SQL CHEATSHEET

What is SQL?

SQL stands for **Structured Query Language** which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is a language to operate databases; it includes Database Creation, Database Deletion, Fetching Data Rows, Modifying & Deleting Data rows, etc.

How SQL Works?

Various components are included in this process.

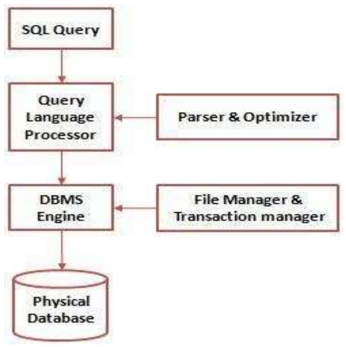
- Query Dispatcher.
- Optimization Engines
- Classic Query Engine
- SQL Query Engine

What Is RDBMS?

RDBMS stands for **Relational Database Management System.** It's used to maintain a relational database.

The data in an RDBMS is stored in database objects known as **tables**. Its is in the form of rows and columns.

Architecture of SQL: -



These SQL commands are mainly categorized into five categories:

1} DDL: - Data definition language helps you to define the database structure or schema.

Create, Drop, Alter, Truncate. These 4 commands come under DDL.

2} DML: - Data Manipulation Language (DML) allows you to modify the database instance by inserting, modifying, and deleting its data.

Insert, Update, Delete. Come under DML.

3} DCL: - Data control Language includes commands like GRANT and REVOKE, which are useful to give “rights & permissions.

Grant, Revoke. Come under DCL.

4} TCS: - Transaction control language or TCL commands deal with the transaction within the database.

Commit, Rollback, Savepoint. Come under TCL.

5} DQL: - Data Query Language (DQL) is used to fetch the data from the database. It uses only one command **Select.**

Command	Meaning	Example Syntax
CREATE DATABASE	statement used to create a new database in SQL.	CREATE DATABASE DatabaseName;
DROP DATABASE	statement in SQL is used to delete a database along with all the data.	DROP DATABASE DatabaseName;
CREATE TABLE	statement is used to create a new table in a database.	CREATE TABLE *table_name* (*column1 datatype*, *column2 datatype*, *column3 datatype*,);
TRUNCATE TABLE	is used to delete all the records from an existing table	TRUNCATE TABLE table_name;
ALTER TABLE	statement is used to add, delete, drop various constraints or modify columns in an existing table.	ALTER TABLE *table_name* ADD *column_name datatype*;
DROP TABLE	statement is used to drop an existing table in a database.	DROP TABLE *table_name*;
DELETE TABLE	statement is used to delete existing records in a table.	DELETE FROM *table_name* WHERE *condition*;
Constraints	Constraints are used to limit the type of data that can go into a table.	
NOT NULL	constraint enforces a column to NOT accept NULL values.	CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255) NOT NULL, Age int);
UNIQUE	constraint ensures that all values in a column are different.	CREATE TABLE Persons (ID int NOT NULL UNIQUE, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int);

PRIMARY KEY	Constraint uniquely identifies each record in a table. It must contain UNIQUE values, and cannot contain NULL values.
	A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).
		CREATE TABLE Persons (
		    ID int NOT NULL,
		    LastName varchar(255) NOT NULL,
		    FirstName varchar(255),
		    Age int,
		    PRIMARY KEY (ID)
		);

FOREIGN KEY	constraint is used to prevent actions that would destroy links between tables.	CREATE TABLE Orders (
		    OrderID int NOT NULL,
		    OrderNumber int NOT NULL,
		    PersonID int,
		    PRIMARY KEY (OrderID),
		    FOREIGN KEY (PersonID)
		REFERENCES Persons(PersonID)
		);

CHECK	constraint is used to limit the value range that can be placed in a column.	CREATE TABLE Persons (
		    ID int NOT NULL,
		    LastName varchar(255) NOT NULL,
		    FirstName varchar(255),
		    Age int,
		    CHECK (Age>=18)
		);

DEFAULT	constraint is used to set a default value for a column.	CREATE TABLE Persons (
		    ID int NOT NULL,
		    LastName varchar(255) NOT NULL,
		    FirstName varchar(255),
		    Age int,
		    City varchar(255) DEFAULT 'Sandnes'
		);

INDEX	Used to create and retrieve data from the database very quickly.	CREATE INDEX *index_name*
		ON *table_name* (*column1*, *column2*, ...);

INSERT	statement is used to insert new records in a table.	INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
		VALUES (*value1*, *value2*, *value3*, ...);

SELECT	statement is used to select data from a database.	SELECT * FROM *table_name*;

UPDATE	statement is used to modify the existing records in a table.	UPDATE *table_name*
		SET *column1* = *value1*, *column2* = *value2*, ...
		WHERE *condition*;

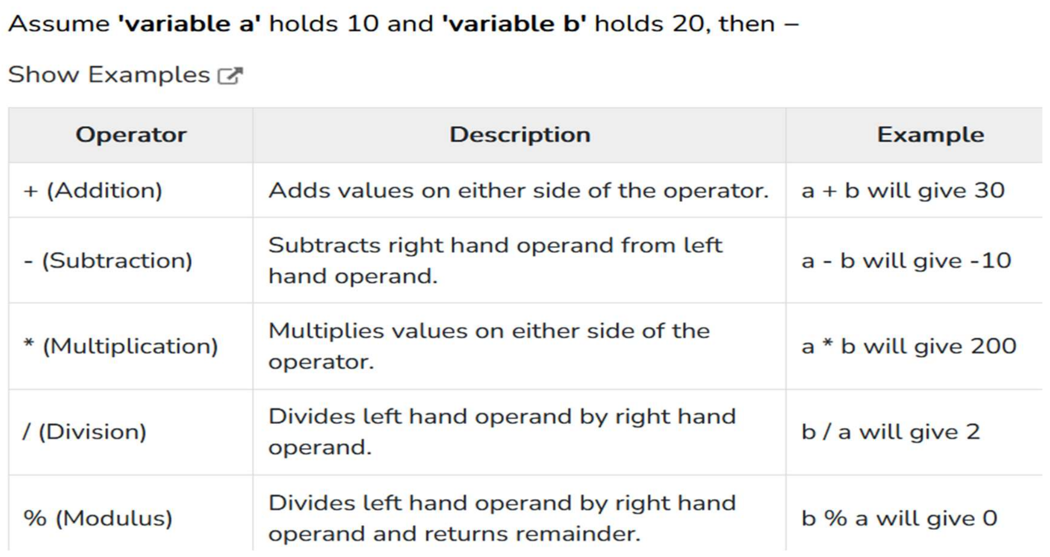
COMMIT	the transactional command used to save changes invoked by a transaction to the database.	
ROLLBACK	the transactional command used to undo transactions that have not already been saved to the database.	

Operators And Clauses: -

A clause in SQL is a built-in function that helps to fetch the required records from a database table.

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, arithmetic operations and Logical operations.

- **Arithmetic operators: -**



Assume **'variable a'** holds 10 and **'variable b'** holds 20, then –

Show Examples

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

- **Comparison Operators:** -

Operator	Description	Example
=	Equal To	SELECT * FROM Products WHERE Price = 18;
>	Greater than	SELECT * FROM Products WHERE Price > 30;
<	Less than	SELECT * FROM Products WHERE Price < 30;
>=	Greater than equal to	SELECT * FROM Products WHERE Price >= 30;
<=	Less than equal to	SELECT * FROM Products WHERE Price <=30;
<>	Not equal to	SELECT * FROM Products WHERE Price <> 18;

- **Logical Operators: -**

Operator	Description	Example
ALL	The ALL operator is used to compare a value to all values in another value set.	SELECT ProductName FROM Products WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);
AND	Operator allows the existence of multiple conditions in an SQL statement's WHERE clause.	SELECT * FROM Customers WHERE City = "London" AND Country = "UK";

ANY	Is used to compare a value to any applicable value in the list as per the condition.	SELECT * FROM Products WHERE Price > ANY (SELECT Price FROM Products WHERE Price > 50);
BETWEEN	Is used to search for values that are within a set of values, given the minimum value and the maximum value.	SELECT * FROM Products WHERE Price BETWEEN 50 AND 60;
EXISTS	Is used to search for the presence of a row in a specified table that meets a certain criterion.	SELECT SupplierName FROM Suppliers WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);
IN	Is used to compare a value to a list of literal values that have been specified.	SELECT * FROM Customers WHERE City IN ('Paris','London');
LIKE	Is used to compare a value to similar values using wildcard operators.	SELECT * FROM Customers WHERE City LIKE 's%';
NOT	Operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**	SELECT * FROM Customers WHERE City NOT LIKE 's%';
OR	Is used to combine multiple conditions in an SQL statement's WHERE clause.	SELECT * FROM Customers WHERE City = "London" OR Country = "UK";

UNION & UNION ALL	UNION operator is used to combine the result-set of two or more SELECT statements. Union values select only distinct value must also have similar data only.	SELECT *column_name(s)* FROM *table1*
		UNION
		SELECT *column_name(s)* FROM *table2*;

Clauses: -

Clauses	Description	Example

WHERE	The WHERE clause is used to filter records.
	Its is also used in Update, Delete statement.
		SELECT *column1*, *column2, ...*
		FROM *table_name*
		WHERE *condition*;

DISTINCT	The SELECT DISTINCT statement is used to return only distinct (different) values.	SELECT DISTINCT *column1*, *column2, ...*
		FROM *table_name*;

ORDER BY	The ORDER BY keyword is used to sort the result-set in ascending or descending order.	SELECT *column1*, *column2, ...*
		FROM *table_name*
		ORDER BY *column1, column2, ...* ASC|DESC;

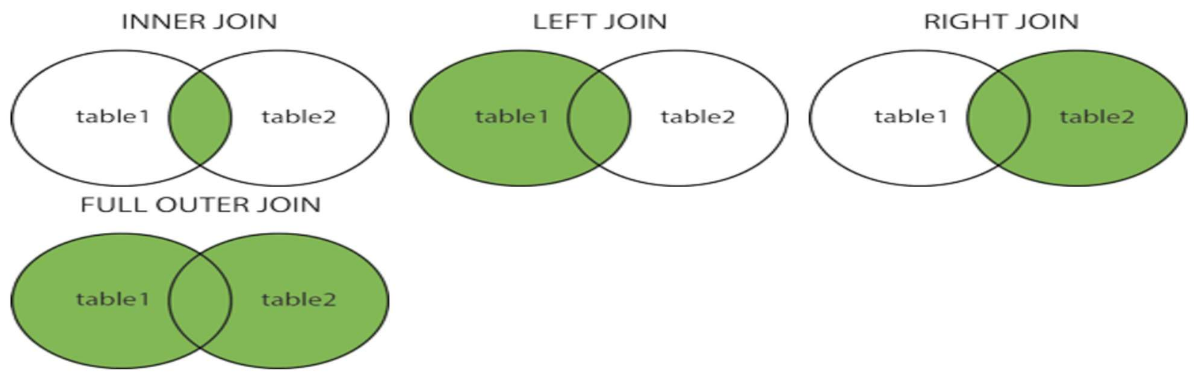
LIMIT	clause to select a limited number of records.	SELECT *column_name(s)* FROM *table_name* WHERE *condition* LIMIT *number*;

ALIASES	SQL aliases are used to give a table, or a column in a table, a temporary name. Aliases is created with AS keyword.	SELECT *column_name* AS *alias_name*
		FROM *table_name;*

GROUP BY	The GROUP BY statement groups rows that have the same values into summary rows.	SELECT *column_name(s)* FROM *table_name* WHERE *condition*
		GROUP BY *column_name(s)*
		ORDER BY *column_name(s);*

HAVING	The `HAVING` clause was added to SQL because the `WHERE` keyword cannot be used with aggregate functions.	SELECT *column_name(s)* FROM *table_name* WHERE *condition* GROUP BY *column_name(s)* HAVING *condition* ORDER BY *column_name(s);*

Joins: - A JOIN clause is used to combine rows from two or more tables, based on a related column between them.



Command	**Description**	**Example**
INNER JOIN	The INNER JOIN keyword selects records that have matching values in both tables.	SELECT *column_name(s)* FROM *table1* INNER JOIN *table2* ON *table1.column_name* = *table2.column_name*;
LEFT JOIN	The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2).	SELECT *column_name(s)* FROM *table1* LEFT JOIN *table2* ON *table1.column_name* = *table2.column_name*;
RIGHT JOIN	The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1).	SELECT *column_name(s)* FROM *table1* RIGHT JOIN *table2* ON *table1.column_name* = *table2.column_name*;
FULL JOIN	The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.	SELECT *column_name(s)* FROM *table1* FULL OUTER JOIN *table2* ON *table1.column_name* = *table2.column_name* WHERE *condition*;

Self-Join: - A self-join is a regular join, but the table is joined with itself.

Example: - SELECT *column_name(s)*
FROM *table1 T1, table1 T2*
WHERE condition;

Aggregate Function: - Returns one value after calculating multiple values of a column.

Functions	Description	Example
COUNT	The COUNT () function returns the number of rows that matches a specified criterion.	SELECT COUNT(*column_name*) FROM *table_name* WHERE *condition*;
AVG	The AVG () function returns the average value of a numeric column.	SELECT AVG(*column_name*) FROM *table_name* WHERE *condition*;
SUM	The SUM () function returns the total sum of a numeric column.	SELECT SUM(*column_name*) FROM *table_name* WHERE *condition*;
MIN	The MIN () function returns the smallest value of the selected column.	SELECT MIN(*column_name*) FROM *table_name* WHERE *condition*;
MAX	The MAX () function returns the largest value of the selected column.	SELECT MAX(*column_name*) FROM *table_name* WHERE *condition*;

Windows Function: -SQL provides a set of functions that can be used to perform calculations across a set of rows that are related to the current row.

Function	Description	Example
RANK ()	Assigns a unique rank to each row within a result set, based on the values in one or more columns	SELECT ProductID, ProductName, Price, RANK () OVER (ORDER BY Price DESC) price_rank FROM Products;
DENSE RANK ()	function also assigns a ranking within the partition, with no gaps and the same ranking for tied values.	SELECT ProductID, ProductName,Price, DENSE_RANK () OVER (ORDER BY Price DESC) price_rank FROM Products;
ROW_NUMBER ()	The function assigns a unique number for each row within the partition, with different numbers for tied values.	SELECT ROW_NUMBER () OVER (ORDER BY Price) ProductID, ProductName, Price FROM Products;

Case Expression: - The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

Example: - CASE

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
    END;
```