**DATABASE**

Database is a collection of interrelated data.

**What is DBMS?**

DBMS (Database Management System) is software used to create, manage, and organize databases.

**Types of Databases:**

1. Relational Database(RDBMS):

* Data stored in tables with rows (records) and columns (attributes).
* We use SQL to work with relational DBMS.
* Eg - MySQL, PostgreSQL, Oracle etc.

1. Non-relational (NoSQL):
   * data not stored in tables.
   * Eg. – MongoDB

**Database related Queries:**

**1. To Create a database**

* CREATE DATABASE db\_name;
* CREATE DATABASE IF NOT EXISTS db\_name;

Eg. - CREATE DATABASE IF NOT EXISTS College;

**2. To delete a database:**

* DROP DATABASE db\_name;
* DROP DATABASE IF EXISTS db\_name;

**3. To see all databases on the server:**

SHOW DATABASES;

**4. To use a database:**

USE db\_name;

**5. To see all tables in a database:**

SHOW TABLES;

**SQL (Structured Query Language):**

**SQL** is a programming language used to interact with relational databases.

It is used to perform 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.

**Types of SQL Commands:**

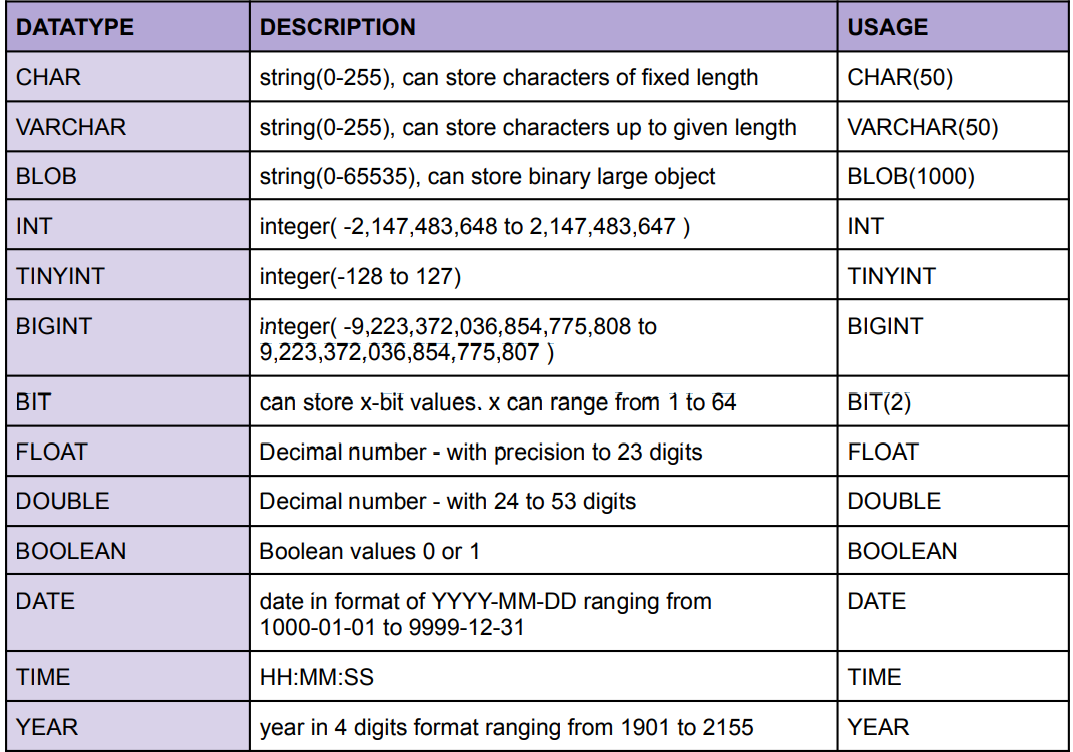
1. **DDL (Data Definition Language) :** Used to create, alter, and delete database objects like tables, indexes, etc. (CREATE, DROP, ALTER, RENAME, TRUNCATE)
2. **DQL (Data Query Language) :** Used to retrieve data from databases. (SELECT)
3. **DML (Data Manipulation Language):** Used to modify the database. (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, SAVEPOINT)

**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:



**\*Note** –

1. CHAR is for fixed length & VARCHAR is for variable length strings. Generally, VARCHAR is better as it only occupies necessary memory & works more efficiently.
2. We can also use UNSIGNED with datatypes when we only have positive values to add. Eg - UNSIGNED INT

**Keys**

**1. Primary Key** :

* It is a column (or set of columns) in a table that uniquely identifies each row. (a unique id)
* There is only 1 PK & it should be NOT null.

2. **Foreign Key :**

* A foreign key is a column (or set of columns) in a table that refers to the primary key in another table.
* FKs can have duplicate & null values. There can be multiple FKs.

# **Constraints :**

SQL constraints are used to specify rules for data in a table.

**1. NOT NULL**:- columns cannot have a null value.

**2. UNIQUE**:- all values in column are different.

**3. PRIMARY KEY:-**  makes a column unique & not null but used only for one column.

**4. FOREIGN KEY**

**5. DEFAULT:-** sets the default value of a column.

6. **CHECK**:- it can limit the values allowed in a column.

**Table related Queries:**

**1. To create a new table:**

CREATE TABLE table\_name ( column\_name1 datatype constraint, column\_name2 datatype constraint);

Eg. CREATE TABLE student (rollno INT PRIMARY KEY, name VARCHAR(50),

marks INT NOT NULL, grade VARCHAR(1), city VARCHAR(20));

**2. Select & View ALL columns:**

1. View data of specific columns

SELECT col1, col2 FROM table\_name;

1. View data of all coumns in the table:

SELECT \* FROM table\_name;

1. View distinct entries in a column:

SELECT DISTINCT col\_name FROM table\_name

**3. Delete the whole table:**

DROP TABLE table\_name;

**WHERE clause:**

To define some conditions.

SELECT col1, col2 FROM table\_name WHERE condition(s);

Eg. Query to get all students details whose marks are greater than equal to 80.

Select \* From student where marks >= 80;

**Operators:**

**1) Arithmetic Operators:** + (addition), - (subtraction), \* (multiplication), / (division), % (modulus).

**2) Comparison Operators :** = (equal to), != (not equal to), > , >= , <=

**3) Logical Operators :** AND, OR , NOT,

Eg. SELECT \* FROM student WHERE marks >= 80 AND city = "mumbai";

SELECT \* FROM student WHERE city NOT IN ("delhi", "mumbai");

IN (matches any value in the list),

Eg. SELECT \* FROM student WHERE city IN ("delhi", "mumbai");

BETWEEN (select from a given range inclusive),

Eg. SELECT \* FROM student WHERE marks BETWEEN 78 AND 85;

ALL,

LIKE,

ANY

**4) Bitwise Operators :** & (Bitwise AND), | (Bitwise OR)

## Limit Clause:

Sets an upper limit on number of rows (tuples) to be returned.

SELECT col1, col2 FROM table\_name LIMIT number;

Eg. SELECT \* FROM student WHERE marks >= 80 LIMIT 3;

**Order By Clause:**

To sort output in ascending (ASC) or descending order (DESC).

SELECT col1, col2 FROM table\_name ORDER BY col\_name(s) ASC;

Eg. Query to get top 3 students with highest marks:

SELECT \* FROM student ORDER BY marks DESC LIMIT 3;

# **Aggregate Functions:**

Aggregare functions perform a calculation on a set of values, and return a single value.

1. COUNT( )
2. MAX( )
3. MIN( )
4. SUM( )
5. AVG( )

### Group By Clause

It collects data from multiple records and groups the result by one or more column.

\*Generally we use group by with some aggregation function.

Eg. SELECT city, count(name) FROM student GROUP BY city;

Note: You can view only those column’s data which are in GROUP BY clause.

Eg. SELECT city, name, count(name) FROM student GROUP BY city; -> will give error.

**Having Clause:**

Similar to Where i.e. applies some condition on rows.

Used when we want to apply any condition after grouping.

Eg. Count number of students in each city where max marks cross 90.

SELECT city, count(rollno) FROM student GROUP BY city HAVING MAX(marks) > 90;

# **General Order of Clauses:**

SELECT column(s)

# FROM table\_name

WHERE condition

GROUP BY column(s)

HAVING condition

ORDER BY column(s) ASC;

**Table related Queries:**

**4. Insert data to table:**

INSERT INTO table\_name (colname1, colname2)

VALUES (col1\_v1, col2\_v1), (col1\_v2, col2\_v2);

Eg. INSERT INTO student (id, name, age)

VALUES(1, "hanni", 30), (2, "Jitu", 28);

Or INSERT INTO student VALUES (3, "himanshu", 26);

**5. To update existing rows:**

UPDATE table\_name SET col1 = val1, col2 = val2 WHERE condition;

Eg. UPDATE student SET grade = "D" WHERE marks < 50;

**Note**: -- To update a table entries, need to off safe update --

SET SQL\_SAFE\_UPDATES = 0;

**6. Delete (to delete existing rows):**

DELETE FROM table\_name WHERE condition;

Eg. DELETE FROM student WHERE marks < 30;

DELETE FROM student; -> to delete all data from table

**Cascading for FK :**

1. **On Update Cascade :** When we create a foreign key using UPDATE CASCADE the referencing rows are updated in the child table when the referenced row is updated in the parent table which has a primary key.
2. **On Delete Cascade:** When we create a foreign key using this option, it deletes the referencing rows in the child table when the referenced row is deleted in the parent table which has a primary key.

Eg. CREATE TABLE teacher(

id INT PRIMARY KEY,

name VARCHAR(50),

dept\_id INT,

FOREIGN KEY (dept\_id) REFERENCES dept(id)

ON UPDATE CASCADE

ON DELETE CASCADE );

**7. Modify Table (ALTER) - Schema:**

**1) Add new column:**

#### ALTER TABLE table\_name

ADD COLUMN col\_name datatype constraints;

Eg. ALTER TABLE student ADD COLUMN age INT NOT NULL DEFAULT 20;

**2) DROP (delete) column:**

#### ALTER TABLE table\_name

DROP COLUMN col\_name;

Eg. ALTER TABLE student DROP COLUMN st\_age;

**3) RENAME table name:**

#### ALTER TABLE table\_name

RENAME TO new\_table\_name;

Eg. ALTER TABLE student RENAME TO studento;

**4) Rename Column (CHANGE):**

ALTER TABLE table\_name

CHANGE COLUMN old\_name new\_name new\_datatype new\_constraint;

Eg. ALTER TABLE student CHANGE age st\_age INT;

**5) MODIFY column (modify datatype/constraints):**

#### ALTER TABLE table\_name

MODIFY col\_name new\_datatype new\_constraint:

Eg. ALTER TABLE student MODIFY age VARCHAR(2);

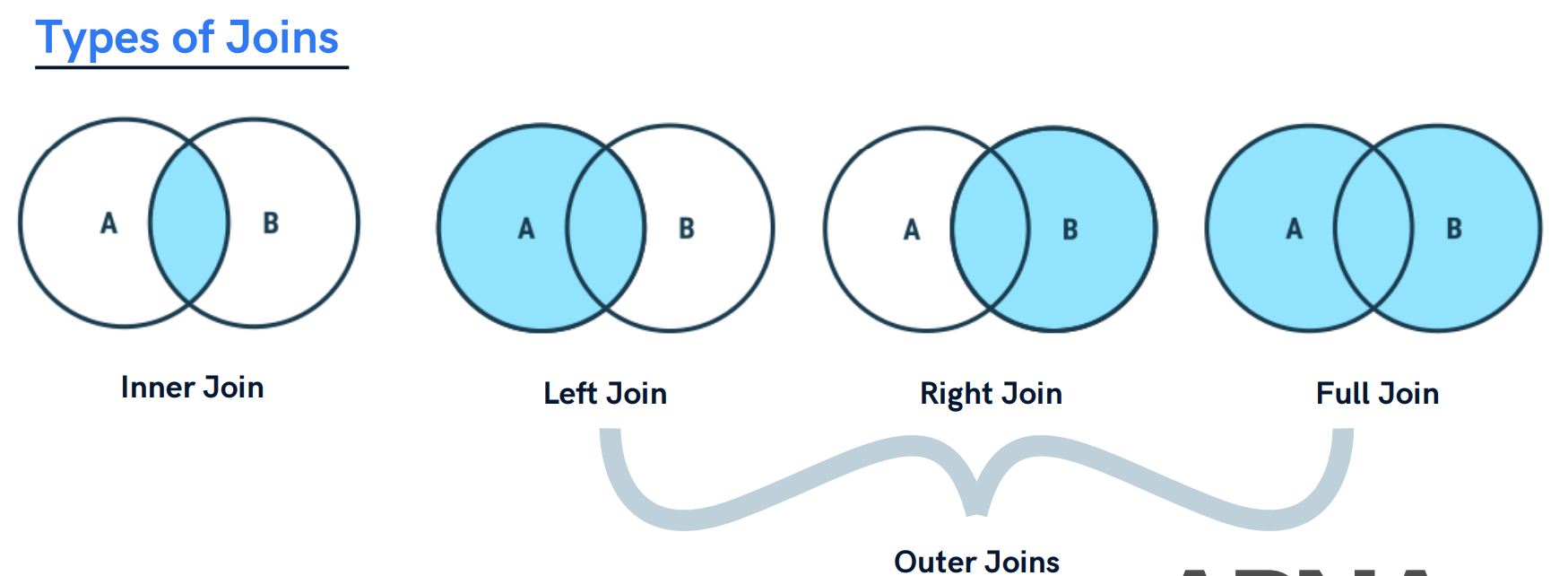
**8. Truncate (to delete table's data):**

TRUNCATE TABLE table\_name ;

**SQL Joins:**

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.

Syntax:

SELECT column(s)

FROM table1

INNER JOIN table2

ON table1.col\_name = table2.col\_name;

Example: Consider two tables: Customers and Orders.

Customers Table:

|  |  |
| --- | --- |
| CustomerID | CustomerName |
| 1 | Alice |
| 2 | Bob |
| 3 | Carol |
| 4 | Danny |

Orders Table:

|  |  |  |
| --- | --- | --- |
| OrderID | CustomerID | Product |
| 101 | 1 | Laptop |
| 102 | 3 | Smartphone |
| 103 | 2 | Headphone |
| 104 |  | Watch |

Inner Join Query:

SELECT c.CustomerName, o.Product

FROM Customers AS c

INNER JOIN Orders AS o

ON c.CustomerID = o.CustomerID;

Result:

|  |  |
| --- | --- |
| CustomerName | Product |
| Alice | Laptop |
| Bob | Headphones |
| Carol | Smartphone |

**2.1 Left (Outer) 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 c.CustomerName, o.Product

FROM Customers AS c

LEFT JOIN Orders AS o

ON c.CustomerID = o.CustomerID;

Result:

|  |  |
| --- | --- |
| CustomerName | Product |
| Alice | Laptop |
| Bob | Headphones |
| Carol | Smartphone |
| Danny | null |

In this example, the left outer join includes all rows from the Customers table.

**2.2 Right (Outer) 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:

SELECT c.CustomerName, o.Product

FROM Customers AS c

RIGHT JOIN Orders AS o

ON c.CustomerID = o.CustomerID;

Result:

|  |  |
| --- | --- |
| CustomerName | Product |
| Alice | Laptop |
| Bob | Headphones |
| Carol | Smartphone |
| null | Watch |

**2.3 Full Join (Full Outer 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.
* In MySQL, there is no keyword like FULL JOIN, so we use union of left join and right join.

Syntax:

LEFT JOIN

UNION

RIGHT JOIN

Example:

SELECT c.CustomerName, o.Product

FROM Customers AS c

LEFT JOIN Orders AS o

ON c.CustomerID = o.CustomerID

UNION

SELECT c.CustomerName, o.Product

FROM Customers AS c

RIGHT JOIN Orders AS o

ON c.CustomerID = o.CustomerID;

Result:

|  |  |  |
| --- | --- | --- |
| CustomerName | Product | |
| Alice | Laptop | |
| Bob | Headphones | |
| Carol | Smartphone | |
| Danny | null | |
| null | | Watch |

**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.

**Syntax**:

SELECT columns

FROM table1

CROSS JOIN table2;

**3. 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.

**Syntax:**

SELECT columns

FROM table1 AS alias1

JOIN table1 AS alias2

ON alias1.column = alias2.column;

**Example**: Consider an Employees table that contains information about employees and their managers.

**Employees Table:**

|  |  |  |
| --- | --- | --- |
| EmployeeID | EmployeeName | ManagerID |
| 1 | Alice | 3 |
| 2 | Bob | 3 |
| 3 | Carol | NULL |
| 4 | Danny | 1 |

**Self Join Query:**

SELECT e1.EmployeeName AS Employee, e2.EmployeeName AS Manager

FROM Employees AS e1

JOIN Employees AS e2

ON e1.ManagerID = e2.EmployeeID;

**Result**:

|  |  |
| --- | --- |
| Employee | Manager |
| Alice | Carol |
| Bob | Carol |
| Danny | Alice |

**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.

**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.

**MySQL View:**

* A view is a **virtual table** created by a query by joining one or more tables.
* It is operated similarly to the base table but does not contain any data of its own.
* If any changes occur in the underlying table, the same changes reflected in the View also.

Syntax:

CREATE [OR REPLACE] **VIEW** view\_name **AS**

**SELECT** columns

**FROM** tables

[**WHERE** conditions];

\*Note:- **OR REPLACE**: It is optional. It is used when a VIEW already exists. If you do not specify this clause and the VIEW already exists, the CREATE VIEW statement will return an error.

**To see the created VIEW:**

SELECT \* FROM view\_name;

**MySQL Update VIEW:**

**ALTER** **VIEW** view\_name **AS**

**SELECT** columns

**FROM** table\_name

**WHERE** conditions;

**MySQL Drop VIEW:**

DROP VIEW [IF EXISTS] view\_name;