Computer Science By SAMPAT LILER

These complete notes have been made for class 12th board computer science exam.

9. Structured Query Language (SQL)

Introduction

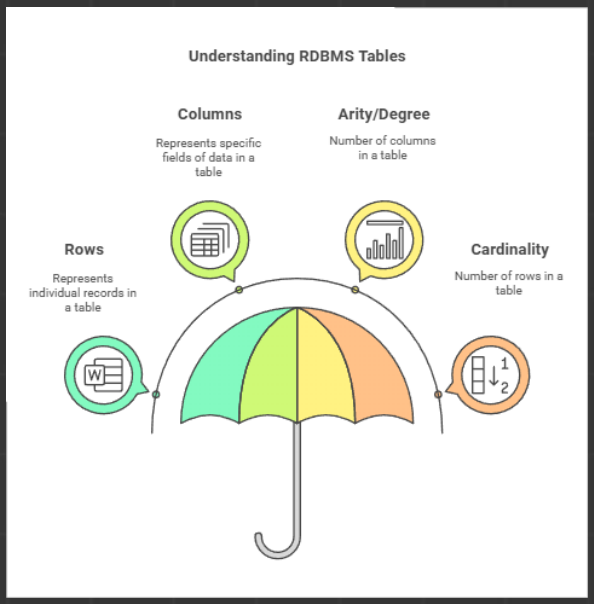
Structured Query Language (SQL) is a standard programming language used to manage and manipulate databases. SQL is used with relational database management systems (RDBMS) such as MySQL, Microsoft SQL Server, PostgreSQL, and Oracle to create, retrieve, update, and delete data efficiently.

Relational Database Management System (RDBMS) Concepts

1. **Table (Relation)**
   A table in RDBMS is a structured collection of related data, organized in rows and columns.
   Example: Student

```
+----+--------+-----+
| ID | Name   | Age |
+----+--------+-----+
| 1  | Sampat | 22  |
| 2  | Sunita | 21  |
| 3  | Prem   | 23  |
+----+--------+-----+
```

2. **Row (Tuple/Record)**
   - A row represents a single record in a table.
   - Example: (1, Alice, 22) is a row in the **Student** table.
3. **Column (Attribute/Field)**
   - A column represents a specific field of data in a table.
   - Example: **ID**, **Name**, and **Age** are columns in the **Student** table.
4. **Arity/Degree**
   - The **degree** of a table is the number of columns (attributes) it has.
   - Example: The **Student** table has **3 columns**, so its **degree = 3**.
5. **Cardinality**
   - The **cardinality** of a table is the number of rows (tuples) in it.
   - Example: The **Student** table has **3 rows**, so its **cardinality = 3**.



Features of SQL:

- SQL is case insensitive.
- SQL statements end with a semicolon (;).
- SQL provides statements for data definition, manipulation, and retrieval.
- SQL allows constraints to maintain data integrity.
- SQL supports functions for performing operations on data.

Data Types and Constraints in MySQL

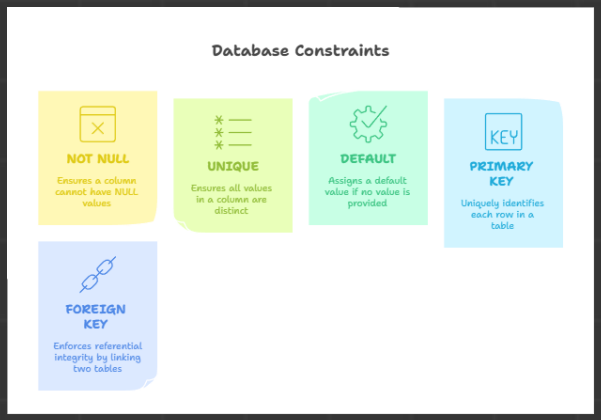
Data Types of Attribute

Data Type	Description
CHAR(n)	Fixed-length character string (0 to 255 characters)
VARCHAR(n)	Variable-length character string (0 to 65535 characters)
INT	Integer values
FLOAT	Floating-point values
DATE	Stores dates in 'YYYY-MM-DD' format

Constraints

Constraints are rules in SQL that help maintain the integrity and accuracy of data in a table. They ensure that the data remains valid and consistent.

Constraint	Description
NOT NULL	Ensures a column cannot have NULL values
UNIQUE	Ensures all values in a column are distinct
DEFAULT	Assigns a default value if no value is provided
PRIMARY KEY	Uniquely identifies each row in a table
FOREIGN KEY	Enforces referential integrity by linking two tables

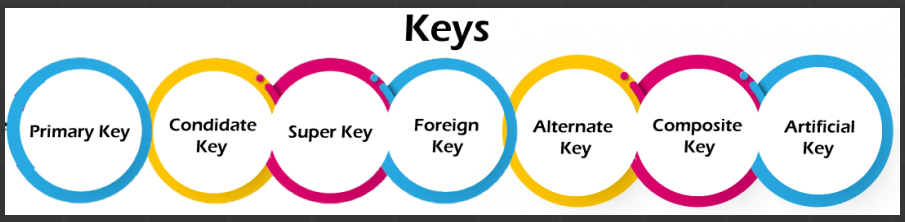


Example:

1. **PRIMARY KEY**
   - Uniquely identifies one or more columns in a table.
   - Example:
   - CREATE TABLE STUDENT (
     RollNumber INT PRIMARY KEY,
     SName VARCHAR(30) NOT NULL
     );
2. **FOREIGN KEY**
   - Links a column in one table to a column in another table.
   - Example:
     ALTER TABLE STUDENT ADD FOREIGN KEY (GUID) REFERENCES GUARDIAN(GUID);
3. **NOT NULL**
   - Prevents a column from having NULL values.
   - Example:
     ALTER TABLE STUDENT MODIFY SName VARCHAR(30) NOT NULL;
4. **UNIQUE**
   - Ensures that all values in a column are unique (no duplicates allowed).
   - Example:
     ALTER TABLE GUARDIAN ADD UNIQUE(GPhone);
5. **DEFAULT**
   - Assigns a default value to a column if no value is provided.
   - Example:
     ALTER TABLE STUDENT MODIFY SDateofBirth DATE DEFAULT '2000-01-01';
6. **CHECK**
   - Enforces a condition on the values of a column.
   - Example:
     ALTER TABLE STUDENT ADD CHECK (Age >= 18);
7. **AUTO_INCREMENT**
   - Automatically generates values, usually for primary keys.
   - Example:

- CREATE TABLE STUDENT (
  RollNumber INT PRIMARY KEY AUTO_INCREMENT,
  SName VARCHAR(30) NOT NULL
  );

Types of keys



1. Candidate Key

A **candidate key** is the **minimal set of attributes** that can uniquely identify a tuple (record) in a table.

Key Points:

✅ A **candidate key** is a **super key** with no extra attributes.

✅ It ensures **uniqueness**, meaning no two records have the same value in the candidate key's columns.

✅ Every table must have **at least one candidate key**.

✅ A table can have **multiple candidate keys**, but only **one primary key** is selected from them.

✅ Example: **STUD_NO** in the **STUDENT** table can be a candidate key if it uniquely identifies each student.

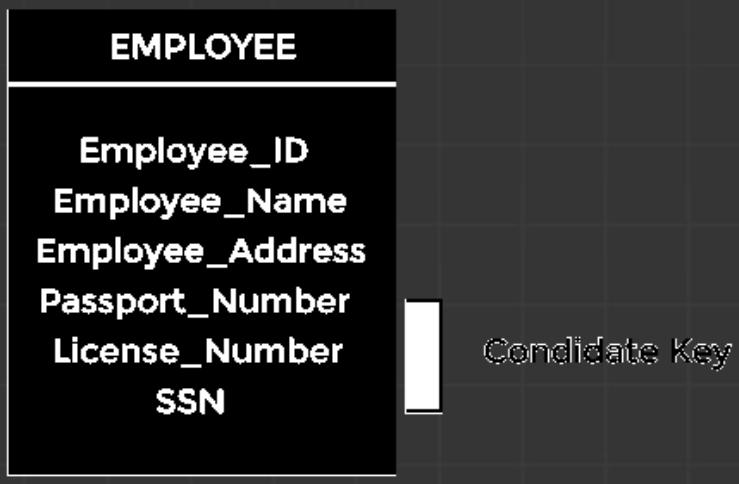
✅ A candidate key **cannot have duplicate or null values**.

➤ **Composite Candidate Key:** Sometimes, a single column is not enough to uniquely identify a record. In such cases, a **combination of columns** forms a **candidate key**.

- Example: (**STUD_ID, COURSE_ID**) together might be a candidate key in a university database.

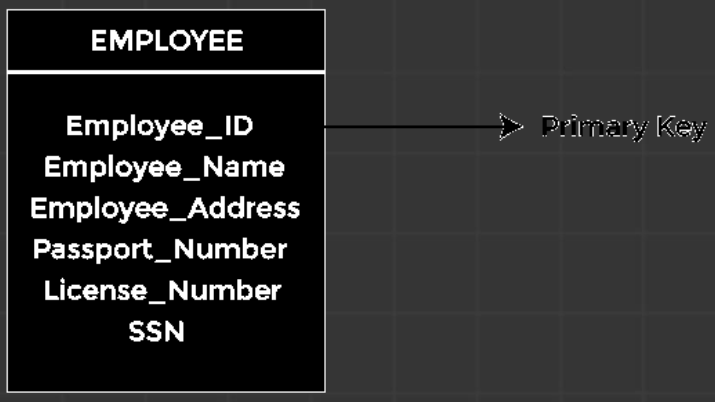
➤ **Difference between Candidate Key & Super Key:**

- **Super Key**: A set of one or more attributes that can uniquely identify a record but may contain **extra attributes**.
- **Candidate Key**: A **minimal** version of a **super key** (i.e., it has **no unnecessary attributes**).



2. Primary Key

A **Primary Key** is a unique identifier for each record in a database table. It ensures that no two rows have the same value for that key and that the value is never NULL.



Key Characteristics of a Primary Key

1. **Uniqueness** – Each value in the primary key column must be unique.
2. **Non-NULL** – A primary key cannot have NULL values.
3. **Single Column or Multiple Columns** – A primary key can consist of a single column (simple key) or multiple columns (composite key).
4. **Automatic Indexing** – Many databases automatically create an index for the primary key to improve search performance.

Example of Primary Key

Consider the following STUDENT table:

Student_ID	Name	Age	Course
101	Rahul	18	B.Tech
102	Priya	19	BCA
103	Amit	18	B.Sc
104	Suman	20	BBA

- Here, Student_ID is the **Primary Key** because each student has a unique ID.
- No two students can have the same Student_ID, and it cannot be NULL.

SQL Query to Define a Primary Key

```
CREATE TABLE STUDENT (
   Student_ID INT PRIMARY KEY,
   Name VARCHAR(50),
   Age INT,
   Course VARCHAR(50)
);
```

Composite Primary Key

When a table requires more than one column to uniquely identify a row, a **Composite Primary Key** is used.

Example: Composite Primary Key

Consider the following ENROLLMENT table where a student enrolls in multiple courses:

Student_ID	Course_ID	Enrollment_Date
101	CSE101	2024-01-15
101	CSE102	2024-01-16
102	CSE101	2024-01-15

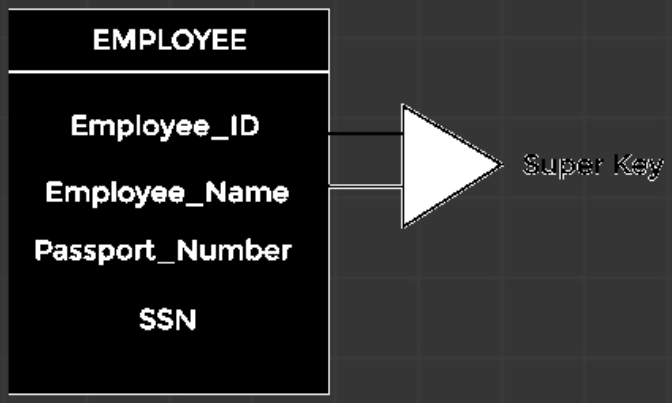
Here, Student_ID alone is not enough because a student can enroll in multiple courses. Similarly, Course_ID alone is not unique. So, we define a **Composite Primary Key** using both Student_ID and Course_ID.

SQL Query for Composite Primary Key

```
CREATE TABLE ENROLLMENT (
   Student_ID INT,
   Course_ID VARCHAR(10),
   Enrollment_Date DATE,
   PRIMARY KEY (Student_ID, Course_ID)
);
```

3. Super Key

A **Super Key** is a set of one or more attributes (columns) that can uniquely identify a tuple (row) in a relation (table).



Key Characteristics of Super Key:

1. **Uniqueness** – A super key uniquely identifies each row in a table.
2. **May Contain Extra Attributes** – A super key may have unnecessary attributes along with the primary key.
3. **Includes Candidate Key & Primary Key** – Every candidate key and primary key is also a super key.
4. **There Can Be Multiple Super Keys** – A table can have multiple super keys.

4. Foreign key

A **Foreign Key** is a column or a set of columns in one table that establishes a link to the **Primary Key** of another table. It helps maintain **referential integrity** between the two tables by ensuring that the value in the foreign key column corresponds to a valid value in the referenced primary key column.

Key Points:

- A **Foreign Key** is used to establish relationships between two tables.
- It references the **Primary Key** of another table.
- Ensures **referential integrity**, preventing invalid data entry.
- Helps in implementing **One-to-Many** and **Many-to-Many** relationships.
- If a referenced row is deleted or updated, the foreign key constraints define how the dependent rows behave (CASCADE, SET NULL, etc.).

Example of Foreign Key in SQL

Consider two tables: Students and Courses

1. Courses Table (Primary Table / Referenced Table)

Course_ID (Primary Key)	Course_Name
101	Mathematics
102	Physics
103	Chemistry

2. Students Table (Child Table / Referenced Table)

Student_ID (Primary Key)	Student_Name	Course_ID (Foreign Key)
1	Amit	101
2	Raj	102
3	Priya	103
4	Rohit	101

Here, **Course_ID** in the Students table is a **Foreign Key** referencing **Course_ID** in the Courses table.

Creating Tables with Foreign Key in SQL

```
CREATE TABLE Courses (
  Course_ID INT PRIMARY KEY,
  Course_Name VARCHAR(50) NOT NULL
);
```

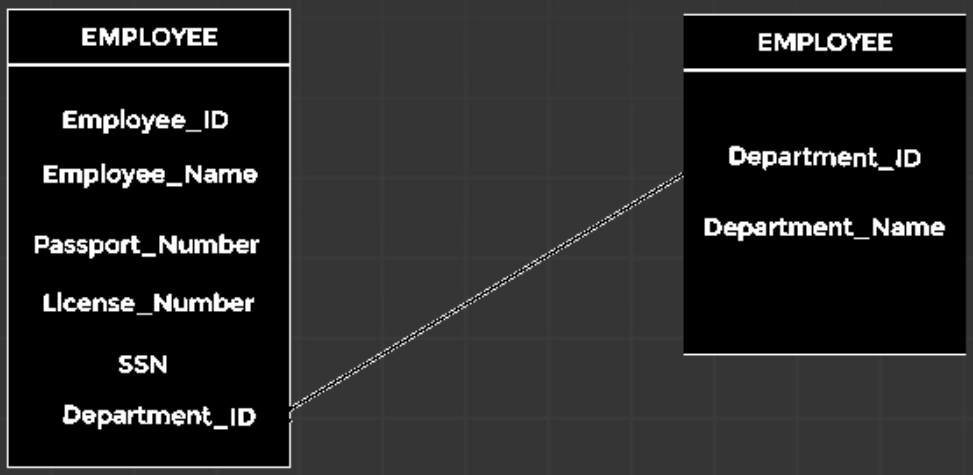
```
CREATE TABLE Students (
  Student_ID INT PRIMARY KEY,
  Student_Name VARCHAR(50) NOT NULL,
  Course_ID INT,
  FOREIGN KEY (Course_ID) REFERENCES Courses(Course_ID) ON DELETE CASCADE
);
```

Explanation:

- Course_ID in the **Courses** table is the **Primary Key**.
- Course_ID in the **Students** table is a **Foreign Key**, referencing the Course_ID of the **Courses** table.
- ON DELETE CASCADE: If a course is deleted from the Courses table, all students enrolled in that course will be deleted automatically.

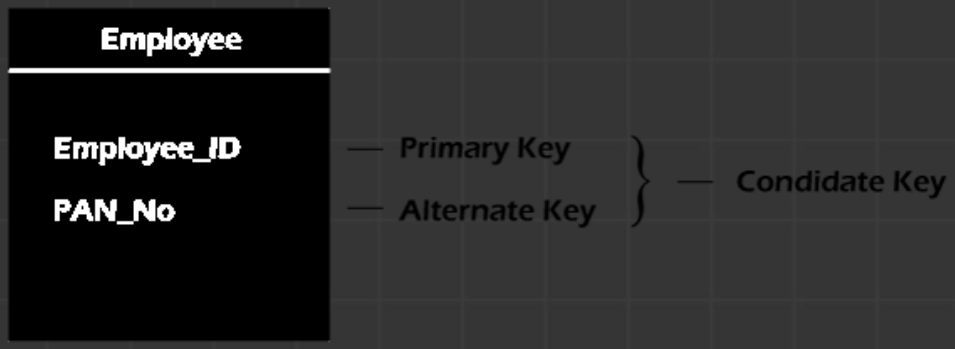
Advantages of Foreign Key

✅ **Maintains Data Integrity** – Prevents invalid data entry.
✅ **Reduces Redundancy** – Eliminates duplicate data storage.
✅ **Ensures Consistency** – Keeps related data synchronized.
✅ **Supports Relationships** – Essential for relational database design.



5. Alternate key

There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key. **In other words,** the total number of the alternate keys is the total number of candidate keys minus the primary key. The alternate key may or may not exist. If there is only one candidate key in a relation, it does not have an alternate key. **For example,** employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.

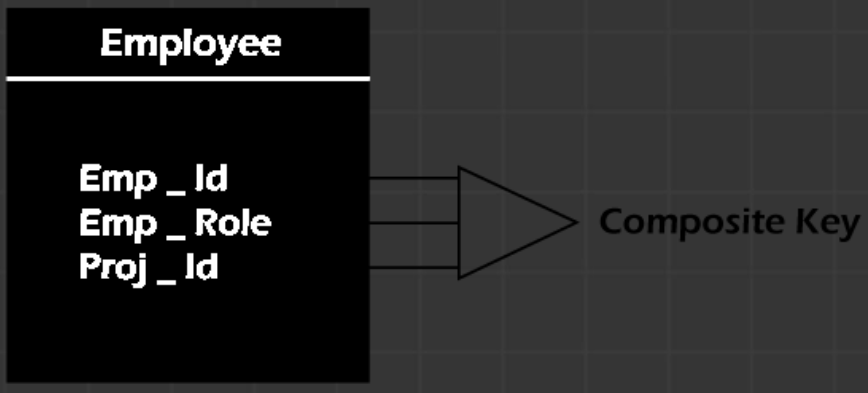


The objective of alternate key is:

- An alternative key's main goal is to make sure that no two entries in a table have the same set of attribute values combined. It provides a different method of record identification from the primary key.
- By providing a variety of possibilities for uniquely identifying entries, alternate keys facilitate flexibility in database architecture. Different qualities may be effectively employed to identify records based on certain use cases.

6. Composite key

Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.



For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.

The objective of alternate key is:

- An alternative key's main goal is to make sure that no two entries in a table have the same set of attribute values combined. It provides a different method of record identification from the primary key.
- By providing a variety of possibilities for uniquely identifying entries, alternate keys facilitate flexibility in database architecture. Different qualities may be effectively employed to identify records based on certain use cases.

7. Artificial key

The key created using arbitrarily assigned data are known as artificial keys. These keys are created when a primary key is large and complex and has no relationship with many other relations. The data values of the artificial keys are usually numbered in a serial order.

For example, the primary key, which is composed of Emp_ID, Emp_role, and Proj_ID, is large in employee relations. So it would be better to add a new virtual attribute to identify each tuple in the relation uniquely.

8. Surrogate Key -

A key is a column, or group of columns, in a database management system (DBMS) that uniquely identifies every row in a table. Natural keys and surrogate keys are the two categories of keys.

Natural Key: A column, or group of columns, that is generated from the table's data is known as a natural key. For instance, since it uniquely identifies every client in the table, the customer ID column in a customer table serves as a natural key.

Surrogate key: A column that is not generated from the data in the database is known as a surrogate key. Rather, the DBMS generates a unique identifier for you. In database tables, surrogate keys are frequently utilized as primary keys.

SQL for Data Definition

SQL (Structured Query Language) is used to create, modify, and delete databases and tables. Data Definition Language (DDL) is a part of SQL that defines the database structure.

CREATE Database

Command:

```
CREATE DATABASE StudentAttendance;
```

Explanation:

This command is used to create a new database. Here, a new database named **"StudentAttendance"** is created.

CREATE Table

Command:

```
CREATE TABLE GUARDIAN (
   GUID CHAR(12) PRIMARY KEY,
   GName VARCHAR(20) NOT NULL,
   GPhone CHAR(10) UNIQUE,
   GAddress VARCHAR(30) NOT NULL
);
```

Explanation:

This command creates a table named **"GUARDIAN"** with the following columns:

Column Name	Data Type	Constraints
GUID	CHAR(12)	PRIMARY KEY
GName	VARCHAR(20)	NOT NULL
GPhone	CHAR(10)	UNIQUE
GAddress	VARCHAR(30)	NOT NULL

Describe Table

Command:

```
DESCRIBE STUDENT;
```

Explanation:

This command is used to view the structure of a table. It displays the column names, data types, and constraints of the table **"STUDENT"**.

ALTER Table

(A) Add Primary Key

Command:

```
ALTER TABLE STUDENT ADD PRIMARY KEY (RollNumber);
```

Explanation:

This command adds a **PRIMARY KEY** constraint to the **"RollNumber"** column in the **"STUDENT"** table.

(B) Add Foreign Key

Command:

```
ALTER TABLE STUDENT ADD FOREIGN KEY (GUID) REFERENCES GUARDIAN(GUID);
```

Explanation:

This command links the **"GUID"** column in the **"STUDENT"** table to the **"GUID"** column in the **"GUARDIAN"** table, making it a **FOREIGN KEY**.

(C) Add UNIQUE Constraint

Command:

```
ALTER TABLE GUARDIAN ADD UNIQUE(GPhone);
```

Explanation:

This command ensures that the **"GPhone"** column in the **"GUARDIAN"** table contains only unique values.

(D) Add an Attribute (Column)

Command:

```
ALTER TABLE GUARDIAN ADD Income INT;
```

Explanation:

This command adds a new column **"Income"** of **integer** data type to the **"GUARDIAN"** table.

Column Name	Data Type	Constraints
Income	INT	None

(E) Modify Data Type

Command:

```
ALTER TABLE GUARDIAN MODIFY GAddress VARCHAR(40);
```

Explanation:

This command modifies the **"GAddress"** column's data type from **VARCHAR(30)** to **VARCHAR(40)**.

(F) Modify Constraint

Command:

```
ALTER TABLE STUDENT MODIFY SName VARCHAR(20) NOT NULL;
```

Explanation:

This command adds a **NOT NULL** constraint to the **"SName"** column in the **"STUDENT"** table, ensuring it cannot contain NULL values.

(G) Add Default Value

Command:

```
ALTER TABLE STUDENT MODIFY SDateofBirth DATE DEFAULT '2000-05-15';
```

Explanation:
This command sets the default value of the **"SDateofBirth"** column in the **"STUDENT"** table to **'2000-05-15'**.

(H) Remove an Attribute (Column)
Command:

```
ALTER TABLE GUARDIAN DROP Income;
```

Explanation:
This command removes the **"Income"** column from the **"GUARDIAN"** table.

(I) Remove Primary Key
Command:

```
ALTER TABLE GUARDIAN DROP PRIMARY KEY;
```

Explanation:
This command removes the **PRIMARY KEY** constraint from the **"GUARDIAN"** table.

DROP Statement
(A) Drop Table
Command:

```
DROP TABLE STUDENT;
```

Explanation: Is command se "STUDENT" table ko permanently delete kar diya gaya hai.
(B) Drop Database
Command:

```
DROP DATABASE StudentAttendance;
```

Explanation: Is command se pura "StudentAttendance" database delete kar diya gaya hai.

Practice Question
Create a database named SchoolRecords. Then, create a table STUDENT with the following columns:

Column Name	Data Type	Constraints
RollNumber	INT	PRIMARY KEY
SName	VARCHAR(30)	NOT NULL
Age	INT	
GUID	CHAR(12)	FOREIGN KEY (References GUARDIAN)
SDateofBirth	DATE	DEFAULT '2000-01-01'

Then, perform the following operations:

1. Modify SName to VARCHAR(50) NOT NULL.
2. Add a new column Email of type VARCHAR(50).
3. Drop the column Age.
4. Delete the STUDENT table.
5. Drop the SchoolRecords database.

Write SQL queries to perform these operations.

SQL for Data Manipulation
Insertion of Records

```
INSERT INTO STUDENT VALUES (1, 'Sampat Liler', '2003-05-15', '444444444444');
```

SQL for Data Query
SELECT Statement
Definition: The SELECT statement is the most basic and powerful SQL command used to retrieve data from a database. It allows us to filter specific columns or rows as needed.
Example:

```
SELECT SName, SDateofBirth FROM STUDENT WHERE RollNumber = 1;
```

STUDENT Table:

RollNumber	SName	SDateofBirth
1	Rahul	2003-05-21
2	Aisha	2004-07-15

Querying Using Database OFFICE

EMPLOYEE Table:

EmpNo	EName	DeptId	Salary	Bonus
101	Sampat	D01	6000	500
102	Mariyam	D02	7500	NULL
103	Insiya	D04	8000	1000
104	Divya	D03	7200	700

(A) Retrieve Selected Columns

Definition: If we need to retrieve only specific columns from a table, we use the SELECT statement with the desired column names.

Example:

```
SELECT EmpNo FROM EMPLOYEE;
```

Output:

EmpNo
101
102
103
104

(B) Renaming Columns

Definition: Using the AS keyword, we can assign an alias name to a column, which will be displayed under that name in the result set.

Example:

```
SELECT EName AS Name FROM EMPLOYEE;
```

Output:

Name
Sampat
Mariyam
Insiya
Divya

(C) DISTINCT Clause

Definition: The DISTINCT clause eliminates duplicate values, ensuring that only unique values appear in the output.

Example:

```
SELECT DISTINCT DeptId FROM EMPLOYEE;
```

Output:

DeptId
D01
D02
D03
D04

(D) WHERE Clause

Definition: The WHERE clause is used to apply specific conditions so that only rows satisfying the condition are retrieved.

Example:

```
SELECT * FROM EMPLOYEE WHERE Salary > 5000 AND DeptId = 'D04';
```

Output:

EmpNo	EName	DeptId	Salary	Bonus
103	Insiya	D04	8000	1000

(E) Membership Operator IN

Definition: The IN operator is used to filter records matching multiple values at once.

Example:

```
SELECT * FROM EMPLOYEE WHERE DeptId IN ('D01', 'D02', 'D04');
```

Output:

EmpNo	EName	DeptId
101	Sampat	D01
102	Mariyam	D02
103	Insiya	D04

(F) ORDER BY Clause

Definition: The ORDER BY clause sorts the data in ascending or descending order.

Example:

```
SELECT * FROM EMPLOYEE ORDER BY Salary DESC;
```

Sorted Output:

EmpNo	EName	Salary
103	Insiya	8000
102	Mariyam	7500
104	Divya	7200
101	Sampat	6000

(G) Handling NULL Values

Definition: To handle NULL values, we use the "IS NULL" or "IS NOT NULL" condition.

Example:

```
SELECT * FROM EMPLOYEE WHERE Bonus IS NULL;
```

Output:

EmpNo	EName	Bonus
102	Mariyam	NULL

(H) Substring Pattern Matching

Definition: The LIKE operator is used for pattern matching with '%' and '_' wildcards.

Example:

```
SELECT * FROM EMPLOYEE WHERE EName LIKE 'K%';
```

Output:

EmpNo	EName
104	Divya

Data Updation and Deletion

Data Updation

Definition:

Data updation refers to modifying existing records in a database table using the UPDATE statement. This is useful when we need to change a specific field of a record without altering the entire row.

Syntax:

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

- table_name: The name of the table where data needs to be updated.
- SET column = value: Specifies which column's value should be changed.
- WHERE condition: Defines which row(s) should be updated (if omitted, all rows will be updated).

Example:

Consider the following STUDENT table:

RollNumber	Name	Age	GUID
1	Aman	18	100100100100
2	Rohan	19	100200200200
3	Priya	17	100300300300

Now, we execute the following SQL command:

```
UPDATE STUDENT
SET GUID = 101010101010
WHERE RollNumber = 3;
```

Updated Table:

RollNumber	Name	Age	GUID
1	Aman	18	100100100100
2	Rohan	19	100200200200
3	Priya	17	101010101010

Here, the GUID of the student with RollNumber = 3 has been updated successfully.

Data Deletion

Definition:

Data deletion is the process of removing specific records from a table using the DELETE statement. This helps in managing unwanted or outdated data in a database.

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

- table_name: The name of the table from which data needs to be deleted.
- WHERE condition: Specifies which records should be deleted (if omitted, all records will be deleted).

Example:

Consider the same STUDENT table before deletion:

RollNumber	Name	Age	GUID
1	Aman	18	100100100100
2	Rohan	19	100200200200
3	Priya	17	101010101010

Now, we execute the following SQL command:

```
DELETE FROM STUDENT
WHERE RollNumber = 2;
```

Updated Table after Deletion:

RollNumber	Name	Age	GUID
1	Aman	18	100100100100
3	Priya	17	101010101010

Here, the record of the student with RollNumber = 2 has been removed from the table.

Functions in SQL
Single Row Functions

Function Name	Description	Example	Example Output
Math Functions			
ABS(x)	Returns the absolute value of x	SELECT ABS(-5);	5
CEIL(x)	Returns the smallest integer greater than or equal to x	SELECT CEIL(4.2);	5
FLOOR(x)	Returns the largest integer less than or equal to x	SELECT FLOOR(4.8);	4
ROUND(x, d)	Rounds x to d decimal places	SELECT ROUND(3.14159, 2);	3.14
POWER(x, y)	Returns x raised to the power y	SELECT POWER(2,3);	8
SQRT(x)	Returns the square root of x	SELECT SQRT(16);	4
String Functions			
UCASE(str)	Converts string to uppercase	SELECT UCASE('hello');	HELLO
LCASE(str)	Converts string to lowercase	SELECT LCASE('HELLO');	hello
LENGTH(str)	Returns length of string in characters	SELECT LENGTH('MySQL');	5
CONCAT(str1, str2, ...)	Concatenates strings	SELECT CONCAT('My', 'SQL');	MySQL
SUBSTRING(str, pos, len)	Extracts substring from str starting at pos for len characters	SELECT SUBSTRING('Database', 2, 4);	atab
TRIM(str)	Removes leading and trailing spaces	SELECT TRIM(' MySQL ');	MySQL
LTRIM(str)	Removes leading spaces from a string	SELECT LTRIM(' MySQL');	MySQL
RTRIM(str)	Removes trailing spaces from a string	SELECT RTRIM('MySQL ');	MySQL
Date Functions			
NOW()	Returns current date and time	SELECT NOW();	2025-02-23 14:30:00
CURDATE()	Returns the current date	SELECT CURDATE();	2025-02-23
CURTIME()	Returns the current time	SELECT CURTIME();	14:30:00
YEAR(date)	Extracts the year from a date	SELECT YEAR('2025-02-23');	2025
MONTH(date)	Extracts the month from a date	SELECT MONTH('2025-02-23');	2
DAY(date)	Extracts the day from a date	SELECT DAY('2025-02-23');	23

Conversion Functions			
CAST(x AS type)	Converts a value to a specified data type	SELECT CAST(123 AS CHAR);	'123'
CONVERT(x, type)	Converts a value to a different type	SELECT CONVERT(123, CHAR);	'123'

Aggregate Functions / Multi Row Function

Function Name	Description	Example Query	Example Output
COUNT()	Returns the number of rows that match a specified condition	SELECT COUNT(*) FROM employees;	50 (Total rows)
SUM()	Returns the total sum of a numeric column	SELECT SUM(salary) FROM employees;	500000 (Total salary)
AVG()	Returns the average value of a numeric column	SELECT AVG(salary) FROM employees;	50000 (Average salary)
MAX()	Returns the maximum value in a column	SELECT MAX(salary) FROM employees;	100000 (Highest salary)
MIN()	Returns the minimum value in a column	SELECT MIN(salary) FROM employees;	20000 (Lowest salary)

GROUP BY Clause and HAVING Clause in SQL

The **GROUP BY** clause is used in SQL to group rows that have the same values in specified columns into aggregated data. This is often used with aggregate functions like COUNT(), SUM(), AVG(), MAX(), and MIN().
The **HAVING** clause is used to filter groups based on aggregate functions, similar to how the WHERE clause filters individual rows.

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```

Example with Table

Employee Table (employees)

EmployeeID	Name	Department	Salary
1	Raj	IT	50000
2	Priya	HR	45000
3	Amit	IT	55000
4	Neha	Sales	40000
5	Rahul	HR	48000
6	Sonal	IT	60000

Query using GROUP BY and HAVING

Find the total salary of each department and show only those departments where the total salary exceeds 1,00,000.

```
SELECT Department, SUM(Salary) AS Total_Salary
FROM employees
GROUP BY Department
HAVING SUM(Salary) > 100000;
```

Output

Department	Total_Salary
IT	165000

Explanation

1. The GROUP BY clause groups records by the **Department** column.
2. The SUM(Salary) function calculates the total salary for each department.
3. The HAVING clause filters out groups where the total salary is **less than or equal to 1,00,000**.

Key Differences Between WHERE and HAVING

Feature	WHERE Clause	HAVING Clause
Filters	Filters individual rows before grouping	Filters groups after applying aggregate functions
Used With	SELECT, UPDATE, DELETE	SELECT (with GROUP BY)
Aggregate Functions	Cannot use aggregate functions	Can use aggregate functions

---

Operations on Relations in Database Management System (DBMS)

Relations in DBMS refer to tables that store data in structured formats. Various operations can be performed on these relations to retrieve meaningful data. The most common relational operations include **Union, Intersection, Set Difference, Cartesian Product, and Join.** These operations are fundamental to relational algebra and SQL queries.

---

1. Union (U)

Definition

The **Union** operation combines two relations and returns all unique rows present in either of the relations. Duplicate records are eliminated in the final result.

Conditions for Union:

- Both relations must have the **same number of attributes** (columns).
- The attributes must have the **same data type**.

Example

Consider two tables **Student_A** and **Student_B** representing students from two different batches.

Roll_No	Name	Course
101	Alex	CS
102	Bob	IT
103	Carol	CS

Table: Student_A

Roll_No	Name	Course
103	Carol	CS
104	David	IT
105	Eva	CS

Table: Student_B

Union Output (Student_A ∪ Student_B)

Roll_No	Name	Course
101	Alex	CS
102	Bob	IT
103	Carol	CS
104	David	IT
105	Eva	CS

Carol appears in both tables, but in the final result, duplicates are removed.

---

2. Intersection (∩)

Definition

The **Intersection** operation returns only those rows that are present in both relations.

Example

Using the same tables **Student_A** and **Student_B**, the intersection will retrieve only the common records.

Intersection Output (Student_A ∩ Student_B)

Roll_No	Name	Course
103	Carol	CS

Carol is the only common record in both tables, so it appears in the output.

---

3. Set Difference (-)

Definition

The **Set Difference** operation finds the records that exist in one relation but not in the other. The operation **Student_A - Student_B** will return records that are in **Student_A** but not in **Student_B**.

Example

Set Difference Output (Student_A - Student_B)

Roll_No	Name	Course
101	Alex	CS
102	Bob	IT

These students are present in **Student_A** but not in **Student_B**.

---

4. Cartesian Product (×)

Definition

The **Cartesian Product** operation returns a combination of each row from the first relation with every row from the second relation.

Example

Consider two small tables **Employee** and **Department**.

Emp_ID	Name
1	John
2	Alice

Table: Employee

Dept_ID	Dept_Name
101	HR
102	IT

Table: Department

Cartesian Product Output (Employee × Department)

Emp_ID	Name	Dept_ID	Dept_Name
1	John	101	HR
1	John	102	IT
2	Alice	101	HR
2	Alice	102	IT

Each employee is paired with each department, resulting in **m × n** rows (2 × 2 = 4).

---

5. Join (⋈)

Join (⋈) in DBMS

Definition

The **Join** operation in Database Management Systems (DBMS) is used to combine rows from two or more tables based on a common column. Unlike the Cartesian Product, which combines every row of one table with every row of another, a Join operation applies a condition to filter relevant records. This helps in retrieving meaningful data efficiently.

Joins are crucial in relational databases because data is often stored in multiple tables to maintain normalization. Using Joins, we can reconstruct meaningful information by linking tables logically.

Types of Joins

There are several types of Joins in SQL, each serving a specific purpose:

1. Inner Join

An **Inner Join** returns only those records where there is a match in both tables based on the given condition. If a row in one table does not have a corresponding match in the other table, it will be excluded from the result.

Syntax

```
SELECT A.column1, A.column2, B.column1, B.column2
FROM TableA A
INNER JOIN TableB B
ON A.common_column = B.common_column;
```

Example

Employee Table

Emp_ID	Name	Dept_ID
1	John	101
2	Alice	102
3	Bob	103
4	David	104

Department Table

Dept_ID	Department_Name
101	HR
102	IT
103	Finance

Inner Join Output

```
SELECT Employee.Emp_ID, Employee.Name, Department.Department_Name
FROM Employee
INNER JOIN Department
ON Employee.Dept_ID = Department.Dept_ID;
```

Emp_ID	Name	Department_Name
1	John	HR
2	Alice	IT
3	Bob	Finance

👉 **Note:** Employee "David" (Dept_ID = 104) is not included in the result because there is no matching department.

---

2. Left Join (Left Outer Join)

A **Left Join** returns all the records from the left table and only the matching records from the right table. If there is no match, NULL values are returned for columns from the right table.

Syntax

```
SELECT A.column1, A.column2, B.column1, B.column2
FROM TableA
LEFT JOIN TableB
ON A.common_column = B.common_column;
```

Example Output

```
SELECT Employee.Emp_ID, Employee.Name, Department.Department_Name
FROM Employee
LEFT JOIN Department
ON Employee.Dept_ID = Department.Dept_ID;
```

Emp_ID	Name	Department_Name
1	John	HR
2	Alice	IT
3	Bob	Finance
4	David	NULL

👉 **Note:** "David" is included, but since there is no matching Dept_ID in the Department table, NULL appears in the Department_Name column.

---

3. Right Join (Right Outer Join)

A **Right Join** returns all records from the right table and only the matching records from the left table. If a match is not found, NULL values appear for columns from the left table.

Syntax

```
SELECT A.column1, A.column2, B.column1, B.column2
FROM TableA A
RIGHT JOIN TableB B
ON A.common_column = B.common_column;
```

Example Output

```
SELECT Employee.Emp_ID, Employee.Name, Department.Department_Name
FROM Employee
RIGHT JOIN Department
ON Employee.Dept_ID = Department.Dept_ID;
```

Emp_ID	Name	Department_Name
1	John	HR
2	Alice	IT
3	Bob	Finance

NULL	NULL	Sales

👉 **Note:** If there was a department "Sales" in the Department table without any employees assigned, it would appear in the result with NULL values in the Employee columns.

---

4. Full Outer Join

A **Full Outer Join** returns all records from both tables. If there is a match, the corresponding data is displayed; otherwise, NULL values appear where a match is not found.

Syntax

```
SELECT A.column1, A.column2, B.column1, B.column2
FROM TableA A
FULL OUTER JOIN TableB B
ON A.common_column = B.common_column;
```

Example Output

```
SELECT Employee.Emp_ID, Employee.Name, Department.Department_Name
FROM Employee
FULL OUTER JOIN Department
ON Employee.Dept_ID = Department.Dept_ID;
```

Emp_ID	Name	Department_Name
1	John	HR
2	Alice	IT
3	Bob	Finance
4	David	NULL
NULL	NULL	Sales

👉 **Note:** This result includes all employees (even those without a department) and all departments (even those without employees).

Comparison of Join Types

Join Type	Includes Non-Matching Rows?	NULL Values Possible?
Inner Join	No	No
Left Join	Yes (from left table)	Yes (for right table)
Right Join	Yes (from right table)	Yes (for left table)
Full Join	Yes (from both tables)	Yes (both sides)

Finally We say :

Joins are essential in database queries to fetch meaningful data spread across multiple tables. Depending on the requirement, we can choose:

- **Inner Join** for strict matches,
- **Left Join** if we want all data from the left table,
- **Right Join** if we want all data from the right table,
- **Full Outer Join** if we need everything from both tables.

Conclusion

Operation	Description
Union (∪)	Combines two sets and removes duplicates.
Intersection (∩)	Retrieves only common records.
Set Difference (-)	Finds records in one set but not in another.
Cartesian Product (×)	Returns all possible row combinations.
Join (⋈)	Merges tables based on a common column.

Cartesian Product in DBMS

In **DBMS (Database Management System)**, the **Cartesian Product** is a type of **join operation** that returns the **cross combination** of rows from two tables. Each row from the first table is paired with **every row** from the second table.

Formula for Cartesian Product:

If **Table A** has **m** rows and **Table B** has **n** rows, then the result of the Cartesian product will have:

Total Rows=m×n\text{Total Rows} = m \times n

Example:

Consider two tables:

Table: Students

Student_ID	Student_Name
1	Rahul
2	Priya

Table: Courses

Course_ID	Course_Name
101	DBMS
102	Operating System

Cartesian Product Result:

Each row from the **Students** table is combined with each row from the **Courses** table.

Student_ID	Student_Name	Course_ID	Course_Name
1	Rahul	101	DBMS
1	Rahul	102	Operating System
2	Priya	101	DBMS
2	Priya	102	Operating System

Key Points:

1. **No Condition Applied** – Cartesian product does **not use any condition** (like ON or WHERE).
2. **Large Output** – The result set grows **exponentially** (m × n), which can be inefficient for large tables.
3. **Used in Joins** – Cartesian product is often used as an **intermediate step** in **INNER JOIN** or **CROSS JOIN**.

SQL Query Example:

```
SELECT * FROM Students, Courses;
```

This will generate the **Cartesian Product** of the two tables.

When to Use Cartesian Product?

- When you need **all possible combinations** of two tables.
- As an intermediate step in **JOIN operations**.

👉 Subscribe Youtube Channel - Anvira Education - YouTube

👉 Join Course - Https://Anviraeducation.Com/

👉 Follow Us On Facebook - Https://Www.Facebook.Com/Anviraedu

👉 Follow Us On Instagram - https://www.instagram.com/anvira_edu/

👉 Sampat Sir Instagram - https://www.instagram.com/writersampat/

👉 Join Our Telegram Channel - https://t.me/Anviraeducation20