5.1 INTRODUCTION TO SQL

SQL (Structured Query Language) :

DBMS requires a language to access the data.

Structured Query Language (SQL) is the language used by relational database system (RDBMS)

- It is developed by IBM company in early 1970's
- Initially it is named as System R and later it is named as SQL.

5.1.1 FEATURES OF SQL

1. SQL can be used by a range of users, including those with little or no programming experience.
2. It is a non procedural language.
3. It reduces the amount of time required for creating and maintaining systems.
4. It is an English-Like Language.
5. It is used to access, create and retrieve data from database.
6. It is used to execute queries.
7. It is used to insert, update and delete records from database.

5.1.2 BENEFITS OF SQL

1. SQL is an english like language
2. It is easy to learn.
3. SQL is non procedural language.

   i.e., SQL specifices only what is required and not how it should be done.
4. More productivity

   i.e., with a single statement we can perform multiple work/task like creation of tables etc.
5. With SQL statements we can retrive multiple rows at a time and even modify multiple rows at a time.
6. It is more powerful then COBOL.
7. All RDBMS supports SQL.
8. It can be used by specialist and non specialist person.

Structured Query Language (SQL) is a language that provides an interface to relational database systems. SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, and Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Why SQL ?

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

In common usage **SQL also encompasses :**

- DML (Data Manipulation Languages) for INSERTs, UPDATEs, DELETEs
- DDL (Data Definition Language) used for creating and modifying tables and other database structures.

5.1.3 COMPONENTS OF SQL

There are following components of SOL.

1. DDL **(Data Definition Language) :** It is a set of SQL commands used to create, modify and delete data base structure but not data.

   For **Examples :**

   (i) **Create :** To create objects in the database.

   (ii) **Alter :** Alters the structure of the database.

   (iii) **Drop :** Delete objects from the database.

   (iv) **Truncate :** Remove all records from a table, including all spaces allocated for the records are removed.

   (v) **Comment :** Add comments to the data dictionary.

2. DML **(Data Manipulation Language) :** It is the area of SOL that allows changing data within the database.

For Examples :

(i) **Insert :** Insert data into a table.

(ii) **Update :** Updates existing data within a table.

(iii) **Delete :** Deletes all records from a table, the space for the records remain.

(iv) **Lock TableL :** Control concurrency.

3. **DCL (Data Control Language) :** It is the components of SOL statements that control access to data and to the database.

For Examples :

(i) **Commit :** Save work done

(ii) **Save Point :** Identify a point in a transaction to which you can later roll back.

(iii) **Roll Back :** Restore database to original since the last COMMIT.

(iv) **Grant/Revoke :** Grant or take back permissions to or from the oracle users.

(v) **Set Transaction :** Change transaction options like what rollback segment to use.

4. **DQL (Data Query Language) :** It is the component of SQL statement that allows getting data from the database and imposing ordering upon it.

For Example :

1. **Select :** Retrieve data from the database.

Naming conventions (Rules) :

- It should start with an alphabet.
- The name should not start with digit
- We should not use keywords as names of variables.
- No special symbols in starting position.
- No spaces in the begining, in the middle, at the end.
- The variable sixe is limited i.e., upto 30 characters.
- A variable can have alphanumeric value

Literals, Integers and Datatypes :

Literals : A literal is a numeric value or a character string used to represent itself literals are of 4 types.

1. **Numeric literals (integers/float) :** Range of integer 32, 767
2. **String literals :** These are represented by one or more legal values and must be enclosed within single coats.

   Ex : 'abc' - valid

   'xyz' - Invalid

3. **Character literals :** It consists of single character

   Eg : 'g'

4. **Logical Literals :** Logical literals are pre-determined constants

   Ex : True, False

5.1.4 NULL AND PSEUDO COLUMNS

Null : A null value is a value which is either unavailable, unassigned or inapplicable. A null value is not seen as zero, zero is a number.

Null values are handled by SQL. When a column name is defined as not null that column becomes a mandatory column and the user must enter data into it.

Pseudo Columns and SQL :

1. **Null :** A null value.
2. **Sysdate :** It returns system date.
3. **UID :** User ID (or) User Identification number it returns a unique number assigned to your session used by DBA.
4. **User :** It contains user log in name.
5. **ROWID :** It is a pointer to where row is stored and it is the complete row description.
6. **ROWNUM :** It returns the row number of the query.

5.2 DATA TYPES IN SQL

Data types comes in several forms and sizes, allowing the programmer to create tables suited to the scope of the project.

Char (Size) : This data type is used to store character strings values of fixed length. The maximum number of characters this data type can hold is 255 characters.

e.g., In case of 'Name Char(15)" then data held in the variable Name is only 15 characters in length.

Varchar (Size) Narchar2 (size) : This data type is used to store variable length alphanumeric data. The maximum type this can hold is 2000 characters.

Number (P,S) : The Number data type is used to store numbers. The precision (P), determines the maximum length of the data, whereas the scale, (S), determines the number of places to the right of the decimal. The maximum precision (P), is 38 digits.

These are of two types integer, float

Ex : int ---------- Number (3) =123

Float ---------- Number(3,2) = 123.12

Date : This data type is used to represent date and time. The standard format is DD-MM-YY as in 26-June-07.

The Date time stores date in the 24-hour format. By default, the time in a date field is 12 : 00 : 00 AM, if no time portion is specified.

Long : The LONG data type is used to store variable length character strings containing upto 2 GB.

Long data can be used to store arrays of binary data in ASCII format.

Raw/Long Raw : The RAW/LONG RAW data types is used to store binary data, such as digitized picture or image.

Raw data type can have a maximum length of 255 bytes.

Long Raw data type can contain up to 2 GB.

Rowid (rowid) : The format of the rowid is :

BBBBBBB. RRRR. FFFFF

where BBBBBBB is the block in the database file;

RRRR is the row in the block;

FFFFF is the data base file.

Boolean : Valid in PL/SQL but this data type does not exist in oracle 8 i or oracle 9i

Datatype	Description	Size
Number(p, s)	It is used to store numeric data types. p stands for precision (total number of decimal digits) and s stands for scale (total number of digits after decimal point).	Range of p is from 1 to 38. And s is from - 84 to 127.
Date	It is used to store date and time values.	Range of date is from jan 1, 47 B.C. to Dec. 31. 9999 A.D.
Char (size)	It is used to store fixed size character data.	Range of char is 1 (By default) to 2000 bytes.
Varchar 2 (size)	It is used to store variable size character data.	Range of varchar 2 is 1 (By default) to 4000 bytes.
Long	It is used to store variable size character data.	Range of long is upto 2 GB.
Clob	It is used to store variable size character data.	Range of clob is upto 4 GB.
Raw (size)	It is used to store fixed binary data.	Maximum size is upto 2000 bytes.
Long raw	It is used to store variable binary data.	Maximum size is upto 2 GB.

5.3 OPERATORS IN SQL

There are different types of operators are available in SQL,

1. **Arithmetic Operator :**

   Arithmetic operators are +, –, *, /, %, **

2. **Logical Operator :**

   Logical operator are AND, OR, NOT

3. **Comparison Operator :**

   Comparison operator are <, >, =, ≥, ≤

4. **Between Operator :**

   Between – – AND

5. **NOT Between Operator :**

   NOT between – – AND

6. **Partial Equality :**

   IN, NOT, IS, NULL, IS NOT NULL

7. **Set Operator :**

   Union, Intersect

Examples on Arithmetic Operators :

> Select 2 + 3;

2+3
5

> Select 4 + 3;

4+3
7

> Select 15 – 5;

15-5
10

> Select 5 – 2;

5-2
3

> Select 5 * 2; (or) Select 2 * 5;

5*2
10

2*5
10

> Select 6/3 (or) Select 2/2;

6/3
2

2/2
1

Logical Operators Examples :

> Select 1 && 1;

1&&1
1

> Select 1 && 0;

1&&0
0

> Select 1 && NULL;

1 & & NULL
NULL

> Select 0 && NULL;

0 & & NULL
0

> Select NULL && 0;

NULL & & 0
0

> Select 1 || 0;

1 | | 0
1

> Select 0 || 0;

0 | | 0
0

> Select 1 || NULL;

1 | | NULL
1

> Select 0 || NULL;

0 | | NULL
NULL

SQL SET Operators : The SQL SET operators are used to combine the two or more SQL SELECT statements.

Types of Set Operation :

1. Union
2. UnionAll
3. Intersect
4. Minus.

1. Union :

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax :

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

Example :

The First table

ID	Name
1	Jack
2	Harry
3	Jackson

The Second table

ID	Name
3	Jackson
4	Stephen
5	David

Union SQL query will be :

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

The resultset table will look like :

ID	Name
1	Jack
2	Harry
3	Jackson
4	Stephen
5	David

2. ***Union All :*** Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax :

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

Example :

Using the above First and Second table.

Union All query will be like:

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

The resultset table will look like :

ID	Name
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephen
5	David

3. **Intersect :**

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

Syntax :

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Example :

Using the above **First and Second table**

Intersect query will be:

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

The resultset table will look like :

ID	Name
3	Jackson

4. **Minus :**

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax :

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

Example :

Using the above First and Second table.

Minus query will be :

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

The resultset table will look like :

ID	Name
1	Jack
2	Harry

5.4 DATA DEFINITION LANGUAGE STATEMENTS IN SQL

Shema : It is actually a logical representation of a database. When we are using relational database management system we are storing data logically in the form of tables. When we are using E-R model we are storing data in the form of entities.

Examples :

Student table schema

Rollnumber	name	address

Course table schema

Cid	Cname	Cduration

We can implement the above schema using SQL data definition language (DDL) commands.

CREATE

ALTER

DROP

Create :

SQL Create Statement : It is basically used to create tables.

Syntax :

CREATE TABLE TABLENAME (column1name datatype (size), column2name datatype (size),

:
:
:
:

columnname datatype (size));

EXAMPLE - 1

Create a student table with PIN, Name, Date of birth, Age, Sex, Marks, Phone no, Address.

```
CREATE   TABLE   STUDENT
                 (PIN      Varchar(20),
                 Name      Varchar(20),
                 dob       Date,
                 age       Number(2),
                 sex       Char(1),
                 marks     Number(3),
                 phno.     Number(18),
                 address   Varchar(50));
```

EXAMPLE - 2

Create employee table having fields name, id, salary, address, phno, department.

```
> CREATE
CREATE    TABLE    STUDENT
          (NAME    Varchar(20),
           Id      Number(8),
           salary  Number(5),
           address Varchar(50),
           phno    Number(10),
           department Varchar(20),
          );
```

SQL ALTER Statement : It is used basically for adding, deleting (or) modifying columns in a existing table.

You can add a column (or) delete a column (or) modify a column in existing table.

Adding a column :

Syntax :

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example :

```
ALTER TABLE EMP
ADD deptname varchar2(10);
```

Delete a column :

Syntax :

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

Example :

```
ALTER TABLE emp
DROP COLUMN deptname;
```

Change datatype :

Syntax :

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```


Example :

```
ALTER TABLE EMP
ALTER COLUMN deptname NUMBER (2);
```

SQL DROP Statment : It is used to delete indexes, tables, database from the system.

DROP Indexes :

Syntax : DROP INDEX index-name ON table_name;

DROP Table : used to delete table from database.

Syntax :

```
DROP TABLE table_name;
DROP TABLE EMP;
```

DROP Database : It is used to delete database from system.

Syntax :

```
DROP DATABASE database_name;
DROP DATABASE Employee;
```

TRUNCATE TABLE : It is used to delete the data inside the table

Not the table structure/schema.

```
TRUNCATE TABLE table_name;
TRUNCATE TABLE Emp;
```

DROP View : It is used to delete view.

```
DROP VIEW view_name;
DROP VIEW viewzo;
```

5.5 DATA MODIFICATION LANGUAGE STATEMENTS IN SQL

INSERT Command :

Once a table is created, the most natural thing to do is load this table with data to be manipulated later.

Syntax : INSERT INTO table_name(column_name1, column_name2...) VALUES (expression, expression);

Example :

INSERT INTO Employee(E-ID, ENAME, ADDRESS, CITY,STATE, PINCODE) VALUES (115, 'VIJAY', 'C-6 Gupta Road', 'New Delhi', 'DELHI');

Another method: INSERT INTO Employee VALUES(& E-ID, '& NAME', '& ADDRESS', '& CITY', '& STATE');

Note :

(i) The character or varchar expression must be enclosed in single quotes (').

(ii) In the insert into SQL statement the columns and values have a one to one relationship.

Inserting Data into a Table from another Table

Syntax : INSERT INTO table_name SELECT column_name, column_name, FROM table_name;

Example : Insert into table student1 from the table student;

INSERT INTO student 1 SELECT Roll-No, Name, Address, Sex, City, Ph-No, State FROM Student;

Insertion of a Data Set into a Table from another Table

Syntax : INSERT INTO table_name SELECT column_name, column_name FROM table_ name WHERE Column = Expression;

Example : Insert records into the table student 1 from the table student where the field Roll-No contains the value '115';

INSERT INTO Student1 SELECT Roll - No, Name, Address, Sex, City, Ph-No, State FROM Student WHERE Roll-No = '115';

DELETE Command :

The DELETE commands deletes rows from the table that satisfies the condition provided by its WHERE clause, and returns the number of records deleted.

The Verb DELETE in SQL is used to remove rows from table. To remove

- All the rows from a table.

OR

- A select set of rows from a table.

Removal of All Rows :

Syntax : DELETE FROM table_name;

Example : (1) Delete all rows from the table student

DELETE FROM Student;

Removal of a Specified Rows :

Syntax : DELETE FROM table_name WHERE search condition;

Example : Delete rows from the table student where the Roll-No> 115.

DELETE FROM student where Roll-No> 115;

UPDATE Command : Updating the Contents of a Table: The UPDATE command is used to change or modify data values in a table.

To update :

- All the rows from a table.

OR

- A select set of rows from a table.

Updating of All Rows :

Syntax : UPDATE table_name SET column_name = expression, column_name = expression;

Example : Give every employee a bonus of 10%. Update the values held in the column net-salary.

UPDATE Employee SET Netsal = net-salary + Basic salary * 0.10;

Updating Records Conditionally :

Syntax : UPDATE table_name SET column_name = expression, column_name = expression

WHERE column_name = expression;

Example : Update the table student change, the contents of the field name to 'Vijay Krishna' and the contents of field Address to 'Hyderabad' for the record identified by the field Roll-No containing the value 115;

UPDATE student SET name = 'Krishna', Address = 'Hyderabad'

WHERE Roll-No = 115;

5.6 BASIC QUERIES WITH EXAMPLES IN SQL

CREATE TABLE COMMAND :

Syntax : CREATE TABLE table_name(column_name datatype (size), column_name data type (size), column_name datatype (size));

EXAMPLE - 1

Create a Employee Table.

CREATE TABLE Employee (E-ID number (6), ENAME char (15), ADDRESS varchar (15), CITY char (15), STATE char (15), PINCODE number (6));

Output : Table Created

EXAMPLE - 2

Create a Student table

CREATE TABLE Student (Roll-No number (15), Name char (15), Address varchar (15), Sex char (2), City char (15), Phone number (15), State char (1));

Output : Table Created.

INSERTION OF DATA INTO TABLES :

Once a table is created, the most natural thing to do is load this table with data to be manipulated later.

Syntax : INSERT INTO table_name(column_name1, column_name2 ...) VALUES (expression, expression);

EXAMPLE - 1

INSERT INTO Employee (E-ID, ENAME, ADDRESS,CITY,STATE, PINCODE) VALUES (115, 'VIJAY', 'C-6 Gupta Road', 'New Delhi', 'DELHI');

Another method: INSERT INTO Employee VALUES(& E-ID, '&NAME', '&ADDRESS', '& CITY', '& STATE');

Note :

(i) The character or varchar expression must be enclosed in single quotes (').

(ii) In the insert into SQL statement the columns and values have a one to one relationship.

SELECT COMMAND :

Once data has been inserted into a table, the next most logical operation would be to view what has been entered. This is achieved by SELECT SQL Verb.

(i) View global table data the syntax is :

SELECT * FROM table_name;

e.g., SELECT * FROM Employee;

(ii) Retrieve ID, name, city of the employee

SELECT E-ID, ENAME, CITY FROM Employee;

Selected Columns and Selected Rows:

WHERE Clause

Syntax : SELECT * FROM table_name

WHERE search condition;

(i) SELECT * FROM Employee WHERE E-ID > 112;

(ii) SELECT Roll-No, Name FROM Student WHERE Roll No. < = 150;

Elimination of Duplicates from the Select Statement

A table could contain duplicate rows. We can eliminate using select statement.

Syntax : SELECT DISTINCT column_name1, column_name2 FROM tablename;

Syntax : SELECT DISTINCT * FROM table_name

EXAMPLE - 1

Select only unique rows from the table student :

SELECT DISTINCT * FROM Student;

SORTING DATA IN A TABLE :

Oracle allows data from a table to be viewed in a sorted order. The rows retrieved from the table will be sorted in either ascending or descending order depending on the condition specified in the select statement.

Syntax : SELECT * FROM table_name ORDER BY column_name1, column_name2 [sortorder];

EXAMPLE - 1

Retrieve all rows from student and display this data sorted on the value contained in the field Roll-No. in ascending order;

SELECT * FROM Student ORDER BY Roll-No;

Note : Oracle engine sorts in ascending order by default.

EXAMPLE - 2

For viewing the data in descending sorted order the word desc.

SELECT * FROM Student ORDER BY Roll-No desc;

CREATING A TABLE FROM A TABLE :

Syntax : CREATE TABLE table_name [(column_name, column_name)] AS SELECT column name, columnname FROM Tablename;

EXAMPLE - 1

Create a table student1 from student.

CREATE TABLE Student1(SRoll-No, SName, Address, Sex, City, Ph.No, State)

AS SELECT Roll-No, Name, Address, Sex, City, Ph. No., State FROM Student;

INSERTING DATA INTO A TABLE_FROM ANOTHER TABLE :

Syntax : INSERT INTO table_name SELECT column_name, column_name, FROM table name;

EXAMPLE - 1

Insert into table student1 from the table student;

INSERT INTO student1 SELECT Roll-No, Name, Address, Sex, City, Ph-No, State FROM Student;

INSERTION OF A DATA SET INTO A TABLE FROM ANOTHER TABLE :

Syntax : INSERT INTO table_name

SELECT column name, column name FROM table name WHERE Column = Expression;

EXAMPLE - 1

Insert records into the table student1 from the table student where the field Roll-No contains the value '115';

INSERT INTO Student1 SELECT Roll-No, Name, Address, Sex, City, Ph-No, State FROM Student WHERE Roll-No = '115';

DELETE OPERATIONS :

The DELETE commands deletes rows from the table that satisfies the condition provided by its WHERE clause, and returns the number of records deleted.

The Verb DELETE in SQL is used to remove rows from table. To remove

- All the rows from a table.

OR

- A select set of rows from a table.

Removal of All Rows :

Syntax : DELETE FROM table_name;

EXAMPLE - 1

Delete all rows from the table student

DELETE FROM Student;

Removal of a Specified Rows :

Syntax : DELETE FROM table_name WHERE search condition;

EXAMPLE - 2

Delete rows from the table student where the Roll-No> 115.

DELETE FROM student where Roll-No> 115;

UPDATE COMMAND :

Updating the Contents of a Table : The UPDATE command is used to change or modify data values in a table.

To update :

- All the rows from a table.

OR

- A select set of rows from a table.

Updating of All Rows :

Syntax : UPDATE table_name SET column_name = expression, column_name = expression;

EXAMPLE - 1

Give every employee a bonus of 10%. Update the values held in the column net-salary.

UPDATE Employee SET Netsal = net-salary + Basic salary * 0.10;

Updating Records Conditionally :

Syntax : UPDATE table_name SET column_name = expression, column_name = expression WHERE column_name = expression;

EXAMPLE - 2

Update the table student change, the contents of the field name to 'Vijay Krishna' and the contents of field Address to 'Hyderabad' for the record identified by the field Roll-No containing the value 115;

UPDATE student SET name = 'Krishna', Address = 'Hyderabad' WHERE Roll-No = 115;

MODIFYING THE STRUCTURE OF TABLES :

Adding New Columns :

Syntax : ALTER TABLE table_name

ADD (new_column_name data type/size, new_column_name data type (size ...);

EXAMPLE - 1

Add the field Fax which is a field that can hold number upto 15 digits in length and Mobile-No, which is a field that can hold a number upto 10 digits in length.

ALTER TABLE student

ADD (Mobile-No number (10), Fax number (15));

Modifying Existing Columns :

Syntax : ALTER TABLE table_name

MODIFY (column_name new data type (New size));

EXAMPLE - 2

Modify the field fax of the table student to now hold maximum of 25 character values.

ALTER TABLE student MODIFY (Fax Varchar (25));

Limitation of the ALTER TABLE : Using the ALTER TABLE clause the following tasks cannot be performed:

- Change the name of the table.
- Change the name of the column.
- Drop a column.
- Decrease the size of a column if table data exists.

RENAMING COMMAND :

To rename a table, the syntax is :

Syntax : RENAME old_table_name to new_table_name

EXAMPLE - 1

Rename the table Employee to Employee 1;

RENAME Employee TO Employee1;

DESTROYING TABLES :

Syntax : DROP TABLE table_name;

EXAMPLE - 1

Destroy the table Employee and all the data held in it;

DROP TABLE Employee;

DESCRIBE COMMAND :

To find information about the column defined in the table uses the following syntax;

Syntax : DESCRIBE table name;

This command displays the column names, the data types and the special attributes connected to the table.

EXAMPLE - 1

Displays the columns and their attributes of the table student.

DESCRIBE Student;

QUERIES USING LOGICAL OPERATORS :

There are following logical operators used in SQL.

- **The AND Operator :** The oracle engine will process all rows in a table and display the result only when all of the conditions specified using the AND operator are satisfied.

EXAMPLE - 1

Retrieve the contents of the columns product-no, profit-percent, sell-price from the table product-master where the values contained in the field profit percent in between 10 and 20.

SELECT Product-no, profit-percent, sell-price

FROM Product-master

WHERE profit-percent> = 10 AND profit-percent < = 20;

- **The OR Operator :** The oracle engine will process all rows in a table and diplay the result only when any of the conditions specified using the OR operator are satisfied.

EXAMPLE - 2

Retrieve the all fields of the table student where the field Roll-No has the value 115 OR 200;

SELECT Roll-No, Name, Address, Sex, City, Ph-No,

State FROM Student WHERE (Roll-No = 115 OR Roll-No = 200);

- The NOT Operator: The oracle engine will process all rows in a table and display the result only when none of the conditions specified using the NOT operator are satisfied.

EXAMPLE - 3

Retrieve specified student information for the clients, who are NOT in 'New-Delhi' OR 'Noida';

SELECT Roll-No, Name, Address, City, State

FROM Student WHERE NOT (City = 'New Delhi' OR City = 'Noida');

Range Searching

BETWEEN Operator :

EXAMPLE - 4

Retrieve Roll-No, Name, Address, Ph-No, State from the table student where the values contained within the field Roll-No is between 100 and 200 both inclusive.

SELECT Roll-No, Name, Address, Ph-No, State FROM Student

WHERE Roll-No BETWEEN 100 AND 200;

Pattern Matching : The use of the LIKE predicate: The LIKE predicate allows for a comparison of one string value with another string value, which is not identical.

For the character data types :

The percent sign (%) matches any string.

The underscore (_) matches any single character.

EXAMPLE - 5

1. Retrieve all information about students whose names begins with the letters 'vi' from student table.

SELECT * FROM Student

WHERE Name LIKE 'vi %';

2. Retrieve all information about students where the second character of names are either 'V' (or) 'S'.

SELECT * FROM Student

WHERE Name LIKE '_V%' OR

Name LIKE '_S%';

THE IN PREDICATES :

In case of value needs to be compared to a list of values then the IN predicate is used.

EXAMPLE - 1

Retrieve the Roll-No, Name, Address, City, Ph-No from the table student where name is either Vijay or Santosh or Gopal or Sanjay.

SELECT Roll-No, Name, Address, City, Ph-No

FROM Student WHERE Name IN ('Vijay', 'Santosh', 'Gopal', 'Sanjay');

The NOT IN prediates: The NOT IN predicate is the opposite of the IN predicate. This will select all the rows where values do not match all of the values in the list.

EXAMPLE - 2

SELECT Roll-No, Name, Address, City, Ph-No FROM Student

WHERE Name NOT IN ('Vijay', 'Santosh', 'Gopal', 'Sanjay');

5.7 PROCESS OF SPECIFYING CONSTRAINTS IN SQL

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into the following two types,

1. Column level constraints: Limits only column data.
2. Table level constraints: Limits whole table data.

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

NOT NULL Constraint : NOT NULL constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

UNIQUE Constraint : UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. This constraint can be applied at column level or table level.

Primary Key Constraint : Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Foreign Key Constraint : FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see its use, with help of the below tables :

CHECK Constraint : CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. It's like condition checking before saving data into a column.

SQL Constraints : SQL constraints are used to specifiy rules for the data in a table. Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the date in the table.

- Constraints can be columnlevel or table level.

The following constraints are commonly used in SQL

(i) NOT NULL (ii) UNIQUE

(iii) PRIMARY KEY (iv) FOREIGN KEY

(v) CHECK (vi) DEFAULT.

SQL NOT NULL : By default, a column can hold null values. The NOTNULL constraint enforces a column to not accept null values. This enforces a field to always contain a value.

EXAMPLE - 1

```
> CREATE TABLE Hero(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  fname varchar(30) NOT NULL,
  age int);
```


SQL UNIQUE CONSTRAINT : It ensures that all values in a column are different. Both unique and primary key constraints provide a guarantee for uniqueness for a column or set of columns. However you can have many unique constraints for table.

EXAMPLE - 2

Oracle :

```
> CREATE TABLE Hero(
  ID int NOT NULL UNIQUE,
  lname varchar(30) NOT NULL,
  fname varchar(30) NOT NU' ',
  age int);
```

EXAMPLE - 3

MySQL :

```
> CREATE TABLE Hero(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  fname varchar(30) NOT NULL,
  age int,
  UNIQUE(ID);
```

Defining a unique constraint on multiple columns;

Syntax :

```
Constraint Constraintname
> CREATE TABLE Hero(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  fname varchar(30) NOT NULL,
  age int,
CONSTRAINT UK UNIQUE (ID, lname));
```

Creating unique key when table is already created

```
> ALTER TABLE student/hero
  ADD UNIQUE (pin);
```

Syntax :

```
Alter table tablename
ADD constraintname (columnname);
```

Defining a unique key on multiple column when table is already created

```
> ALTER TABLE student
  ADD
  CONSTRAINT UK UNIQUE (pin, phone no);
```

Deleting unique constraint :

```
> ALTER TABLE Student/Hero
  DROP
  UNIQUE UK;
```

SQL PRIMARY KEY CONSTRAINT :

It uniquely identify each record in a database table. Primary key must contain unique values, cannot contain null values.

EXAMPLE - 1

MySQL :

```
> CREATE TABLE Hero(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  age int,
  PRIMARY KEY (ID));
```

EXAMPLE - 2

Oracle :

```
> CREATE TABLE Hero(
  ID int NOT NULL PRIMARY KEY,
```



```
  lname varchar(30) NOT NULL,
  fname varchar(30) NOT NULL,
  age int);
```

Defining a primary key constraint on multiple column :

```
> CREATE TABLE Hero(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  fname varchar(30) NOT NULL,
  age int,
  CONSTRAINT PK PRIMARY KEY(ID, lname);
```

Creating Primary key when table is already created :

```
> ALTER TABLE STUDENT/Hero
  ADD PRIMARY KEY(pin);
```

Defining creating primary key when table is already created on multiple columns:

```
> ALTER TABLE Student/Hero
  ADD
  CONSTRAINT PR PRIMARY KEY(pin, phoneno);
```

Deleting a primary key :

```
> AUTER TABLE Student/Hero
  Drop
  PRIMARY KEY PR;
```

FOREIGN KEY CONSTRAINT :

A foreign key is a key used to link two tables together. A foreign is a field or collection of fields in one table that refers to the primary key in another table.

The table containing the foreign key is called child table, and the table containing the candidate key is called referenced (or) parent table.

EXAMPLE - 1

Hero (id, lname, fname, age)

Heroine (hid, phone no, id)

MySQL :

```
> CREATE TABLE Heroine(
  hid int NOT NULL,
  phoneno bigint NOT NULL,
  id int,
  PRIMARY KEY(hid),
  FOREIGN KEY(ID) REFERENCES Hero(id));
```

Oracle :

```
> CREATE TABLE Heroine(
  hid int NOT NULL PRIMARY KEY,
  Phoneno bigint NOT NULL,
  id int FOREIGN KEY REFERENCES
  Hero(id));
```

Defining a Foreign key constraint on multiple columns :

```
> CREATE TABLE Heroinc(
  hid int NOT NULL,
  Phoneno bigint NOT NULL,
  id int, PRIMARYKEY(hid),
  CONSTRAINT FK FOREIGNKEY (id)
  REFERENCES Hero(id));
> ALTER TABLE Heroine
  ADD FOREIGN KEY(id) REFERENCES Hero(id);
```

Defining foreign key constraint on multiple columns when table is already created :

```
> ALTER TABLE Heroine
  ADD CONSTRAINT FK FOREIGN KEY(id)
  REFERENCES Hero(id);
```


Deleting a foreign key constraint :

```
> ALTER TABLE Heroine
  DROP FOREIGN KEY FK;
```

SQL 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 single column it allows only certain value for that column.

EXAMPLE - 1

MySQL :

```
> CREATE TABLE Vilan(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  fname varchar(30), age int,
  CHECK (age > = 18));
```

EXAMPLE - 2

Oracle :

```
CREATE TABLE Vilan(
ID int NOT NULL,
lname varchar(30) NOT NULL,
fname varchar(30),
age int check (age > = 18));
```

Defining a check constraint on multiple columns :

```
> CREATE TABLE Vilan(
ID int NOT NULL,
lname varchar(30) NOT NULL,
fname varchar(30),
age int,
```

```
city varchar(30),
Constraint CHK CHECK (age > = 18 AND City = 'Hyderabad'));
```

Creating a check constraint when table is already created :

EXAMPLE - 3

```
> ALTER TABLE Vilan
  ADD CHECK (age > = 18);
```

Defining a check constraint on multiple columns when table is already created

```
> ALTER TABLE Vilan ADD
  CONSTRAINT CHK CHECK (age > = 18 AND city = 'Hyderabad');
```

Deleting a check constriant :

```
> ALTER TABLE Vilan
  DROP CHECK CHK; /DROP CONSTRAINT CHK;
```

DEFAULT CONSTARAINT :

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

EXAMPLE - 1

```
> CREATE TABLE Comedian(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  fname varchar(30),
  age int,
  city varchar(30) DEFAULT 'Hyderabad');
```

The default constraint can also be used to insert system values by using functions.

EXAMPLE - 2

```
> CREATE TABLE Comedian(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  fname varchar(30),
```



```
  age int,
  city varchar(30) DEFAULT 'Hyderabad';
  dob date DEFAULT GETDATE( ));
```

Creating a default constraint when table is already created :

MySQL :

```
> ALTER TABLE Comedian
  ALTER City SET DEFAULT 'Hyderabad';
```

Oracle :

```
> ALTER TABLE Comedian
  MODIFY city DEFAULT 'Hyderabad';
```

Deleting a default constraint :

MySQL :

```
> ALTER TABLE Comedian
  ALTER city DROP DEFAULT;
```

Oracle :

```
> ALTER TABLE Comedian
  ALTER COLUMN city DROP DEFAULT;
```

5.8 SQL FUNCTIONS

They are five types of SQL functions,

(i) Numeric function.

(ii) Aggregate function.

(iii) Scalar function.

(iv) Date function.

(v) String function.

(i) **Numeric Function :** These are used primarly for numeric manipulation and mathematical calculation.

1. **ABC :** This function returns absolute value of (x).

   ABS (x)

   Ex : > Select ABS (– 2);

ABS (–2)
2

   Ex : ABS (x)

   > Select ABS(2);

ABS (2)
2

2. **CEILING (x) :** This function returns the smallest integer value and it is not smaller than x.

   Ex : > Select CEIL (3.4);

CEIL (3.4)
4

3. **FLOOR (x) :** This function returns the largest value.

   Ex : > Select FLOOR (3.6);

FLOOR (3.6)
3

4. **COS(x) :** It returns the cosine of x.

   The value of x is given is radiants.

   Ex : Select cos(90);

cos (90)
0

5. **GREATEST (n_1, n_2 .....) :** It returns the greatest value in set of input parameters.

   Ex : > Select GREATEST (1, 10, 20, 5);

GREATEST (1,10,20,5)
20

6. **LEAST (n_1, n_2 .....) :** It returns the least value in set of input parameters.

   Ex : > Select LEAST (1, 10, 20, 5);

LEAST (1,10,20,5)
1

7. **MOD (N, M) :** It returns the remainder of N divided by M.

   Ex : > Select MOD (29, 3);

MOD (29,3)
2

8. **Power (x, y) :** It returns (x, y) value of x to the power of y.

   Ex : > Select power (3, 3);

Power (3,3)
27

9. **Round (x, y) :** This function returns round-nearest integer and the second argument in this function returns x rounded to y decimal place.

   Ex : > Select ROUND (5.7);

ROUND (5.7)
6

   Ex : > Select Round (5.68145, 2);

Round (5.68145,2)
5.68

10. **PI :** It simply returns the value of Pi.

    Ex : > Select PI ( );

PI
3.141593

11. **SQRT (x) :** It returns the non negative square root of x.

    Ex : > Select SQRT (49);

SQRT (49)
7

12. **TRUNCATE (X, D) :** It is same as round as (x, y).

Ex : Select Truncate (7.53678, 3);

TRUNCATE (7.53678,3)
7.536

(ii) Aggregate Function : Aggregate functions are functions that take a collection of values as input and return a single value. SQL offers five suit-in aggregate functions.

- **Average :** AVG;
- **Minimum :** MIN
- **MAXIMUM :** MAX
- **Total :** SUM
- **Count :** COUNT

COUNT :

Syntax : COUNT ([DISTINCT/ALL] expr) returns the number of rows where 'expr' is not NULL.

EXAMPLE - 1

SELECT COUNT (Product-no) "No. of product".

FROM product-master;

Output : No. of product = 10

AVG :

Syntax : AVG ([DISTINCT/ALL]n)

EXAMPLE - 2

SELECT AVG (sell-price) "Average"

FROM product-master;

Output : Average

(205.137)

MIN :

Syntax : MIN ([DISTINCT\ALL] expr)

EXAMPLE - 3

SELECT MIN (Bal-due) "Minimum Balance" FROM client-master;

Output : Minimum Balance 3

MAX :

Syntax : MAX ([DISTINCT\ALL] expr)

EXAMPLE - 4

SELECT MAX (Bal-due) "Maximum"

FROM client-master;

Output : Maximum 25000

SUM :

Syntax : SUM ([DISTINCT\ALL]n)

EXAMPLE - 5

SELECT SUM (Bal-due) "Total Balance Due"

FROM client-master;

Output : Total Balance Due 30000

POWER :

Syntax : POWER (m, n)

Rcturns 'm' raised to 'nth' power 'n' must be an integer, else an error is returned.

EXAMPLE - 6

SELECT POWER (3,2) "RESULT =" FROM math;

Output : RESULT = 9

ABS :

Syntax : ABS (n)

Returns the absolute value of 'n'

EXAMPLE - 7

SELECT ABS (-10) "Absolute =" FROM math;

Output : Absolute = 10

LOWER :

Syntax : LOWER (char)

Returns char, with all letters in lowercase

EXAMPLE - 8

SELECT LOWER ('KRISHNA') "Lower Case" FROM math;

Output : Lower Case krishna

UPPER :

Syntax : UPPER (char)

Returns char with all letters forced to upper case

EXAMPLE - 9

SELECT UPPER ('Krishna') "Result" FROM math;

Output : Result

KRISHNA

INITCAP :

Syntax : INITCAP (char)

Returns string with the first letter in upper case

EXAMPLE - 10

SELECT INITCAP ('KRISHNA') "Result" FROM math;

Output : Result

Krishna

SQRT :

Syntax : SQRT (n)

Returns square root of 'n'. If n < 0, NULL. SQRT returns a real result.

EXAMPLE - 11

SELECT SQRT (49) "Square root ="

FROM Math;

Output : Square Root = 7

LENGTH :

Syntax : LENGTH (char)

Returns the length of char.

EXAMPLE - 12

'SELECT LENGTH ('VIJAY') "Length"

FROM Math;

Output : Length 5

LTRIM :

Syntax : LTRIM (char [, set])

- Removes character from the left of char with initial characters removed upto the first character not in set.

Example :

LTRIM(' tech')

Result : 'tech'

LTRIM(' tech', ' ')

Result : 'tech'

LTRIM('000123', '0')

Result : '123'

LTRIM('123123Tech', '123')

Result : 'Tech'

LTRIM('123123Tech123', '123')

Result : 'Tech123'

LTRIM('xyxzyyyTech', 'xyz')

Result : 'Tech'

LTRIM('6372Tech', '0123456789')

Result : 'Tech'

The LTRIM function may appear to remove patterns, but this is not the case as demonstrated in the following example.

LTRIM('xxyyxzyxyyxTech', 'xyz')

Result : 'Tech'

It actually removes the individual occurrences of 'x', 'y', and 'z', as opposed to the pattern of 'xyz'.

The LTRIM function can also be used to remove all leading numbers as demonstrated in the next example.

LTRIM('637Tech', '0123456789')

Result : 'Tech'

In this example, every number combination from 0 to 9 has been listed in the trim_string parameter. By doing this, it does not matter the order that the numbers appear in string1, all leading numbers will be removed by the LTRIM function.

RTRIM :

Syntax : RTRIM (char, [set])

- Returns char, with final characters removed after the last character not in the set.

Example :

RTRIM('tech ')

Result : 'tech'

RTRIM('tech ', ' ')

Result : 'tech'

RTRIM('123000', '0')

Result : '123'

RTRIM('Tech123123', '123')

Result : 'Tech'

RTRIM('123Tech123', '123')

Result : '123Tech'

RTRIM('Techxyxzyyy', 'xyz')

Result : 'Tech'

RTRIM('Tech6372', '0123456789')

Result : 'Tech'

The RTRIM function may appear to remove patterns, but this is not the case as demonstrated in the following example.

RTRIM('Techxyxxyzyyyxx', 'xyz')

Result : 'Tech'

It actually removes the individual occurrences of 'x', 'y', and 'z', as opposed to the pattern of 'xyz'.

The RTRIM function can also be used to remove all trailing numbers as demonstrated in the next example.

RTRIM('Tech6372', '0123456789')

Result : 'Tech'

In this example, every number combination from 0 to 9 has been listed in the trim_string parameter. By doing this, it does not matter the order that the numbers appear in string1, all trailing numbers will be removed by the RTRIM function.

(iii) **Scalar Function :** It returns a single value based on input value.

1. **UCASE :** It converts a text field to upper case.
2. **LCASE :** It converts a text field to lower case.
3. **MID :** It extracts character from a text field.
4. **LEN :** It returns the length of a text field.

Examples :

Select UCASE('suresh'); ---------SURESH

Select LCASE('SURESH'); ------suresh

Select LENGTH('suresh'); -------6

Select MID('suresh',2); ----------r

(iv) **Date Function :**

1. **CURDATE :** It returns current date as a value in YYY-MM-DD format.

   Ex : > Select CURDATE ( );

CURDATE( )
2019-05-04

2. **CURTIME :** It returns the current time as a value in HH:MM:SS

Ex : > Select CURTIME ( );

CURTIME( )
11:05:30

3. **DAY OF YEAR :** It returns the day of the year

Ex : > Select DAYOFYEAR ('2018 - 07 - 15');

DAYOFYEAR('2018-07-15')
196

(So range is upto 365)

4. **HOUR (TIME) :** It returns the hour for time.

The range of the return value is 0 to 23 how ever the range of time values actually much large so hour can return vale > 23.

Ex : > Select HOUR ('11:05:30');

HOUR('11:05:30')
11

5. **MONTH (DATE) :** It returns the month for date in the rage 1 to 12 for January to december.

Ex : > Select MONTH ('2014 - 07 - 30');

MONTH('2014-07-30')
07

6. **MINUTE (TIME) :** It returns the minute from the time in the range 0 to 59.

Ex : > Select MINUTE ('2014 - 07 - 30 11:15:42');

MINUTE
15

7. **NOW :** It returns the current date and time as the value in YYY-MM-DD, HH:MM:SS.

Ex : > Select NOW ( );

NOW( )
2014-07-31, 09:45:50

8. **SYSDATE :** Same as above function.

Ex : > Select SYSDATE ( );

SYSDATE( )
2014-07-31, 09:46:50

9. **TIMESTAMP :** It returns the date or date time expression.

TIMESTAMP (exp1, exp2)

Here expr is a date time value which as two arguments expr1, expr2, which is used to add the time and return the result as the date time value.

Ex : Select TIMESTAMP ('2014 - 07 - 31');

TIMESTAMP
2014-07-31, 00:00:00

Ex : > Select TIMESTAMP ('2014 - 12 - 31', '12:00:00')

TIMESTAMP
2015-01-01, 00:00:00

10. **YEAR :** It returns the current year.

Ex : > Select YEAR ('2014 - 07 - 31');

YEAR('2014-07-31')
2014

(v) **String Functions :**

1. **LENGTH (String) :** It returns number of characters in a string.

Ex : > Select LENGTH ('vani');

LENGTH('vani')
4

2. **LOWER (String) :** It converts string to lower case.

Ex : > Select LOWER ('SWETHA')

LOWER('Swetha')
Swetha

3. **UPPER (String) :** It converts into upper case.

Ex : Select UPPER ('Swetha');

UPPER('Swetha')
SWETHA

4. **INITCAP (String) :** It converts first character to capital letter.

Ex : > Select INITCAP ('Swetha')

INITCAP('swetha')
Swetha

5. **REVERSE (String) :** It returns the string in reverse order.

Ex : > Select REVERSE ('Shivani');

REVERSE('Shivani')
inavihS

6. **STRCMP (String1, String2) :** It compare two strings and returns 0 if both strings are equal, it returns-1 if first argument is smaller than second According to the current sort order otherwise it returns 1.

Ex : > Select STRCMP ('Rani', 'Raju');

STRCMP('Rani','Raju')
1

Ex : > Select STRCMP ('Kaif', 'Katrina');

STRCMP('Kaif','Katrina')
–1

Ex : > Select STRCMP ('Anushka', 'Arina');

STRCMP('Anushka','Arina')
–1

7. **SUBSTRING :**

(a) SUBSTRING (String, POS).

(b) SUBSTRING (String, from POS).

(c) SUBTRING (String, POS, LEN).

(a) **SUBSTRING (String, POS) :** The first function will return substring from the given string starting at position (POS).

Ex : > Select SUBSTRING ('bhavani', 4);

SUBSTRING('bhavani', 4)
Vani

(b) **SUBSTRING (String, from POS) :**

Ex : > Select SUBSTRING ('bhavani', From 4);

SUBSTRING('bhavani', from 4)
Vani

(c) **SUBSTRING (String, POS, Len) :**

Ex : > Select SUBSTRING ('bharat', 2, 6);

SUBSTRING('bharat', 2, 6)
harat

8. **TRIM :** It returns the string without prefix or suffix. It remove spaces.

Ex : SELECT TRIM (' Raju');

TRIM('Raju')
Raju

(a) **LTRIM :** It removes the left prefix (or) Space from the given string.

Ex : > Select LTRIM (' Raju')

LTRIM(Raju)
Raju

(b) **RTRIM :** It removes the right space from the given string.

Ex : > Select RTRIM ('Raju ')

RTRIM('Raju')
Raju

9. **CONCAT (strin1, string2) :** It returns the string that results from CONCATINATING (adding) the arguments.

Ex : > Select CONCAT ('kajol', 'agarwal');

CONCAT('kajol','agarwal')
Kajolagarwal

10. **INSERT (string, POS, len, new string) :** It returns a new string by adding a string from position and length to the given string.

Ex : > Select INSERT ('bharat', 4, 6, van)

INSERT(Bharat,4,6,'van')
Bhavan

> Select INSERT ('Bharat', 2, 4, 'indu');

INSERT('Bharat',2,4,'indu')
bindut

11. **REPLACE (string from string to string) :**

Ex : > Select REPLACE ('Vani', 'V', 'R');

REPLACE('Vani','V','R')
Rani

12. **LPAD (string, len, pad string) :** Padding (adding or pumping) the string onleft.

Ex : > Select LPAD ('hi', 4, '$$');

LPAD('hi',4,$$)
$$hi

Ex : > Select LPAD ('hi', 3, '$$');

LPAD('hi',3,'$$')
$hi

Ex : > Select LPAD ('hi', 2, '&&');

LPAD('hi',2,'&&')
hi

Ex : > Select LPAD ('hi', 6, '@@');

LPAD('hi',6,'@@')
@@@@hi

13. **RPAD (string, length, padstring) :** Padding (adding or pumping) the string on right.

Ex : > Select RPAD ('hi', 4, '$$')

RPAD('hi',4,'$$')
hi$$

5.9 SUB QUERIES AND SQL JOIN STATEMENTS

Subqueries : A subquery (or) innerquery (or) nested query is a query with in another SQL query and embedid within the where clause. It is used to return data that will be used in the main query as a condition to further restricts the data to be retrieved. Subquery can be used with select, insert, update, delete statements along with operators like =, <, >, < =, > =, in between

- Subquery is an approach which provides the capability of embedding the first query into another query.
- Subquery must be enclosed with in paranthes.
- A subquery can have only one column in the select statement unless multiple columns are in main query for the subquery to compare it's selected column.
- An order by clause cannot be used in a subquery although the main query can have order by clause and also group by clause can be used to perform the same function as order by clause in a subquery.
- A subquery that returns more than one row can only be used with multiple value operator such as IN operator.
- The Between operation cannot be used with in subquery.

Sub Queries : A Subquery is a query within another SQL query and embedded within the WHERE clause.

Important Rule :

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.

- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

1. **Subqueries with the Select Statement :** SQL subqueries are most frequently used with the SELECT statement.

Syntax :

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
(SELECT column_name  from table_name WHERE ... );
```

EXAMPLE - 1

Consider the EMPLOYEE table have the following records :

```
CREATE TABLE EMPLOYEE(ID integer,NAME varchar(20), AGE integer,ADDRESS varchar(20), SALARY float);
CREATE TABLE EMPLOYEE_BKP(ID integer,NAME varchar(20), AGE integer,ADDRESS varchar(20), SALARY float);
INSERT INTO EMPLOYEE VALUES(1,'John',20,'US',2000.00);
INSERT INTO EMPLOYEE VALUES(2,'Stephen',26,'Dubai',1500.00);
INSERT INTO EMPLOYEE VALUES(3,'David',27,'Bangkok',2000.00);
INSERT INTO EMPLOYEE VALUES(4,'Alina',29,'UK',6500.00);
INSERT INTO EMPLOYEE VALUES(5,'Kathrin',34,'Bangalore',8500.00);
INSERT INTO EMPLOYEE VALUES(6,'Harry',42,'China',4500.00);
INSERT INTO EMPLOYEE VALUES(7,'Jackson',25,'Mizoram',10000.00);
```

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
6	Harry	42	China	4500.00
7	Jackson	25	Mizoram	10000.00

The subquery with a SELECT statement will be :

```
SELECT *
FROM EMPLOYEE
WHERE ID IN (SELECT ID
FROM EMPLOYEE
WHERE SALARY > 4500);
```

This would produce the following result :

ID	NAME	AGE	ADDRESS	SALARY
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
7	Jackson	25	Mizoram	10000.00

Subqueries with the INSERT Statement :

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax :

```
INSERT INTO table_name (column1, column2, column3....)
SELECT *
FROM table_name
WHERE VALUE OPERATOR
```

EXAMPLE - 2

Consider a table EMPLOYEE_BKP with similar as EMPLOYEE.

Now use the following syntax to copy the complete EMPLOYEE table into the EMPLOYEE_BKP table.

```
INSERT INTO EMPLOYEE_BKP
SELECT * FROM EMPLOYEE
WHERE ID IN (SELECT ID
FROM EMPLOYEE);
```

Subqueries with the UPDATE Statement : The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax :

```
UPDATE table
SET column_name = new_value
WHERE VALUE OPERATOR
(SELECT COLUMN_NAME
FROM TABLE_NAME
WHERE condition);
```

EXAMPLE - 3

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example updates the SALARY by .25 times in the EMPLOYEE table for all employee whose AGE is greater than or equal to 29.

```
UPDATE EMPLOYEE
SET SALARY = SALARY * 0.25
WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
WHERE AGE >= 29);
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	1625.00
5	Kathrin	34	Bangalore	2125.00
6	Harry	42	China	1125.00
7	Jackson	25	Mizoram	10000.00

Subqueries with the DELETE Statement : The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

Syntax :

```
DELETE FROM TABLE_NAME
WHERE VALUE OPERATOR
(SELECT COLUMN_NAME
FROM TABLE_NAME
WHERE condition);
```

EXAMPLE - 4

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example deletes the records from the EMPLOYEE table for all EMPLOYEE whose AGE is greater than or equal to 29.

```
DELETE FROM EMPLOYEE
WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
WHERE AGE >= 29 );
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
7	Jackson	25	Mizoram	10000.00

SQL joins : A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are :

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

Consider the two tables below :

CREATE TABLE EMPLOYEE(EMP_ID integer, EMP_NAME varchar(20), CITY varchar(20), SALARY bigint, AGE integer);

```
CREATE TABLE PROJECT(PROJECT_NO integer,EMP_ID integer, DEPARTMENT
varchar(20));
insert into EMPLOYEE values(1,'Angelina','Chicago',200000,30);
insert into EMPLOYEE values(2,'Robert','Austin',300000,26);
insert into EMPLOYEE values(3,'Christian','Denver',100000,42);
insert into EMPLOYEE values(4,'Kristen','Washington',500000,29);
insert into EMPLOYEE values(5,'Russell','Los angels',200000,36);
insert into EMPLOYEE values(6,'Marry','Canada',600000,48);
insert into PROJECT values(101,1,'Testing');
insert into PROJECT values(102,2,'Development');
insert into PROJECT values(103,3,'Designing');
insert into PROJECT values(104,4,'Development');
```

EMPLOYEE :

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

PROJECT :

PROJECT_NO	EMP_ID	DEPARTMENT
101	1	Testing
102	2	Development
103	3	Designing
104	4	Development

1. **Inner Join :** In SQL, INNER JOIN selects records that have matching values in both

tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

Syntax :

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query :

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
INNER JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output :

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

1. **Left Join :** The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.

Syntax :

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query :

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
```

```
LEFT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output :

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

3. ***Right Join :*** *In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.*

Syntax :

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query :

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
RIGHT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output :

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

Full Join : *In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.*

Syntax :

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query :

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
FULL JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output :

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

5.10 MANAGEMENT OF SCHEMA OBJECTS

The logical database is determined by schema objects which directly refers to database. A schema is a collection of logical structures of data, or schema objects. Schema objects can be created and manipulated with SQL. The schema objects are :

1. Tables
2. Index
3. Synonyms
4. Sequence
5. View
6. Clusters
7. Triggers
8. Stored functions, procedures and packages.

5.10.1 MANAGEMENT OF INDEXES

Index is SQL is created on existing tables to retrieve the rows quickly. When there are thousands of records in a table retrieving information will take long time.

- Indexes are created on columns which are accessed frequently. So that the information can be retrieved quickly.
- Indexes can be created on single column (or) group of columns.
- When an index is created it first sorts the data and then it assigns a row id for each row.
- Even the SQL index are created to access the rows in table quickly, they slowdown when DML operations like insert, update and delete are performed on a table because the indexes and table both are updated, where DML operation is performed.
- We use indexes only on columns which are used to search the table frequently.
- We can create index for upto 16 columns. The user cannot see index. They are just used to speed up search for query.

Types of Index : There are two types of index namely,

1. **Implicit Index :** These are created when a column is explicitly defined as primary key and unique key constraints.
2. **Explicit Index :** They are created by using create index syntax.

Syntax :

```
> CREATE  INDEX     indexname
  ON      tablename (column1, column2 ......);
```

Ex :
```
CREATE  INDEX     studentindex
ON      Student   (pin, name);
```

(or)

```
> CREATE  UNIQUE  INDEX
  ON      tablename (column1, column2 ......);
```

Ex :
```
CREATE  UNIQUE  INDEX     Studentindex1
ON      Student   (pin, name);
```

If you want to delete the created index.

```
> DROP INDEX indexname;
> DROP INDEX studentindex;
```

```
ALTER TABLE tablename
DROP INDEX indexname;
```

Cluster Index : A cluster index determines the physical order of data in a table. It is analogous to a telephone directory.

The cluster index defines the physical storage order of data in the table. The table can contain only one cluster index, however index can comprise of multiple columns.

- Cluster index sort and store the data of rows in table based on their keyvalue.
- When a primary key is created a cluster index is automatically created.
- If the table is under heavy data modification, primary key is used to search a cluster index.

Creating a Cluster :

Syntax : The following statement creates a cluster named personal with the clusterkey column department and a cluster size of 512 bytes along with the storage parameters.

Syntax :

```
> CREATE  CLUSTER  clustername
(clusterkeycolumn  datatype(size)
SIZE value
STORAGE (initial value next value);
```

Ex :
```
CREATE  Cluster  Personal
(department  varchar(30))
SIZE  512
STORAGE (INITIAL 100k NEXT 50k);
```

Creating a Cluster Index : The following statement creates the cluster index on cluster personal.

```
> CREATE  INDEX  indexname
  ON  CLUSTER  clustername;
```

Ex :
```
> Create  INDEX  personalcluster
  ON  CLUSTER  personal;
```

After creating the cluster index we can add tables to the index and perform DML operations.

Adding table to a Cluster :

Syntax :

```
> CREATE  TABLE  tablename
```

```
CLUSTER clustername (columnname)
AS
SELECT columnlist FROM
WHERE condition;
```

Ex :

```
> CREATE TABLE dept10
 CLUSTER personal (department)
 AS
 SELECT * FROM employee
 WHERE id = 10;
```

HASH Cluster : The following statement creates a Hash cluster named language with the clusterkey column clang, a maximum of ten hash keyvalues each of which is allocated 512 bytes along with storage parameters. This is used for faster retrieval of information.

Syntax :

```
> CREATE CLUSTER language(
 Clang varchar(20))
 SIZE 512
 HASHKEY 10
 STORAGE (INITIAL 100k NEXT 50k);
```

Deleting a cluster :

```
> DROP CLUSTER clustername;
> DROP CLUSTER personal;
```

5.10.2 MANAGEMENT OF SEQUENCES

Sequence (auto increment) : A sequence is a database object that generate numbers in sequential order.

Applications :

- Most often use these numbers when they require a unique value in a table.

Characteristics of Sequences :

- Sequences are available to users.

- Sequences are created.
- Sequences have a minimum and maximum value minimum value = 0, maximum value = $2^{63} - 1$.
- They can be dropped but cannot be reset i.e., once the sequence written a value and that value will never be written again.
- Sequences are incremented by the specific value when created by default value is one (1).

Creating a Sequence :

Syntax :

```
> CREATE SEQUENCE Sequencename
 MINIVALUE value
 MAXVALUE value
 START WITH value
 INCREMENT by value
 CACHE value | NO CACHE
 CYCLE value | NO CYCLE;
```

Ex :

```
> CREATE SEQUENCE Sn
 MINVALUE 1
 MAXVALUE 200
 START WITH 1
 INCREAMENT BY 1
 CACHE 20
 NO CYCLE;
```

Altering the sequence :

Syntax :

```
> ALTER SEQUENCE Sequencename
```

RESTART WITH value

MINVALUE value | NO MINVALUE

MAXVALUE value | NO MAXVALUE

INCREMENT BY value

CACHE value | NO CACHE

CYCLE value | NO CYCLE;

Ex :

> ALTER SEQUENCE Sn

RESTART WITH 50

MINVALUE 50

MAXVALUE 200

INCREMENT by 2

NO CACHE

NO CYCLE;

Deleting Sequence :

Syntax :

> DROP SEQUENCE Sequencename;

Ex : > DROP SEQUENCE Sn;

AUTO-increment : (sequence) **(mySQL)**

> CREATE TABLE producer (

Id INT NOT NULL AUTO_INREMENT,

lname varchar(30) NOT NULL,

fname varchar(30),

age INT,

PRIMARYKEY(id));

How to insert value for that particular table ?

> INSERT INTO producer (lname, fname, age)
VALUES('Amarendra', 'bahubali', 30);

> INSERT INTO producer (lname, fname, age)
VALUES('Mahendra', 'bahubali', 60);

> INSERT INTO producer (lname, fname, age)
VALUES('Balala', 'Deva', 65);

> SELECT * FROM producer;

id	lname	fname	age
1.	Amarendra	Bahubali	30
2.	Mahendra	Bahubali	60
3.	Balala	Deva	65

> DELETE FROM producer WHERE age = 65;

id	lname	fname	age
1.	Amarendra	Bahubali	30
2.	Mahendra	Bahubali	60

> INSERT INTO producer (lname, fname, age)
VALUES ('deva', 'sena', 28);

> SELECT * FROM producer;

id	lname	fname	age
1.	Amarendra	Bahubali	30
2.	Mahendra	Bahubali	60
4.	Deva	Sena	28

5.10.3 MANAGEMENT OF SYNONYMS

A synonym is simply an Alias (alternative name) for a table, view, sequence, stored procedure function and package. When there is a need to remove an application the synonym can be used.

The synonym plays an important role in distributed databases where applications can reference both local and remote database objects, to accomplish work the usage of

synonym hides the location of data structure from application and enables location transperancy.

Creating Synonym :

> CREATE SYNONYM synonymname FOR object name tablename;

Ex : > CREATE SYNONYM vidhyarthi FOR student;

> CREATE SYNONYM worker FOR employee;

Deleting a Synonym :

> DROP SYNONYM synonymname;

Ex : DROP SYNONYM vidhyarthi;

5.10.4 MANAGEMENT OF VIEW

View : A view is a virtual table or logical table. A view is dynamic because it is not related to physical schema. A view is similar to a database table which consists of rows and columns, you can query data against it.

- One of the main reason for making use of views in an application schema is to isolate application logic from direct dependence on table structure.
- Views are like virtual window through which we can view or change information of a table.

Advantages of view :

- We use the view to hide the complexity of underline table to the end user and external application. i.e., view allows you to simplify complex queries.
- We can use a view to expose only nonsensitive data to a specific group of users i.e., view helps to limit data access to specific users.
- View allows you to create the read only expose read only data to specific users. Hence providing extra security layer.
- A view enables backward compability

Disdadvantages :

- Performance : Querying data from the view can be slow. If a view is created based on other view
- You cannot create an index on a view.
- Table dependency.

- You cannot use sub-queries.

Creating a view :

> CREATE VIEW viewname

AS

select coulmnlist FROM

tablename

[WHERE condition];

EXAMPLE - 1

> CREATE VIEW sview

AS

SELECT pin, name, branch, marks, dob, address FROM student;

Modifying View :

Syntax :

CREATE (OR) REPLACE VIEW viewname

AS

SELECT columnlist FROM tablename

[WHERE condition];

Alterning View :

ALTER VIEW viewname

AS

SELECT columnlist FROM

tablename

[WHERE condition];

CREATING A VIEW BASED ON ANOTHER VIEW :

Syntax :

> CREATE VIEW newviewname

```
AS
SELECT   columnlist
FROM   oldviewname
WHERE   condition;
```

EXAMPLE - 1

```
> CREATE VIEW SPO
AS
SELECT ordenno, sum (quantity * price) total
FROM
orderdetails;
```

EXAMPLE - 2

```
> CREATE VIEW BSO
AS
SELECT order no, ROUND (total,2) AS total FROM SPO
WHERE total > 6000;
```