Day 12- Facilitation Guide

Index

For (1.5 hrs) ILT

During the previous session, we explored Database management system,Sql, MySQL and primary key along with Basic Sql queries.

We covered a variety of important concepts:

SQL (Structured Query Language): SQL is a specialized language for working with relational databases, allowing tasks like data querying, insertion, update, and deletion.

MySQL: MySQL is a popular open-source relational database system that uses SQL for database management. It's known for its speed, reliability, and scalability, making it widely used in web applications.

Primary Key: A primary key is a unique identifier for each record in a database table, ensuring uniqueness and enabling efficient data retrieval. It's a crucial part of maintaining data integrity and facilitating queries.

In this session, we will delve into more advanced SQL queries, the concept of foreign keys, table joins, and data filtering techniques.

I. More advanced SQL queries

Let's start by understanding why advanced queries matter in MySQL:

- **Complex Data Retrieval:** In real-world applications, data retrieval needs can be complex. You may need to retrieve data based on multiple conditions, apply sorting, filtering, and join data from multiple tables to get meaningful insights. Advanced queries allow you to express these complex retrieval requirements.

- **Efficiency:** Advanced queries can be optimized for performance. Instead of retrieving a large dataset and then filtering it in your application code, you can use SQL queries to filter and aggregate data directly on the database server. This reduces the amount of data transferred over the network and improves query performance.

- **Data Transformation:** SQL queries enable you to transform data in various ways. You can calculate aggregates (e.g., sums, averages), pivot data, and format results to meet specific reporting or presentation needs.

- **Data Integrity:** Complex queries can enforce data integrity rules at the database level. For example, you can use constraints and triggers to ensure that data is consistent and follows business rules, reducing the chances of data corruption or errors.

- **Reduced Code Complexity:** Using advanced SQL queries can simplify your application code. Instead of writing complex data manipulation logic in your application, you can delegate some of that work to the database, making your application code cleaner and more maintainable.

- **Reporting and Analytics:** In business applications, you often need to generate various reports and perform analytics on data. Advanced queries are essential for aggregating and summarizing data for reporting purposes.

- **Data Integration:** When working with multiple data sources or systems, you may need to join and consolidate data from different databases or data stores. Advanced queries can facilitate this data integration process.

- **Security:** SQL queries allow you to implement security measures, such as row-level security and access control, to restrict who can access and modify specific data within your database.

- **Scalability:** As your application and data grow, the ability to write efficient queries becomes crucial for maintaining good system performance. Advanced query optimization techniques, indexing, and query execution plans are essential for scalability.

- **Data Validation:** Complex queries can be used to validate data integrity and correctness before it is stored in the database. This ensures that only valid data is entered.

Let’s explore more advanced SQL query with examples:

ALTER TABLE

To modify the structure of a MySQL database table, you can use the ALTER TABLE SQL statement. Here are some common uses of the ALTER TABLE statement:
Adding a New Column:

ALTER TABLE table_name
ADD column_name data_type;

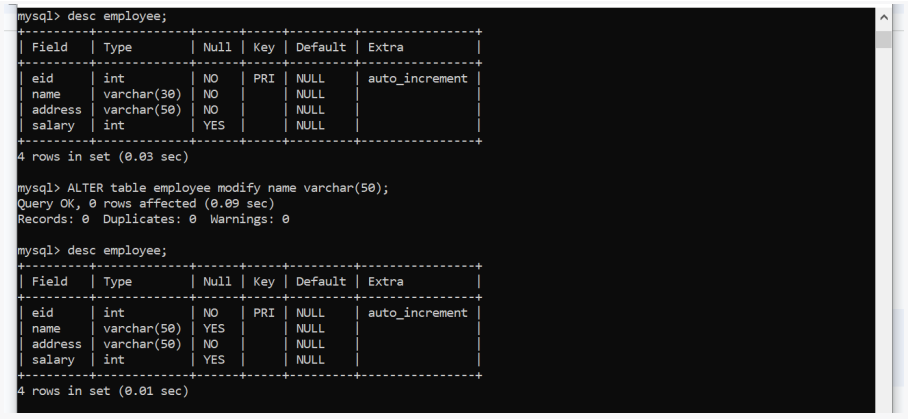
Let's take an example of employee Table:

ALTER table employee add salary int;

Modifying an Existing Column:

ALTER TABLE table_name
MODIFY column_name new_data_type;

ALTER table employee modify name varchar(50);



This query modifies an existing column's data type in the table_name.

<u>Dropping a Column</u>:

ALTER TABLE table_name
DROP column_name;

alter table employee drop column salary;

```
mysql> alter table employee drop column salary;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc employee;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| eid     | int         | NO   | PRI | NULL    | auto_increment |
| name    | varchar(50) | YES  |     | NULL    |                |
| address | varchar(50) | NO   |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql>
```

This query removes an existing column from the table_name.

Clause:
In SQL, a clause is a specific part of a SQL statement that defines various aspects of the query or operation you want to perform on a database. SQL statements are constructed by combining different clauses to form a complete query.

Here are some commonly used SQL clauses:

FROM Clause: Specifies the table(s) from which you want to retrieve data.
Example:

mysql> Select * from student;

```
mysql> select * from student;
+-----------+-----------+----------+---------------------+--------+----------------------+------------+
| StudentID | FirstName | LastName | DateOfBirth         | Gender | Email                | Phone      |
+-----------+-----------+----------+---------------------+--------+----------------------+------------+
| S101      | John      | Doe      | 2000-10-10 00:00:00 | M      | john@example.com     | 9878457945 |
| S102      | Jane      | Smith    | 2013-08-08 00:00:00 | M      | jane@example.com     | 9977457745 |
| S103      | Alice     | Johnson  | 2011-09-08 00:00:00 | F      | alice@example.com    | 9876457845 |
| S104      | Jim       | Doe      | 2011-07-08 00:00:00 | F      | jim.doe@india.com    | 9876457845 |
| S105      | Peter     | Parker   | 2011-06-05 00:00:00 | F      | p_parker@example.com | 9876457845 |
+-----------+-----------+----------+---------------------+--------+----------------------+------------+
5 rows in set (0.00 sec)

mysql>
```

WHERE Clause: Filters the rows returned by a query based on a specified condition.
Example:
mysql> Select * from student where StudentId='S101';

```
ERROR 1054 (42S22): Unknown column 'S101' in 'where clause'
mysql> Select * from student where StudentId='S101';
+-----------+-----------+----------+---------------------+--------+------------------+------------+
| StudentID | FirstName | LastName | DateOfBirth         | Gender | Email            | Phone      |
+-----------+-----------+----------+---------------------+--------+------------------+------------+
| S101      | John      | Doe      | 2000-10-10 00:00:00 | M      | john@example.com | 9878457945 |
+-----------+-----------+----------+---------------------+--------+------------------+------------+
1 row in set (0.01 sec)

mysql>
```

ORDER BY Clause: Specifies the sorting order for the result set.

Example-Sort in descending order:

select StudentID, FirstName,LastName from student order by StudentID desc;;

```
mysql> select StudentID, FirstName,LastName from student order by StudentID desc;
+-----------+-----------+----------+
| StudentID | FirstName | LastName |
+-----------+-----------+----------+
| S105      | Peter     | Parker   |
| S104      | Jim       | Doe      |
| S103      | Alice     | Johnson  |
| S102      | Jane      | Smith    |
| S101      | John      | Doe      |
+-----------+-----------+----------+
5 rows in set (0.00 sec)
```

Example-Sort in ascending order:
select StudentID, FirstName,LastName from student order by StudentID;

```
mysql> select StudentID, FirstName,LastName from student order by StudentID;
+-----------+-----------+----------+
| StudentID | FirstName | LastName |
+-----------+-----------+----------+
| S101      | John      | Doe      |
| S102      | Jane      | Smith    |
| S103      | Alice     | Johnson  |
| S104      | Jim       | Doe      |
| S105      | Peter     | Parker   |
+-----------+-----------+----------+
5 rows in set (0.00 sec)
```

LIMIT (or TOP) Clause: Restricts the number of rows returned by a query.
Example (MySQL):

Select * from student limit 2;

```
mysql> Select * from student limit 2;
+-----------+-----------+----------+---------------------+--------+------------------+------------+
| StudentID | FirstName | LastName | DateOfBirth         | Gender | Email            | Phone      |
+-----------+-----------+----------+---------------------+--------+------------------+------------+
| S101      | John      | Doe      | 2000-10-10 00:00:00 | M      | john@example.com | 9878457945 |
| S102      | Jane      | Smith    | 2013-08-08 00:00:00 | M      | jane@example.com | 9977457745 |
+-----------+-----------+----------+---------------------+--------+------------------+------------+
2 rows in set (0.00 sec)
```

These are some of the fundamental SQL clauses used to construct queries for data retrieval, modification, and analysis in relational databases. The combination and arrangement of these clauses determine the behavior and results of your SQL statements.

<u>UPDATE query</u>:

update records in a MySQL database table, you can use the UPDATE SQL statement. Here's the basic syntax for an UPDATE query in MySQL:

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

table_name: Name of the table where you want to update records.
SET column1 = value1, column2 = value2, ...: Specifies the columns you want to update and their new values.

WHERE condition: An optional condition that specifies which records should be updated. If omitted, all records in the table will be updated.

Here are a few examples:

Update All Records in a Table:

update employee set salary=10000;

This query will update the salary column for all records in the employee table by 10000.
Update Records Based on a Condition:

mysql> select * from employees;

```
mysql> select * from employee;
+-----+------------+-----------+--------+
| eid | name       | address   | salary |
+-----+------------+-----------+--------+
|   2 | riya       | bangalore |  10000 |
|   3 | Ram sharma | tamilnadu |  10000 |
+-----+------------+-----------+--------+
2 rows in set (0.00 sec)

mysql>
```

Update a Specific Record by Matching a Column Value:

update employee set salary=20000 where eid=3;

This query will update the salary column to '20000' for records in the employee table where the eid is=3.

```
mysql> select * from employee;
+-----+------------+-----------+--------+
| eid | name       | address   | salary |
+-----+------------+-----------+--------+
|   2 | riya       | bangalore |  10000 |
|   3 | Ram sharma | tamilnadu |  20000 |
+-----+------------+-----------+--------+
2 rows in set (0.00 sec)

mysql>
```

Always exercise caution when using the UPDATE statement, especially without a WHERE clause, as it can update all records in a table. Make sure to specify the appropriate conditions to update only the records you intend to modify.

II. Aggregate Functions

Aggregate functions perform calculations on sets of values and return a single result.

Before delving into practical examples, it's crucial to grasp the significance of aggregate functions.

Aggregate functions are essential in SQL for several important reasons:

- **Summarizing Data:** Aggregate functions allow you to summarize and condense large sets of data into meaningful insights. They provide a way to answer questions about your data, such as calculating totals, averages, or finding extreme values, which are crucial for decision-making.

- **Data Analysis:** They help in performing data analysis tasks like identifying trends, patterns, and anomalies in your dataset. For instance, you can use aggregate functions to determine the average sales per month, the highest-selling product, or the total revenue for a specific period.

- **Reporting:** In business and reporting applications, aggregate functions are vital for generating reports and dashboards. You can use them to generate summaries and statistics that are easily consumable by stakeholders and management.

- **Performance Optimization:** Aggregate functions can improve query performance. Instead of retrieving and processing a massive amount of data, you can use aggregate functions to extract the specific information you need. This can lead to faster query execution and reduced resource consumption.

- **Data Validation:** Aggregate functions can be used for data validation and quality assurance. For example, you can use COUNT() to check if the number of records in a related table matches your expectations, ensuring data consistency.

- **Grouping and Segmentation:** Aggregate functions work in conjunction with GROUP BY, allowing you to segment data into groups based on specific criteria. This is especially useful for creating summary reports by categories or dimensions, such as sales by region or department.

- **Filtering and Decision Support:** Aggregate functions help in filtering and making data-driven decisions. For instance, you can use HAVING with GROUP BY to filter aggregated results based on conditions, allowing you to focus on specific subsets of data that meet your criteria.

- **Query Efficiency:** By using aggregate functions, you can often achieve the desired results with fewer lines of SQL code, making queries more concise and easier to read and maintain.

In summary, aggregate functions are fundamental tools in SQL that enable you to extract valuable insights, make informed decisions, generate reports, and optimize query performance. They are indispensable for working with relational databases and handling large datasets effectively.

Let's dive into practical demonstrations of aggregate functions to explore their real-world applications:

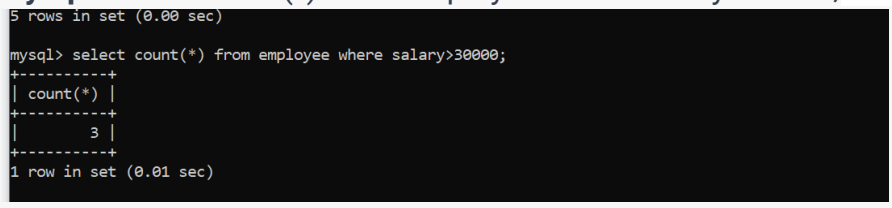
1. **COUNT()**

Counts the number of rows in a specified column or the number of rows that match a condition.

Let's explore with an example: Count all records in a table.
mysql> select count(*) from employee;

Let's explore with another example: Count the number of employees whose salary are greater than 30000;

mysql> select count(*) from employee where salary>30000;

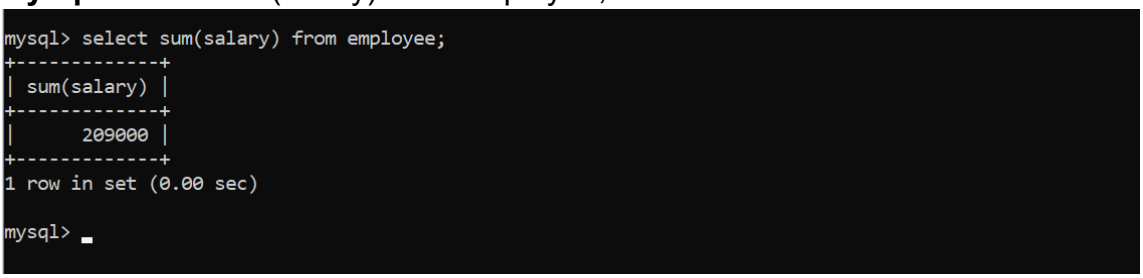


2. **SUM()**

- Calculates the sum of values in a numeric column.

Let's explore with an example: Calculate the total salary of all employees.

mysql> select sum(salary) from employee;



3. **AVG()**

- Computes the average (mean) of values in a numeric column.
  Let's explore with an example: Calculate the average salary of emploee.

mysql> select avg(salary) from employee;

```
mysql> select avg(marks) from student
    -> ;
+------------+
| avg(marks) |
+------------+
|    47.6000 |
+------------+
1 row in set (0.00 sec)
```

4. **MAX()**

- Retrieves the maximum value from a column.
  Let's explore with an example: Find the highest salary among all employees.
  mysql> select max(salary) from employee;

```
mysql> select max(salary) from employee;
+-------------+
| max(salary) |
+-------------+
|       70000 |
+-------------+
1 row in set (0.00 sec)

mysql>
```

5. **MIN()**

- Retrieves the minimum value from a column.
  Let's explore with an example: Find the lowest salary among all employees.
  mysql> select min(salary) from employee;

```
mysql> select min(salary) from employee;
+-------------+
| min(salary) |
+-------------+
|       10000 |
+-------------+
1 row in set (0.00 sec)
```

Knowledge check…

Now that we've covered advanced sql queries and basic understanding of aggregate functions, it's time to test your understanding. The Trainer will conduct a short poll quiz to assess your knowledge on these topics.

1.What SQL statement is used to retrieve data from a database table?

- a) FETCH
- b) SELECT
- c) EXTRACT
- d) SEARCH

2.Which aggregate function calculates the total number of rows in a table?

- a) COUNT
- b) SUM
- c) AVG
- d) MAX

3.What does the SUM() aggregate function do in MySQL?

- a) Calculates the average of a column's values.
- b) Computes the total sum of values in a numeric column.
- c) Retrieves the highest value from a column.
- d) Counts the number of distinct values in a column.

At the end of the quiz,the Trainer will provide feedback and explanations to the participants, enhancing the learning experience and understanding of the content.

Food for thought..

The trainer can ask the students the following question to engage them in a discussion:
Can you establish a foreign key relationship between two tables in different databases or even on different database servers?

III. Foreign Key

In MySQL, a foreign key is a column or a set of columns that establishes a link between two tables in a relational database. The foreign key in one table references the primary key of another table, creating a relationship between them. The purpose of a foreign key is to enforce referential integrity, ensuring that data remains consistent between related tables.

Here are some key points about foreign keys in MySQL:

- **Data Integrity:** Foreign keys enforce referential integrity, ensuring that relationships between tables are maintained correctly. They prevent actions that would leave orphaned or inconsistent data in the database, thus preserving data integrity.

- **Data Consistency:** By establishing relationships between tables, foreign keys help maintain data consistency by enforcing rules that ensure only valid and related data can be inserted or updated. This prevents data anomalies and errors.

- **Data Validation:** Foreign keys act as a form of data validation. They ensure that values in a column correspond to valid entries in another table, preventing the insertion of incorrect or irrelevant data.

- **Relationships:** Foreign keys define relationships between tables, which is essential for modeling complex data structures and capturing the real-world associations between entities. This enables the database to represent more accurate and meaningful information.

- **Data Retrieval:** Foreign keys facilitate efficient data retrieval and querying by providing a means to join related tables. This simplifies the process of retrieving data from multiple tables and allows for more complex and insightful queries.

- **Data Integrity Checks:** Foreign keys can be used to perform integrity checks and identify inconsistencies or missing data in related tables. This is especially useful during data quality assurance and validation processes.

- **Data Navigation:** They aid in data navigation and exploration, making it easier to traverse through related data across different tables in a database, thus improving the usability and accessibility of data.

- **Maintainability:** Foreign keys enhance database maintainability by reducing the likelihood of errors and inconsistencies. They make it easier to update, modify, or extend the database schema without compromising data integrity.

- **Enforced Constraints:** Foreign keys enable the enforcement of specific constraints and rules on how data can be manipulated, ensuring that the database adheres to predefined business logic and standards.

- **Security:** Foreign keys can contribute to data security by preventing unauthorized access or modification of critical data through relationships and constraints.

In summary, foreign keys are vital components of a relational database system because they ensure data accuracy, maintain relationships, enhance data quality, and facilitate efficient data retrieval and manipulation. They play a central role in preserving data integrity and ensuring that the database accurately reflects the real-world relationships between entities.

Question: What is a foreign key in a relational database, and how does it differ from a primary key?

Syntax for Defining a Foreign Key: To create a foreign key constraint when defining a table, you can use the following syntax:

```
CREATE TABLE child_table (
    ...
    foreign_key_column data_type,
    ...
    FOREIGN KEY (foreign_key_column) REFERENCES
parent_table(primary_key_column)
    ...
);
```

- child_table: The table that contains the foreign key.
- foreign_key_column: The column in child_table that references the primary key of the parent table.
- parent_table: The table being referenced (the parent table).
- primary_key_column: The primary key column in the parent table that the foreign key references.

Note: We will observe a hands-on demonstration illustrating the concept of table joins.

IV. Joining Table

A "joining table" in a database is often referred to as a "junction table" or an "association table." It's a table used in a relational database to manage many-to-many relationships between two other tables. The joining table contains foreign keys from both of the related tables, allowing you to connect records from one table to records in another. This is a common technique in database design to represent complex relationships between entities.

For example, if you have a database for a student management system, you might have a "Course" table and a "Instructor" table. Since one Instructor can teach many courses, and many courses are taught by one Instructor, you would use a joining table to link them together, indicating which Instructor is associated with which Courses.

Here's a simple representation of such tables:

Student

- **Attributes:**
  StudentID (Primary Key)
  FirstName
  LastName

DateOfBirth
Gender
Email
Phone

- **Relationships:**
  One **Student** can enroll in more than one **Course** (One-to-Many)

Course

- **Attributes:**
  CourseID (Primary Key)
  CourseTitle
  Credits
- **Relationships:**
  Many **Course** is taught by one **Instructor** (**Many-to-One**)

Instructor

- **Attributes:**
  InstructorID (Primary Key)
  FirstName
  LastName
  Email
- **Relationships:**
  One **Instructor** teaches many **Courses** (**One-to-Many**)
  One **Instructor** has many **Students** (**One-to-Many**)

Enrollment

- **Attributes:**
  EnrollmentID (Primary Key)
  EnrollmentDate
  StudentID(Foreign key)
  CourseID(Foreign Key)
  InstructorID(Foreign key)
- **Relationships:**
  One **Student** maps to Many **Enrolment ids** (**One-to-Many**)
  Many **Enrolment ids** map to one **Course** (**Many-to-One**)

Score

- **Attributes**:
  ScoreID (Primary Key)
  CourseID (Foreign key)
  StudentID (Foreign Key)
  DateOfExam
  CreditObtained

- **Relationships:**
  Many **ScoreIDs** will map to one **Student (Many-to-one)**
  Many **ScoresIDs** will map to one **Course (Many-to-one)**

Feedback

- **Attributes**:
  FeedbackID (Primary Key)
  StudentID (Foreign key)
  Date
  InstructorName
  Feedback

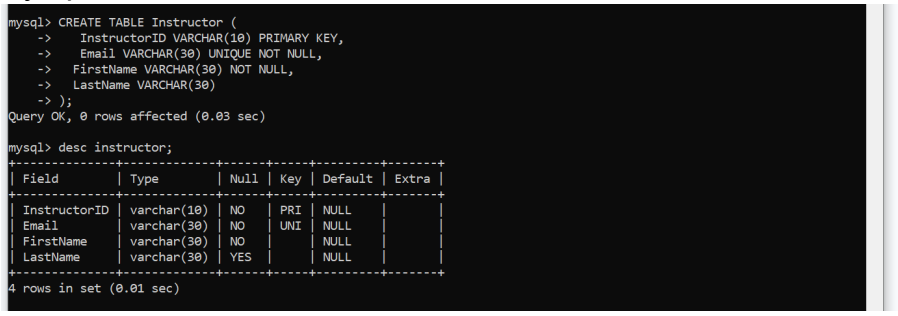
- **Relationships**:
  One **Student** will maps to Many **Feedbacks** (**One-to-Many**)

Joining tables play a crucial role in relational database design and are used to normalize data and maintain referential integrity in complex relationships.

Instructor Table:

```
CREATE TABLE Instructor (
   InstructorID VARCHAR(10) PRIMARY KEY,
   Email VARCHAR(30) UNIQUE NOT NULL,
  FirstName VARCHAR(30) NOT NULL,
  LastName VARCHAR(30)
);
```

mysql> desc Instructor;



```
mysql> CREATE TABLE Instructor (
    ->    InstructorID VARCHAR(10) PRIMARY KEY,
    ->    Email VARCHAR(30) UNIQUE NOT NULL,
    ->   FirstName VARCHAR(30) NOT NULL,
    ->   LastName VARCHAR(30)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> desc instructor;
+--------------+------------+------+-----+---------+-------+
| Field        | Type       | Null | Key | Default | Extra |
+--------------+------------+------+-----+---------+-------+
| InstructorID | varchar(10) | NO   | PRI | NULL    |       |
| Email        | varchar(30) | NO   | UNI | NULL    |       |
| FirstName    | varchar(30) | NO   |     | NULL    |       |
| LastName     | varchar(30) | YES  |     | NULL    |       |
+--------------+------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Add Some Sample Data with nested Insert query:

```
INSERT INTO Instructor (InstructorID ,Email,FirstName,LastName) VALUES
   ('I101','sunil@example.com','Sunil','Rawat'),
   ('I102','nida@example.com','Nida','Fatima'),
   ('I103','shiv@example.com','Shiv','Kumar');
```

mysql> select * from Instructor;

```
mysql> select * from instructor;
+--------------+-------------------+-----------+----------+
| InstructorID | Email             | FirstName | LastName |
+--------------+-------------------+-----------+----------+
| I101         | sunil@example.com | Sunil     | Rawat    |
| I102         | nida@example.com  | Nida      | Fatima   |
| I103         | shiv@example.com  | Shiv      | Kumar    |
+--------------+-------------------+-----------+----------+
3 rows in set (0.00 sec)
```

Course Table

```
CREATE TABLE Course (
    CourseID VARCHAR(10) PRIMARY KEY,
    CourseTitle VARCHAR(30) NOT NULL,
   Credits INT NOT NULL
);
```

mysql> desc Course;

```
mysql> desc course;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| CourseID    | varchar(10) | NO   | PRI | NULL    |       |
| CourseTitle | varchar(30) | NO   |     | NULL    |       |
| Credits     | int         | NO   |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Add Some Sample Data with nested Insert query:

```
INSERT INTO Course (CourseID,CourseTitle,Credits) VALUES
   ('C101','Math101',12),
   ('C102','History101',13),
   ('C103','Computer Science101',11);
```

mysql> select * from Course;

```
mysql> select * from course;
+----------+--------------------+---------+
| CourseID | CourseTitle        | Credits |
+----------+--------------------+---------+
| C101     | Math101            |      12 |
| C102     | History101         |      13 |
| C103     | Computer Science101 |      11 |
+----------+--------------------+---------+
3 rows in set (0.00 sec)

mysql>
```

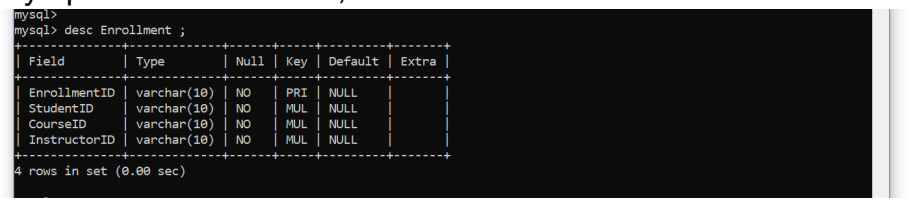
Enrollment Table: (Junction table)

We are creating a third table as a junction table to represent the relationship between the two related tables.This table typically contains three columns that serve as foreign keys, each referencing a primary key from the related tables.

For our example, here we have created a "Enrollment" junction table with columns "StudentID" and "CourseID" and "InstructorID" along with EnrollmentId and EnrollmentDate.

```
CREATE TABLE Enrollment (
    EnrollmentID VARCHAR(10) PRIMARY KEY,
    StudentID VARCHAR(10) NOT NULL,
    CourseID VARCHAR(10) NOT NULL,
    InstructorID VARCHAR(10) NOT NULL,
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID),
    FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)
);
```

mysql> desc Enrollment;

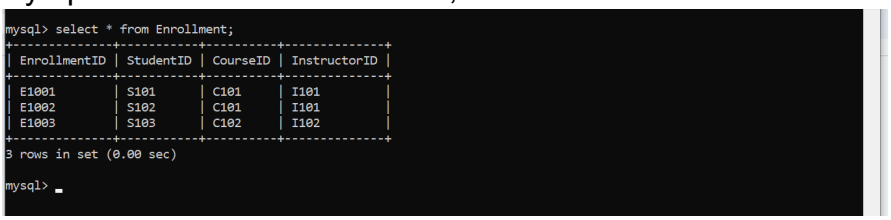


```
mysql>
mysql> desc Enrollment ;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| EnrollmentID | varchar(10) | NO   | PRI | NULL    |       |
| StudentID    | varchar(10) | NO   | MUL | NULL    |       |
| CourseID     | varchar(10) | NO   | MUL | NULL    |       |
| InstructorID | varchar(10) | NO   | MUL | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Add Some Sample Data with nested Insert query:

```
INSERT INTO Enrollment (EnrollmentID,StudentID, CourseID,InstructorID) VALUES
    ('E1001','S101','C101','I101'),
    ('E1002','S102','C101', 'I101'),
    ('E1003','S103','C102','I102');
```

mysql> select * from Enrollment;



```
mysql> select * from Enrollment;
+--------------+-----------+----------+--------------+
| EnrollmentID | StudentID | CourseID | InstructorID |
+--------------+-----------+----------+--------------+
| E1001        | S101      | C101     | I101         |
| E1002        | S102      | C101     | I101         |
| E1003        | S103      | C102     | I102         |
+--------------+-----------+----------+--------------+
3 rows in set (0.00 sec)

mysql>
```

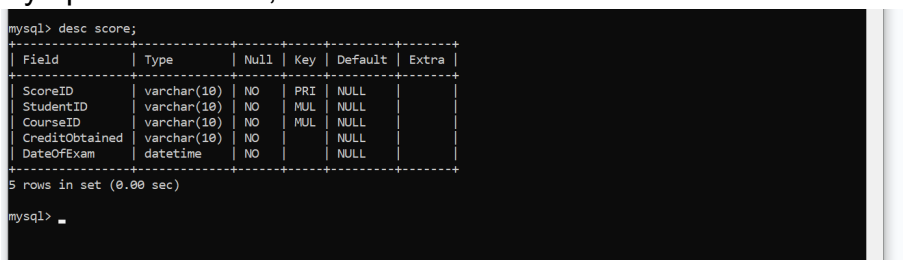
V. Joins

SQL JOIN is a SQL operation used to combine rows from two or more tables based on a related column between them. It allows you to retrieve data from multiple tables in a single result set, making it a powerful tool for querying and analyzing data stored across different tables within a relational database. SQL JOINs are crucial for working with normalized databases and handling relationships between tables.

We already have the Student and Course tables established. Now, let's create an additional table called "score" where we will include StudentId and CourseId as foreign keys.

```
CREATE TABLE Score(
        ScoreID VARCHAR(10) PRIMARY KEY,
        StudentID VARCHAR(10) NOT NULL,
        CourseID VARCHAR(10) NOT NULL,
        FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
        FOREIGN KEY (CourseID) REFERENCES Course(CourseID),
        CreditObtained VARCHAR(10) NOT NULL,
        DateOfExam DateTime NOT NULL
);
```

mysql> desc Score;

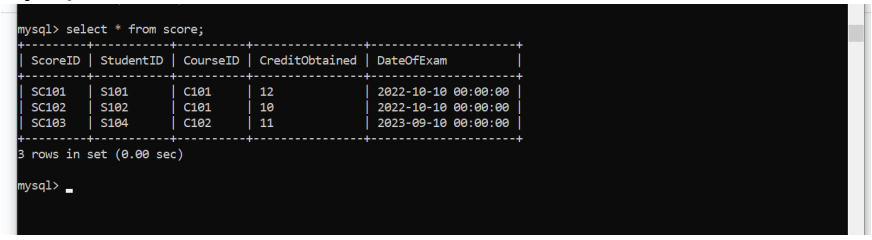


```
mysql> desc score;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| ScoreID        | varchar(10) | NO   | PRI | NULL    |       |
| StudentID      | varchar(10) | NO   | MUL | NULL    |       |
| CourseID       | varchar(10) | NO   | MUL | NULL    |       |
| CreditObtained | varchar(10) | NO   |     | NULL    |       |
| DateOfExam     | datetime    | NO   |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql>
```

Add Some Sample Data with nested Insert query:

```
INSERT INTO Score
(ScoreID,StudentID,CourseID,CreditObtained,DateOfExam)VALUES
('SC101','S101','C101','12','2022-10-10'),
('SC102','S102','C101','10','2022-10-10'),
('SC103','S104','C102',11,'2023-09-10');
```

mysql> select * from Score;



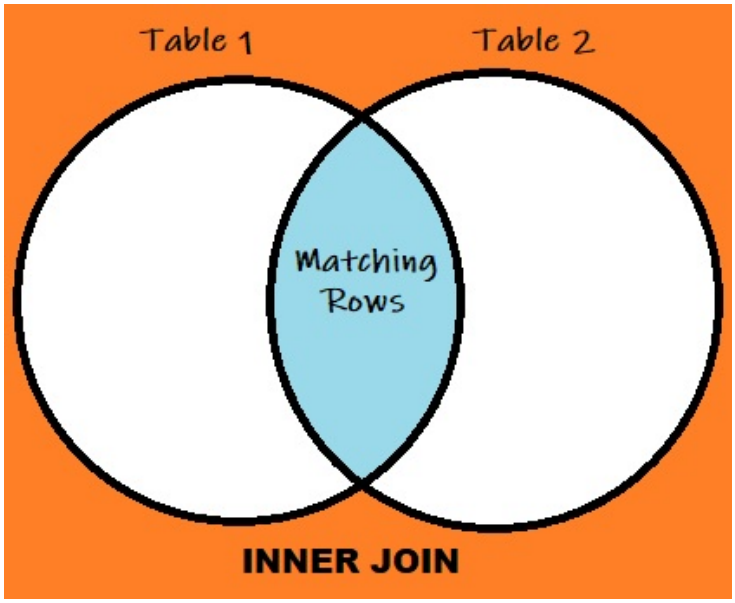
```
mysql> select * from score;
+---------+-----------+----------+----------------+---------------------+
| ScoreID | StudentID | CourseID | CreditObtained | DateOfExam          |
+---------+-----------+----------+----------------+---------------------+
| SC101   | S101      | C101     | 12             | 2022-10-10 00:00:00 |
| SC102   | S102      | C101     | 10             | 2022-10-10 00:00:00 |
| SC103   | S104      | C102     | 11             | 2023-09-10 00:00:00 |
+---------+-----------+----------+----------------+---------------------+
3 rows in set (0.00 sec)

mysql>
```

Types of Joins

In MySQL and other relational database management systems, there are several types of joins that you can use to combine data from two or more tables based on a related column between them.

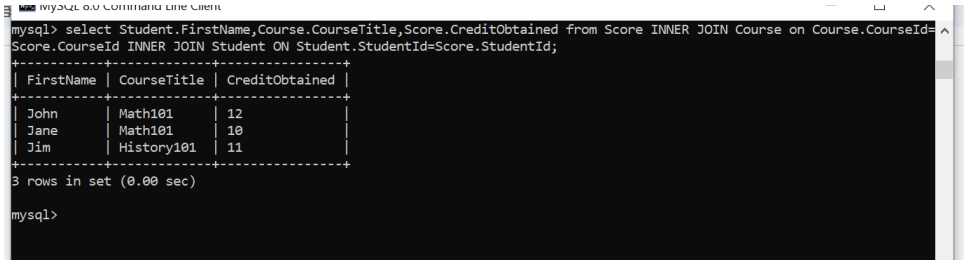
The most commonly used types of joins in MySQL are:

1. **INNER JOIN:** This type of join returns only the rows that have matching values in both tables. Rows from the tables that don't have a match in the other table are excluded from the result set.



SELECT * FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;

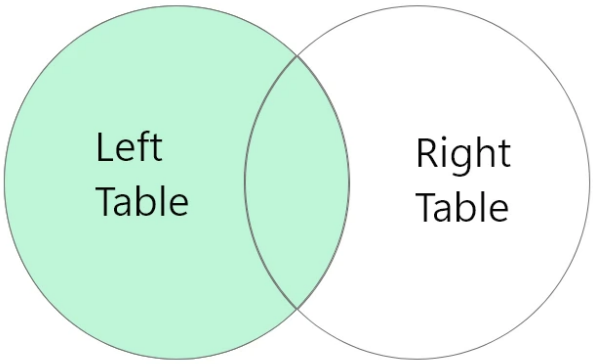
select Student.FirstName,Course.CourseTitle,Score.CreditObtained from Score INNER JOIN Course on Course.CourseId=Score.CourseId INNER JOIN Student ON Student.StudentId=Score.StudentId;



```
mysql> select Student.FirstName,Course.CourseTitle,Score.CreditObtained from Score INNER JOIN Course on Course.CourseId=
Score.CourseId INNER JOIN Student ON Student.StudentId=Score.StudentId;
+-----------+-------------+----------------+
| FirstName | CourseTitle | CreditObtained |
+-----------+-------------+----------------+
| John      | Math101     | 12             |
| Jane      | Math101     | 10             |
| Jim       | History101  | 11             |
+-----------+-------------+----------------+
3 rows in set (0.00 sec)

mysql>
```

2. **LEFT JOIN (or LEFT OUTER JOIN):** This type of join returns all the rows from the left table (table1), and the matched rows from the right table (table2). If there's no match in the right table, NULL values are returned for the columns from the right table.

LEFT JOIN IN DBMS

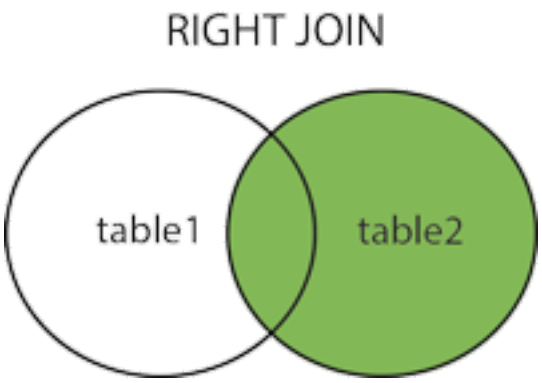


SELECT * FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;

select * from Course LEFT JOIN Score on Score.CourseId=Course.CourseId;

```
mysql> select * from Course LEFT JOIN Score on Score.CourseId=Course.CourseId;
+----------+--------------------------------+---------+---------+-----------+----------+----------------+---------------------+
| CourseID | CourseTitle                    | Credits | ScoreID | StudentID | CourseID | CreditObtained | DateOfExam          |
+----------+--------------------------------+---------+---------+-----------+----------+----------------+---------------------+
| C101     | Math101                        |      12 | SC101   | S101      | C101     | 12             | 2022-10-10 00:00:00 |
| C101     | Math101                        |      12 | SC102   | S102      | C101     | 10             | 2022-10-10 00:00:00 |
| C102     | History101                     |      13 | SC103   | S104      | C102     | 11             | 2023-09-10 00:00:00 |
| C103     | Computer Science101            |      11 | NULL    | NULL      | NULL     | NULL           | NULL                |
| C104     | Data Science101                |      10 | NULL    | NULL      | NULL     | NULL           | NULL                |
| C105     | Geography101                   |      13 | NULL    | NULL      | NULL     | NULL           | NULL                |
| C106     | Java Fullstack With Angular101 |      13 | NULL    | NULL      | NULL     | NULL           | NULL                |
+----------+--------------------------------+---------+---------+-----------+----------+----------------+---------------------+
7 rows in set (0.00 sec)
```

3. **RIGHT JOIN (or RIGHT OUTER JOIN):** This is the opposite of a LEFT JOIN. It returns all the rows from the right table and the matched rows from the left table. If there's no match in the left table, NULL values are returned for the columns from the left table.

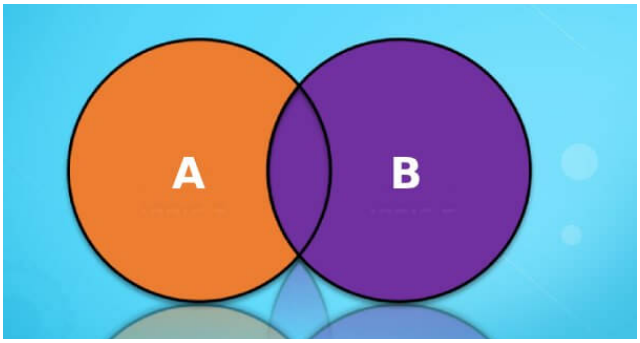


SELECT * FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;

select * from Score RIGHT JOIN Course on Score.CourseId=Course.CourseId;

```
mysql> select * from Score RIGHT JOIN Course on Score.CourseId=Course.CourseId;
+---------+-----------+----------+----------------+---------------------+----------+-------------------------------+---------+
| ScoreID | StudentID | CourseID | CreditObtained | DateOfExam          | CourseID | CourseTitle                   | Credits |
+---------+-----------+----------+----------------+---------------------+----------+-------------------------------+---------+
| SC101   | S101      | C101     | 12             | 2022-10-10 00:00:00 | C101     | Math101                       |      12 |
| SC102   | S102      | C101     | 10             | 2022-10-10 00:00:00 | C101     | Math101                       |      12 |
| SC103   | S104      | C102     | 11             | 2023-09-10 00:00:00 | C102     | History101                    |      13 |
| NULL    | NULL      | NULL     | NULL           | NULL                | C103     | Computer Science101           |      11 |
| NULL    | NULL      | NULL     | NULL           | NULL                | C104     | Data Science101               |      10 |
| NULL    | NULL      | NULL     | NULL           | NULL                | C105     | Geography101                  |      13 |
| NULL    | NULL      | NULL     | NULL           | NULL                | C106     | Java Fullstack With Angular101 |      13 |
+---------+-----------+----------+----------------+---------------------+----------+-------------------------------+---------+
7 rows in set (0.00 sec)
```

4. **SELF JOIN:** This is used to join a table with itself. It can be useful when you have hierarchical or recursive data structures within a single table.



select Student.FirstName, Course.CourseTitle,Score.CreditObtained from Course INNER JOIN Score on score.CourseId=course.CourseId INNER JOIN Student ON Student.StudentID=Score.StudentID;

```
mysql> select Student.FirstName, Course.CourseTitle,Score.CreditObtained from Course INNER JOIN Score on score.CourseId=course.CourseId INNER JOIN Student ON Student.Studen
tID=Score.StudentID;
+-----------+-------------+----------------+
| FirstName | CourseTitle | CreditObtained |
+-----------+-------------+----------------+
| John      | Math101     | 12             |
| Jane      | Math101     | 10             |
| Jim       | History101  | 11             |
+-----------+-------------+----------------+
3 rows in set (0.00 sec)

mysql>
```

These are the main types of joins in MySQL. You can use them to combine data from multiple tables based on the relationships defined by their columns. The choice of which join to use depends on the specific requirements of your query and the desired result set.

Note:I've employed the "AS" keyword in the query, which is referred to as an alias.In SQL, an alias is a temporary name or label assigned to a table or column for the duration of a database query. Aliases are used to make the SQL query more readable or to provide a shorter, more convenient name for a table or column.

Note: Full Outer join is not supported by MySQL.

Question: What are some best practices for optimizing JOIN queries to improve query performance, especially when dealing with large datasets?

VI. Filtering Data

Filtering data in MySQL involves using the WHERE clause in your SQL queries to specify conditions that must be met for a row to be included in the result set. This allows you to retrieve only the data that meets certain criteria.

Here's how you can filter data in MySQL:

Comparison Operators:
You can use various comparison operators to filter data, such as =, !=, >, <, >=, and <=.

select * from Instructor where InstructorID>'I101';

```
mysql> select * from Instructor where InstructorID>'I101';
+--------------+-----------------+-----------+----------+
| InstructorID | Email           | FirstName | LastName |
+--------------+-----------------+-----------+----------+
| I102         | nida@example.com | Nida      | Fatima   |
| I103         | shiv@example.com | Shiv      | Kumar    |
+--------------+-----------------+-----------+----------+
2 rows in set (0.00 sec)

mysql>
```

Logical Operators:
You can use logical operators like AND, OR, and NOT to create complex conditions.

select * from student where FirstName='Peter' and StudentID='S105';

```
mysql>  select * from student where FirstName='Peter' and StudentID='S105';
+-----------+-----------+----------+---------------------+--------+---------------------+------------+-------+
| StudentID | FirstName | LastName | DateOfBirth         | Gender | Email               | Phone      | marks |
+-----------+-----------+----------+---------------------+--------+---------------------+------------+-------+
| S105      | Peter     | Parker   | 2011-06-05 00:00:00 | F      | p_parker@example.com | 9876457845 |    40 |
+-----------+-----------+----------+---------------------+--------+---------------------+------------+-------+
1 row in set (0.00 sec)

mysql>
```

Pattern Matching:
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

Select all Instructor that starts with the letter "S"
mysql> select * from Instructor where FirstName like 's%';

```
mysql> select * from Instructor where FirstName like 's%';
+--------------+-------------------+-----------+----------+
| InstructorID | Email             | FirstName | LastName |
+--------------+-------------------+-----------+----------+
| I101         | sunil@example.com | Sunil     | Rawat    |
| I103         | shiv@example.com  | Shiv      | Kumar    |
+--------------+-------------------+-----------+----------+
2 rows in set (0.01 sec)

mysql>
```

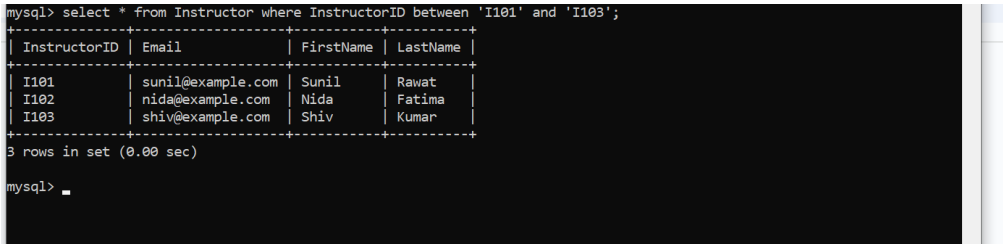
IN Operator:
The IN operator allows you to filter data where a column matches any value in a list.

Select All Instructor details where id is I101 and I103;
mysql> select * from Instructor where InstructorID in('I101','I103');

```
mysql> select * from Instructor where FirstName like 's%';
+--------------+-------------------+-----------+----------+
| InstructorID | Email             | FirstName | LastName |
+--------------+-------------------+-----------+----------+
| I101         | sunil@example.com | Sunil     | Rawat    |
| I103         | shiv@example.com  | Shiv      | Kumar    |
+--------------+-------------------+-----------+----------+
2 rows in set (0.00 sec)

mysql>
```

BETWEEN Operator:
The BETWEEN operator filters data within a specified range.

Select All Instructor details where id is in between I101 and I103;
mysql> select * from Instructor where InstructorID between 'I101' and 'I103';



```
mysql> select * from Instructor where InstructorID between 'I101' and 'I103';
+--------------+------------------+-----------+----------+
| InstructorID | Email            | FirstName | LastName |
+--------------+------------------+-----------+----------+
| I101         | sunil@example.com | Sunil     | Rawat    |
| I102         | nida@example.com  | Nida      | Fatima   |
| I103         | shiv@example.com  | Shiv      | Kumar    |
+--------------+------------------+-----------+----------+
3 rows in set (0.00 sec)

mysql>
```

Remember that you can combine multiple conditions to create more complex filters in your WHERE clause. Properly filtering data is essential for obtaining the specific information you need from your database tables.

Exercise:

Use ChatGPT to explore NoSQL database:

Put the below problem statement in the message box and see what ChatGPT says.
I was learning Database Management Systems and I was wondering what is NoSQL, and how it differs from traditional SQL Databases?