**SQL Notes**

**Database(DB):**

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

**Database Management System(DBMS)**:

It is a software application that is used to manage our DB.

**Why do we need DBMS?**

* Efficient data storage and retrieval
* Data management and manipulation
* Ensuring data integrity and security
* Managing large amounts of data
* Supporting multi-user access

**Types of Databases**

Relational Databases(RDBMS) NoSQL Databases

**Relational Databases(RDBMS):**

* Based on a structured schema and tables with rows and columns.
* We use SQL to work with relational DBMS.
* Ex: MySQL, PostgreSQL, Oracle Databases, Microsoft SQL server.

**NoSQL Databases:**

* Designed for unstructured, semi-structured, or rapidly changing data.
* Offers high scalability and flexibility.
* Ex: document in a json format (MongoDB), key-value (Redis), Graph(neo4j) etc.

**What is SQL?**

Structured Query Language(SQL) is a programming language used to interact with **relational databases.**

It is used to perform CRUD operations:

**Create**

**Read**

**Update**

**Delete**

**How to Create Database in SQL?**

CREATE DATABASE IF NOT EXISTS db\_name;

**How can we drop it?**

DROP DATABASE IF EXISTS db\_name;

**What is a Table?**

A table is a structured way of organizing and storing data. It consists of rows and columns, making it easy to manage and query information.

**How can we create a table using SQL?**

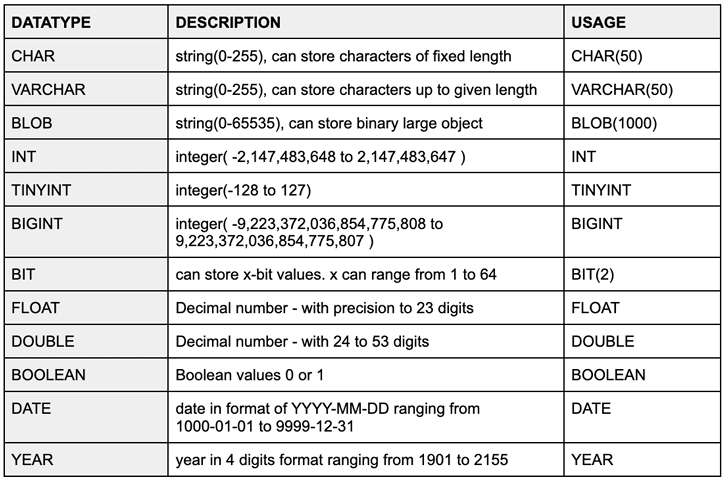
CREATE TABLE table\_name (

column\_name1 datatype constraint,

column\_name2 datatype constraint);

**NOTE:** Before creating any table, we first have to select our db in which we want to create our table using ***USE******db\_name*** command.

**SQL Datatypes:**



**Types of SQL Commands:**

* **Data Definition Language(DDL):** create, alter, rename, truncate & drop
  + **Note:** TRUNCATE removes all rows from a table, effectively resetting the table to an empty state. It is faster.
* **Data Query Language(DQL):** select
* **Data Manipulation Language(DML):** insert, update & delete
* **Data Control Language(DCL):** grant & revoke permission to users
* **Transaction Control Language(TCL):** It is used to manage transactions in a database, ensuring data integrity. The TCL commands include start transaction, commit, rollback etc.

**Data Definition Language(DDL) Example:**

**CREATE TABLE Students (**

**StudentID INT PRIMARY KEY,**

**Name VARCHAR(50),**

**Age INT,**

**Department VARCHAR(50)**

**);**

Here:

* StudentID is the primary key (unique identifier)
* FirstName and LastName are character fields with a maximum length of 50
* Age is an integer
* Department stores the department name

**Data Query Language(DQL) Example:**

*SELECT StudentID, Name FROM Students WHERE Department = 'Sales';*

**To see all the tables:**

*SHOW tables;*

**To See all the dbs:**

*SHOW DATABASES*

*SELECT DISTINCT Name FROM Students;*

**Data Manipulation Language(DML) Example:**

**INSERT INTO Students (StudentID, Name, Age, Department)**

**VALUES (1, 'John Doe', 20, 'Computer Science');**

UPDATE Students

SET Age = 21

WHERE StudentID = 1;

DELETE FROM Students

WHERE StudentID = 2;

**Data Control Language(DCL) Example:**

**REVOKE INSERT ON Students FROM user1;**

**GRANT SELECT, INSERT ON Students TO user1;**

**Transaction Control Language(TCL) Example:**

**BEGIN TRANSACTION;**

**UPDATE accounts**

**SET balance = balance - 500**

**WHERE account\_id = 1;**

**UPDATE accounts**

**SET balance = balance + 500**

**WHERE account\_id = 2;**

**-- If everything is successful, commit the changes**

**COMMIT;**

**-- If there is an error, rollback the transaction**

**ROLLBACK;**

**Constraints:**

Constraints are rules enforced on table columns to maintain the integrity, validity, and consistency of the data.

* **PRIMARY KEY**: Ensures a unique & not null identifier for each record.
* **FOREIGN KEY**: Establishes relationships between tables.
* **NOT NULL**: Ensures that a column cannot have NULL values.
* **UNIQUE**: Ensures all values in a column(s) are unique.
* **CHECK**: Validates data based on specific conditions.
* **DEFAULT**: Provides a default value for a column if no value is specified.

**Super Key, Candidate Key, and Alternate Key:**

**Example**

**Student Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **StudentID** | **FullName** | **Email** | **Phone** |
| **101** | **Sudhanshu Gupta** | [**sudhanshu@gmail.com**](mailto:sudhanshu@gmail.com) | **1111111XXXX** |
| **102** | **Sayan Kundu** | [**sayan@gmail.com**](mailto:sayan@gmail.com) | **2222222XXXX** |
| **103** | **Aryan Roy** | **aryan@gmail.com** | **3333333XXXX** |

**Super Keys:** A Super Key is any combination of columns that can uniquely identify each row.

* **{StudentID}**
* **{StudentID, Email}**
* **{StudentID, FullName}**
* **{StudentID, Email, Phone}**
* **{Email}**
* **{Email, Phone}**

**Candidate Key:** A Candidate Key is the minimal **subset of super keys**, meaning it must uniquely identify each row and contain no extra columns.

* **{StudentID}**
* **{Email}**

**Alternate Key:** When one of the Candidate Keys is chosen as the Primary Key, the other becomes the Alternate Key. For example:

- If **{StudentID}** is selected as the Primary Key, then **{Email}** becomes the Alternate Key.

**Operator Used with Where Clause:**

* **Arithmetic Operators:** +(addition) , -(subtraction), \*(multiplication), /(division), %(modulus)
* **Comparison Operators:** =(equal to), != (not equal to), > , >=, <=
* **Logical Operators:** AND, OR , NOT, IN, BETWEEN, ALL, LIKE(%, \_ , [], [^ ] or [! ], -), ANY
* **Bitwise Operators:** & (Bitwise AND), | (Bitwise OR)
* **Wildcard characters**(used for pattern matching): %, \_ , [],

**1. %:** Matches zero or more characters.

* Example: `'A%'` matches any string starting with 'A' (e.g., "Arjun", "Anil", "Amrita").

**2. \_:** Matches exactly one character.

* Example: `'A\_'` matches any string with two characters starting with 'A' (e.g., "An", "Aj").

**3. [ ]:** Matches any single character within the brackets.

* Example: `'J[aeiou]n'` matches "Jan", "Jen", "Jon", etc.

**4. [^ ] or [! ]:** Matches any single character not within the brackets.

* Example: `'J[^aeiou]n'` matches "Jkn", "Jmn", etc.

**5. - :** Specifies a range of characters within brackets.

* Example: `'J[a-c]n'` matches "Jan", "Jbn", "Jcn".

**Aggregate functions:**

It performs a calculation on a set of values, and return a single value. Eg:

* COUNT( )
* MAX( )
* MIN( )
* SUM( )
* AVG( )

**Get Maximum marks:**

*SELECT max(marks) FROM students;*

**Clauses:**

In SQL, clauses are used to specify **actions, conditions, and constraints** while querying or manipulating data in a database. Some most commonly used clauses:

1. **SELECT Clause:** Specifies the columns to retrieve from a table.

- Example: `SELECT Name, Age FROM Employees;`

2. **FROM Clause:** Indicates the table(s) to retrieve data from.

- Example: `SELECT \* FROM Employees;`

3. **WHERE Clause:** Filters rows based on a specified condition.

- Example: `SELECT \* FROM Employees WHERE Age > 30;`

4. **GROUP BY Clause:** Groups rows that have the same values in specified columns.

- Example: `SELECT Department, COUNT(\*) FROM Employees GROUP BY Department;`

5. **HAVING Clause:** Filters groups of data based on aggregate functions.

- Example: `SELECT Department, AVG(Salary) FROM Employees GROUP BY Department HAVING AVG(Salary) > 50000;`

6. **ORDER BY Clause:** Sorts the result set in ascending (`ASC`) or descending (`DESC`) order.

- Example: `SELECT \* FROM Employees ORDER BY Salary DESC;`

7. **LIMIT / OFFSET Clause:** Limits the number of rows returned or skips rows in the result set (commonly used in MySQL and PostgreSQL).

- Example: `SELECT \* FROM Employees LIMIT 5 OFFSET 10;`

8**. JOIN Clause:** Combines rows from two or more tables based on a related column.

- Example: `SELECT Employees.Name, Departments.DepartmentName FROM Employees JOIN Departments ON Employees.DepartmentID = Departments.ID;`

9. **UNION Clause:** Combines the results of two or more `SELECT` statements.

- Example: `SELECT Name FROM Employees UNION SELECT Name FROM Managers;`

10. **INSERT INTO, UPDATE, and DELETE Clauses:** Used for manipulating data in a table.

- Example (INSERT): `INSERT INTO Employees (Name, Age) VALUES ('Rahul', 28);`

**General Order:**

*SELECT column(s)*

*FROM table\_name*

*WHERE condition*

*GROUP BY column(s)*

*HAVING condition*

*ORDER BY column(s) ASC*

*LIMIT num*

*OFFSET num;*

**Foreign Key:**

A foreign key is a column(or set of columns) in a table that refers to the primary key of another table. It can have duplicate & null values. It is used to enforce relationships between tables.

**Purpose**:

* Enforces referential integrity between tables.
* Ensures valid and consistent relationships between tables.

**Key Features**:

* A foreign key can accept NULL values (unless specified otherwise).
* It allows you to define parent-child relationships between tables.

**Ex:**

**Table 1: Departments**

|  |  |
| --- | --- |
| **DepartmentID (Primary Key)** | **DepartmentName** |
| 1 | HR |
| 2 | Sales |
| 3 | IT |

**Table 2: Employees**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

EmployeeName VARCHAR(100),

DepartmentID INT,

**FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)**

);

|  |  |  |
| --- | --- | --- |
| **EmployeeID (Primary Key)** | **EmployeeName** | **DepartmentID (Foreign Key)** |
| 101 | John | 1 |
| 102 | Sarah | 2 |
| 103 | Mike | NULL |

**Explanation:**

* **DepartmentID** in the Employees table is a foreign key referencing the **DepartmentID** in the Departments table.
* Currently, the Employees table allows NULL values in the **DepartmentID** column, meaning an employee may not belong to any department.

**Constraint Example:**

ALTER TABLE Employees

MODIFY DepartmentID INT NOT NULL;

|  |  |  |
| --- | --- | --- |
| **EmployeeID** | **EmployeeName** | **DepartmentID** |
| 101 | John | 1 |
| 102 | Sarah | 2 |

**Cascading for FK**

**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.

**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

*CREATE TABLE Student(*

*Id INT PRIMARY KEY,*

*CourseID INT,*

*FOREIGN KEY(CourseID) REFERENCES course(ID)*

*ON DELETE CASCADE*

*ON UPDATE CASCADE*

*);*

**Relations:**

A relation in SQL is a table consisting of rows (tuples) and columns (attributes), representing entities and their data.

Relation Types:

* **One-to-One:** Each row in one table is linked to exactly one row in another table.
* **One-to-Many:** A row in one table can be associated with multiple rows in another.
* **Many-to-Many:** Rows in one table can relate to multiple rows in another, often requiring a junction table.

Example of Relationship (One-to-Many):

Suppose you have Departments and Employees tables, each department can have multiple employees (e.g., *DepartmentID* 2 has Priya and Meena).

If you want to retrieve all employees along with their department names, you'd use a JOIN:

*SELECT Employees.Name, Departments.DepartmentName*

*FROM Employees*

*JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;*

**Joins:**

Join is used to combine rows from two or more tables, based on a related column between them.

**Types of joins:**

**In SQL, joins are used to combine data from two or more tables based on a related column. Here are the types of joins:**

**1. INNER JOIN:**

* Retrieves records that have matching values in both tables.
* Example: Employees and their Departments where IDs match.

*SELECT Employees.Name, Departments.DepartmentName*

*FROM Employees*

*INNER JOIN Departments*

*ON Employees.DepartmentID = Departments.DepartmentID;*

**2. LEFT JOIN (or LEFT OUTER JOIN):**

Retrieves all records from the left table, and matching records from the right table. Non-matching records from the right table are set to NULL.

Example: Employees with departments (even if no department is assigned).

**3. RIGHT JOIN (or RIGHT OUTER JOIN):**

Retrieves all records from the right table, and matching records from the left table. Non-matching records from the left table are set to NULL.

**4. FULL JOIN (or FULL OUTER JOIN):**

Retrieves all records where there's a match in either table. Non-matching rows from both tables are included, with NULLs where no match exists.

**5. CROSS JOIN:**

Produces the Cartesian product, combining every row from the first table with every row from the second table.

**6. SELF JOIN:**

Joins a table to itself based on a related column.

