Unit - III

Interactive SQL and Performance Tuning

18 Marks

Syllabus Details

3.1 SQL: -Data-types, Data Definition Language (DDL),Data Manipulation language (DML), Data Control Language (DCL), Transaction Control Language (TCL).

3.2 Clauses & Join:- Different types of clauses - Where, Group by ,Order by, Having Joins: Types of Joins, Nested queries.

3.3 Operators:- Relational, Arithmetic, Logical, Set operators.

3.4 Functions:- Numeric , Date and time, String functions, Aggregate Functions.

3.5 Views, Sequences, Indexes: -Views : Concept ,Create ,Update Drop Views. Sequences :- Concept ,Create, Alter , Drop, Use of Sequence in table, Index: Concept ,Types of Index , Create ,Drop Indexes

Introduction to SQL

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Components of SQL

1.Data Definition Language (DDL)

a. CREATE b. ALTER c. DROP d. RENAME e.TRUNCATE f. DESC

2.Data Manipulation Language (DML)

a. INSERT b. UPDATE c. DELETE

3.Data Control Language (DCL)

a.COMMIT b.SAVEPOINT c.ROLLBACK d. GRANT e. REVOKE

4.Transaction Control Language (TCL)

a. BEGIN Transaction b. COMMIT Transaction c. ROLLBACK Transaction

5.Data Query Language (DQL)

a. SELECT

Data Definition Language

- The Data Definition Language (DDL) contains commands that are less frequently used. DDL commands modify the actual structure of a database, rather than the database's contents.
- Examples of commonly used DDL commands include those used to generate a new database table (CREATE TABLE), modify the structure of a database table (ALTER TABLE), and delete a database table (DROP TABLE).

Data Manipulation Language

- The Data Manipulation Language (DML) contains the subset of SQL commands used most frequently — those that simply manipulate the contents of a database in some form.
- The four most common DML commands retrieve information from a database (the SELECT) command, add new information to a database (the INSERT command), modify information currently stored in a database (the UPDATE command), and remove information from a database (the DELETE command)

Data Control Language

- The Data Control Language (DCL) is used to manage user access to databases.
- It consists of two commands: the GRANT command, used to add database permissions for a user, and the REVOKE command, used to remove existing permissions.
- These two commands form the core of the relational database security model.

Data Query Language

- The commands of SQL that are used to retrieve data from the database are collectively called DQL. So all Select statements come under DQL.
- The purpose of DQL Command is to get some schema relation based on the query passed to it.

DDL Commands:

CREATE

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

Syntax
CREATE TABLE table_name (column1 datatype, column datatype,column datatype,....);

EG:The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

CREATE TABLE Persons (PersonID int,LastName varchar(255),FirstName varchar(255), Address varchar(255),City varchar(255));

PersonID	LastName	FirstName	Address	City

DDL Commands:

ALTER

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

SYNTAX:-ALTER TABLE - ADD Column
ALTER TABLE table_name ADD column_name datatype;

Now we want to add a column named "DateOfBirth" in the "Persons" table. We use the following SQL statement:

ALTER TABLE Persons ADD DateOfBirth date;

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

DDL Commands:

ALTER

SYNTAX:-ALTER TABLE - DROP COLUMN
ALTER TABLE table_name DROP COLUMN column_name;

we want to delete the column named "DateOfBirth" in the "Persons" table. We use the following SQL statement:

ALTER TABLE Persons DROP COLUMN DateOfBirth;

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

DDL Commands:

ALTER

Change Data Type Example

Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table. We use the following SQL statement:

ALTER TABLE Persons ALTER COLUMN DateOfBirth year;

Notice that the "DateOfBirth" column is now of type year and is going to hold a year
in a two- or four-digit format.

DDL Commands:

DROP

The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

TRUNCATE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

```
TRUNCATE TABLE table_name;
```

DDL Commands:

DESC

SQL DESC statement use for describe the list of column definitions for specified table.

You can use either DESC or DESCRIBE statement. both are return same result.

Syntax

SQL DESCRIBE Table Column use following syntax,

DESC table_name

Example

DESC users_info;

Example

```
DESC users_info;
```

Name	Null?	Type
NO	NOT NULL	NUMBER(3)
NAME		VARCHAR2(30)
ADDRESS		VARCHAR2(70)
CONTACT_NO		VARCHAR2(12)

RENAME

RENAME TABLE syntax is used to change the name of a table. Sometimes, we choose non-meaningful name for the table. So it is required to be changed.

Syntax

RENAME old_table _name TO new_table_name;

Example

RENAME STUDENTS TO ARTISTS;

DDL Commands:

DESC

SQL DESC statement use for describe the list of column definitions for specified table.

You can use either DESC or DESCRIBE statement. both are return same result.

Syntax

SQL DESCRIBE Table Column use following syntax,

DESC table_name

Example

DESC users_info;

Example

```
DESC users_info;
```

Name	Null?	Type
NO	NOT NULL	NUMBER(3)
NAME		VARCHAR2(30)
ADDRESS		VARCHAR2(70)
CONTACT_NO		VARCHAR2(12)

RENAME

RENAME TABLE syntax is used to change the name of a table. Sometimes, we choose non-meaningful name for the table. So it is required to be changed.

Syntax

RENAME old_table _name TO new_table_name;

Example

RENAME STUDENTS TO ARTISTS;

DML Commands:

INSERT

- The INSERT INTO statement is used to insert new records in a table.
- It is possible to write the INSERT INTO statement in two ways.
- The first way specifies both the column names and the values to be inserted:

Syntax

INSERT INTO table_name VALUES (value1, value2, value3, ...);

Below is a selection from the "Customers" table in the sample database:

I**NSERT INTO Customers** VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

DML Commands:

DELETE

The DELETE statement is used to delete existing records in a table.

Syntax

DELETE FROM table_name WHERE condition;

EG:--DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM table_name;

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

DELETE FROM Customers;

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

DML Commands:

UPDATE

The UPDATE statement is used to modify the existing records in a table.

Syntax

UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;

UPDATE Customers SET ContactName = 'Alfred Schmidt', City= 'Frankfurt' **WHERE** CustomerID = 1;

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

DCL Commands:

GRANT

- You can GRANT and REVOKE privileges on various database objects in SQL Server.
- You can grant users various privileges to tables. These permissions can be any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, orALL.
- The **syntax for granting privileges** on a table in SQL Server is:
- **GRANT privileges ON object TO user;**

For example, if you wanted to grant SELECT, INSERT, UPDATE, and DELETE privileges on a table called employees to a user name smithj, you would run the following GRANT statement:

GRANT SELECT, INSERT, UPDATE, DELETE ON employees TO smithj;

You can also use the ALL keyword to indicate that you wish to grant the permissions (ie: SELECT, INSERT, UPDATE, DELETE, and REFERENCES) to a user named smithj.

For example:

GRANT ALL ON employees TO smithj;

DCL Commands:

GRANT

If you wanted to grant only SELECT access on the employees table to all users, you could grant the privileges to the public role. For example:

GRANT SELECT ON employees TO public;

REVOKE

Once you have granted privileges, you may need to revoke some or all of these privileges. To do this, you can run a revoke command. You can revoke any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, or ALL.

The syntax for revoking privileges on a table in SQL Server is:

REVOKE privileges ON object FROM user;

For example, if you wanted to revoke DELETE privileges on a table called employees from a user named anderson, you would run the following REVOKE statement:

REVOKE DELETE ON employees FROM anderson;

TCL Commands:

COMMIT

- COMMIT command is used to permanently save any transaction into the database.
- When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

- To avoid that, we use the COMMIT command to mark the changes as permanent.
  Following is commit command's syntax,
  SQL>> COMMIT;

SAVEPOINT

- SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.
- Following is savepoint command's syntax,
  SAVEPOINT savepoint_name;
- In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

TCL Commands:

ROLLBACK

This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.
Following is rollback command's syntax,

ROLLBACK TO savepoint_name;

DQL Commands:

SELECT

The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set.

Syntax

SELECT column1, column2, ... FROM table_name;

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following.

Syntax:

SELECT * FROM table_name;

Example

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table:

SELECT CustomerName, City FROM Customers;

SELECT * Example

The following SQL statement selects all the columns from the "Customers" table:

SELECT * FROM Customers;

The SQL WHERE Clause

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

Syntax

SELECT column1, column2, ... FROM table_name WHERE condition;

Example

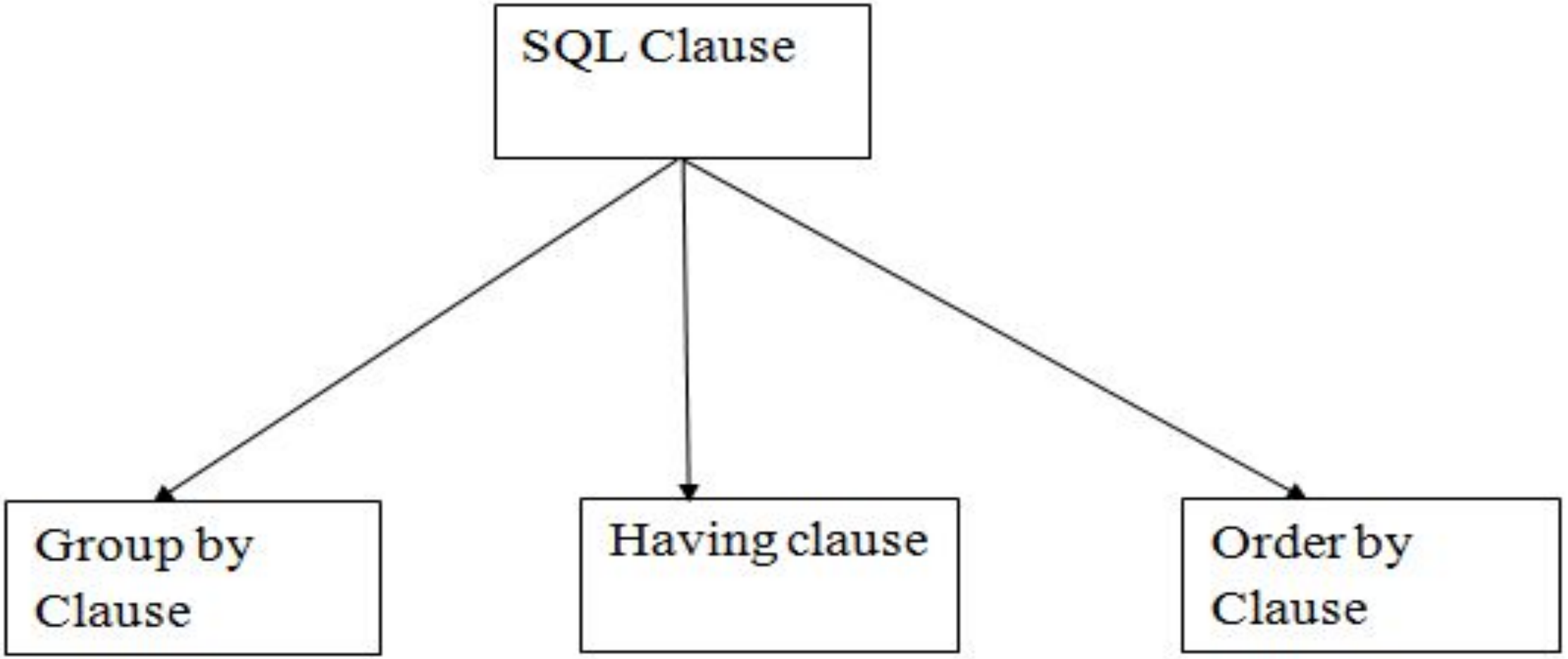
The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

SELECT * FROM Customers WHERE Country='Mexico';

Number of Records: 5

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country	Email
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico	*null*
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico	*null*
13	Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.	05022	Mexico	*null*
58	Pericles Comidas clásicas	Guillermo Fernández	Calle Dr. Jorge Cash 321	México D.F.	05033	Mexico	*null*
80	Tortuga Restaurante	Miguel Angel Paolino	Avda. Azteca 123	México D.F.	05033	Mexico	*null*

Group by, Having and order by clauses:



GROUP BY

- SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
- The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- The GROUP BY statement is used with aggregation function.

Syntax:- Select column , sum(column) from table group by column;

Ex:-sales

Company	Amount
Wipro	5500
IBM	4500
Wipro	7100

Sql>>Select company, sum (amount) from sales;

Company	Amount
Wipro	17100
IBM	17100
Wipro	17100

Sql>>Select company,sum(amount) from sales

GROUP BY company;

Company	Amount
Wipro	12600
IBM	4500

MySQL 5.5 Command Line Client

```
mysql> select * from emp;
+-----+----------+--------------+-------+--------+
| eno | name     | dept         | sal   | deptno |
+-----+----------+--------------+-------+--------+
| 101 | Anurag   | HR           | 65000 |     40 |
| 102 | Sakshi   | Sales        | 50000 |     10 |
| 103 | Amit     | Parchase     | 55000 |     20 |
| 104 | Tom      | Sales        | 50000 |     10 |
| 105 | Ravindra | HR           | 36520 |     40 |
| 106 | Sunil    | Manfacturing | 52000 |     50 |
| 107 | Vedant   | Swiper       | 26000 |     60 |
| 108 | Anil     | Admin        | 85000 |     30 |
+-----+----------+--------------+-------+--------+
8 rows in set (0.00 sec)

mysql> select deptno,sum(sal) from emp group by deptno;
```

OP:-

```
mysql> select deptno,sum(sal) from emp group by deptno;
+--------+----------+
| deptno | sum(sal) |
+--------+----------+
|     10 |   100000 |
|     20 |    55000 |
|     30 |    85000 |
|     40 |   101520 |
|     50 |    52000 |
|     60 |    26000 |
+--------+----------+
6 rows in set (0.00 sec)
```

Having Clause

- Having clause was added to sql because the where keyword could not be used against aggregate functions(like sum) and without having clause would be impossible to test for result conditions.

Syntax:- Select column , sum(column) from table group by column

Having sum(column) condition value;

Ex:-sales

Company	Amount
Wipro	5500
IBM	4500
Wipro	7100

Sql>>Select company, sum (amount) from sales group by Company having sum(amount)>10000;

Company	Amount
Wipro	12600

MySQL 5.5 Command Line Client

```
mysql> select * from emp;
+-----+----------+--------------+-------+--------+
| eno | name     | dept         | sal   | deptno |
+-----+----------+--------------+-------+--------+
| 101 | Anurag   | HR           | 65000 |     40 |
| 102 | Sakshi   | Sales        | 50000 |     10 |
| 103 | Amit     | Parchase     | 55000 |     20 |
| 104 | Tom      | Sales        | 50000 |     10 |
| 105 | Ravindra | HR           | 36520 |     40 |
| 106 | Sunil    | Manfacturing | 52000 |     50 |
| 107 | Vedant   | Swiper       | 26000 |     60 |
| 108 | Anil     | Admin        | 85000 |     30 |
+-----+----------+--------------+-------+--------+
8 rows in set (0.00 sec)

mysql> select deptno,sum(sal) from emp group by deptno;
```

OP:-

```
mysql> select deptno,sum(sal) from emp group by deptno having deptno=10;
+--------+----------+
| deptno | sum(sal) |
+--------+----------+
|     10 |   100000 |
+--------+----------+
1 row in set (0.00 sec)
```

Order by clause:

- The ORDER BY clause sorts the result-set in ascending or descending order.
- It sorts the records in ascending order by default. DESC keyword is used to sort the records in descending order.

Syntax:- SELECT column1, column2 FROM table_name

WHERE condition

ORDER BY column1, column2... ASC|DESC;

Ex:-orders

Company	Amount
sega	3412
MBT	5678
Wipro	2312
Wipro	6798

Ex1) To display company names in alphabetical order

Sql>>Select company, Amount from orders Order by company;

Company	Amount
MBT	5678
sega	3412
Wipro	**6798**
Wipro	**2312**

Ex2) To display company names in alphabetical order and the amount in numerical order

Sql>>Select company, amount from orders Order by company,amount;

Ex:-orders

Company	Amount
sega	3412
MBT	5678
Wipro	2312
Wipro	6798

Company	Amount
Wipro	**6798**
Wipro	**2312**
sega	3412
MBT	5678

Ex1) To display company names in reverse alphabetical order

Sql>>Select company, Amount from orders Order by company Desc;

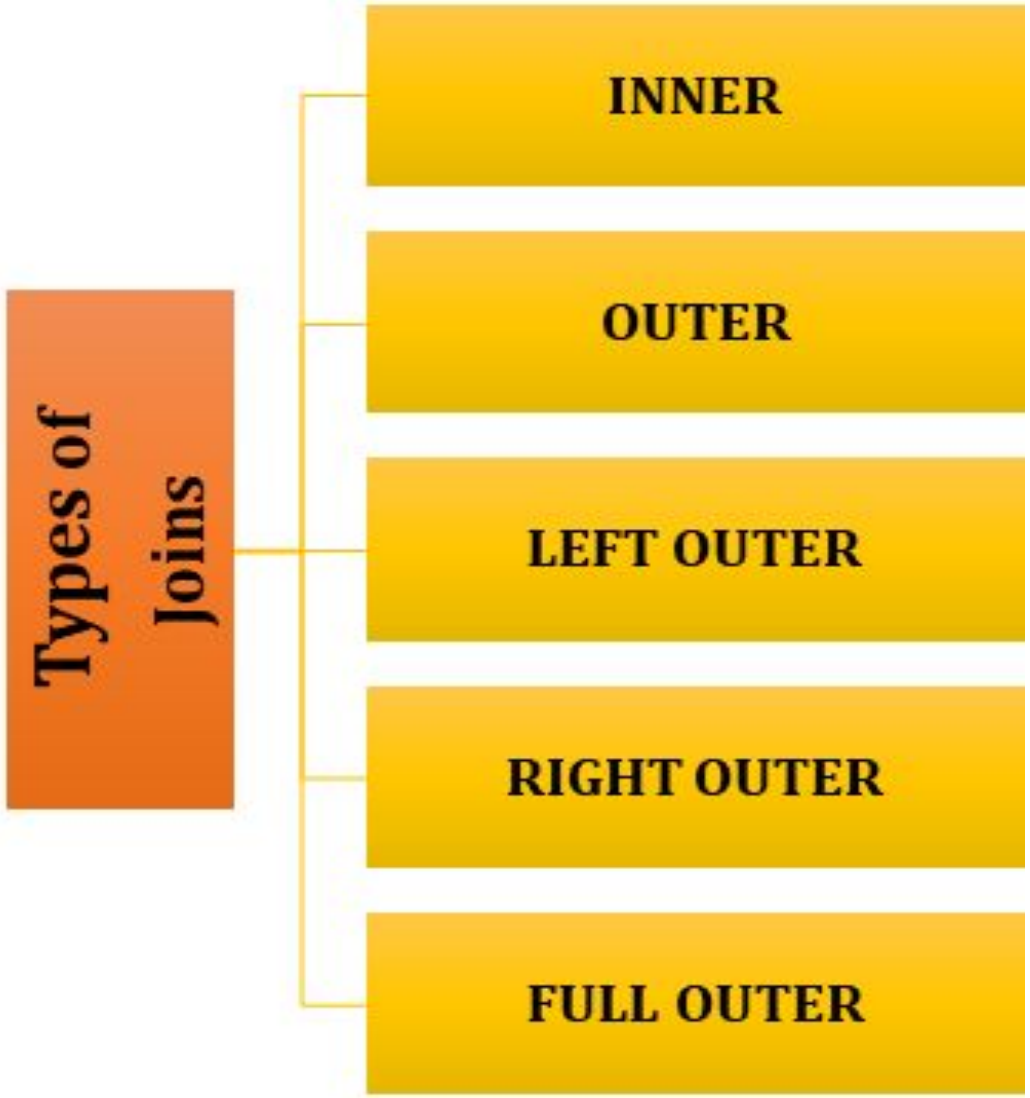
Ex2) To display company names in reverse alphabetical order and the amount in numerical order

Sql>>Select company, amount from orders Order by company desc ,amount desc;

Sql Joins

As the name shows, JOIN means *to combine something*. In case of SQL, JOIN means "**to combine two or more tables**".

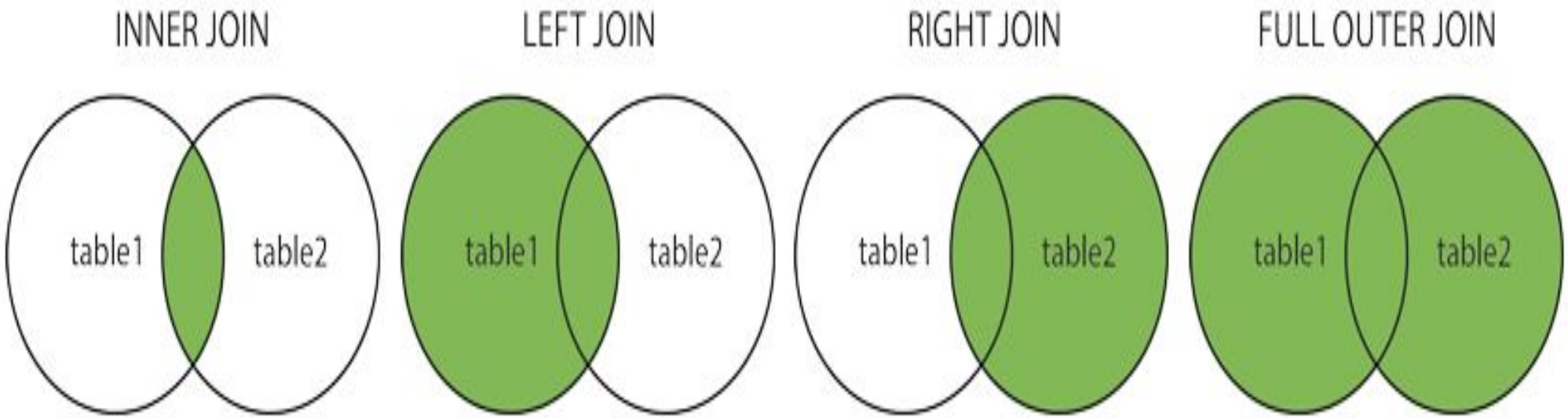
The SQL JOIN clause takes records from two or more tables in a database and combines it together.


Types of Joins
INNER
OUTER
LEFT OUTER
RIGHT OUTER
FULL OUTER

Sql Joins

There are different types of joins available in SQL –

- INNER JOIN – returns rows when there is a match in both tables.
- LEFT outer JOIN – returns all rows from the left table, even if there are no matches in the right table.
- RIGHT outer JOIN – returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN – returns rows when there is a match in one of the tables.
- SELF JOIN – is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN – returns the Cartesian product of the sets of records from the two or more joined tables.


INNER JOIN
table1
table2
LEFT JOIN
table1
table2
RIGHT JOIN
table1
table2
FULL OUTER JOIN
table1
table2

SQL - INNER JOINS

The most important and frequently used of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN.

The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate.

The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Consider the following two tables –

Table 1 – CUSTOMERS Table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 – ORDERS Table

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using the INNER JOIN as follows –

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS INNER JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+----+----------+--------+---------------------+
```

SQL - LEFT OUTER JOINS

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax

```
SELECT table1.column1, table2.column2...

FROM table1

LEFT JOIN table2

  ON table1.common_field = table2.common_field;
```

Consider the following two tables –

Table 1 – CUSTOMERS Table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 – ORDERS Table

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using the Left JOIN as follows –

```
SQL> SELECT  ID, NAME, AMOUNT, DATE   FROM CUSTOMERS
   LEFT JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

SQL - RIGHT OUTER JOINS

The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table. This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax

The basic syntax of a RIGHT JOIN is as follow.

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

Consider the following two tables –

Table 1 – CUSTOMERS Table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 – ORDERS Table

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using the RIGHT JOIN as follows –

```
SQL> SELECT  ID, NAME, AMOUNT, DATE   FROM CUSTOMERS
   RIGHT JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

SQL - FULL JOINS

The SQL FULL JOIN combines the results of both left and right outer joins.

The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.

Syntax

The basic syntax of a FULL JOIN is as follows –

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

Consider the following two tables –

Table 1 – CUSTOMERS Table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 – ORDERS Table

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using the FULL JOIN as follows –

```
SQL> SELECT  ID, NAME, AMOUNT, DATE   FROM CUSTOMERS
  FULL JOIN ORDERS  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

What is an Operator in SQL?

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Range Searching Operators
- Pattern Matching operator
- Set Operators

SQL Arithmetic Operators

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
+	It adds the value of both operands.	a+b will give 30
-	It is used to subtract the right-hand operand from the left-hand operand.	a-b will give 10
*	It is used to multiply the value of both operands.	a*b will give 200
/	It is used to divide the left-hand operand by the right-hand operand.	a/b will give 2
%	It is used to divide the left-hand operand by the right-hand operand and returns reminder.	a%b will give 0

SQL Arithmetic Operators

Examples :

Display salary of employees by adding 5000 to it.

Sql>> select salary + 5000 from emp;

Display salary of employee by subtracting 5000 to it. **Answer**

Display salary of employee by giving 5% raise in salary to employee.

>>Select salary + 0.05 * salary from employee;

Display salary by dividing the salary by 2,

>>Select salary/2 from employe;

SQL Comparison Operators:

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
=	It checks if two operands values are equal or not, if the values are queal then condition becomes true.	(a=b) is not true
!=	It checks if two operands values are equal or not, if values are not equal, then condition becomes true.	(a!=b) is true
<>	It checks if two operands values are equal or not, if values are not equal then condition becomes true.	(a<>b) is true
>	It checks if the left operand value is greater than right operand value, if yes then condition becomes true.	(a>b) is not true
<	It checks if the left operand value is less than right operand value, if yes then condition becomes true.	(a<b) is true

SQL Comparison Operators:

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

>=	It checks if the left operand value is greater than or equal to the right operand value, if yes then condition becomes true.	(a>=b) is not true
<=	It checks if the left operand value is less than or equal to the right operand value, if yes then condition becomes true.	(a<=b) is true
!<	It checks if the left operand value is not less than the right operand value, if yes then condition becomes true.	(a!=b) is not true
!>	It checks if the left operand value is not greater than the right operand value, if yes then condition becomes true.	(a!>b) is true

SQL Comparison Operators:

Company	Order_Number
sega	3412
wipro	2312
TCS	4678
wipro	6798

Eg. Find company of the order_Number 6789
>> select company from order where order_number=6789;

Find company for order_number between 4000 to 7000.
>> Select company from orders where order_number >=4000 and ordernumber>=7000;

SQL Logical Operators

AND and OR join two or more conditions in a WHERE clause.The AND operator display a row if all conditions listed are true.The OR operator displays a row if ANY of the conditions listed are true.

The AND operator allows the existence of multiple conditions in any sql statements WHERE clause.

LastName	FirstName	Address	City
sharma	Arun	Camp 23	Mumbai
Mane	Vijay	GM Road18	Mumbai
Mane	Pooja	Decaan 19	Mumbai

Example1: Find a person with firstname equal to “Pooja **and** the lastname equal to “mane”;
sql>> Select * from persons where firstname=”Pooja” and lastname=”mane”

Example1: Find a person with firstname equal to “Pooja **or** the lastname equal to “mane”;
sql>> Select * from persons where firstname=”Pooja” or lastname=”mane”

Combine And OR :- Select * from persons WHERE(Firstname=”Pooja or Firstname=”Vijay” AND Lastname=”Mane”);

Range Searching Operators

IN Operator

The IN operator may be used if you know the exact value you want to return for at least one of the Columns

Syntax:-select column_name from table_name WHERE column_name IN(value1,value2.);

Example :- To display the persons name with LastName equal to “Sharma” or “Johnson” use the following SQL

Syntax:- select * from Persons WHERE LastName IN(‘Sharma’,’Johnson’);

LastName	FirstName	Address	City
sharma	Arun	Camp 23	Mumbai
Mane	Vijay	GM Road18	Mumbai
Mane	Pooja	Decaan 19	Mumbai

Range Searching Operators

Not IN Operator:- Not IN is negative operator

Syntax:-select column_name from table_name WHERE column_name NOT IN(value1,value2.);

Example :- To display the persons name with LastName NOT equal to “Sharma” or “Johnson” use the following SQL

Syntax:- select * from Persons WHERE LastName NOT IN(‘Sharma’,’Johnson’);

LastName	FirstName	Address	City
sharma	Arun	Camp 23	Mumbai
Gupta	Vijay	GM Road18	Pune
Johnson	Kari	RTO Road20	Banglore
Mane	Pooja	Decaan 19	Mumbai

Range Searching Operators

Between and Not Between Operator:-The between and not between operator select a range of data between two values and not between the two values.These values can be number, text or dates.

The between operator is used to search for value that are within that a set of values, given the minimum value and the maximum value.

Syntax:-select column_name from table_name WHERE column_name between value1 and value2;

Syntax:-select column_name from table_name WHERE column_name not between value1 and value2;

Range Searching Operators

LastName	FirstName	Address	City
sharma	Arun	Camp 23	Mumbai
Gupta	Vijay	GM Road18	Pune
Johnson	Kari	RTO Road20	Banglore
Mane	Pooja	Decaan 19	Mumbai

Example :- To display the persons alphabetically **between** (and including) “sharma” and exclusive “johnson” use the following sql

Select * from persons where LastName Between ‘ Sharma’ and ‘Johnson’;

Example :- To display the persons outside the range used the not operator:

Select * from persons where LastName Not Between ‘ Sharma’ and ‘Johnson’;

Pattern Matching in SQL

- LIKE clause is used to perform the pattern matching task in SQL.
- A WHERE clause is generally preceded by a LIKE clause in an SQL query.
- LIKE clause searches for a match between the patterns in a query with the pattern in the values present in an SQL table. If the match is successful, then that particular value will be retrieved from the SQL table.
- LIKE clause can work with strings and numbers.

The LIKE clause uses the following symbols known as wildcard operators in SQL to perform this pattern-matching task in SQL.

1. To represent zero, one or more than one character, % (percentage) is used.
2. To represent a single character _ (underscore) is used

ID	Name	City	Salary	Age
1	Priyanka Bagul	Nasik	26000	20
2	Riya Sharma	Mumbai	72000	28
3	Neha Verma	Varanasi	37000	19
4	Neeta Desai	Nasik	39500	21
5	Priya Wagh	Udaipur	60000	32

A) Using LIKE clause with % (percentage) (Draw Table)

Example 1: Write a query to display employee details in which name starts with 'Pr

```
SELECT * FROM employee_details WHERE Name LIKE 'Pr%';  '
```

Example 2: Write a query to display employee details in which 'ya' is a substring in a name.

```
SELECT * FROM employee_details WHERE Name LIKE '%ya%';
```

Example 3: Write a query to display employee details in which city name ends with 'i'.;

```
SELECT * FROM employee_details WHERE City LIKE '%i';
```

Example 4: Write a query to display employee details in which age number starts with 2.

```
SELECT * FROM employee_details WHERE Age LIKE '2%';
```

Example 5:

Write a query to display employee details in which salary contains a number 50 in between.

```
SELECT * FROM employee_details WHERE Salary LIKE '%50%';
```

(B) Using LIKE clause with _ (underscore)

Example 1:

Write a query to display employee details in which city name starts with 'Na', ends with 'ik', and contains any single character between 'Na' and 'ik'.

Query:

```
mysql> SELECT * FROM employee_details WHERE City LIKE 'Na_ik';
```

Example 2:

Write a query to display employee details in which salary contains a number starting with '3' succeeding any two digits and finally ends with '00'.

```
SELECT * FROM employee_details WHERE Salary LIKE '3__00';
```

(C) Using LIKE clause with % and _ operator in a single query

Example 1: Write a query to display employee details in which employee name contains 'a' at fifth position.

Query:

```
1. mysql> SELECT * FROM employee_details WHERE Name LIKE '___a%';
```

Example 2:

Write a query to display employee details in which salary contains a substring '00' starting from the 5^{th} position.

Query:

```
1. mysql> SELECT * FROM employee_details WHERE Salary LIKE '_00%';
```

(D) Using LIKE clause with NOT operator

Example 1:

Write a query to display employee details in which employee name is not like 'Priya'.

Query:

```
1. mysql> SELECT * FROM employee_details WHERE Name NOT LIKE 'Priya%';
```

Example 2:

Write a query to display employee details in which employee name is not like any single character followed by 'Na' and ending with 'ik'.

Query:

```
1. mysql> SELECT * FROM employee_details WHERE City NOT LIKE 'Na_ik';
```

SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

Syntax

The basic syntax of a table alias is as follows.

```
SELECT column1, column2....FROM table_name AS alias_name WHERE [condition];
```

The basic syntax of a column alias is as follows.

```
SELECT column_name AS alias_name FROM table_name  WHERE [condition];
```

LastName	FirstName	Address	City
sharma	Arun	Camp 23	Mumbai
Gupta	Vijay	GM Road18	Pune
Mane	Pooja	Decaan 19	Mumbai

Example :Using a Column Alias:

Select LastName AS Family, Firstname AS Name from Persons:

Family	Name
sharma	Arun
Gupta	Vijay
Mane	Pooja

LastName	FirstName	Address	City
sharma	Arun	Camp 23	Mumbai
Gupta	Vijay	GM Road18	Pune
Mane	Pooja	Decaan 19	Mumbai

Example :Using a Table Alias:

Select LastName,Firstname from person AS Employee:

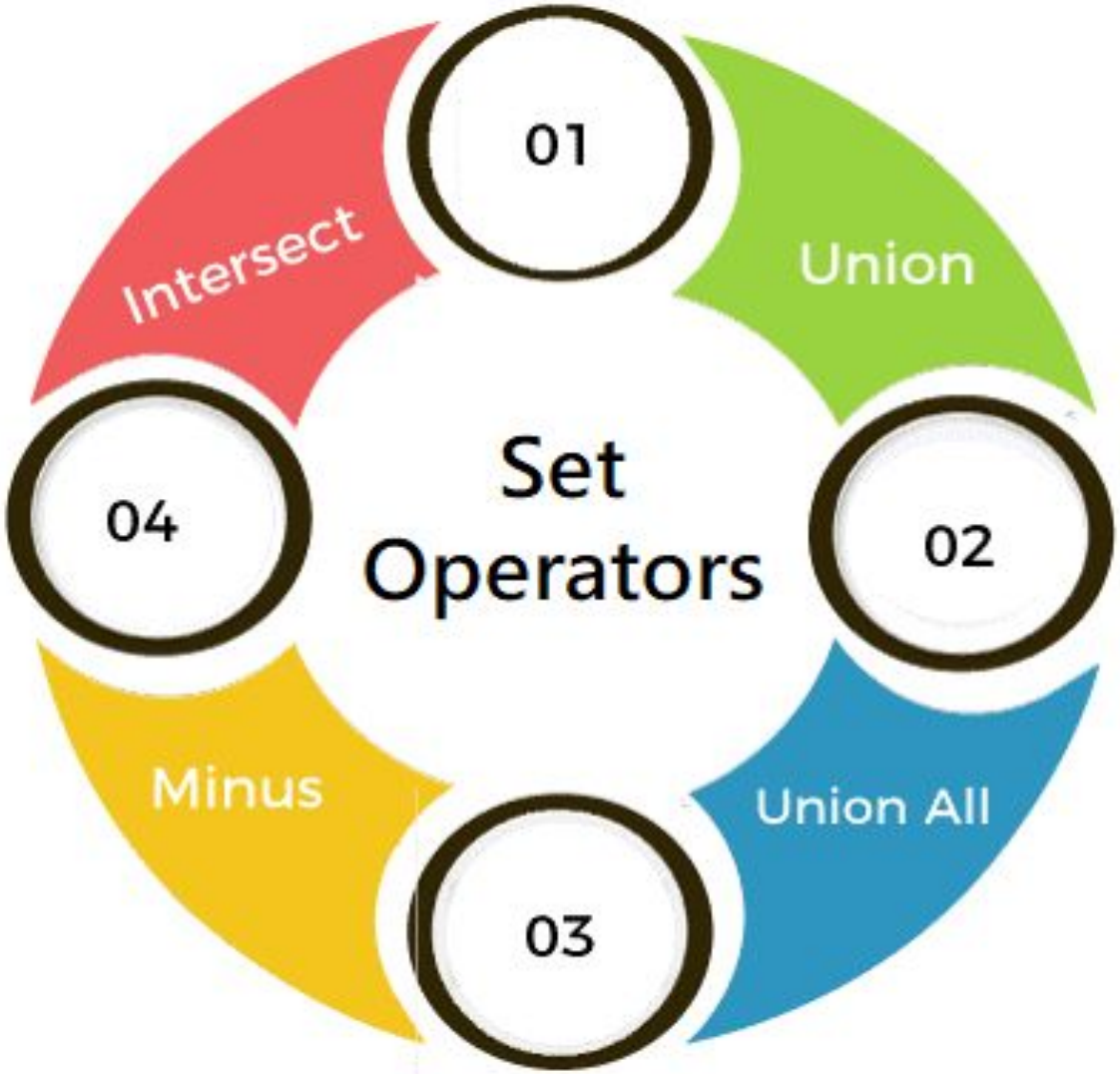
Employee

Lastname	Firstname
sharma	Arun
Gupta	Vijay
Mane	Pooja

SET Operators in SQL

There are certain rules which must be followed to perform operations using SET operators in SQL. Rules are as follows:

1. **The number and order of columns must be the same.**
2. **Data types must be compatible.**



1. UNION:

- UNION will be used to combine the result of two select statements.
- Duplicate rows will be eliminated from the results obtained after performing the UNION operation.
- The UNION operator is used to combine the result-set of two or more SELECT statements.
- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

ID	Name	Department	Salary	Year_of_Experienc
1	Aakash Singh	Development	72000	2
2	Abhishek Pawar	Production	45000	1
3	Pranav Deshmukh	HR	59900	3
4	Shubham Mahale	Accounts	57000	2
5	Sunil Kulkarni	Development	87000	3
6	Bhushan Wagh	R&D	75000	2
7	Paras Jaiswal	Marketing	32000	1

ID	Name	Department	Salary	Year_of_Experience
1	Prashant Wagh	R&D	49000	1
2	Abhishek Pawar	Production	45000	1
3	Gautam Jain	Development	56000	4
4	Shubham Mahale	Accounts	57000	2
5	Rahul Thakur	Production	76000	4
6	Bhushan Wagh	R&D	75000	2
7	Anand Singh	Marketing	28000	1

SQL Statement>>SELECT * FROM t_employees UNION SELECT *FROM t2_employees;

ID	Name	Department	Salary	Year_of_Experience
1	Aakash Singh	Development	72000	2
2	Abhishek Pawar	Production	45000	1
3	Pranav Deshmukh	HR	59900	3
4	Shubham Mahale	Accounts	57000	2
5	Sunil Kulkarni	Development	87000	3
6	Bhushan Wagh	R&D	75000	2
7	Paras Jaiswal	Marketing	32000	1
1	Prashant Wagh	R&D	49000	1
3	Gautam Jain	Development	56000	4
5	Rahul Thakur	Production	76000	4
7	Anand Singh	Marketing	28000	1

2. UNION ALL:

- This operator combines all the records from both the queries.
- Duplicate rows will be not be eliminated from the results obtained after performing the UNION ALL operation.

Example 1:

Write a query to perform union all operation between the table t_employees and the table t2_employees.

Query:

1. mysql> SELECT *FROM t_employees UNION ALL SELECT *FROM t2_employees;

ID	Name	Department	Salary	Year_of_Experience
1	Aakash Singh	Development	72000	2
2	Abhishek Pawar	Production	45000	1
3	Pranav Deshmukh	HR	59900	3
4	Shubham Mahale	Accounts	57000	2
5	Sunil Kulkarni	Development	87000	3
6	Bhushan Wagh	R&D	75000	2
7	Paras Jaiswal	Marketing	32000	1
1	Prashant Wagh	R&D	49000	1
2	Abhishek Pawar	Production	45000	1
3	Gautam Jain	Development	56000	4
4	Shubham Mahale	Accounts	57000	2
5	Rahul Thakur	Production	76000	4
6	Bhushan Wagh	R&D	75000	2
7	Anand Singh	Marketing	28000	1

3. INTERSECT:

- It is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements.

Example 1:

Write a query to perform intersect operation between the table t_employees and the table t2_employees.

Query:

1. mysql> SELECT * FROM t_employees INTERSECT SELECT * FROM t2_employees;

ID	Name	Hometown	Percentage	Favourite_Subject
2	Abhishek Pawar	Production	45000	1
4	Shubham Mahale	Accounts	57000	2
6	Bhushan Wagh	R&D	75000	2

4. MINUS

- It displays the rows which are present in the first query but absent in the second query with no duplicates.

Example 1:

Write a query to perform a minus operation between the table t_employees and the table t2_employees.

Query:

1. mysql> SELECT * FROM t_employees MINUS SELECT * FROM t2_employees;

ID	Name	Department	Salary	Year_of_Experience
1	Aakash Singh	Development	72000	2
3	Pranav Deshmukh	HR	59900	3
5	Sunil Kulkarni	Development	87000	3
7	Paras Jaiswal	Marketing	32000	1

Since we have performed Minus operation between both the tables, so only the unmatched records from both the tables are displayed.

There are four types of single row functions. They are:

1) **Numeric Functions:** These are functions that accept numeric input and return numeric values.

2) **Character or Text Functions:** These are functions that accept character input and can return both character and number values.

3) **Date Functions:** These are functions that take values that are of datatype DATE as input and return values of datatype DATE, except for the MONTHS_BETWEEN function, which returns a number.

4) **Conversion Functions:** These are functions that help us to convert a value in one form to another form. For Example: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE etc.

You can combine more than one function together in an expression. This is known as nesting of functions.

String Functions:

Character or text functions are used to manipulate text strings. They accept strings or characters as input and can return both character and number values as output.

Few of the character or text functions are as given below:

Function Name	Return Value
LOWER (string_value)	All the letters in *'string_value'* is converted to lowercase.
UPPER (string_value)	All the letters in *'string_value'* is converted to uppercase.
INITCAP (string_value)	All the letters in *'string_value'* is converted to mixed case.
LTRIM (string_value, trim_text)	All occurrences of *'trim_text'* is removed from the left of *'string_value'*.
RTRIM (string_value, trim_text)	All occurrences of *'trim_text'* is removed from the right of *'string_value'* .
TRIM (trim_text FROM string_value)	All occurrences of *'trim_text'* from the left and right of *'string_value'* , *'trim_text'* can also be only one character long
SUBSTR (string_value, m, n)	Returns *'n'* number of characters from *'string_value'* starting from the *'m'* position.
LENGTH (string_value)	Number of characters in *'string_value'* in returned.
LPAD (string_value, n, pad_value)	Returns *'string_value'* left-padded with *'pad_value'* . The length of the whole string will be of *'n'* characters.
RPAD (string_value, n, pad_value)	Returns *'string_value'* right-padded with *'pad_value'* . The length of the whole string will be of *'n'* characters.

Table Name :- Person

LastName	FirstName	Address	City
Sharma	Oli	GM Road 10	Pune
Mante	Edison	Camp 23	Pune
Johnson	Kari	RTO Road 20	Banglore

1) InitCap: It Display the initial letter as Capital.

SYNTAX:-Initcap(String)

Ex:- SELECT initcap(City) from Person;

City
Pune
Pune
Banglore

Table Name :- Person

LastName	FirstName	Address	City
Sharma	Oli	GM Road 10	Pune
Mante	Edison	Camp 23	Pune
Johnson	Kari	RTO Road 20	Banglore

2) Lower and upper: It Display the string into either lower or uppercase.

SYNTAX:-lower(String)
upper(String)

Ex:- SELECT lower(City) from Person;
SELECT upper(City) from Person;

City

Table Name :- Person

LastName	FirstName	Address	City
Sharma	Oli	GM Road 10	Pune
Mante	Edison	Camp 23	Pune
Johnson	Kari	RTO Road 20	Banglore

3) Ltrim And Rtim: It trims or cuts the character from left or right.

SYNTAX:-ltrim(String,trim text)
rtrim(String,trim text)

Ex:- SELECT ltrim('Pune', 'P') from Person;
SELECT rtrim('Pune', 'e') from Person;

City

Table Name :- Person

LastName	FirstName	Address	City
Sharma	Oli	GM Road 10	Pune
Mante	Edison	Camp 23	Pune
Johnson	Kari	RTO Road 20	Banglore

4) TRIM (trim_text FROM string_value)

SYNTAX:-trim(String,trim text)

Ex:- SELECT trim(‘o’) from Person;

City

Table Name :- Person

LastName	FirstName	Address	City
Sharma	Oli	GM Road 10	Pune
Mante	Edison	Camp 23	Pune
Johnson	Kari	RTO Road 20	Banglore

5) Substr:-It returns the substring from the specified position

SYNTAX:-Substr(Main String, Position, Characters to be replaced)

Ex:- SELECT substr(City,1,3) from Person;

City
Pun
Pun
Ban

Table Name :- Person

LastName	FirstName	Address	City
Sharma	Oli	GM Road 10	Pune
Mante	Edison	Camp 23	Pune
Johnson	Kari	RTO Road 20	Banglore

6) Lpad and Rpad:-It is used for formatting from left and right side by using any character

SYNTAX:-
lpad(Main string, length , character to be padded)
rpad(Main string, length , character to be padded)
Length is including the total size of the city column

Ex:- SELECT lpad(City,15,'*') from Person;

City
********Pune
********Pune
******Banglore

Table Name :- Person

LastName	FirstName	Address	City
Sharma	Oli	GM Road 10	Pune
Mante	Edison	Camp 23	Pune
Johnson	Kari	RTO Road 20	Banglore

7) Concatenation:-It is used to Concatenate several expression together.

SYNTAX:-
CONCAT(expr1,expr2,expr3,……)

Ex:- SELECT concat(firstName,City) from Person;

Oli Pune
Edison Pune
Kari Banglore

Table Name :- Person

LastName	FirstName	Address	City
Sharma	Oli	GM Road 10	Pune
Mante	Edison	Camp 23	Pune
Johnson	Kari	RTO Road 20	Banglore

8) Length:-It is used to Calculate the length of the given string.

SYNTAX:-
Length(string)

Ex:- SELECT length(City) from Person;

City
4
4
8

Numeric or Arithmetic Functions:

Numeric functions are used to perform operations on numbers.

They accept numeric values as input and return numeric values as output.

Few of the Numeric functions

Function Name	**Return Value**
ABS (x)	Absolute value of the number '*x*'
CEIL (x)	Integer value that is Greater than or equal to the number '*x*'
FLOOR (x)	Integer value that is Less than or equal to the number '*x*'
TRUNC (x, y)	Truncates value of number '*x*' up to '*y*' decimal places
ROUND (x, y)	Rounded off value of the number '*x*' up to the number '*y*' decimal places

Example:

1) ABS():

This SQL ABS() returns the absolute value of a number passed as argument.

Ex:- SELECT ABS(-17.38) FROM dual;

Output:- 17.38

2) CEIL():

This SQL CEIL() will rounded up any positive or negative decimal value within the function upwards.

Ex:- SELECT CEIL(-17.38) FROM dual;

Output:- 18

Example:

3) FLOOR():

This SQL FLOOR() rounded up any positive or negative decimal value down to the next least integer value.

Ex:- SELECT FLOOR(17.38) FROM dual;

Output:- 17

4) Trunc(x,y):

Truncates the value of number x to y decima places.

Ex:- SELECT TRUNC(125.456,1) FROM dual;

Output:- 125.4

Example:

5) Round(x,y):

Round the value of number x to y decimal places.

Ex:- SELECT Round (140.234,2) FROM dual;

Output:- 140.23

6) Exp(x):

Returns e raised to the xth power where e =2.71828183

Ex:- SELECT EXP (5) FROM dual;

Output:- 148.413159

Example:

7) Power(x,y):

Returns x raised to the nth power .y must be an integer else an error is returned.

Ex:- SELECT Power (4,2) FROM dual;

Output:- 16

8) SQRT(x):

Returns square root of x

Ex:- SELECT sqrt (25) FROM dual;

Output:- 5

Example:

9) Mod(x,y):

Returns the remainder of a first number divided by second no. passes a parameter.If the second no. is zero the result of the same as the first no.

Ex:- SELECT Mod (10,3) FROM dual;

Output:- 1

Date and Time Functions:

These are functions that take values that are of data type DATE as input and return value of data types DATE,expect for the months_between function , which return a number as output:

1) Add_Months:-Adds the specified number of months to the date value

   Syntax:- Add-Months(date,number of months)

   Ex;-Select add_months(sysdate,5) from dual;

   This will add 5 months to the existing date.

Date and Time Functions:

2) Greatest :-Returns the greatest value in a list of expression

Syntax:- Greatest (expr1 ,expr2 ,expr3)

Ex;-Select Greatest('10-JAN-07','12-OCT-07') from dual; Output;-

3) Least :-Returns the Least value in a list of expression

Syntax:- Least(expr1 ,expr2 ,expr3)

Ex;-Select least('10-JAN-07','12-OCT-07') from dual; Output;-

4) Datediff:- Return the difference in days between two days values.

Syntax:- Datediff(date1 ,date2)

Ex;-Select datediff('10-JAN-07','12-OCT-07') from dual; Output;

Aggregate Functions:

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

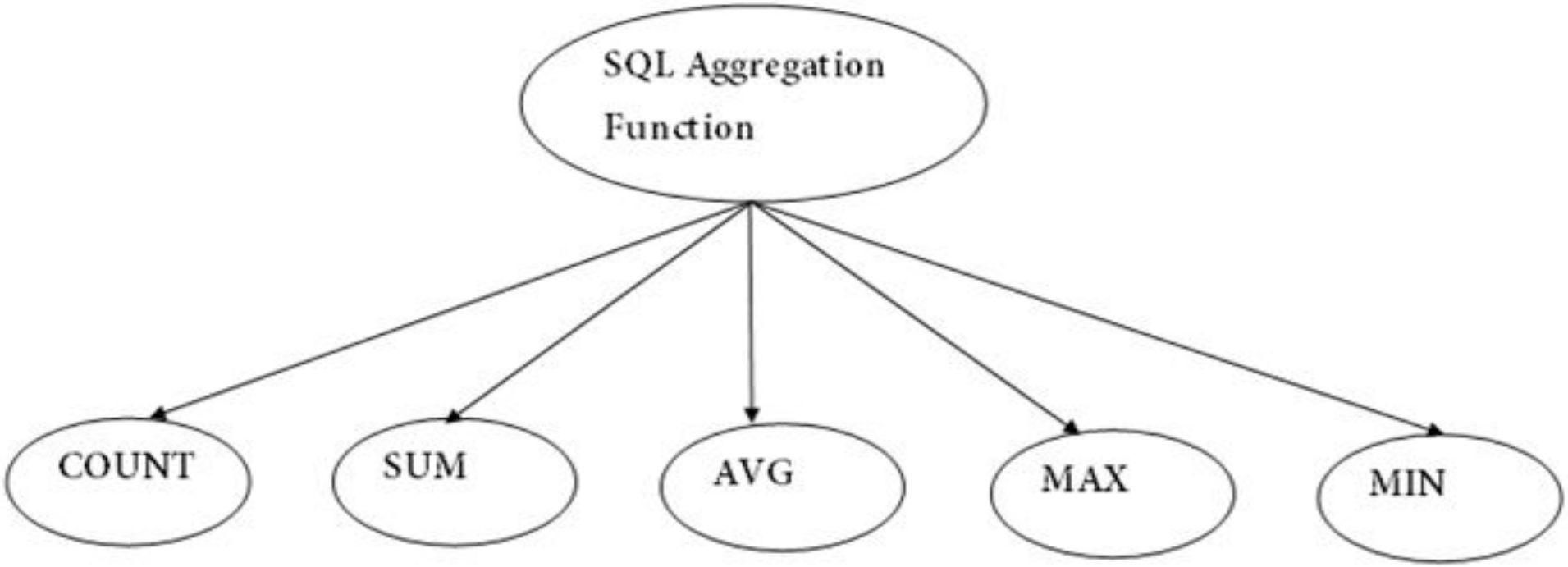


Table Name :- Persons

LastName	FirstName	Age	City
Sharma	Oli	34	Pune
Mante	Edison	45	Pune
Johnson	Kari	19	Banglore

1) AVG: The Avg function returns the average value of a column in a selection.NULL Values are not included in the calculation.

SYNTAX:-Select avg(column) from table_name;

Ex1: Find the average age of the persons in the persons table:

Sql >>Select AVG(Age) from persons;

Values
32.67
39.5

Exp2:Find the average age for the persons that are older than 20 years:

Sql>> Select AVG(Age) from persons where Age>20;

Table Name :- Persons

LastName	FirstName	Age	City
Sharma	Oli	34	Pune
Mante	Edison		Pune
Johnson	Kari	19	Banglore

2) Count: The Count function returns the number of rows without a null value in the specified column.

SYNTAX:-Select count(column) from table_name;

Values
2

Ex1: Find the number of persons with a value in the age filed in the persons table:
Sql >>Select count(Age) from persons;

Table Name :- Persons

LastName	FirstName	Age	City
Sharma	Oli	34	Pune
Mante	Edison	45	Pune
Johnson	Kari	19	Banglore

3) Count(*) Function: The Count(*) function returns the number of selected rows in a selection.

SYNTAX:-Select count(*) from table_name;

Values
3

Ex1: Find the number of persons with a value in the age filed in the persons table:
Sql >>Select count(*) from persons;
Ex2: Find the number of persons that are older than 20 yrs:
Sql>> Select count(*) from persons where age>>20;

Table Name :- Persons

LastName	FirstName	Age	City
Sharma	Oli	34	Pune
Mante	Edison	45	Pune
Johnson	Kari	19	Banglore

4. Max Function: The max function returns the highest value in a column.Null Values are not included in the calculation
SYNTAX:-Select max(column) from table_name;

Values
45

Ex1: Find the maximum age from the persons table:
Sql >>Select max(age) from persons;

Table Name :- Persons

LastName	FirstName	Age	City
Sharma	Oli	34	Pune
Mante	Edison	45	Pune
Johnson	Kari	19	Banglore

5. Min Function: The min function returns the lowest value in a column.Null Values are not included in the calculation

SYNTAX:-Select min(column) from table_name;

Ex1: Find the minimum age from the persons table:

Sql >>Select min(age) from persons;

Values
19

Table Name :- Persons

LastName	FirstName	Age	City
Sharma	Oli	34	Pune
Mante	Edison	45	Pune
Johnson	Kari	19	Banglore

6. Sum Function: The sum function returns the total sum of a column in a given selection.Null Values are not included in the calculation
SYNTAX:-Select sum(column) from table_name;

Ex1: Find the sum of all ages from the persons table:
Sql >>Select sum(age) from persons;
Ex2: Find the sum of ages for persons that are more than 20 yrs old
Sql>> Select sum(age) from persons where age >20;

Values
98
79

SQL -Sub Query

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.

SQL -Sub Query

- **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

Consider the EMPLOYEE table 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	6500.00
5	Kathrin	34	Bangalore	8500.00
6	Harry	42	China	4500.00
7	Jackson	25	Mizoram	10000.00

```
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

SQL -Sub Query

2. 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

Consider the EMPLOYEE table 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	6500.00
5	Kathrin	34	Bangalore	8500.00
6	Harry	42	China	4500.00
7	Jackson	25	Mizoram	10000.00

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

SQL -Sub Query

3. 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

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
CUSTOMERS_BKP  WHERE AGE >= 29);
```

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

SQL -Sub Query

4. 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

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 );
```

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

3.5 Views

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

Views, which are a type of virtual tables allow users to do the following –

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows –

CREATE VIEW view_name AS SELECT column1, column2.....

FROM table_name WHERE [condition];

Example

Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SQL > CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age FROM CUSTOMERS;

SQL > SELECT * FROM CUSTOMERS_VIEW;

name	age
Ramesh	32
Khilan	25
kaushik	23
Chaitali	25
Hardik	27
Komal	22
Muffy	24

Updating Views

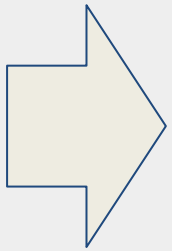
A view can be updated under certain conditions which are given below –

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

Example

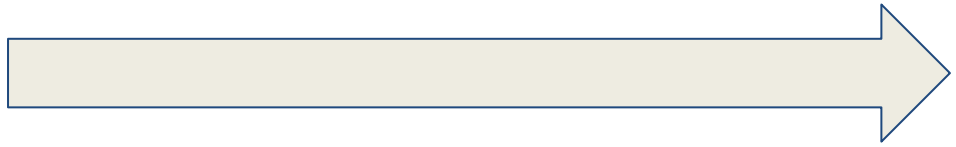
Consider the CUSTOMERS table having the following records −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



```
SQL > UPDATE
CUSTOMERS_VIEW
  SET AGE = 35
 WHERE name = 'Ramesh';
```

```
SQL > SELECT * FROM
CUSTOMERS_VIEW;
```



OUTPUT

Inserting Rows into a View

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here, we cannot insert rows in the CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.

Deleting Rows into a View

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE = 22.

SQL > DELETE FROM CUSTOMERS_VIEW WHERE age = 22;

Dropping Views

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple and is given below –

```
DROP VIEW view_name;
```

Following is an example to drop the CUSTOMERS_VIEW from the CUSTOMERS table.

```
DROP VIEW CUSTOMERS_VIEW;
```

SEQUENCES

Sequence is a set of integers 1, 2, 3, … that are generated and supported by some database systems to produce unique values on demand.

- A sequence is a user defined schema bound object that generates a sequence of numeric values.
- Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.
- The sequence of numeric values is generated in an a**scending or descending order** at defined intervals and can be configured to restart when exceeds max_value.

Syntax:

```
CREATE SEQUENCE
sequence_name

START WITH
initial_value

INCREMENT BY
increment_value

MINVALUE minimum
value

MAXVALUE maximum
value

CYCLE | NOCYCLE ;
```

sequence_name: Name of the sequence.

initial_value: starting value from where the sequence starts. Initial_value should be greater than or equal to minimum value and less than equal to maximum value.

increment_value: Value by which sequence will increment itself. Increment_value can be positive or negative.

minimum_value: Minimum value of the sequence.

maximum_value: Maximum value of the sequence.

cycle: When sequence reaches its set_limit it starts from beginning.

nocycle: An exception will be thrown if sequence exceeds its max_value.

Example 1:

```
CREATE SEQUENCE
EID

start with 1

increment by 1

minvalue 0

maxvalue 100

cycle;
```

CREATE TABLE students (ID number (10) Primary key ,NAME char(20));

Now insert values into table

INSERT into students VALUES(EID.nextval,'Ramesh');

INSERT into students VALUES(EID.nextval,'Suresh');

where *EID.nextval* will insert id's in id column in a sequence as defined in sequence_1.

ID	NAME
1	Ramesh
2	Suresh

ALTER SEQUENCE

Use the ALTER SEQUENCE statement to change the increment, minimum and maximum values, cached numbers, and behavior of an existing sequence.

```
ALTER SEQUENCE  sequence_name
    [ START  WITH Start_num ]
    [ INCREMENT BY increment_num ]
    [ { MINVALUE minimum_num  | NO MINVALUE } ]
    [ { MAXVALUE maximum_num |NO MAXVALUE } ]
    [ CYCLE | { NO CYCLE } ]
    [ { CACHE  cache_num | NO CACHE } ]
    [ {ORDER | NOORDER}];
```

Maxvalue=1500

Dropping Sequences

If a sequence is no longer required, you can drop the sequence using the DROP Sequence statement.

Syntax:-DROP Sequence;

Following is an example to drop the EID Sequence

SQL> Drop Sequence EID;

INDEXES

Indexes are optional structures associated with tables and clusters that allow sql statements to execute more quickly against a table.

Just as the index in this manual helps you locate information faster than if there were no index, an oracle index provides a faster access path to table data.

You can use indexes without rewriting any queries.

Being independent structures,they require storage space. You can create or drop an Index without affecting the base tables,database applications, or other indexes.

Oracle automatically maintains indexes when you insert, update and delete rows of the associated tables.

The CREATE INDEX Command

The basic syntax of a **CREATE INDEX** is as follows.

CREATE INDEX index_name ON table_name;

1.Single-Column Indexes

A single-column index is created based on only one table column. The basic syntax is as follows.

Syntax:-CREATE INDEX index_name ON table_name (column_name);

Sql> Create index emp on employee(emp_id);

2.Unique Indexes

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows.

CREATE UNIQUE INDEX index_name on table_name (column_name);

Sql> Create unique index emp on employee(emp_id);

3.**Compsite Indexes**

A composite index is an index on two or more columns of a table.

A composite index also called a concatenated index.

Syntax:-CREATE INDEX index_name ON table_name (column_name1,column_name 2);

Sql> Create index emp on employee(emp_id,Dept_id);

DROPPING INDEXES

An index can be dropped using SQL **DROP** command. Care should be taken when dropping an index because the performance may either slow down or improve.

The basic syntax is as follows –Syntax:-DROP INDEX index_name;

Synonyms

A Synonyms is an alias for a table, view,snapshot,sequence, procedure,function,or package.

Synonyms can provide a level of security by masking the name and owner of an object and by providing location transparency for remote objects of a distributed database.

They are convenient to use and reduce the complexity of sql statements for database users.

You can create both public and private synonyms. A public synonym is owned by the special user group named public is accessible to every user in a database.

A private synonym is contained in the schema of a specific user and available only to the user and the user's grantees.

Creating Synonyms

- To create a private synonym you must have the create synonym privilege.
- To create a public synonym you must have the create public synonym system privilege.

The syntax for creating the synonym is:

Create synonym synonym_name for table_name;

Example:- The following example create the synonym emp for the relation employee.

SQL> Create synonym emp for employee;

Synonym Created. SQL> Select * from emp;

*You can insert the rows in the synonym emp as you insert the rows in the employee table.

Dropping Synonyms

- If a synonym is no longer required,you can drop the sequence using the drop synonym statement.
- The original table for which synonym is created does not get deleted when you delete the synonym.
- When you drop a synonym, its definition is removed from the data dictionary.

The syntax for dropping the synonym is:

Drop synonym synonym_name;

Example:- The following example drop the synonym emp for the relation employee.

SQL> Drop synonym emp;

Synonym dropped.