SQL Notes

Database

Database is collection of data in a format that can be easily accessed (Digital).

A software application used to manage our DB is called DBMS (Database Management System).

Types of Databases

Relational	Non-relational (NoSQL)
Data stored in tables	data not stored in tables
Eg - MySQL, PostgreSQL, Oracle etc.	Eg - MongoDB

We use SQL to work with relational DBMS.

What is SQL?

SQL is Structured Query Language - used to store, manipulate and retrieve data from RDBMS. (

It is not a database, it is a language used to interact with database) We use SQL for CRUD Operations:

- ✓ ● CREATE - To create databases, tables, insert tuples in tables etc
- ✓ ● READ - To read data present in the database.
- ✓ ● UPDATE - Modify already inserted data.
- ✓ ● DELETE - Delete database, table or specific data point/tuple/row or multiple rows.

Note - SQL keywords are NOT case sensitive. Eg: select is the same as SELECT in SQL.

SQL v/s MySQL

SQL is a language used to perform CRUD operations in Relational DB, while MySQL is a RDBMS that uses SQL.

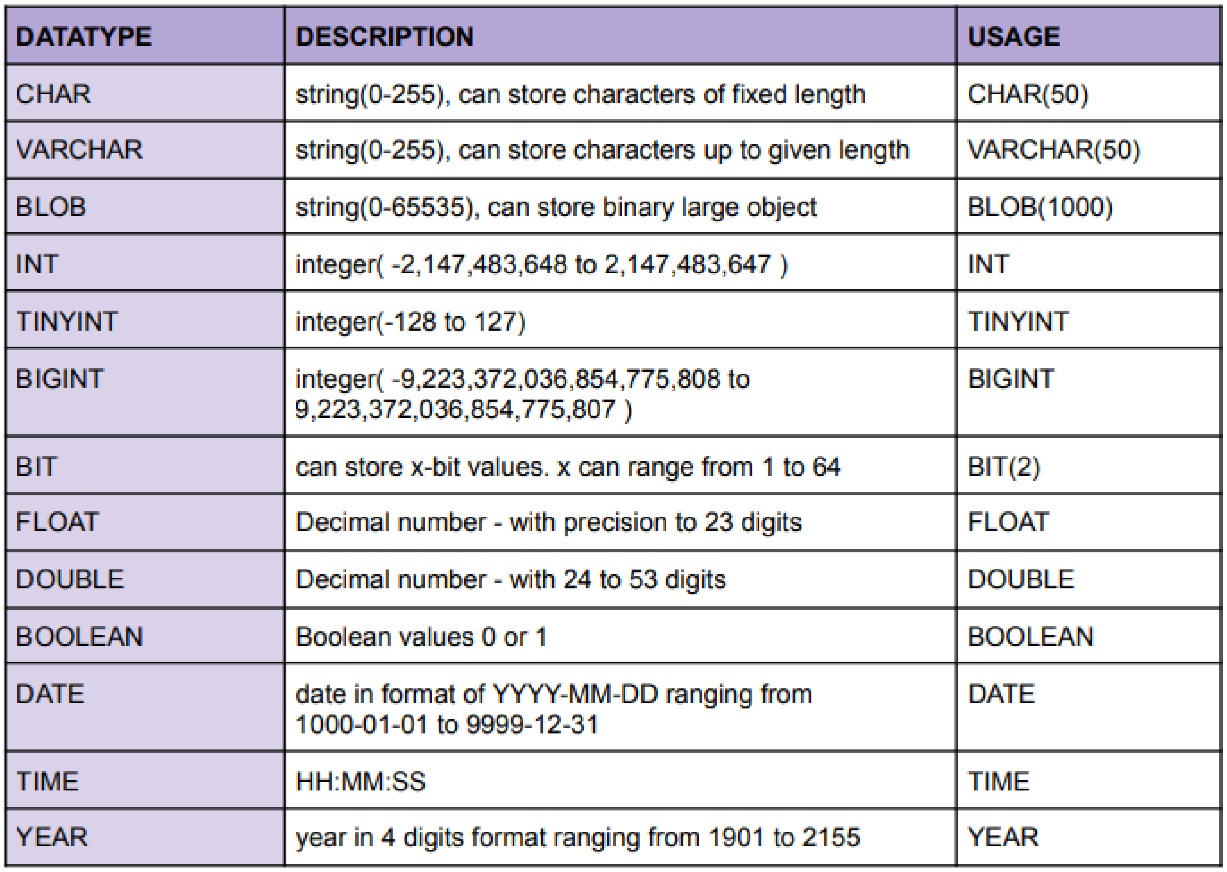
SQL Data Types

In SQL, data types define the kind of data that can be stored in a column or variable.

To see all data types of MYSQL,

Visit: https://dev.mysql.com/doc/refman/8.0/en/data-types.html

Here are the frequently used SQL data types:



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

Note - CHAR is for fixed length & VARCHAR is for variable length strings. Generally, VARCHAR is better as it only occupies necessary memory & works more efficiently.

We can also use UNSIGNED with datatypes when we only have positive values to add. Eg - UNSIGNED INT.

Types of SQL Commands

1. DQL (Data Query Language)

Used to retrieve data from databases. (SELECT)

2. DDL (Data Definition Language)

Used to create, alter, and delete database objects like tables, indexes, etc. (CREATE, DROP, ALTER, RENAME, TRUNCATE).

3. DML (Data Manipulation Language)

Used to modify the database. (INSERT, UPDATE, DELETE).

4. DCL (Data Control Language)

Used to grant & revoke permissions. (GRANT, REVOKE).

5. TCL (Transaction Control Language)

Used to manage transactions. (COMMIT, ROLLBACK, START TRANSACTIONS, AND SAVEPOINT).

1. Data Definition Language (DDL)

Data Definition Language (DDL) is a subset of SQL (Structured Query Language) responsible for defining and managing the structure of databases and their objects.

DDL commands enable you to create, modify, and delete database objects like tables, indexes, constraints, and more. Key DDL Commands are:

● CREATE TABLE

- ➢ Used to create a new table in the database.
- ➢ Specifies the table name, column names, data types, constraints, and more.

- Example: CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50), salary DECIMAL(10, 2));

● ALTER TABLE

- Used to modify the structure of an existing table.
- You can add, modify, or drop columns, constraints, and more.
- Example: ALTER TABLE employees ADD COLUMN email VARCHAR(100);

● DROP TABLE

- Used to delete an existing table along with its data and structure.
- Example: DROP TABLE employees;

● CREATE INDEX

- Used to create an index on one or more columns in a table.
- Improves query performance by enabling faster data retrieval.
- Example: CREATE INDEX idx_employee_name ON employees (name); ● DROP INDEX
- Used to remove an existing index from a table.
- Example: DROP INDEX idx_employee_name;

● CREATE CONSTRAINT

- Used to define constraints that ensure data integrity.
- Constraints include PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, and CHECK.
- Example: ALTER TABLE orders ADD CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES customers(id);

● DROP CONSTRAINT

- Used to remove an existing constraint from a table.
- Example: ALTER TABLE orders DROP CONSTRAINT fk_customer;

● TRUNCATE TABLE

- Used to delete the data inside a table, but not the table itself.

- Syntax – TRUNCATE TABLE table_name;

2. DATA QUERY/RETRIEVAL LANGUAGE (DQL or DRL)

DQL (Data Query Language) is a subset of SQL focused on retrieving data from databases.

The SELECT statement is the foundation of DQL and allows us to extract specific columns from a table.

- SELECT

The SELECT statement is used to select data from a database.

Syntax: SELECT column1, column2, ... FROM table_name; Here, column1, column2, ... are the field names of the table. If you want to select all the fields available in the table, use the following syntax: SELECT * FROM table_name; Ex: SELECT CustomerName, City FROM Customers;

- WHERE
  The WHERE clause is used to filter records. Syntax:
  SELECT column1, column2, ... FROM table_name WHERE condition; Ex: SELECT * FROM Customers WHERE Country=Skardu';
  Operators used in WHERE are
  - = : Equal
  - : Greater than
  - < : Less than
  - >= : Greater than or equal
  - <= : Less than or equal
    - : Not equal.

  Note: In some versions of SQL this operator may be written as !=
  - AND, OR and NOT:

  The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:
The AND operator displays a record if all the conditions separated by AND are TRUE.
The OR operator displays a record if any of the conditions separated by OR is TRUE.
The NOT operator displays a record if the condition(s) is NOT TRUE.
Syntax:
SELECT column1, column2, ... FROM table_name WHERE condition1 AND condition2 AND condition3 ...;
SELECT column1, column2, ... FROM table_name WHERE condition1 OR condition2 OR condition3 ...;
SELECT column1, column2, ... FROM table_name WHERE NOT condition; Example:
SELECT * FROM Customers WHERE Country='Pakistan' AND City='Skardu'; SELECT * FROM Customers WHERE Country='Pakistan' AND (City='Skardu' OR City='Gilgit');

- DISTINCT

  Removes duplicate rows from query results.
  Syntax: SELECT DISTINCT column1, column2 FROM table_name;

- LIKE

  The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

  There are two wildcards often used in conjunction with the LIKE operator.
  The percent sign (%) represents zero, one, or multiple characters - The underscore sign (_) represents one, single character.
  Example: SELECT * FROM employees WHERE first_name LIKE 'J%';
  WHERE CustomerName LIKE 'm%' .

Finds any values that start with "m”.
WHERE CustomerName LIKE '%m'
Finds any values that end with "m"
WHERE CustomerName LIKE '%or%' –
Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'
Finds any values that have "r" in the second position.
WHERE CustomerName LIKE 'm_%'
Finds any values that start with "m" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'
Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'
Finds any values that start with "a" and ends with "o"

- IN

Filters results based on a list of values in the WHERE clause. Example: SELECT * FROM products WHERE category_id IN (1, 2, 3);

- BETWEEN

Filters results within a specified range in the WHERE clause. Example: SELECT * FROM orders WHERE order_date BETWEEN '2023-01-01' AND '2023-06-30';

- IS NULL

Checks for NULL values in the WHERE clause. Example: SELECT * FROM customers WHERE email IS NULL;

- AS

Renames columns or expressions in query results. Example: SELECT first_name AS "First Name", last_name AS "Last Name" FROM employees

- ORDER BY

The ORDER BY clause allows you to sort the result set of a query based on one or more columns.

Basic Syntax:
 The ORDER BY clause is used after the SELECT statement to sort query results.
Syntax
 SELECT column1, column2 FROM table_name ORDER BY column1 [ASC|DESC];

- Ascending and Descending Order
  - By default, the ORDER BY clause sorts in ascending order (smallest to largest).
  - You can explicitly specify descending order using the DESC keyword.
  - Example: SELECT product_name, price FROM products ORDER BY price DESC;
- Sorting by Multiple Columns
  - You can sort by multiple columns by listing them sequentially in the ORDER BY clause.
  - Rows are first sorted based on the first column, and for rows with equal values, subsequent columns are used for further sorting.
  - Example: SELECT first_name, last_name FROM employees ORDER BY last_name, first_name;
- Sorting by Expressions
  - It's possible to sort by calculated expressions, not just column values.
  - Example: SELECT product_name, price, price * 1.1 AS discounted_price FROM products ORDER BY discounted_price;
- Sorting NULL Values
  - By default, NULL values are considered the smallest in ascending order and the largest in descending order.
  - You can control the sorting behaviour of NULL values using the NULLS FIRST or NULLS LAST options.

- Example: SELECT column_name FROM table_name ORDER BY column_name NULLS LAST;

- Sorting by Position
    - Instead of specifying column names, you can sort by column positions in the ORDER BY clause.
    - Example: SELECT product_name, price FROM products ORDER BY 2 DESC, 1 ASC;

- GROUP BY
    - The GROUP BY clause in SQL is used to group rows from a table based on one or more columns.

Syntax

- The GROUP BY clause follows the SELECT statement and is used to group rows based on specified columns.

Syntax

- SELECT column1, aggregate_function(column2) FROM table_name GROUP BY column1;
    - Aggregation Functions
- Aggregation functions (e.g., COUNT, SUM, AVG, MAX, MIN) are often used with GROUP BY to calculate values for each group.
- Example: SELECT department, AVG(salary) FROM employees GROUP BY department;
    - Grouping by Multiple Columns
    - You can group by multiple columns by listing them in the GROUP BY clause. ○
    - This creates a hierarchical grouping based on the specified columns.

- Example: SELECT department, gender, AVG(salary) FROM employees GROUP BY department, gender;

- Combining GROUP BY and ORDER BY
  - You can use both GROUP BY and ORDER BY in the same query to control the order of grouped results.
  - Example: SELECT department, COUNT(*) FROM employees GROUP BY department ORDER BY COUNT(*) DESC;
- HAVING Clause
  - The HAVING clause is used with GROUP BY to filter groups based on aggregate function results.
  - It's similar to the WHERE clause but operates on grouped data.
  - Example: SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 50000;
- AGGREGATE FUNCTIONS

These are used to perform calculations on groups of rows or entire result sets. They provide insights into data by summarising and processing information.

Common Aggregate Functions

- COUNT()

Counts the number of rows in a group or result set.

- SUM()

Calculates the sum of numeric values in a group or result set.

- AVG()

Computes the average of numeric values in a group or result set.

- MAX()

Finds the maximum value in a group or result set.

- MIN()

Retrieves the minimum value in a group or result set.

DATA MANIPULATION LANGUAGE

INSERT

The INSERT statement adds new records to a table.

Syntax: INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);

Example: INSERT INTO employees (first_name, last_name, salary) VALUES ('Munawar', 'Hussain', 50000);

UPDATE

The UPDATE statement modifies existing records in a table.

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

Example: UPDATE employees SET salary = 550000 WHERE first_name = 'Munawar';

DELETE

The DELETE statement removes records from a table.

Syntax: DELETE FROM table_name WHERE condition;

Example: DELETE FROM employees WHERE last_name = 'Johar';

JOINS

In a DBMS, a join is an operation that combines rows from two or more tables based on a related column between them.

Joins are used to retrieve data from multiple tables by linking them together using a common key or column.

Types of Joins:

1. Inner Join
2. Outer Join
3. Cross Join
4. Self-Join

1) Inner Join

An inner join combines data from two or more tables based on a specified condition, known as the join condition.

The result of an inner join includes only the rows where the join condition is met in all participating tables.

It essentially filters out non-matching rows and returns only the rows that have matching values in both tables.

Syntax: SELECT columns FROM table1 INNER JOIN table2 ON table1.column = table2.column;

2) Outer Join

Outer joins combine data from two or more tables based on a specified condition, just like inner joins. However, unlike inner joins, outer joins also include rows that do not have matching values in both tables.

Outer joins are particularly useful when you want to include data from one table even if there is no corresponding match in the other table.

Types:

There are three types of outer joins: left outer join, right outer join, and full outer join.

- ### Left Outer Join (Left Join):

A left outer join returns all the rows from the left table and the matching rows from the right table.

If there is no match in the right table, the result will still include the left table's row with NULL values in the right table's columns.

Example: SELECT Customers.CustomerName, Orders.Product FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

- ### Right Outer Join (Right Join):

A right outer join is similar to a left outer join, but it returns all rows from the right table and the matching rows from the left table.

If there is no match in the left table, the result will still include the right table's row with NULL values in the left table's columns.

Example: Using the same Customers and Orders tables. SELECT Customers.CustomerName, Orders.Product FROM Customers RIGHT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

- ### Full Outer Join (Full Join)

A full outer join returns all rows from both the left and right tables, including matches and non-matches.

If there's no match, NULL values appear in columns from the table where there's no corresponding value.

Example: Using the same Customers and Orders tables. SELECT Customers.CustomerName, Orders.Product FROM Customers FULL OUTER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

3) Cross Join

A cross join, also known as a Cartesian product, is a type of join operation in a Database Management System (DBMS) that combines every row from one table with every row from another table.

Unlike other join types, a cross join does not require a specific condition to match rows between the tables. Instead, it generates a result set that contains all possible combinations of rows from both tables.

Cross joins can lead to a large result set, especially when the participating tables have many rows.

Syntax:

SELECT columns FROM table1 CROSS JOIN table2;

In this syntax:

- columns refers to the specific columns you want to retrieve from the cross-joined tables.
- table1 and table2 are the names of the tables you want to combine using a cross join. Example: Consider two tables: Students and Courses.

4) Self Join

A self join involves joining a table with itself. This technique is useful when a table contains hierarchical or related data and you need to compare or analyse rows within the same table.

Self joins are commonly used to find relationships, hierarchies, or patterns within a single table.

In a self join, you treat the table as if it were two separate tables, referring to them with different aliases.

Syntax:

The syntax for performing a self join in SQL is as follows:

SELECT columns FROM table1 AS alias1 JOIN table1 AS alias2 ON alias1.column = alias2.column;

SET OPERATIONS

Set operations in SQL are used to combine or manipulate the result sets of multiple SELECT queries.

They allow you to perform operations similar to those in set theory, such as union, intersection, and difference, on the data retrieved from different tables or queries.

Set operations provide powerful tools for managing and manipulating data, enabling you to analyse and combine information in various ways.

There are four primary set operations in SQL:

● UNION

● INTERSECT

● EXCEPT (or MINUS)

● UNION ALL

1. UNION

The UNION operator combines the result sets of two or more SELECT queries into a single result set.

It removes duplicates by default, meaning that if there are identical rows in the result sets, only one instance of each row will appear in the final result.

Example: Assume we have two tables: Customers and Suppliers.

UNION Query

SELECT CustomerName FROM Customers UNION SELECT SupplierName FROM Suppliers;

2. INTERSECT

The INTERSECT operator returns the common rows that exist in the result sets of two or more SELECT queries. It only returns distinct rows that appear in all result sets.

Example: Using the same tables as before.

SELECT CustomerName FROM Customers INTERSECT SELECT SupplierName FROM Suppliers;

3. EXCEPT (or MINUS)

The EXCEPT operator (also known as MINUS in some databases) returns the distinct rows that are present in the result set of the first SELECT query but not in the result set of the second SELECT query. Example: Using the same tables as before.

SELECT CustomerName FROM Customers EXCEPT SELECT SupplierName FROM Suppliers;

4. UNION ALL

The UNION ALL operator performs the same function as the UNION operator but does not remove duplicates from the result set.

It simply concatenates all rows from the different result sets.

Example: Using the same tables as before. SELECT CustomerName FROM Customers UNION ALL SELECT SupplierName FROM Suppliers;

SUB QUERIES

Subqueries, also known as nested queries or inner queries, allow you to use the result of one query (the inner query) as the input for another query (the outer query).

Subqueries are often used to retrieve data that will be used for filtering, comparison, or calculation within the context of a larger query.

They are a way to break down complex tasks into smaller, manageable steps.

Syntax: SELECT columns FROM table WHERE column OPERATOR (SELECT column FROM table WHERE condition);

Thank You