By Sumit Ojha

Introduction to SQL :

SQL Structured Query language is a programming language designed to manage data stored in relational databases.
SQL operates through simple, declarative statements. This keeps data accurate and secure, and helps maintain the integrity of databases, regardless of size.

Relational database:- It's a database that organizes information into one or more tables.

All data stored in a relational database is of a certain data. Some of the most common data types are:

i) Integer, text, date, Real etc

Statements :- A statement is text that the database recognizes as a valid command. Statements always ends in a semicolon ;

Eg:- CREATE TABLE table_name (
    column_1 data_type,
    column_2 data_type,
    column_3 data_type,);

Let break down the Components of a Statement

1. CREATE_TABLE is a clause. Clauses perform specific tasks in SQL. By convention, clauses are written in Capital letters. Clauses can also be referred to as Commands.
2. table_name refers to the name of the table that the Command is applied to.
3. (Column_1 data, ...) is a parameters. A Parameters is a list of Columns, data types, or Values that are passed to a clause as an argument. here, the Parameters is a list of Column names & the associated data types.

CREATE :- Create Statement allows us to Create a new table in the database. You can use the Create Statement anytime you want to create a new table from Scratch.

Ex:- CREATE TABLE celebs (
id INTEGER, name TEXT, age INTEGER);

1. CREATE TABLE is a clause that tells SQL u want to create a new table.
2. Celebs is the name of the table.
3. (id INTEGER, name TEXT) is a list of parameters defining each column, or attribute in the table and its data type

INSERT :- use to insert a new row in a table

specified row → name of table

Ex:- INSERT INTO celebs (id, name, age) → parameters
Values (1, 'Sumit ojha', 22);
data being inserted

SELECT :- Its use to fetch data from a database.

Ex. SELECT name FROM celebs;
clause column name

If we want two column

SELECT column1name, column2name
FROM celebs;

Alter :- The Alter table statements adds a new Column to a table.

Ex:- ALTER TABLE celebs
ADD COLUMN twitter_handle TEXT;

* NULL is a special value in SQL that represents missing or unknown data. here the rows that existed before the Column was added have NULL (0) values for twitter_handle

UPDATE :- the update statements edits a row in a table. you can use the update statement when you want to change existing records.

Ex:- UPDATE celebs
SET twitter_handle = '@sumitsojha8'
WHERE id = 4;

DELETE :- the DELETE FROM statement deletes one or more rows from a table. You can use the statement when you want to delete existing records.

Ex:- DELETE FROM Celebs
WHERE twitter_handle IS NULL;

* IS NULL is a Cond" in SQL that returns true when the Value is NULL and false otherwise.

Constraints :- Constraints that add information about how a Column can be used are invoked after specifying the data type for a Column. They can be used to tell the database to reject inserted data that does not adhere to a certain restriction.

Ex:- CREATE TABLE celebs(
id INTEGER PRIMARY KEY,
name TEXT UNIQUE,
date_of_birth TEXT NOT NULL,
date_of_death TEXT DEFAULT 'Not Applicable');

1) Primary key Columns are used to uniquely identify the row. If already in table will result in a Constraint Violation which will not allow you to insert the new row.
2) Unique Columns have diff value for every row. this is similar to primary key except a table can have many diff. UNIQUE Columns.
3) NOT NULL Columns must have a value.
4) DEFAULT assumed value for an inserted row.

AS :- AS is a Keyword use to rename a Column or table using an alias. the new name can be anything u want as long as u put it inside a Single quote.

* when using AS, the Columns are not being renamed in the table. the aliases only appear in the result.

Ex:- SELECT name AS 'Titles'
FROM movies;

Distinct :- It's used to return unique value in the output. it filters out all duplicate values in the Specified Column

Ex:- SELECT DISTINCT title
FROM movies

Where :- we can restrict our query results using the WHERE clause in order to obtain only the information we want.

Ex:- SELECT *
FROM movies
WHERE imdb_rating > 8;

* '^[^aeiou]'
* where City REGEXP '^[aeiou].*[aeiou]$';
* Where City 'REGEXP '[aeiou]$';

LIKE I :- LIKE Can be a useful operator when u want to Compare Similar Values.

Ex:- SELECT *
FROM movies
WHERE name LIKE 'Se_en';

LIKE II :- The % sign is another wildcard characters that Can be used with LIKE.
* A% :- all letters start with A,
* %a :- all ends with a. * REGEXP_LIKE(City, '^[AOE]')

Is NULL :- we use IS NULL & IS NOT NULL.
Ex:- SELECT name
from movies
WHERE imdb_rating IS NOT FULL;

Between : the between operator is used in a WHERE clause to filter the result set within a certain range. It accepts two values that are either numbers, text or dates.
Ex:- SELECT *
FROM movies
WHERE year BETWEEN 1990 AND 1999;

AND :- If all Cond^n are true

Eg SELECT *
FROM movies
WHERE year BETWEEN 1990 AND 1999
AND genre = 'romance';

OR :- If any Cond^n is true

Eg- SELECT *
FROM movies
WHERE year > 2014
OR genre = 'action';

Order By :- We can sort the results using ORDER BY, either alphabetically or numerically.

Eg SELECT *
FROM movies
ORDER BY name; // ORDER BY year DESC

* DESC is a keyword to sort in descending order
* ASC is ascending order

Limit :- Limit is a clause that lets you specify the maximum no. of rows the result set will have. this saves space on our screen & makes our queries run faster

* We can't have more than 10 rows.
* It always goes at the very end of the query - also, its not supported in all SQL databases.

Eg:- SELECT *
FROM movies
LIMIT 10;

Case :- A case statement allows us to create different outputs (usually in the SELECT statement). It's SQL's way of handling if-then logic.

Eg: SELECT name,
CASE
WHEN imdb_rating > 8 THEN 'Fantastic'
WHEN 'imdb_rating > 6 THEN 'Poorly Received'
ELSE 'Avoid at All Costs'
END
FROM movies;

Count :- the fastest way to calculate how many rows are in a table is to use the COUNT() func^{n}.

Eg: SELECT COUNT(*)
FROM table_name;

SUM :- It takes the name of a column as an argument & returns the sum of all the values in that column.

Eg: SELECT SUM(downloads)
FROM fake_apps;

Max / Min :- The MAX() & MIN() func^{n} return the highest & lowest values in a column.

Eg: SELECT MAX(download)
FROM fake_apps;

Eg: SELECT MIN(download)
FROM fake_apps;

REPLAC :- REPLACE(SALARY, '0', '')

Average :- SQL uses the AVG() func^n to quickly calculate the average value of a particular column.

Eg:- SELECT AVG(download)
FROM fake_apps;

Round :- SQL tries to be as precise as possible without rounding.

Round() func^n takes two arguments inside the parenthesis
1. Column name
2. an integer.

It rounds the values in the column to the no. of decimal places specified by the integer.

Eg:- SELECT ROUND(price, 0)
FROM fake_apps;

GROUP BY :- It's a clause in SQL that is used with aggregate func^n. It's used in collaboration with the SELECT statement to arrange identical data into groups.

* the Group by statement comes after any WHERE statements, but before ORDER BY or LIMIT.

Eg: SELECT Year
AVG(imdb_rating)
FROM movies
GROUP BY Year
ORDER BY Year;

GROUP BY :- Sometimes we want to group By a Calculation done on a Column

Eg SELECT Round(imdb_rating),
COUNT(name)
FROM movies
GROUP BY 1
ORDER BY 1;

here the 1 refers to the first Column in our SELECT Statement,

Having :- It allows u to filter which grp to include & which to exclude.

Eg:- SELECT Year,
genre,
COUNT(name)
FROM movies
GROUP BY 1,2
Having COUNT(name) > 10

* having statement always comes after GROUP BY but before ORDER BY and LIMIT.

Combining Tables Manually :

↳ By ourself

Combining Tables with SQL :-

Combining tables manually is time-consuming. Luckily, SQL gives us an easy sequence for this; it's called a JOIN.

Ex.
```
SELECT *
FROM orders
JOIN customers
    ON orders.customer_id = customers.customer_id;
```

INNER JOINS :- When we perform a simple JOIN (often called an inner join) our result only includes rows that match our ON condition.

LEFT Joins :- A left join will keep all rows from the first table, regardless of whether there is a matching row in the second table.

Ex:
```
SELECT *
FROM table1
LEFT JOIN table2
    ON table.c2 = table2.c2;
```

Primary Key Vs Foreign Key

* Each of these tables has a Column that uniquely identifies each row of that table. these Special Columns are called Primary Keys.

Primary Keys have a few requirements :-

* None of the Values can be NULL.
* Each Value must be unique.
* A table can not have more than one primary key Column.

* When the Primary key for one table appears in a different table, it's called a foreign key.

Why is this important?
The most Common types of joins will be joining a foreign key from one table with the primary key from another table.

Cross Join :- Sometime, we just want to combine all rows of one table with all rows of another table.

Ex:- SELECT shirts.shirt_color,
pants.pants_color
FROM shirts
CROSS JOIN pants;

* A more common usage of CROSS JOIN is when we need to compare each row of a table to a list of values.

Union :- Sometimes we just want to stack one dataset on top of the other. Well the UNION operator allows us to do that.

Ex:- SELECT *
FROM table1
UNION
SELECT *
FROM table2;

Rules:-
* tables must have the same no of columns.
* the columns must have the same data types in the same order as the first table.

With :- Often times, we want to Combine two tables, but one of the table is the result of another Calculation.

```
SELECT Customer_id,
       COUNT(Subscription_id) AS 'Subscription'
FROM order
GROUP BY Customer_id;
```

Ex:-

```
WITH previous_result AS (
    SELECT ...
    ...
    ...
    ...
)
SELECT *
FROM previous_result
JOIN Customers
  ON ____ = ____;
```

* The WITH statement allows us to perform a seperate query
* previous_results is the alias that we will use to reference any Columns from the query inside of the WITH clause
* We can then go on to do whatever we want with this temporary table.