SQL COMMANDS

DATA DEFINITION LANGUAGE (DDL)

DATATYPE	DESCRIPTION	USAGE
CHAR	string(0-255), can store characters of fixed length	CHAR(50)
VARCHAR	string(0-255), can store characters up to given length	VARCHAR(50)
BLOB	string(0-65535), can store binary large object	BLOB(1000)
INT	integer(-2,147,483,648 to 2,147,483,647)	INT
TINYINT	integer(-128 to 127)	TINYINT
BIGINT	integer(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)	BIGINT
BIT	can store x-bit values. x can range from 1 to 64	BIT(2)
FLOAT	Decimal number - with precision to 23 digits	FLOAT
DOUBLE	Decimal number - with 24 to 53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN
DATE	date in format of YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31	DATE
TIME	HH:MM:SS	TIME
YEAR	year in 4 digits format ranging from 1901 to 2155	YEAR

CREATE COMMAND: Used for creating objects in the database.

1.) CREATE DATABASE <database_name>
USE <database_name>

2.) CREATE TABLE <table_name>
(column_name1 datatype,
column_name2 datatype,
.
.
column_name_n datatype

DROP COMMAND: Allows to remove entire database objects from the database. It removes the entire data structure from the database.

1.) DROP DATABASE <database_name>

2.) DROP TABLE <table_name>

ALTER COMMAND : Allows to alter or modify the structure of the database.

1.) ALTER TABLE <table_name>
ADD <column_name datatype>;

2.) ALTER TABLE <table_name>
CHANGE <old_col_name> <new_col_name>;

3.) ALTER TABLE <table_name>
MODIFY COLUMN <col_name> <data_type>;

4.) ALTER TABLE <table_name>
DROP COLUMN <column_name>;

RENAME COMMAND : Used to rename an object.

1.) RENAME TABLE <old_table_name> TO <new_table_name>;

TRUNCATE COMMAND : Used to delete all the rows from the table permanently.

1.) TRUNCATE TABLE <table_name>;

DATA MANIPULATION LANGUAGE (DML)

SELECT COMMAND : Used to retrieve data from the database.

Clause	Description
WHERE	It specifies which rows to retrieve.
GROUP BY	It is used to arrange the data into groups.
HAVING	It selects among the groups defined by the GROUP BY clause.
ORDER BY	It specifies an order in which to return the rows.
AS	It provides an alias which can be used to temporarily rename tables or columns.

1.) SELECT * FROM <table_name>;

2.) SELECT * FROM <table_name>
WHERE salary >= 10000;

3.) SELECT *
FROM student
where s_marks BETWEEN 75 and 80; //Selects numbers between 75 and 80

4.) SELECT *
FROM Students
WHERE NOT city='Pune' //Negation Operator

5.) SELECT column1,column2
FROM table
WHERE column LIKE pattern

PATTERN	DESCRIPTION
LIKE 'a%'	Customer name starting with letter 'a'
LIKE '%a'	Customer name ending with letter 'a'
LIKE '__a%'	Customer name having third letter as 'a'
LIKE 'a__%'	Starting with a and having atleast 3 characters in length
LIKE 'a%b'	Customer name starting with letter 'a' and ending with 'b'

6.) SELECT * FROM Employee
WHERE e_city IN ('Mumbai','Pune'); // Selects multiple values in WHERE

INSERT Command: Used for inserting data into a table. Using this command, you can add one or more records to any single table in a database. It is also used to add records to existing code.

1.) INSERT INTO <table_name>
VALUES
(value1, value2, …, valuen);

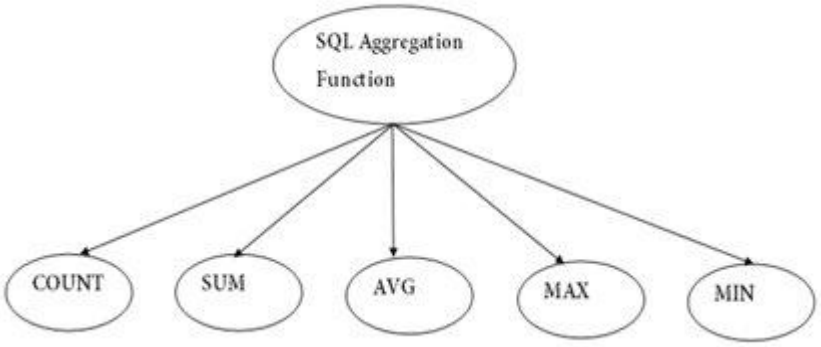
UPDATE COMMAND: Used to modify the records present in the existing table.

1.) UPDATE <table_name>
SET <column_name> = value
WHERE condition;

DELETE COMMAND: Used to delete some or all records from the existing table.

1.) DELETE FROM <table_name>
WHERE condition;
{ If “WHERE” is not added, then all the rows will get deleted. }

AGGREGATE FUNCTIONS (USE GROUP BY)



1.) SELECT **MIN(column_name)**
FROM table_name
WHERE condition;

2.) SELECT **MAX(column_name)**
FROM table_name
WHERE condition;

3.) SELECT **COUNT(column_name)**
FROM table_name
GROUP BY column_name
HAVING condition;

4.) SELECT **AVG(column_name)**
FROM table_name
GROUP BY column_name
HAVING condition;

5.) SELECT **SUM(column_name)**
FROM table_name
GROUP BY column_name
HAVING condition;

ORDER BY (SORT)

1.) SELECT *
FROM table_name
ORDER BY column_name ASC/DESC //By default, ASC

GROUP BY : The GROUP BY statement groups rows that have the same values into summary rows

{**HAVING :** Having is used in the combination of Group by to restrict the groups of returned rows to only those whose the condition is true.}

1.) SELECT column_name
FROM table_name
WHERE condition
GROUP BY column_name;

2.) SELECT DeptID, AVG(salary)
FROM Employee
GROUP BY DeptID
HAVING AVG(salary) > 3000;

ALIASES : Temporary name to tables.

1.) SELECT s_name
FROM Student **s**, Book **b**

SQL - JOINS

CROSS JOIN : CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used along with CROSS JOIN. This kind of result is called as Cartesian Product. Means for every value in table1 there will be all the rows in table2.

```
SELECT *
FROM table1
CROSS JOIN table2;
```

INNER JOIN : Basically a CROSS JOIN with a condition which selects all rows from both the tables as long as the condition staisfies

```
SELECT tabl1.column1, table1.column2, table2.column1, …
FROM table1 INNER JOIN table2
ON
table1.matching_column = table2.matching_column;
```

NATURAL JOIN : Columns with the same name of the associated tables will appear only once. (DON'T USE ON)

```
SELECT *
FROM table1
NATURAL JOIN table2;
```

LEFT JOIN : Returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join.

```
SELECT tabl1.column1, table1.column2, table2.column1, …
FROM table1 LEFT JOIN table2
ON
table1.matching_column = table2.matching_column;
```

RIGHT JOIN : Returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join.

```
SELECT tabl1.column1, table1.column2, table2.column1, …
FROM table1 RIGHT JOIN table2
ON
table1.matching_column = table2.matching_column;
```

FULL JOIN : Returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join.

```
SELECT tabl1.column1, table1.column2, table2.column1, …
FROM table1 RIGHT JOIN table2
ON
table1.matching_column = table2.matching_column;
```

SELF JOIN : A SELF JOIN is another type of join in SQL that is used to join a table to itself, especially when the table has a FOREIGN KEY that references its own PRIMARY KEY.

```
SELECT a.company_name,b.company_name,a.company_city
FROM company a, company b
WHERE a.company_city=b.company_city;
```