and sum of SAL+COMM and multiply it by 2:

SELECT ENAME, SAL, COMM, (SAL+COMM)*2 NETSAL FROM EMP

ENAME	SAL	COMM	NETSAL
SMITH	800		
ALLEN	1600	300	3800
WARD	1250	500	3500

NULL VALUES

A null value is a value which is unavailable, unassigned, unknown, or inapplicable. It is not same as zero or blank space.

Like in query 4, the commission column has null values, i.e.: it implies that, that particular employee is not given commission.

NOTE: If there is a null value in an arithmetic expression the result is null.

In this query 5, employee who does not take commission will have the NETSAL as null.

CONCATENATION OPERATOR

A concatenation Operator is used to concatenate columns or character strings to other columns. It is represented by two vertical bars (||). The Columns on either side of the operator is combined to make one single column.

Query 7: Query to concatenate ENAME and JOB the naming to column as NAME_JOB. The output should be like “SMITH WORKS AS CLERK”.

```
SELECT ENAME || 'WORKS AS' || JOB AS "NAME_JOB" FROM EMP;
```

NAME_JOB
SMITH WORKS AS CLERK
ALLEN WORKS AS SALESMAN
WARD WORKS AS SALESMAN
JONES WORKS AS MANAGER

RESTRICTING ROWS: WHERE CLAUSE

If we want to know the employee details of employees working in DEPTNO 10. Then we need to restrict the rows giving the condition that the DEPTNO should be 10.

This method of restricting rows is done using WHERE clause. WHERE clause follows the FROM clause.

Syntax:

```
SELECT * | { [DISTINCT] column | expression [alias], ... }
```

```
From table
```

```
[WHERE condition(s)];
```

Here,

WHERE restricts the query to rows that meet a condition

Condition is composed of column names, expressions, constants and a comparison operator.

A WHERE clause has three elements:

- ✓ Column name
- ✓ Comparison condition
- ✓ Column name, constants, or list of values

In the WHERE clause:

- ✓ Character string and date values are to be enclosed in single quotation ('...').
- ✓ Character strings are case sensitive.
- ✓ Data values are format sensitive.

Query 8: Query to display the list of employee's works in DEPTNO 10.

```
SELECT ENAME, DEPTNO FROM EMP  
WHERE DEPTNO=10;
```

ENAME	DEPTNO
-----	-----
CLARK	10
KING	10
MILLER	10

Query 9: Query to display the list of employees who works as CLERK.

```
SELECT ENAME, JOB FROM EMP
```

WHERE JOB='CLERK';

ENAME	JOB
-----	-----
SMITH	CLERK
ADAMS	CLERK
JAMES	CLERK
MILLER	CLERK

COMPARISION OPERATORS

Operator	Meaning
=	Equal to
<	Less than
>	More than
<=	Less than or equal to
>=	More than or equal to
<>	Not equal to
BETWEEN ... AND ...	Between two values(inclusive)
IN(set)	Match any of the list values
LIKE	Match a character pattern
IS NULL	Is a null value

Note:

In like operator:

% represents zero or many characters

_ represents one character Examples

of using Comparisons operators:

Query 10: Query to display the ENAME taking SAL more than 2000.

```
SELECT ENAME, SAL FROM EMP
WHERE SAL>2000;
```

ENAME	SAL
-----	-----
JONES	2975
BLAKE	2850
CLARK	2450

Query 11: Query to display the ENAME taking SAL between 2000 and 3000.

```
SELECT ENAME, SAL FROM EMP WHERE
SAL BETWEEN 2000 AND 3000;
```

ENAME	SAL
-----	-----
JONES	2975
BLAKE	2850
CLARK	2450

Query 12: Query to display the ENAME working in DEPTNO 10 and 20.

```
SELECT ENAME, DEPTNO FROM EMP
WHERE DEPTNO IN (10, 20);
```

ENAME	DEPTNO
SMITH	20
JONES	20
CLARK	10

Query 13: Query to display the ENAME not taking COMM.

```
SELECT ENAME FROM EMP
WHERE COMM IS NULL;
```

ENAME
SMITH
JONES
BLAKE
CLARK

Query 14: Query to display ENAME of employee whose name start from A.

```
SELECT ENAME FROM EMP
WHERE ENAME LIKE 'A%';
```

ENAME
ALLEN
ADAMS

Query 15: Query to display ENAME of employee whose name has last second letter E.

```
SELECT ENAME FROM EMP
WHERE ENAME LIKE '%E_';
```

ENAME
ALLEN
JONES
TURNER

LOGICAL OPERATORS

Operator	Meaning
AND	Returns TRUE if <u>both</u> component conditions are true
OR	Returns TRUE if <u>either</u> component condition is true
NOT	Returns true if the following condition is false

Examples of using Logical Operators:

Query 16: Query to display ENAME taking SAL more than 500 and JOB is CLERK.

```
SELECT ENAME, SAL, JOB FROM EMP
WHERE SAL>500 AND JOB='CLERK';
```

ENAME	SAL	JOB
SMITH	800	CLERK
ADAMS	1100	CLERK
JAMES	950	CLERK
MILLER	1300	CLERK

Query 17: Query to display ENAME taking SAL more than 2000 or JOB is CLERK.

```
SELECT ENAME, SAL, JOB FROM EMP
WHERE SAL>2000 OR JOB='CLERK';
```

ENAME	SAL	JOB
SMITH	800	CLERK
JONES	2975	MANAGER
BLAKE	2850	MANAGER
CLARK	2450	MANAGER

Query 18: Query to display ENAME whose COMM is not null.

```
SELECT ENAME, COMM FROM EMP
WHERE COMM IS NOT NULL;
```

ENAME	COMM
ALLEN	300
WARD	500
MARTIN	1400
TURNER	0

RULES OF PRECEDENCE

ORDER EVALUATED	OPERATOR
1	Parentheses [...]
2	Multiply (*) or Divide (/)
3	Addition (+) or Subtraction (-)
4	Concatenation ()
5	Comparison Operators(< or > or <= or >= or = or <>)
6	IS [NOT] NULL, [NOT] LIKE, [NOT] IN
7	[NOT] BETWEEN ... AND
8	NOT Logical Condition
9	AND Logical Condition
10	OR Logical Condition

Override rules of precedence by parentheses.

Query 19: Query to display ENAME whose JOB is CLERK or ANALYST and takes SAL more than 1000.

```
SELECT ENAME, JOB, SAL FROM EMP
WHERE JOB='CLERK'
OR JOB='ANALYST'
AND SAL>1000;
```

ENAME	JOB	SAL
SMITH	CLERK	800
SCOTT	ANALYST	3000
ADAMS	CLERK	1100
JAMES	CLERK	950

Here, the select statement will read the WHERE clause as:

“Select the rows where JOB is equal to CLERK and SAL is more than 1000, or if JOB is equal to ANALYST.”

ORDER BY CLAUSE

To sort the rows in ascending or descending order according to the column(s) we use ORDER BY clause.

- ✓ ASC: ascending order (the default order)
- ✓ DESC: descending order

ORDER BY clause comes last in the select statement. Syntax:

```
SELECT * | { [DISTINCT] column | expression [alias], ... }
```

```
FROM table
```

```
[WHERE condition(s)]
```

```
[ORDER BY column(s) | expression | alias];
```

Query 20: Query to display ENAME, JOB, and SAL in the ascending order of SAL.

```
SELECT ENAME, JOB, SAL FROM EMP
```

```
ORDER BY SAL;
```

ENAME	JOB	SAL
SMITH	CLERK	800
JAMES	CLERK	950
ADAMS	CLERK	1100
WARD	SALESMAN	1250

Query 21: Query to display ENAME, JOB, and SAL in DESCENDING order of SAL. SELECT ENAME, JOB, SAL FROM EMP

```
ORDER BY SAL DESC;
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
FORD	ANALYST	3000
SCOTT	ANALYST	3000
JONES	MANAGER	2975

Query 22: Query to display ENAME, JOB, and SAL in ASCENDING order of SAL AND JOB.

(Sorting by multiple columns)

```
SELECT ENAME, JOB, SAL FROM EMP  
ORDER BY SAL, JOB;
```

ENAME	JOB	SAL
SMITH	CLERK	800
JAMES	CLERK	950
ADAMS	CLERK	1100
WARD	SALESMAN	1250

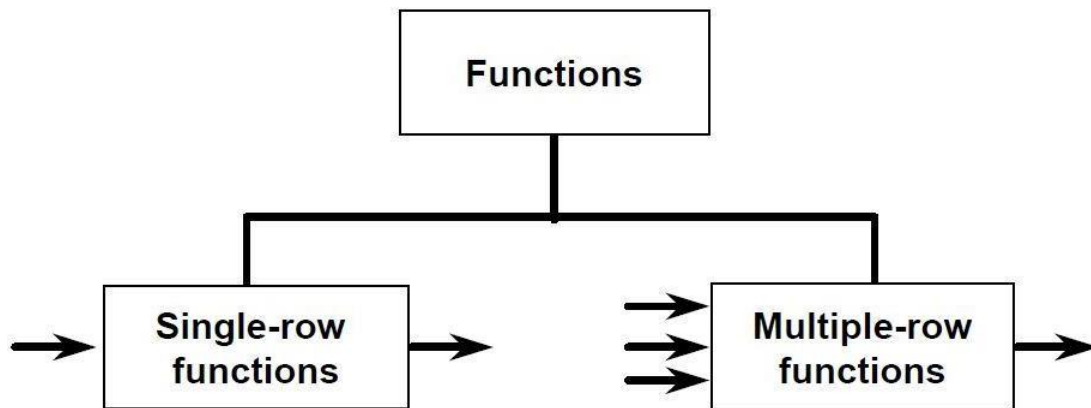
Query 23: Query to display ENAME, JOB, and SAL in DECENDING order of SAL. Give an alias to the column SAL and order using the alias. (Sorting using alias)

```
SELECT ENAME, JOB, SAL SALARY FROM EMP  
ORDER BY SALARY DESC;
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
FORD	ANALYST	3000
SCOTT	ANALYST	3000
JONES	MANAGER	2975

SQL FUNCTIONS

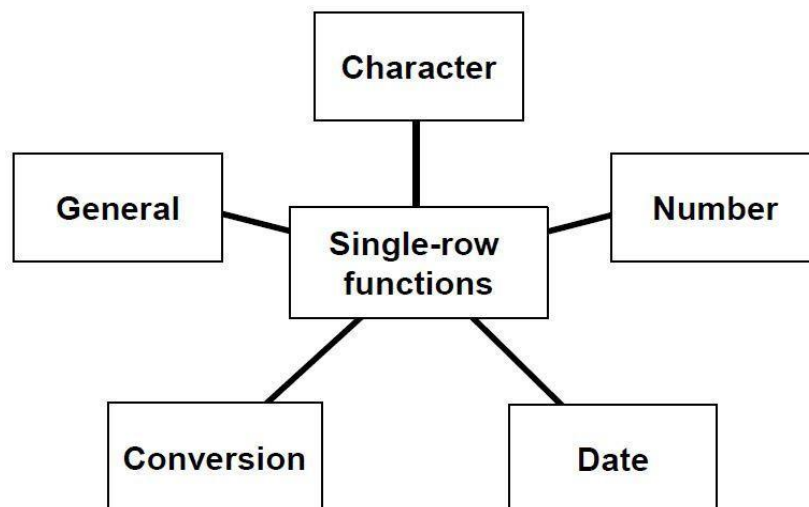
Functions is a feature of SQL which can be used to perform calculations, modify individual data items, manipulate output for groups of rows, format date and umbers for display, convert column data type, etc. Some SQL functions take arguments and some does not take arguments.



There are two types of SQL functions:

1. **Single Row Functions:** Functions which are operated on single rows only and returns one result per row.
2. **Multiple Row Functions:** Function which can manipulate groups of rows to give one result per group of rows. These functions are known as group functions.

1. Single-Row Functions



Character Functions:

- **Case-Manipulation Functions:** These functions are case-conversion function.
 - i. LOWER: Converts all alpha character to lower case
Syntax: LOWER (column | expression)
Example: LOWER ('HELLO') □ hello
 - ii. UPPER: Converts all alpha character to upper case
Syntax: UPPER (column | expression)
Example: UPPER ('Hello') □HELLO
 - iii. INITCAP: Converts the first letter to upper case and remaining to lower case.
Syntax: INITCAP (column | expression)
Example: INITCAP ('hello everyone') □Hello Everyone

Query 24: Query to display the ENAME and JOB as “SMITH works as clerk”.

SELECT UPPER (ENAME) || ' works as ' || LOWER (JOB) FROM EMP;

UPPER(ENAME) 'WORKSAS' LOWER

JAMES works as clerk
FORD works as analyst
MILLER works as clerk

Character-Manipulation Function:**i. CONCAT: Concatenates two values together.**

Syntax: CONCAT (column1 | expression1, column2 | expression2)

Example: CONCAT ('HELLO', 'World') □HELLOWorld

ii. SUBSTR: Extracts a string of determined length. Syntax:

SUBSTR (column | expression, M, N)

Here, M=Starting character position N=length of the substring

Example: SUBSTR ('HELLO',2,3) □ELL

iii. LENGTH: Extracts the length of the string. Syntax:

LENGTH (column | expression)

Example: LENGTH ('HELLOWorld') □10

iv. INSTR: Find numeric position of a named character.

Syntax: INSTR (column |expression, 'character' [,M, N])

Here, M=starting character position of the search N=occurrence of the character

In default values of M and N are 1.

Example: INSTR ('SALARY', 'A', 2, 2) □4

v. LPAD: Pads the character value right-justified to total width of N character positions.

Syntax: LPAD (column |expression, N, 'character')

Example: LPAD (1000, 10, '/') □////////1000

vi. RPAD: Pads the character value left-justified to total width of N character positions.

Syntax: RPAD (column |expression, N, 'character')

Example: RPAD (1000, 10, '\') □1000\\\\\\\\

vii. REPLACE: Searches a text expression for a character string and if found, replaces it with a specified replacement string.

Syntax: REPLACE (text, search_string, replacement_string) Note: Character functions can be used in SELECT clause or WHERE clause.

Number Functions:**i. ROUND: Rounds values to specific decimals.**

Syntax: ROUND (column | expression, n)

Here, n=number of decimal places to be rounded off.

ii. TRUNC: Truncates values to specified decimals.

Syntax: TRUNC (column | expression, n)

Here, n=number of decimal places to be truncated.

iii. MOD: Returns the remainder when M is divided by N.

Syntax: MOD (m, n)

Query 25: Query to round off 35.943 to 2, 0 and -1 decimal places.

```
SELECT ROUND (35.943, 2), ROUND (35.943, 0) , ROUND (35.943, -1) FROM
DUAL;
```

ROUND(35.943,2)	ROUND(35.943,0)	ROUND(35.943,-1)
35.94	36	40

Query 26: Query to truncate 35.943 to 2, 0 and -1 decimal places.

```
SELECT TRUNC (35.943, 2) , TRUNC (35.943, 0) , TRUNC (35.943, -1)
FROM DUAL;
```

TRUNC(35.943,2)	TRUNC(35.943,0)	TRUNC(35.943,-1)
35.94	35	30

Query 27: Query to find the remainder when 3 is divided by 2.

```
SELECT MOD (3,2) FROM DUAL;
```

MOD(3,2)
1

Date Functions:

- i. **SYSDATE**: Returns current system date and time.
- ii. **ROUND**: Returns date rounded off according to fmt. Syntax: ROUND (date [, 'fmt'])
Here, n=number of decimal places to be rounded off.
- iii. **TRUNC**: Returns date truncated according to fmt. Syntax: TRUNC (date [, 'fmt'])
Here, n=number of decimal places to be truncated.
- iv. **MONTHS_BETWEEN**: Returns the number of months between two dates.
Syntax: MONTHS_BETWEEN(date1, date2)
- v. **ADD_MONTHS**: Adds calendar month to date. Syntax: ADD_MONTHS(date, n)
Here, n=number of months, it can be negative or positive integer
- vi. **LAST_DATE**: Returns the last day of the month. Syntax: LAST_DATE(date)
- vii. **NEXT_DAY**: Returns the next day of the date specified. Syntax: NEXT_DAY (date, 'char')

Query 28: Query to fetch current system date and time.

```
SELECT SYSDATE FROM DUAL;
```

SYSDATE

28-FEB-13

Query 29: Query to round off and truncate the SYSDATE according to months.

```
SELECT ROUND (SYSDATE, 'MONTHS'), TRUNC(SYSDATE, 'MONTH')
FROM DUAL;
```

ROUND OFF	TRUNCATE
-----	-----
01-MAR-13	01-FEB-13

Query 30: Query to find months between SYSDATE and 15-JAN-2011

```
SELECT MONTHS_BETWEEN (SYSDATE,'15-JAN-2011')
```

FROM DUAL;

MONTHS_BETWEEN(SYSDATE, '15-JAN-2011')

25.4497502

Query 31: Query to find the date after 8 months.

```
SELECT ADD_MONTHS (SYSDATE, 8) FROM DUAL;
```

ADD_MONTH

31-OCT-13

Query 32: Query to find the date before 5 months.

```
SELECT ADD_MONTHS (SYSDATE, -5) FROM DUAL;
```

ADD_MONTH

30-SEP-12

Query 33: Query to find the date of next Thursday.

```
SELECT      NEXT_DAY      (SYSDATE,
'THURSDAY') FROM DUAL;
```

NEXT_DAY

07-MAR-13

Query 34: Query to display the last date of the month. SELECT LAST_DAY (SYSDATE) FROM DUAL;

LAST_DAY

28-FEB-13

Query 35: Query to display the number of years SMITH worked as CLERK. (DATE ARITHMETIC)

```
SELECT ENAME, (SYSDATE-HIREDATE)/365 FROM EMP
WHERE ENAME='SMITH';
```

ENAME	(SYSDATE-HIREDATE)/365
-----	-----
SMITH	32.2245052

Conversion Functions:

✓ Implicit Conversion:

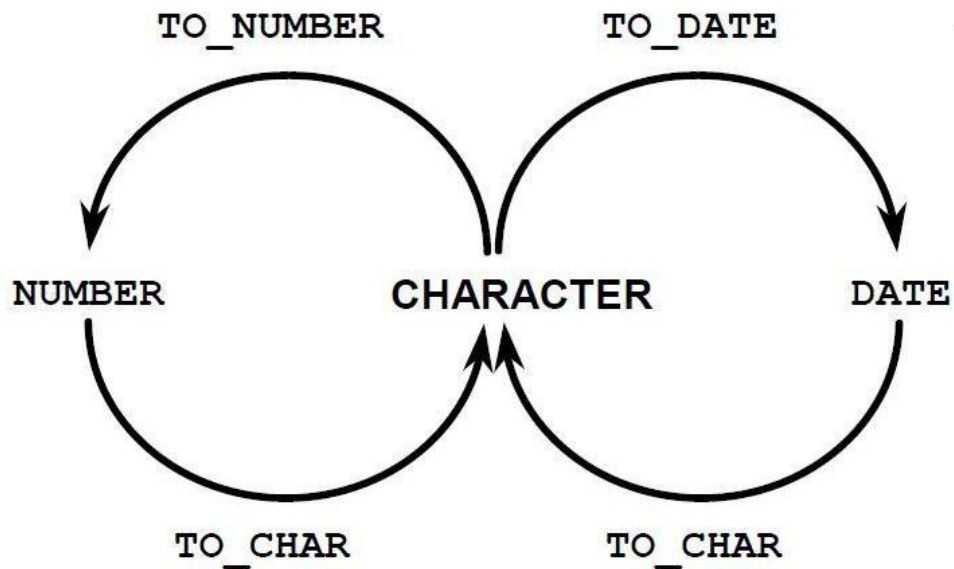
For assignments, the oracle server can automatically convert the follows:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2 or CHAR
DATE	VARCHAR2 or CHAR

For expression evaluation, the oracle server can automatically convert the following:

FROM	TO
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

✓ Explicit Conversion:



Using TO_CHAR function with dates-

It is used to convert from date to character. Syntax:

TO_CHAR(date, 'format-model')

Elements of the date format model:

YYYY	Full year in number
YEAR	Year spelled out
MM	Two-digit value of the month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day week
DAY	Full name of the day of the week
DD	Numeric day of the month

1. Time element format the time portion of the date

Syntax: HH24:MI:SS AM

Example: 16:15:20 PM

2. Add character string by enclosing them in double quotations

Syntax: DD “of” MOM

Example: 20 of FEB

3. Number Suffixes spelled out

number Syntax: ddsph

Example: twentieth

Query 36: Query to display ENAME, HIREDATE in the format “20th FEB 2012”

```
SELECT ENAME, TO_CHAR (HIREDATE, 'FMDDTH MONTH YYYY') AS
HIREDATE FROM EMP;
```

ENAME	HIREDATE
SMITH	17TH DECEMBER 1980
ALLEN	20TH FEBRUARY 1981
WARD	22ND FEBRUARY 1981

Using TO_CHAR with numbers:

It is used to convert from number to character. Syntax:

TO_CHAR(number, 'format-model')

Elements of number format model:

9	Represents of number
0	Forces a zero to be displayed
\$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a thousand indicator

Query 37: Query to display the salary of SCOTT as \$9,999.00

```

SELECT          ENAME,
TO_CHAR(SAL,'$9,999.00') SAL FROM EMP
WHERE ENAME='SCOTT';

```

ENAME	SAL
-----	-----
SCOTT	\$3,000.00

Using TO_DATE function:

This converts a string to date format.

Syntax: TO_DATE (char [, 'format-model'])

Example: TO_DATE(HIREDATE)

Using TO_NUMBER function:

This converts string to number format.

Syntax: TO_NUMBER (char [, 'format-model'])

Example: TO_NUMBER(SALARY)

General Functions:

NVL: This function converts a null to an actual value. Syntax:

Query: Query to calculate NETSAL of SCOTT. If he gets COMM then
NETSAL=SAL+COMM else NETSAL=SAL.

```
SELECT ENAME, NVL2(COMM,SAL+COMM,SAL) NETSAL
FROM EMP
WHERE ENAME='SCOTT';
```

ENAME	NETSAL
SCOTT	3000

✓ NULLIF: This function compares two expressions. If exp1 is equal to exp2 then the result is NULL.

Syntax: NULLIF (exp1, exp2)

✓ COALESCE: This function returns the first non-null expression in the
• list. Syntax: COALESCE (exp1, exp2, exp3, exp n)

CONDITIONAL EXPRESSION:

If we want IF-ELSE-IF logic within a SQL statement we use conditional expression. The conditional expressions are:

✓ CASE:

Syntax: CASE exp WHEN comparision_exp1 THEN
return_exp1 [WHEN comparision_exp2 THEN
return_exp2 WHEN comparision_expn THEN
return_expn ELSE else_expr]

END [alias]

✓ DECODE:

Syntax: DECODE (exp | column, search1, result1
[, search 2, result2.....] [,
default]) [alias]

Query 40: Query to display the revised salary as: CLERK – 10%, ANALYST – 20%,

SALESMAN – 15%.✓ **Using Case**

```

SELECT ENAME, JOB, SAL,
CASE JOB WHEN 'CLERK' THEN 0.10*SAL
        WHEN 'ANAYLST' THEN 0.20*SAL
        WHEN 'SALESMAN' THEN 0.15*SAL
        ELSE SAL
END      "REVISED_SALARY"
FROM EMP;

```

✓ **Using Decode**

```

SELECT  ENAME,  JOB,  SAL
DECODE (JOB, 'CLERK', 0.10*SAL,
        'ANAYLST', 0.20*SAL,
        'SALESMAN', 0.15*SAL, SAL) "REVISED_SALARY"
FROM EMP;

```

ENAME	JOB	SAL	REVISED_SALARY
SMITH	CLERK	800	80
ALLEN	SALESMAN	1600	240
WARD	SALESMAN	1250	187.5

GROUP FUNCTIONS

Group functions are functions which operate on a set of rows to give one result per group.

The group functions are:

- ✓ **AVG:** Average of the values in a column, ignoring null values. Syntax: AVG ([DISTINCT] column)
- ✓ **SUM:** Sum of the values in a column, ignoring null values. Syntax: SUM ([DISTINCT] column)
- ✓ **MIN:** Minimum value in a column, ignoring null values. Syntax: MIN ([DISTINCT] column)
- ✓ **MAX:** Maximum value in a column, ignoring null

values. Syntax: MAX ([DISTINCT] column)

✓ COUNT: It returns the number of rows in a table. Syntax: COUNT ([DISTINCT] * | column)

Query 41: Query to find minimum, maximum, sum, average of the salaries of the DEPTNO 10.

And also count the number of employees in that department.

```
SELECT MIN (SAL), MAX (SAL), AVG (SAL), SUM (SAL), COUNT (SAL)
FROM EMP
WHERE DEPTNO=10;
```

MIN(SAL)	MAX(SAL)	AVG(SAL)	SUM(SAL)	COUNT(SAL)
1300	5000	2916.66667	8750	3

NESTING FUNCTIONS

Function inside another function is called nesting function. It can be nested to any depth.

Query 42: Query to calculate average commission of the employees of DEPTNO 10.

```
SELECT AVG (NVL (COMM, 0)) FROM EMP
WHERE DEPTNO=10;
```

CREATING GROUPS OF DATA

When we want to divide the table of information into smaller groups we use GROUP BY clause.

Syntax: SELECT * | { [DISTINCT] column | expression [alias], ...}

From table

[WHERE condition(s)]

[GROUP BY group_by_expression]

[ORDER BY column(s)];

Query 43: Query to display the average salary of all the departments.

```
SELECT DEPTNO, AVG (SAL)
FROM EMP
GROUP BY DEPTNO;
```

DEPTNO	AVG(SAL)
30	1566.66667
20	2175
10	2916.66667

Query 44: Query to display the average salary department wise and job wise. (using GROUP BY clause on multiple columns)

```
SELECT DEPTNO, JOB, AVG (SAL)
FROM EMP
GROUP BY DEPTNO, JOB;
```

DEPTNO	JOB	AVG(SAL)
20	CLERK	950
30	SALESMAN	1400
20	MANAGER	2975
30	CLERK	950

RESTRICTING GROUP RESULTS

We use WHERE clause to restrict rows, in the same way we use HAVING clause to restrict groups.

Query 45: If we want to display the DEPTNO and average SAL of each department whose Average SAL is more than 1000, the query would be:

```
SELECT DEPTNO, AVG (SAL)
FROM EMP
GROUP BY DEPTNO
HAVING AVG (SAL) >1600;
```

DEPTNO	AVG(SAL)
20	2175
10	2916.66667

Syntax: SELECT * | { [DISTINCT] column | expression [alias], ...}

From table

[WHERE condition(s)]

[GROUP BY group_by_expression]

[HAVING group_condition] [ORDER
BY column(s)];

NESTING GROUP FUNCTIONS

Group functions can be nested to a depth of two.

Query 46: Query to display which DEPTNO gets maximum average salary.

```
SELECT MAX (AVG (SAL)) FROM EMP
GROUP BY DEPTNO;
```

MAX(AVG(SAL))
2916.66667

CHAPTER 2

JOINS

Sometimes we need to use data from more than one table. For example, we need EMPNO from EMP table, DEPTNO from DEPT table and LOC from DEPT table, here two tables are involved EMP and DEPT. To produce this type of reports we need to use the concept called Joins.

The joints available in ORACLE 9i are as follows:

- ✓ Equi Join
- ✓ Non-Equi Join
- ✓ Left Outer Join
- ✓ Right Outer Join
- ✓ Full Outer Join
- ✓ Join On
- ✓ Self Join
- ✓ Cross Join
- ✓ Natural Join
- ✓ Using Clause
- ✓ Inner Join
- ✓ Three way join

CARTESIAN PRODUCT

When all rows of one table is joined with all the rows of another table, it is known as Cartesian product of the two tables.

Query 47: Query to find the Cartesian product of EMP table and DEPT table.

```
SELECT ENAME, LOC
FROM EMP, DEPT;
```

ENAME	DEPTNO	DNAME
SMITH	20	ACCOUNTING
ALLEN	30	ACCOUNTING
WARD	30	ACCOUNTING
JONES	20	ACCOUNTING

GENERAL SYNTAX OF JOINING TABLES

```
SELECT table1 .column[s], table2.column[s],...
FROM table1 [alias1], table2 [alias2]
WHERE table1.column1=table2.column2
```

Notes:

1. We can use the alias given to the table n use the alias wherever required.
2. The WHERE clause is used for the joining condition.
3. If the same column name appears in more than one table then the column name is to

be prefixed with the table name.

EQUIJOIN

To determine an employee's department name, we compare the values in the DEPTNO column of EMP table with the DEPTNO column in the DEPT table. The relationship between the EMP and DEPT table is an EQUIJOIN, that is, the value in the DEPTNO column on both tables must be equal. This type of join usually involves primary and foreign key complements.

Query 48:

```
SELECT EMP.EMPNO, EMP.ENAME, EMP.DEPTNO, DEPT.DEPTNO
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO;
```

EMPNO	ENAME	DEPTNO	DEPTNO
7369	SMITH	20	20
7499	ALLEN	30	30
7521	WARD	30	30
7566	JONES	20	20

NONEQUIJOIN

A nonequijoin is a join that condition containing something other than an equality operator.

For example, to determine the relation between an employee's salary, and the salary grade, we compare the values in the SAL column of EMP table with the HISAL and LOSAL column in the SALGRADE table. The relationship between the EMP and SALGRADE table is an NONEQUIJOIN. This type of join usually involves a strong entity and a weak entity and is obtained using an operator other than equals.

Query 49:

```
SELECT E.EMPNO, E.SAL, S.GRADE
FROM EMP E, SALGRADE S WHERE
E.SAL BETWEEN S.LOSAL AND
S.HISAL;
```

EMPNO	SAL	GRADE
7369	800	1
7521	1250	2
7844	1500	3
7902	3000	4
7839	5000	5

SELF JOINS

Sometimes we need to join a table to itself, this is when we SELF JOIN.

For example, to find the name of each employee's manager, we need to join EMP table to itself and perform self join.

Query 50:

```
SELECT W.EMPNO, M.EMPNO
FROM EMP W, EMP M
WHERE W.EMPNO=M.EMPNO;
```

EMPNO	EMPNO
7369	7369
7499	7499
7521	7521

CROSSJOIN

Crossjoin is same as cartesian product, i.e., all rows of one table is joined with all the rows of another table.

Query 51:

```
SELECT E.ENAME, E.DEPTNO, D.DNAME
FROM EMP E CROSS JOIN DEPT D;
```

NATURAL JOIN

This type of join helps in joining the tables automatically based on column in the two tables which have matching data type and names.

Query 52:

```
SELECT E.EMPNO, E.ENAME, D.DNAME
FROM EMP E NATURAL JOIN DEPT D;
```

EMPNO	ENAME	DNAME
7782	CLARK	ACCOUNTING
7788	SCOTT	RESEARCH
7521	WARD	SALES

INNER JOIN

Inner join is same as the equi-join. It returns all the rows from different tables where there is a match.

Query 53:

```
SELECT E.EMPNO, E.DEPTNO, D.DNAME
FROM EMP E INNER JOIN DEPT D
ON E.DEPTNO=D.DEPTNO;
```

LEFT OUTER JOIN

This type of join returns all the rows in the left table even if there is no match in the right table.

Query 54:

```
SELECT E.EMPNO, E.ENAME, D.DNAME
FROM EMP E LEFT OUTER JOIN DEPT D
ON E.DEPTNO=D.DEPTNO;
```

This query returns all the rows of EMP table even if there is no match in the DEPT table.

EMPNO	ENAME	DNAME
7782	CLARK	ACCOUNTING
7788	SCOTT	RESEARCH
7521	WARD	SALES

RIGHT OUTER JOIN

This type of join returns all the rows in the right table even if there is no match in the left table.

Query 55:

```
SELECT E.EMPNO, E.ENAME, D.DNAME
FROM EMP E RIGHT OUTER JOIN DEPT D
ON E.DEPTNO=D.DEPTNO;
```

This query returns all the rows of DEPT table even if there is no match in the EMP table.

EMPNO	ENAME	DNAME
7782	CLARK	ACCOUNTING
7788	SCOTT	RESEARCH
7521	WARD	SALES
		OPERATIONS

FULL OUTER JOIN

This type of join returns all the rows in the left table even if there is no match in the right table and all the rows in the right table even if there is no match in the left table.

Query 56:

```
SELECT E.EMPNO, E.ENAME, D.DNAME
FROM EMP E FULL OUTER JOIN DEPT D
ON E.DEPTNO=D.DEPTNO;
```

This query returns all the rows of EMP table even if there is no match in the DEPT table and all the rows of DEPT table even if there is no match in the EMP table.

EMPNO	ENAME	DNAME
7782	CLARK	ACCOUNTING
7788	SCOTT	RESEARCH
7521	WARD	SALES
		OPERATIONS

JOIN ON

Join On is same as the equi-join. It returns all the rows from different tables where there is a match.

Query 57:

```
SELECT E.EMPNO, E.DEPTNO, D.DNAME
FROM EMP E JOIN DEPT D
ON E.DEPTNO=D.DEPTNO;
```

USING CLAUSE

Using clause is used in the place of where or on clause. If we use this clause we need to only mention the column with has the match. The column name must be the same.

Query 58:

```
SELECT E.EMPNO, E.DEPTNO, D.DNAME
FROM EMP E JOIN DEPT D
USING DEPTNO;
```

THREE WAY JOIN

If we want to join more than two tables we use the concept of three way join.

Query 59:

```
SELECT E.EMPNO, E.DEPTNO, D.DNAME, E.SAL, S.GRADE FROM
EMP E JOIN DEPT D
ON E.EMPNO=D.DEPTNO JOIN
SALGRADE S
ON E.SAL BETWEEN S.LOSAL AND S.HISAL;
```

SUBQUERIES

A sub-query is a SELECT statement that is embedded in a clause of another SELECT statement. They can be useful when we need to select rows from a table with a condition that depends on the data in the table itself.

We can place the sub-query in the following clauses:

- ✓ The WHERE clause
- ✓ The HAVING clause
- ✓ The FROM clause

Query 60: Query to display the ENAME for the employees getting salary more than SCOTT.

```
SELECT      ENAME
FROM EMP
WHERE SAL > (SELECT SAL FROM EMP
              WHERE NAME='SCOTT');
```

ENAME

KING

TYPES OF SUBQUERIES

1. Single row subqueries:
 - ✓ Sub query which returns only one row.
 - ✓ The operators which can be used are: <, >, =, <=, >=, <>
2. Multiple row subqueries:
 - ✓ Subquery which return more than one row
 - ✓ The operators which can be used are:
 - a. IN- Equal to any member in the list.

- b. ANY- Compare value to each value returned by the sub-query.
- c. ALL- Compare value to every value returned by the sub-query.

Query 61: Query to display to ENAME of the employee earning SAL same as the minimum SAL of each DEPTNO.

```
SELECT ENAME
FROM EMP
WHERE SAL IN (SELECT MIN (SAL)
              FROM EMP
              GROUP BY DEPTNO);
```

ENAME
SMITH
JAMES
MILLER

Query 62: Query to display to ENAME of the employee earning SAL less the SAL given by DEPTNO 10.

```
SELECT ENAME
FROM EMP
WHERE SAL < (SELECT SAL FROM EMP
             WHERE DEPTNO=10);
```

Query 63: Query to display to ENAME of the employee earning SAL more the minimum SAL given by DEPTNO 20.

```
SELECT ENAME
FROM EMP
WHERE SAL > (SELECT SAL
             FROM EMP
             WHERE DEPTNO=20);
```

Note:

- ✓ = ANY (...) is equivalent to IN.
- ✓ < ANY (...) means less than the maximum.
- ✓ > ANY (...) means more than the minimum.
- ✓ < ALL (...) means less than the maximum.
- ✓ > ALL (...) means more than the minimum.

DATA MANIPULATION LANGUAGE

When we want to add, update or delete data in the database, we execute data manipulation language.

The statements available in DML are:

- ✓ Insert
- ✓ Delete
- ✓ Update

INSERT STATEMENT

This statement is used to insert new rows in the existing table. Syntax:

```
INSERT INTO table_name [ (column1 [, columns]) ]  
VALUES (value1 [, values]);
```

Query 64: Query to insert a new department in the DEPT table.

```
INSERT INTO DEPT  
VALUES (50, 'SALES', 'NEW DELHI');
```

Query 65: Query to insert a new DEPTNO and DNAME only in DEPT table.

```
INSERT INTO DEPT(DEPTNO, DNAME)  
VALUES (50,'SALES');
```

Note: While inserting a data in a table, different constraints are to be kept in the mind and then inserted.

Usage of NULL in INSERT statement:

We can give a null value during insertion of data in a table. It can be done by using the keyword null or just by not giving any values.

```
Query 66: Query to insert a new department in DEPT table, with no DNAME.  
  
INSERT INTO DEPT  
VALUES (60,null , 'BANGALORE');
```

Usage of DEFAULT in INSERT statement:

We can give the default value, during insertion of data in a table. A default value is a value given while creating a table. It can be done by using the keyword default or just by not giving any values.

Query 67: Query to insert a new department in DEPT table, where the LOC value is the default value.

```
INSERT INTO DEPT
```

```
VALUES (50,'SALES' ,default);
```

Usage of FUNCTION in INSERT statement:

We can function during insertion of data in a table.

Query 68: Query to insert a new employee in EMP table.

```
INSERT INTO EMP(EMPNO,HIREDATE  
VALUES (1001,SYSDATE);
```

Usage of & in INSERT statement:

& symbol is used when we want to input values at run time.

Query 69: Query to insert a new department in DEPT table. Accept the values at run time.

```
INSERT INTO DEPT  
VALUES (&DEPTNO,'&DNAME' ,&LOC);
```

Inserting rows from another table:

If we want to copy the data from one table to the other table we use DML statements in DML statements to do this.

Query 70: Query to copy the details of EMP table to EMPCLERK table where JOB is CLECK.

```
INSERT INTO EMPCLERK  
SELECT EMPNO, ENAME, SAL  
FROM EMP  
WHERE JOB='CLERK';
```

UPDATE STATEMENT

This statement is used to update the existing rows in the existing table. Syntax:

```
UPDATE table_name  
SEL column=values [, column(s)=value(s) ]  
WHERE condition;
```

Query 71: Query to update the DEPT table by changing to LOC to BANGLORE of DEPTNO 10.

```
UPDATE DEPT  
SEL LOC='BANGALORE'  
WHERE DEPTNO=10;
```

Note: While updating a value in a table, different constraints are to be kept in the mind and then

updated. Example: if we want to change the DEPTNO of an employee to 55 in the EMP table and the DEPTNO 55 do not exist in DEPT table then an ERROR would appear which is the integrity constraint i.e.: DEPTNO in EMP table is a foreign key referring to primary key DEPTNO of DEPT table and the DEPTNO does not exist.

Update columns with sub-query

Query 72: Query to change the SAL of EMPNO 7788 to the SAL to SCOTT.

```
UPDATE EMP
```

```
SEL SAL=(SELECT SAL FROM EMP WHERE ENAME=SCOTT)
```

```
WHERE EMPN=7788;
```

In this query the salary of employee number 7788 will change and his salary will be equal to the salary of Scott.

Update rows based on another table

Query 73: Query to change the DEPTNO of all the employees in EMPCLERK table to the DEPTNO of SCOTT of EMP table.

```
UPDATE EMPCLERK
```

```
SEL DEPTNO=(SELECT DEPTNO FROM EMP WHERE ENAME=SCOTT)
```

In this query the department number of all the employees in the EMPCLERK will change to the department number in which Scott works.

DELETE STATEMENT

This statement is used to delete the existing rows in the existing table. Syntax:

```
DELETE [FROM] table_name
```

```
WHERE condition;
```

Query 74: Query to delete the employee taking SAL less than 500.

```
DELETE      EMP
```

```
WHERE SAL<500;
```

Note: While deleting a value in a table, different constraints are to be kept in the mind and then updated. Example: if we want to delete the department from the DEPT table and employees work in that department then an ERROR would appear which is the integrity constraint i.e.: DEPTNO in EMP table is a foreign key referring to primary key DEPTNO of DEPT table. So until the employees

of EMP table are deleted the department of DEPT table cannot be deleted. In other words a parent table cannot be deleted until a child table is deleted.

TRANSACTION CONTROL

To manage the changes made by data manipulation language we use transaction control statements.

A database transaction begins with a DML statement and ends with transaction control statements.

The transaction control statements are as follows:

- ✓ COMMIT
- ✓ ROLLBACK
- ✓ SAVEPOINT

COMMIT

It helps in ending the transaction by making all pending data changes permanent. Syntax:

COMMIT;

State of data after commit:

- ✓ Data changes are made permanent in the database.
- ✓ The previous state of the data is completely lost.
- ✓ All users can view the result.
- ✓ Locks on the affected rows are released; those rows are available for other users to manipulate.
- ✓ All savepoints are erased.

SAVEPOINT

It marks a save-point within the current transaction. It divides the transaction into smaller sections.

Syntax:

SAVEPOINT savepoint_name;

ROLLBACK

It helps in discarding all the pending changes. Syntax:

ROLLBACK;

State of data after rollback:

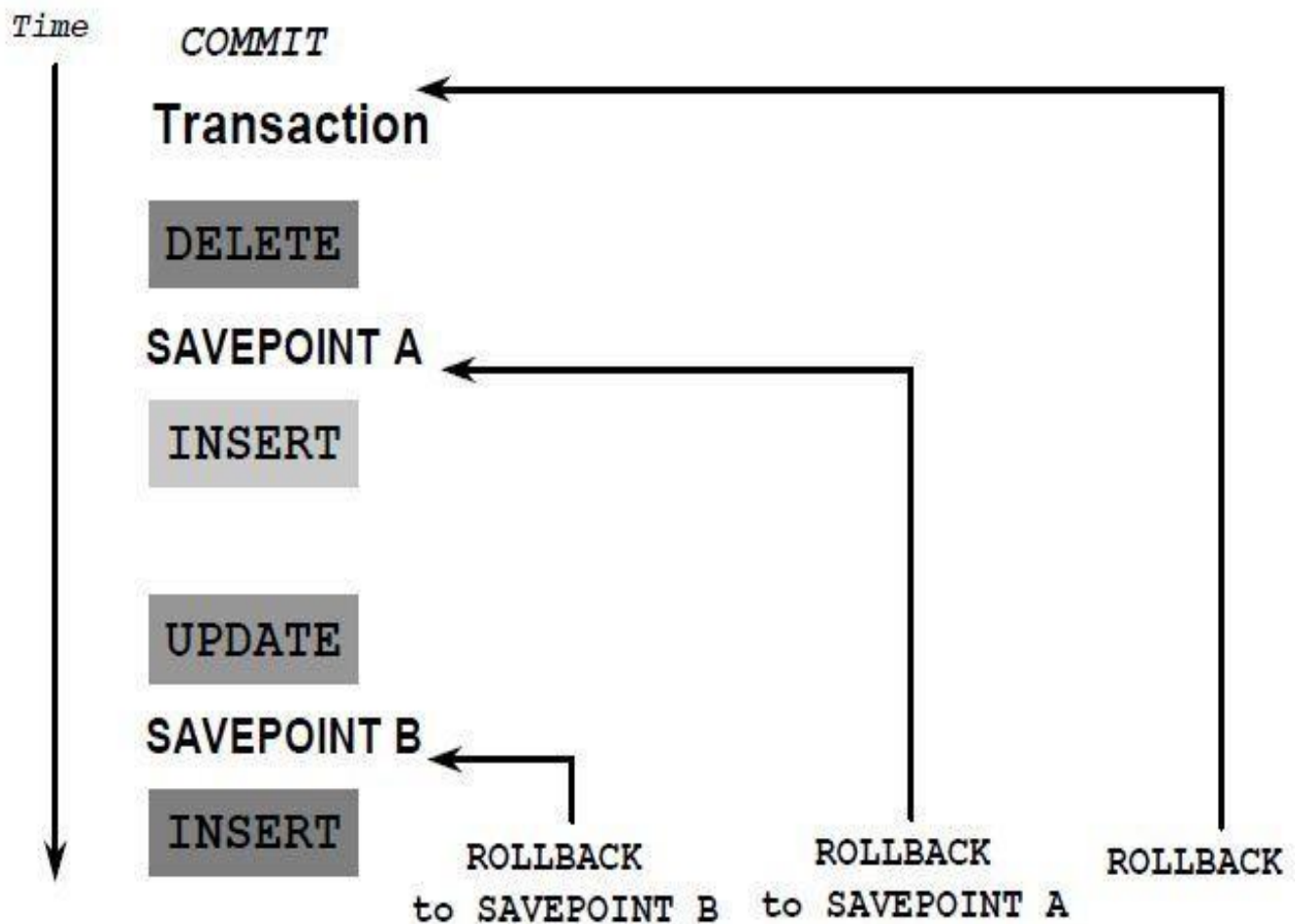
- ✓ Data changes are undone.
- ✓ The previous state of the data is completely restored.

- ✓ Locks on the affected rows are released; those rows are available for other users to manipulate.

If we want to discard the pending changes up the marker, we can do this by using ROLLBACK TO SAVEPOINT statement.

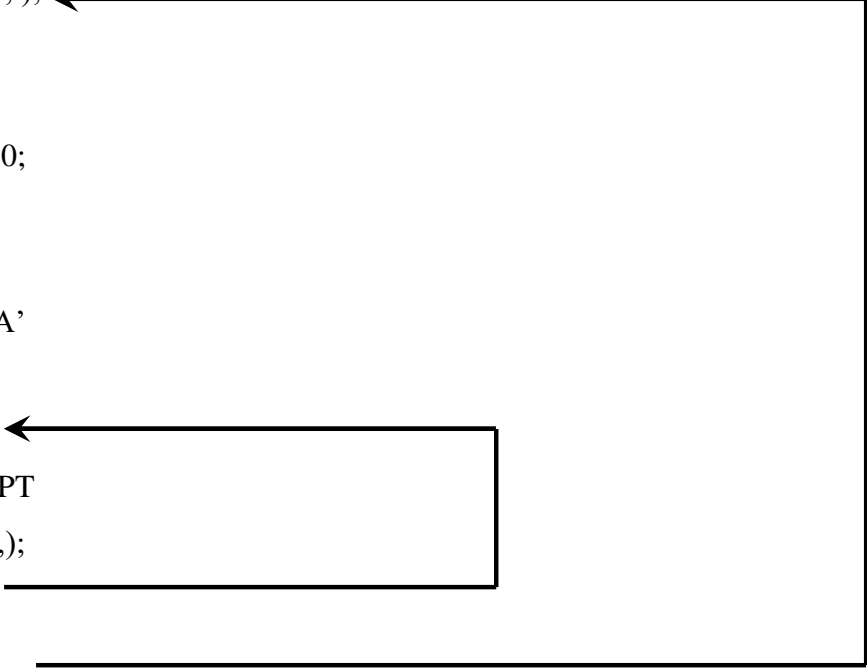
Syntax:

ROLLBACK to savepoint_name



Example:

```
INSERT INTO DEPT
VALUES (50, 'SALES', );
COMMIT;
DELETE DEPT
WHERE DEPTNO=50;
SAVEPOINT A;
UPDATE DEPT
SET DNAME='BCA'
WHERE DEPTNO=10;
SAVEPOINT B;
INSERT INTO DEPT
VALUES (50, 'SALES', );
ROLLBACK TO B;
ROLLBACK;
```



The diagram illustrates the execution flow of the SQL statements. It features two horizontal arrows pointing left. The first arrow originates from the right side of the 'VALUES (50, 'SALES',);' statement and points to the 'COMMIT;' statement. The second arrow originates from the right side of the 'VALUES (50, 'SALES',);' statement and points to the 'ROLLBACK TO B;' statement. Additionally, a vertical line segment connects the right side of the 'VALUES (50, 'SALES',);' statement to the right side of the 'ROLLBACK;' statement, indicating the final state of the transaction.

DATA DEFINATION LANGUAGE

Data Definition Language deals with the setup, change and remove data structure from tables.

DATATYPES IN ORACLE

- | | | |
|------------------|---------|-----|
| • NUMBER(P,S) | • LONG | |
| • DATE | • CLOB | |
| • VARCHAR(size) | • RAW | and |
| • VARCHAR2(size) | LONGRAW | |
| • CHAR(size) | • BLOB | |
| • TIMESTAMP | • BFILE | |
| | • ROWID | |

DATA DEFINITION LANGUAGE

Syntax:

```
CREATE TABLE [schema.]table_name
            (column datatype [ DEFAULT expr ] [...]);
```

Query 75: Query to create a table EMPDUP with column as EMPNO, ENAME.

```
CREATE TABLE EMPDUP
            (EMPNO      NUMBER(10,
            ENAME VARCHAR2(20) );
```

Create a table using Sub-Query:

When we want to create a table with the same structure of the other table we use sub-query.

Syntax:

```
CREATE TABLE [schema.]table_name
            [ (column datatype [ DEFAULT expr ] [...]) ]
AS SUB-QUERY
```

Query 76: Query to create a table EMPDUP with column and data as EMPNO, ENAME taken from EMP table.

```
CREATE TABLE EMPDUP
AS SELECT EMPNO, ENAME FROM EMP;
```

Query 77: Query to create a table EMPDUP with column as EMPNO, ENAME taken from EMP table and no data present.

```
CREATE TABLE EMPDUP
AS SELECT EMPNO, ENAME
FROM EMP
WHERE ROWID IS NULL;
```

ALTER A TABLE

We use ALTER TABLE statement to add a column, modify an existing column, define a default value for the new column, and to drop a column.

Add a column in the existing table:

Syntax:

```
ALTER TABLE table_name
ADD (column datatype [ DEFAULT expr ] [...]);
```

Query 78: Query to add a column DOB to the EMP table.

```
ALTER TABLE EMP
ADD (DOB DATE);
```

Modify a column in the existing table:

Syntax:

```
ALTER TABLE table_name
MODIFY (column datatype [ DEFAULT expr ] [...]);
```

Query 79: Query to change the data type of MGR to VARCHAR (20).

```
ALTER TABLE EMP
```

MODIFY (MGR VARCHAR(20));

Drop a column in the existing table:

Syntax:

```
ALTER TABLE table_name DROP  
COLUMN column_name;
```

Query 80: Query to delete column DOB from EMP.

```
ALTER TABLE EMP  
DROP COLUMN DOB;
```

DROP A TABLE

We use DROP TABLE statement to delete a table. Syntax:

```
DROP TABLE table_name;
```

Query 81: Query to delete the table EMPDUP.

```
DROP TABLE EMPDUP;
```

RENAME A TABLE

We use DROP TABLE statement to delete a table. Syntax:

```
RENAME old_table_name TO new_table_name;
```

Query 82: Query to rename the table EMP to EMPLOYEE.

```
RENAME EMP TO EMPLOYEE;
```


DATA CONSTRAINTS

Constraints help to prevent invalid data entry into the table. The constraints are as follows:

- ✓ NOT NULL: ensures the column that cannot contain null.
- ✓ UNIQUE: ensures the column that values must be unique i.e. no duplicate.
- ✓ PRIMARY KEY: ensures the column which is not null and unique.
- ✓ FOREIGN KEY: establishes and enforces a foreign key relationship between the column and a column of the reference table.
- ✓ CHECK: Specifies a condition that must be true.

Defining Constraints:

- ✓ Column level
 COLUMN [CONSTRAINT constraint-name] constraint-type;
- ✓ Table level
 Column,...

[CONSTRAINT constraint-name] constraint-type (column, ...), Note: Foreign key constraint can be only defined at table level.

Query 83: Query to create a table EMP1 with following columns and constraints:

EMPNO	Primary Key
ENAME	Not Null
EMAIL	Unique
DEPTNO	Foreign Key Referred To DEPT Table
DOB	Check for age more than 18

Column level:

```
CREATE TABLE EMP1
(
EMPNO NUMBER(10) CONSTRAINT empno_pk PRIMARY KEY, ENAME
VARCHAR2(20) CONSTRAINT  ename_nn NOT NULL, EMAIL
VARCHAR2(30) CONSTRAINT email_u UNIQUE, DEPTNO NUMBER(10),
DOB DATE CONSTRAINT dob_c CHECK ( (SYSDATE-DOB/365)>18),
```

```
CONSTRAINT deptno_fk FOREIGN KEY (DEPTNO)
REFERENCES DEPT(DEPTNO) );
```

Table level:

```
CREATE TABLE EMP1
(EMPNO    NUMBER(10),
ENAME    VARCHAR2(20),
EMAIL    VARCHAR2(30),
DEPTNO   NUMBER(10),
DOB DATE CONSTRAINT dob_c CHECK ( (SYSDATE-DOB/365)>18),
CONSTRAINT empno_pk PRIMARY KEY (EMPNO),
CONSTRAINT  ename_nn  NOT  NULL(ENAME),
CONSTRAINT      email_u      UNIQUE(EMAIL),
CONSTRAINT deptno_fk FOREIGN KEY (DEPTNO)
REFERENCES DEPT (DEPTNO));
```

VIEWS

Views are logically represents subset of data from one or more tables. It contains no data of its own but is a window through which data from a table can be viewed or changed. The table on which view is based is called base tables.

Uses of views:

- ✓ To restrict data access
- ✓ To make complex queries easy
- ✓ To provide data independence
- ✓ To present different views of the same data

Types of views:

- ✓ Simple View
 - 1.Number of table: One
 - 2.Contains function: No
 - 3.Contains group function: No
 - 4.DML operation through view: Yes
- ✓ Complex View:
 1. Number of table: One more
 2. Contains function: Yes
 3. Contains group function: Yes
 4. DML operation through view: Not Always

CREATING VIEW

Syntax:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view [(alias [, alias,])]  
AS sub-query  
[WITH CHECK OPTION [CONSTRAINT constraint]]  
[WITH READ ONLY [CONSTRAINT constraint]];
```

In the syntax:

OR REPLACE: re-creates the view if already exists

FORCE: creates the view regardless of whether or not base table exists. UNFORCE: creates the view only if the base case exist. (This is by default) Sub-query: is a complete select statement. (It can contain alias.)

WITH CHECK OPTION: specifies that only rows accessible to the view can be inserted or updated

WITH READ ONLY: ensures no DML operations are been performed.

Query 85: Query to create a view EMP10 with ENAME, EMPNO and SAL for each employee in DEPTNO 10

```
CREATE VIEW EMP10  
AS  
SELECT EMPNO, ENAME, SAL  
FROM EMP  
WHERE DEPTNO=10;
```

RETRIVING DATA FROM A VIEW

For retrieving data from a view we can use DRL statements. Query 86:

Query to display the view EMP10

```
SELECT * FROM EMP10;
```

MODIFY A VIEW

For making changes in a view we use REPLACE keyword.

Query 87: Query to modify a view EMP10 and add column JOB for each employee in DEPTNO 10.

Make the output look like: 7788SCOTT WORKS AS ANALYST2000

```
CREATE OR REPLACE VIEW EMP10
AS
SELECT EMPNO SSN, ENAME || ' WORKS AS ' || JOB, SAL SALARY FROM
EMP
WHERE DEPTNO=10;
```

CREATING A COMPLEX VIEW

Query 88: Query to create a view EMPDETAILS with DNAME and minimum, maximum and average SAL of each department.

```
CREATE OR REPLACE VIEW EMPDETAILS
AS
SELECT D.DEPTNO, D.DNAME, MIN (E.SAL), MAX (E.SAL), AVG (E.SAL)
FROM EMP E, DEPT D
WHERE E.DEPTNO=D.DEPTNO
GROUP BY D.DEPTNO;
```

USING DML OPERATIONS ON VIEW

We can perform DML operation only on simple views.

We cannot remove a row from a view if the view contains the following:

- ✓ Group function
- ✓ GROUP BY clause
- ✓ DISTINCT keyword
- ✓ Pseudo column ROWNUM keyword

We cannot modify data in a view if the view contains the following:

- ✓ Group function

- ✓ GROUP BY clause
- ✓ DISTINCT keyword

- ✓ Pseudo column ROWNUM keyword
- ✓ Column defined by expression

We cannot add data through a view if the view contains the following:

- ✓ Group function
- ✓ GROUP BY clause
- ✓ DISTINCT keyword
- ✓ Pseudo column ROWNUM keyword
- ✓ Column defined by expression
- ✓ NOT NULL column in the base tables that are not selected by the view

USING THE WITH CHECK CLAUSE

We can ensure that the DML operation performed on the view stay with the domain of the view by using the WITH CHECK OPTION clause.

Query 89: Query to create a view EMP10 with ENAME, EMPNO and SAL for each employee in DEPTNO 10. And also check that no employee other than that of DEPTNO 10 is inserted to the table

```
CREATE OR REPLACE VIEW
EMP10 AS
SELECT      EMPNO,
ENAME, SAL FROM EMP
WHERE DEPTNO=10
WITH CHECK OPTION CONSTRAINT emp10_ck;
```

According to this query, any attempt to change the DEPTNO from any row in the view fails because it violates the WITH CHECK OPTION constraint.

DENYING DML OPERATION ON VIEW

To deny the DML operations on a view we use READ ONLY constraint.

Query: Query to modify a view EMP10 and add column JOB for each employee in DEPTNO 10. Make the output look like: 7788SCOTT WORKS AS ANALYST2000

Query 90: Query to create a view EMP10 with ENAME, EMPNO and SAL for each employee in DEPTNO 10. The view cannot be changed.

```
CREATE OR REPLACE VIEW
```

```
EMP10 AS
SELECT      EMPNO,
ENAME, SAL FROM EMP
WHERE
DEPTNO=10
WITH      READ
ONLY;
```

REMOVING A VIEW

We can remove the view without removing the data because a view is based on underlying tables in a database.

Syntax:

```
DROP VIEW view;
```

Query 91: Query to delete view EMP10.

```
DROP VIEW EMP10;
```

INDEX

An index is used by the Oracle Server to speed up the retrieval of rows by a pointer. It can reduce disk I/O by using a rapid path access method to locate data quickly. It is independent of tables.

Types of index

- ✓ Implicit Index: A unique index is created automatically when we define a PRIMARY KEY or UNIQUE constraint in a table definition. (Automatically)
- ✓ Explicit Index: Users can create nonunique indexes on columns to speed up access to the rows. (Manually)

Syntax of index:

```
CREATE INDEX index
ON TABLE (column [, column]);
```

Query 92: Query to create an index with ENAME from EMP. SELECT INDEX INDEX_ENAME
ON TABLE EMP (ENAME);

Removing an Index:

Syntax:

```
DROP INDEX index;
```

```
Query 93: Query to delete the index  
INDEX_ENAME.      DROP      INDEX  
INDEX_ENAME;
```

DATA CONTROL LANGUAGE

Data Control Language gives and removes access right to both oracle database and structure within it.

CREATING USER

Syntax:

```
CREATE      USER  
username IDENTIFIED  
BY password;
```

Query 94: Query to create a new user BCA4.

```
CREATE USER BCA4  
IDENTIFIED      BY  
JGI12345;
```

GRANT

Once the user is created, the DBA can grant specific system privileges to a user. Those privileges are:

1. Create Table
2. Create View
3. Create Sequences
4. Create Index
5. Create Synonym

Syntax:

```
GRANT  privilege  [,  
privilege] TO username;
```

Query 95: Query to grant the user BCA4 to create view and table.

```
GRANT CREATE VIEW, CREATE
TABLE TO BCA4;
```

Change The password:

```
ALTER USER username
IDENTIFIED BY
new_password;
```

Query 96:

```
ALTER USER
BCA4
IDENTIFIED
BY JGI;
```

OBJECT PRIVILEGES

OBJECT PRIVILEGE	TABLE	VIEW	SEQUENCE	PROCEDURE
ALTER	YES	NO	YES	NO
DELETE	YES	YES	NO	NO
EXECUTE	NO	NO	NO	YES
INDEX	YES	NO	NO	NO
INSERT	YES	YES	NO	NO
REFERENCES	YES	YES	NO	NO
SELECT	YES	YES	YES	NO
UPDATE	YES	YES	NO	NO

The DBA has all the privilege on all the objects. But the users can get only specific privileges (mentioned in the table) which are given by the DBA.

Syntax:

```
GRANT privilege [,
privilege] ON table_name
TO user_name;
[WITH GRANT OPTION]
```

A privilege that is granted with the WITH GRANT OPTION clause can be passed on to other users

and roles by grantee. An owner of the table can grant access to all the users using PUBLIC keyword.

Query 97: Query to grant the update, insert, delete, and select privileges to BCA4 user. And also BCA4 can grant these privileges to other users.

```
GRANT UPDATE, SELECT, INSERT,  
DELETE ON DEPT  
TO BCA4  
WITH GRANT OPTION;
```

Query 98: Query to grant the update, insert, delete, and select privileges on SCOTT's DEPT table to all the users.

```
GRANT UPDATE, SELECT, INSERT,  
DELETE ON SCOTT.DEPT  
TO PUBLIC;
```

REVOKE

To remove privileges granted to the users, we use REVOKE statement. When we use the REVOKE statement, the privileges that we specify are revoked from the users we name and from any users to whom those privileges were granted through WITH GRANT OPTION clause.

Syntax:

```
REVOKE {privilege [, privilege...] |  
ALL} ON object  
FROM {user [, user] | role|  
PUBLIC} [CASCADE  
CONSTRAINTS];
```

Query 99: Query to revoke the insert and delete privileges from BCA4 on EMP table.

```
REVOKE INSERT,  
DELETE ON EMP  
FROM BCA4;
```

CHAPTER -3

TRIGGERS

Triggers are stored programs that are fired automatically when some events occur. The code to be fired can be defined as per the requirement. Oracle has also provided the facility to mention the event upon which the trigger needs to be fire and the timing of the execution.

Types of Triggers:

- Triggers can be classified based on the following parameters.
- Classification based on the timing
 - BEFORE Trigger: It fires before the specified event has occurred.
 - AFTER Trigger: It fires after the specified event has occurred.
- Classification based on the level
 - STATEMENT level Trigger: It fires one time for the specified event statement.
 - ROW level Trigger: It fires for each record that got affected in the specified event.
(only for DML)
- Classification based on the Event
 - DML Trigger: It fires when the DML event is specified
(INSERT/UPDATE/DELETE)
 - DDL Trigger: It fires when the DDL event is specified (CREATE/ALTER)
 - DATABASE Trigger: It fires when the database event is specified
(LOGON/LOGOFF/STARTUP/SHUTDOWN)
- So each trigger is the combination of above parameters.

Syntax of Trigger:

Syntax:

```
CREATE [ OR REPLACE ] TRIGGER <trigger_name>
```

```
[ BEFORE | AFTER | INSTEAD OF ]
```

Trigger Timing

```
[ INSERT | UPDATE | DELETE.....]
```

Event

```
ON <name of underlying object>
```

```
[ FOR EACH ROW ]
```

Row Level

```
[ WHEN <condition for trigger to get execute> ]
```

Conditional Clause

```
DECLARE
```

```
<Declaration part>
```

```
BEGIN
```

```
<Execution part>
```

```
EXCEPTION
```

```
<Exception handling part>
```

```
END;
```

Example on Trigger: Create a trigger to stop employee perform DML operations on

```
Create or replace trigger trig_error
```

```
before insert or update or delete
```

```
on emp
```

```
begin
```

```
if to_char(sysdate,'DY') IN ('SAT','SUN') then
```

```
raise_application_error(-20111,'No changes can be made on weekday');
```

```
dbms_output.put_line('Cant insert/delete on Tuesday');
```

```
end if;
```

```
end;
```

Example 2: Create trigger to insert a record to dept_log after inserting to dept table.

```
Create or replace trigger depttrig
```

```
AFTER INSERT ON dept
```

```
FOR EACH ROW
```

```
BEGIN
```

```
INSERT INTO dept_log
```

```
VALUES(:new.deptno,:new.dname,:new.loc);
```

```
END;
```

Example-3: Create a trigger to insert a record to log table after performing update and delete on dept table.

```
Create or replace trigger dept_trig
BEFORE update or delete on dept
FOR EACH ROW
BEGIN
INSERT INTO dept_log
VALUES(:old.deptno,:old.dname,:old.loc);
END;
```

Example 4: Create a trigger to count number employee after deleting record from a emp tatble

```
CREATE OR REPLACE TRIGGER emp_count
AFTER DELETE ON EMP
DECLARE
    n INTEGER;
BEGIN
    SELECT COUNT(*) INTO n FROM emp;
    DBMS_OUTPUT.PUT_LINE(' There are now ' || n || ' employees. ');
END;
```

Example 5: Create a trigger to convert ename to upper

```
CREATE OR REPLACE TRIGGER before_emp
BEFORE UPDATE OR INSERT ON emp
FOR EACH ROW
begin
/* convert character values to upper case */
:new.ename := upper( :new.ename );
:new.job := upper( :new.job);
end;
```

Dictionary for triggers is user_triggers

```
Select trigger_name,trigger_body,table_name,description
from user_triggers
```

Enabling/Diasabling Triggers

To enable a disabled trigger, use the

Syntax:

ALTER TRIGGER statement with the ENABLE clause.

For example, to enable the disabled trigger:

Example:

```
ALTER TRIGGER trig_error ENABLE;
```

To enable all triggers defined for a specific table, use the ALTER TABLE statement with the ENABLE clause and the ALL TRIGGERS option.

```
ALTER TABLE emp ENABLE ALL TRIGGERS;
```

Disabling Triggers

- You might temporarily disable a trigger if:
- You must perform a large data load, and you want it to proceed quickly without firing triggers.

```
ALTER TRIGGER trig_error DISABLE;
```

- To disable all triggers defined for a specific table, use the ALTER TABLE statement with the DISABLE clause and the ALL TRIGGERS option. For example, to disable all triggers defined for the Inventory table, enter the following statement:

```
ALTER TABLE emp DISABLE ALL TRIGGERS;
```

Advance Features:

Database security: It refers to the collective measures used to protect and secure a database or database management software from illegitimate use and malicious threats and attacks. It is a broad term that includes a multitude of processes, tools and methodologies that ensure security within database environment.

Advanced SQL features – (CASE, DECODE, Rollup, Cube) done in SQL

Embedded SQL– Embedded SQL is a method of combining the computing power of a programming language and the database manipulation capabilities of SQL. Embedded SQL statements are SQL statements written inline with the program source code, of the host language.

Need of Embed SQL:

Because the host language cannot parse **SQL**, the inserted **SQL** is parsed by an **embedded SQL** preprocessor. **Embedded SQL** is a robust and convenient method of combining the computing power of a programming language with **SQL's** specialized data management and manipulation capabilities

Dynamic SQL:

Dynamic SQL is a programming methodology for generating and running statements at run-time. It is mainly used to write the general-purpose and flexible programs where the SQL statements will be created and executed at run-time based on the requirement.

Embedded SQL are SQL statements in an application that do not change at run time and, therefore, can be hard-coded into the application. Dynamic SQL is SQL statements that are constructed at run time; for example, the application may allow users to enter their own queries.

Introduction to Distributed Databases:

In a distributed database, there are a number of databases that may be geographically distributed all over the world. A distributed DBMS manages the distributed database in a manner so that it appears as one single database to users.

The two types of distributed systems are as follows:

1. Homogeneous distributed databases system: Homogeneous distributed database system is a network of two or more databases (With same type of DBMS software) which can be stored on one or more machines. ...
2. Heterogeneous distributed database system.

Example: Oracle distributed database systems employ a distributed processing architecture to function. For example, an Oracle server acts as a client when it requests data that another Oracle server manages.

Client/Server Databases:

A **client/server application** is a piece of software that runs on a **client** computer and makes requests to a remote **server**. Many such **applications** are written in high-level visual programming languages where UI, forms, and most business logic reside in the **client application**.

