Nitte Meenakshi Institute of Technology

Yelahanka, Bangalore- 560064, Karnataka- India.

An Autonomous Institution with A+ Grade by NAAC UGC, Approved by VTU, UGC, AICTE, Govt. of India.

Department of Electronics and Communication Engineering

AY: 2025-2026

A Report on the Experiments of the Ability Enhancement Course

Database Management Systems Lab (22ECA563)

Submitted By

Arya Anand Pathak 1NT23EC022

Under the Guidance of

Dr Ramachandra AC

Professor

Department of ECE, NMIT

Contents

Experiment 1: Introduction to DBMS

- What is a Database?

  A database is an organized collection of data that is structured to facilitate efficient storage, retrieval, and management by a computer system. It allows users to store large amounts of information in a systematic manner, enabling quick access and manipulation of that data.

- What is a Database Management System?

  A Database Management System (DBMS) is a software platform designed to facilitate the creation, management, and manipulation of databases. It serves as an intermediary between users and the database, allowing for efficient data storage, retrieval, and management. A DBMS makes it easier to organize and control the data stored in a database, with examples including MySQL, PostgreSQL, and MongoDB.

- Relational DBMS (RDBMS):

  An RDBMS is a type of database management system that organizes data into tables (relations) consisting of rows and columns. It uses Structured Query Language (SQL) to manage and manipulate data. Relationships between tables are established using keys.

We will be looking into two types of models used for Relational DBMS: Relational Model and Entity-Relationship Model.

1. Entity-Relationship Model (ER Model):

   The ER model is a design framework used to visualize and structure the data before implementing it in a database. It represents data as entities, attributes, and relationships. ER diagrams are used to depict this model graphically, aiding in database design.

   Symbols used in ER model:

- Rectangles: Rectangles represent Entities in the ER Model.
- Ellipses: Ellipses represent Attributes in the ER Model.
- Diamond: Diamonds represent Relationships among Entities.

- What is an Entity?

  An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

- What is an Attribute?

  Attributes are the properties that define the entity type. For example, USN, Name, Age, and Address are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.

- What is a Relationship?

  A Relationship represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course.

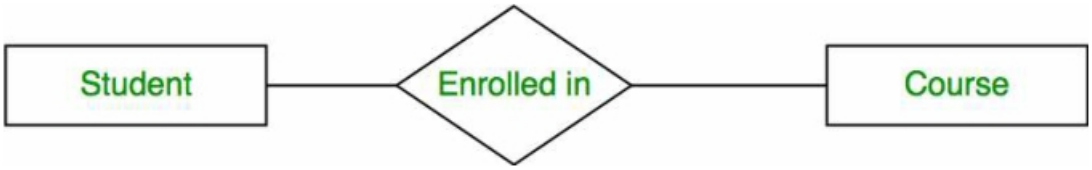

Diagram representing a simple Entity-Relationship model.

2. Relational Model

The relational model is a framework for organizing data in an RDBMS. Data is represented as tables (relations) with rows and columns. Each table has a unique key to identify records, known as primary key and relationships between tables are based on shared attributes.

Terminologies:

- Attribute: Attributes are the properties that define an entity. e.g.; Name, USN, Address.
- Tuple: Each row in the relation is known as a tuple.
- Relation Schema: A relation schema defines the structure of the relation and represents the name of the relation with its attributes.
- Relation Instance: The set of tuples of a relation at a particular instance of time is called a relation instance.
- Degree: The number of attributes in the relation is known as the degree of the relation.
- Cardinality: The number of tuples in a relation is known as cardinality.
- Column: The column represents the set of values for a particular attribute.
- NULL Values: The value which is not known or unavailable is called a NULL value. It is represented by blank space.

About MySQL

MySQL server is an open-source relational database management system. Databases and related tables are the main component of many websites as the data is stored and exchanged over the web. Almost all social networking websites like Facebook, Twitter and Google depend on MySQL data which are designed and optimized for such purpose. For all these reasons, MySQL server becomes the default choice for web applications.

Advantages of MySQL:

- Fast and high-performance database.
- Easy to use, maintain and administer.
- Easily available and maintain integrity of database.
- Provides scalability, usability and reliability.
- Low-cost hardware.
- MySQL can read simple and complex queries and write operations.
- Provides strong indexing support.
- Provides support for secured connections.
- Provides powerful data encryption and accuracy.
- Provides cross-platform compatibility.
- Provides minimized code repetition.

Queries can be understood as the commands which interacts with database tables to work around with data. We will be using MySQL for the experiments that follow.

Experiment 1: Creating a Database with Tables

Design a database and create required tables in it.

Let us create a database named ‘College’, with two tables – Student details and Course details.

See Existing Databases:

Syntax: SHOW DATABASES;

This displays information of all the existing databases in the server.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| nitte_university   |
| nmit               |
| nmit1              |
| performance_schema |
| referential_demo   |
| sakila             |
| student_db         |
| studentdb          |
| sys                |
| university         |
| user_mgmt          |
+--------------------+
13 rows in set (3.288 sec)
```

Screenshot of output to the ‘show databases’ query, displaying list of existing databases.

Creating a Database:

Syntax: CREATE database database_name;

This creates a new database with the name ‘database_name’ in the MySQL server.

```
mysql> create database college;
Query OK, 1 row affected (0.01 sec)
```

Screenshot of output to the ‘create database’ query, creating the new database.

Using the Database:

Syntax: USE database_name;

This sets the database as the current database in the MySQL server.

```
mysql> use college;
Database changed
```

Screenshot of output to the ‘use database’ query, setting the current database.

See Existing Tables in the Database:

Syntax: SHOW TABLES;

This shows all the tables in the selected database as information.

```
mysql> show tables;
Empty set (0.01 sec)
```

Screenshot of output to the ‘show tables’ query, displaying if there are any pre existing tables in the database.

An empty set means, there are no tables existing in our database.

Creating Tables:

Syntax:

CREATE TABLE table_name(column1, column2, column3..);

This statement creates a new table with the given columns.

```
mysql> create table student(id int, name varchar(15), branch varchar(4), age int);
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> create table course(course_id varchar(3), branch_name varchar(4), strength int);
Query OK, 0 rows affected (0.02 sec)
```

Screenshot of outputs to the ‘create table’ query, creating the new table with given columns.

Describing the Table:

Syntax:

DESCRIBE table_name;

This describes the columns of the table_name with respect to Field, Type, Null, Key, Default, Extra.

```
mysql> describe student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | YES  |     | NULL    |       |
| name   | varchar(15) | YES  |     | NULL    |       |
| branch | varchar(4)  | YES  |     | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

```
mysql> describe course;
+-------------+------------+------+-----+---------+-------+
| Field       | Type       | Null | Key | Default | Extra |
+-------------+------------+------+-----+---------+-------+
| course_id   | varchar(3) | YES  |     | NULL    |       |
| branch_name | varchar(4) | YES  |     | NULL    |       |
| strength    | int        | YES  |     | NULL    |       |
+-------------+------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Outputs to the ‘describe table’ query, giving information about the table, its columns, datatypes and constraints.

Inserting Values:

Syntax:

INSERT INTO table_name (column1, column2, column3 . .) VALUES(value1, value2, . .);

This statement inserts a new record into a table with specified values.

```
mysql> insert into student(id, name, branch, age) values (101, 'A', 'ECE', 19), (102, 'B', 'EEE', 21), (103, 'C', 'ECE', 19), (104, 'D
', 'CSE', 20);
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

```
mysql> insert into course(course_id, branch_name) values('E01', 'ECE'), ('E02', 'EEE'),('E03', 'CSE');
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> insert into course(course_id, branch_name, scope) values('E04', 'ISE', 'high');
Query OK, 1 row affected (0.01 sec)
```

Screenshots of outputs to the 'insert into table' query, that inserts new records into the table.

Display All Records of the Table:

Syntax:

SELECT * FROM table_name ;

This query retrieves all records from the table.

```
mysql> select * from student;
+------+------+--------+------+
| id   | name | branch | age  |
+------+------+--------+------+
|  101 | A    | ECE    |   19 |
|  102 | B    | EEE    |   21 |
|  103 | C    | ECE    |   19 |
|  104 | D    | CSE    |   20 |
+------+------+--------+------+
4 rows in set (0.00 sec)
```

```
mysql> select * from course;
+-----------+-------------+----------+-------+
| course_id | branch_name | strength | scope |
+-----------+-------------+----------+-------+
| E03       | CSE         |     NULL | high  |
| E01       | ECE         |     NULL | mid   |
| E02       | EEE         |     NULL | low   |
| E04       | ISE         |     NULL | high  |
+-----------+-------------+----------+-------+
4 rows in set (0.00 sec)
```

Screenshot of outputs to the 'select * from…' query, retrieves all records from the table.

Experiment 2: Primary key and Foreign Key Integrity

Primary Key

A MySQL Primary Key is a unique column/field in a table that should not contain duplicate or NULL values and is used to identify each record in the table uniquely. The role of the primary key constraint is to maintain the integrity of the database by preventing duplicate and null values in the key column. A table can only have one primary key.

Primary key constraint can be set either while creating a table or by using the ALTER command.

1. Using CREATE:

Syntax:

CREATE TABLE table_name (column1 datatype primary key, column2 datatype);

```
mysql> create table students(id int primary key not null, name varchar(10), branch varchar(4));
Query OK, 0 rows affected (0.02 sec)

mysql> describe students;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | NO   | PRI | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
| branch | varchar(4)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Screenshot of output when constraint primary key is given using CREATE command.

2. Using ALTER:

Syntax:

ALTER TABLE table_name ADD PRIMARY KEY (column_name);

```
mysql> alter table course add primary key(branch_name);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe course;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| course_id   | varchar(3)  | NO   |     | NULL    |       |
| branch_name | varchar(4)  | NO   | PRI | NULL    |       |
| strength    | int         | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Screenshot of output when constraint primary key is given using ALTER command.

Foreign Key

A Foreign Key is a field in one table, that refers to the Primary Key in another table. The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

```
mysql> ALTER TABLE student ADD FOREIGN KEY (branch) REFERENCES course(branch_name);
Query OK, 4 rows affected (0.09 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> describe course;
+-------------+------------+------+-----+---------+-------+
| Field       | Type       | Null | Key | Default | Extra |
+-------------+------------+------+-----+---------+-------+
| course_id   | varchar(3) | NO   |     | NULL    |       |
| branch_name | varchar(4) | NO   | PRI | NULL    |       |
| strength    | int        | YES  |     | NULL    |       |
+-------------+------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Screenshot of output when constraint foreign key is given to a field.

NOT NULL

By default, a column can hold NULL values. The NOT NULL constraint enforces a column to NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

```
mysql> create table students(id int primary key not null, name varchar(10), branch varchar(4));
Query OK, 0 rows affected (0.02 sec)

mysql> describe students;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | NO   | PRI | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
| branch | varchar(4)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Screenshot of output when constraint NOT NULL is given to a field.

Now, the field ‘id’ can never be a null value, that means, it must always hold a value.

Experiment 3: ALTER, UPDATE, DELETE

ALTER

This statement can be used to alter the table, that is, adding or deleting columns of the table.

Syntax:
ALTER TABLE table_name ADD(column1, column2, column3..);
This statement adds columns to the existing table.

```
mysql> alter table student add column(city varchar(14));
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | NO   | PRI | NULL    |       |
| name   | varchar(15) | YES  |     | NULL    |       |
| branch | varchar(4)  | YES  | MUL | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
| city   | varchar(14) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> alter table student add column(cgpa float);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Screenshot of output when ALTER TABLE command is used to add columns.

Syntax:
ALTER TABLE table_name DROP(column1);
This statement deletes specified columns from the existing table.

```
mysql> alter table student drop column city;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | NO   | PRI | NULL    |       |
| name   | varchar(15) | YES  |     | NULL    |       |
| branch | varchar(4)  | YES  | MUL | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Screenshot of output when ALTER TABLE command is used to drop columns.

UPDATE

The UPDATE statement is used to modify the existing records in a table.

Syntax:
UPDATE table_name SET column1 = value1, column2 = value2,.. WHERE condition;
This statement updates records in the table with the new given values for the columns.

```
mysql> update course set scope='high' where branch_name='CSE';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update course set scope='low' where branch_name='EEE';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update course set scope='mid' where branch_name='ECE';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from course;
+-----------+-------------+----------+-------+
| course_id | branch_name | strength | scope |
+-----------+-------------+----------+-------+
| E03       | CSE         |     NULL | high  |
| E01       | ECE         |     NULL | mid   |
| E02       | EEE         |     NULL | low   |
+-----------+-------------+----------+-------+
3 rows in set (0.00 sec)
```

```
mysql> update student set cgpa='8.9' where id='101';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update student set cgpa='7.6' where id='102';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update student set cgpa='8.8' where id='103';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update student set cgpa='5.5' where id='104';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from student;
+-----+------+--------+------+------+
| id  | name | branch | age  | cgpa |
+-----+------+--------+------+------+
| 101 | A    | ECE    |   19 |  8.9 |
| 102 | B    | EEE    |   21 |  7.6 |
| 103 | C    | ECE    |   19 |  8.8 |
| 104 | D    | CSE    |   20 |  5.5 |
+-----+------+--------+------+------+
4 rows in set (0.00 sec)
```

Screenshots of outputs when UPDATE command is used to modify records in the table.

WHERE clause in MySQL queries is used to filter rows for a specific condition.

DELETE

The DELETE statement is used to delete existing records in a table.

Syntax:

DELETE FROM table_name WHERE condition;

```
mysql> delete from course where course_id='E04';
Query OK, 1 row affected (0.01 sec)

mysql> select * from course;
+-----------+-------------+----------+-------+
| course_id | branch_name | strength | scope |
+-----------+-------------+----------+-------+
| E03       | CSE         |     NULL | high  |
| E01       | ECE         |     NULL | mid   |
| E02       | EEE         |     NULL | low   |
+-----------+-------------+----------+-------+
3 rows in set (0.00 sec)
```

Screenshot of output when DELETE command is used to delete records from the table.

EXPERIMENT 4: JOINS in SQL

JOINS

Joins are the joining of two or more database tables to fetch data based on a common field. There are various types of joins with different names in different databases. Commonly known joins are self join, left join, right join and inner join.

Let us perform different JOIN operations on our two tables, 'Student' and 'Course'. Our tables are as follows:

```
mysql> select * from course;
+-----------+-------------+----------+-------+
| course_id | branch_name | strength | scope |
+-----------+-------------+----------+-------+
| E03       | CSE         |     NULL | high  |
| E01       | ECE         |     NULL | mid   |
| E02       | EEE         |     NULL | low   |
+-----------+-------------+----------+-------+
3 rows in set (0.00 sec)

mysql> select * from student;
+-----+------+--------+------+
| id  | name | branch | age  |
+-----+------+--------+------+
| 101 | A    | ECE    |   19 |
| 102 | B    | EEE    |   21 |
| 103 | C    | ECE    |   19 |
| 104 | D    | CSE    |   20 |
+-----+------+--------+------+
4 rows in set (0.00 sec)
```

Screenshot of tables that we have in our database, i.e., Student and Course

1. Regular Join:

   It is the join which gets all the records from both the tables which exactly match the given condition.

```
mysql> select id, name, branch, age, course_id, scope from student join course on student.branch=course.branch_name;
+-----+------+--------+------+-----------+-------+
| id  | name | branch | age  | course_id | scope |
+-----+------+--------+------+-----------+-------+
| 104 | D    | CSE    |   20 | E03       | high  |
| 101 | A    | ECE    |   19 | E01       | mid   |
| 103 | C    | ECE    |   19 | E01       | mid   |
| 102 | B    | EEE    |   21 | E02       | low   |
+-----+------+--------+------+-----------+-------+
4 rows in set (0.00 sec)
```

Screenshot of output when JOIN command is used to join tables.

2. Left Join:

   It is the join which gets all the records that match the given condition, and also fetch all the records from the left table.

```
mysql> select id, name, branch, age, course_id, scope from student left join course on student.branch=course.branch_name;
+-----+------+--------+------+-----------+-------+
| id  | name | branch | age  | course_id | scope |
+-----+------+--------+------+-----------+-------+
| 101 | A    | ECE    |   19 | E01       | mid   |
| 102 | B    | EEE    |   21 | E02       | low   |
| 103 | C    | ECE    |   19 | E01       | mid   |
| 104 | D    | CSE    |   20 | E03       | high  |
+-----+------+--------+------+-----------+-------+
4 rows in set (0.00 sec)
```

Screenshot of output when LEFT JOIN command is used to join tables.

3. Right Join:

It is the join which gets all the records that match the given condition, and also fetch all the records from the right table.

```
mysql> select id, name, branch, age, course_id, scope from student right join course on student.branch=course.branch_name;
+------+------+--------+------+-----------+-------+
| id   | name | branch | age  | course_id | scope |
+------+------+--------+------+-----------+-------+
|  104 | D    | CSE    |   20 | E03       | high  |
|  101 | A    | ECE    |   19 | E01       | mid   |
|  103 | C    | ECE    |   19 | E01       | mid   |
|  102 | B    | EEE    |   21 | E02       | low   |
+------+------+--------+------+-----------+-------+
4 rows in set (0.00 sec)
```

Screenshot of output when RIGHT JOIN command is used to join tables.

4. Inner Join:

The INNER JOIN keyword selects records that have matching values in both tables.

```
mysql> select id, name, branch, age, course_id, scope from student inner join course on student.branch=course.branch_name;
+-----+------+--------+------+-----------+-------+
| id  | name | branch | age  | course_id | scope |
+-----+------+--------+------+-----------+-------+
| 104 | D    | CSE    |   20 | E03       | high  |
| 101 | A    | ECE    |   19 | E01       | mid   |
| 103 | C    | ECE    |   19 | E01       | mid   |
| 102 | B    | EEE    |   21 | E02       | low   |
+-----+------+--------+------+-----------+-------+
4 rows in set (0.00 sec)
```

Screenshot of output when INNER JOIN command is used to join tables.

EXPERIMENT 5: Functions in SQL

Queries to implement different functions in SQL: MAX(), MIN(), AVG() and COUNT()

Let us consider our 'Students' table. We will perform these function operations on the CGPA column.

```
mysql> select * from student;
+-----+------+--------+------+------+
| id  | name | branch | age  | cgpa |
+-----+------+--------+------+------+
| 101 | A    | ECE    |   19 |  8.9 |
| 102 | B    | EEE    |   21 |  7.6 |
| 103 | C    | ECE    |   19 |  8.8 |
| 104 | D    | CSE    |   20 |  5.5 |
+-----+------+--------+------+------+
4 rows in set (0.00 sec)
```

Screenshot shows our table Student, holding student details including the CGPA column.

1. MAX()
   It is used to get the maximum numeric value of a particular column of table.

```
mysql> select max(cgpa) from student;
+-----------+
| max(cgpa) |
+-----------+
|       8.9 |
+-----------+
1 row in set (0.00 sec)

mysql> select max(cgpa) from student where branch='ECE';
+-----------+
| max(cgpa) |
+-----------+
|       8.9 |
+-----------+
1 row in set (0.00 sec)
```

Above screenshots show outputs when MAX() command is used on the 'Student' table.

2. MIN()
   It is used to get the minimum numeric value of a particular column of table.

```
mysql> select min(cgpa) from student;
+-----------+
| min(cgpa) |
+-----------+
|       5.5 |
+-----------+
1 row in set (0.00 sec)

mysql> select min(cgpa) from student where branch='ECE';
+-----------+
| min(cgpa) |
+-----------+
|       8.8 |
+-----------+
1 row in set (0.00 sec)
```

Above screenshots show outputs when MIN() command is used on the 'Student' table.

3. AVG()
   The AVG() function returns the average value of a numeric column.

```
mysql> select avg(cgpa) from student;
+------------------+
| avg(cgpa)        |
+------------------+
| 7.699999928474426 |
+------------------+
1 row in set (0.01 sec)

mysql> select avg(cgpa) from student where branch='ECE';
+------------------+
| avg(cgpa)        |
+------------------+
| 8.849999904632568 |
+------------------+
1 row in set (0.00 sec)
```

Above screenshots show outputs when AVG() command is used on the 'Student' table.

4. COUNT()

It returns the number of records in the table or column.

```
mysql> select count(cgpa) from student where branch='ECE';
+-------------+
| count(cgpa) |
+-------------+
|           2 |
+-------------+
1 row in set (0.00 sec)

mysql> select count(*) from student;
+----------+
| count(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

Above screenshots show outputs when COUNT() command is used on the 'Student' table.

Experiment 6: Integrity Constraints

1. Primary Key:

Primary Key is a unique column/field in a table that should not contain duplicate or NULL values and is used to identify each record in the table uniquely.

```
mysql> alter table course add primary key(branch_name);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe course;
+-------------+------------+------+-----+---------+-------+
| Field       | Type       | Null | Key | Default | Extra |
+-------------+------------+------+-----+---------+-------+
| course_id   | varchar(3) | NO   |     | NULL    |       |
| branch_name | varchar(4) | NO   | PRI | NULL    |       |
| strength    | int        | YES  |     | NULL    |       |
+-------------+------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Screenshot of output when constraint primary key is given.

2. Foreign Key:

A Foreign Key is a field in one table, that refers to the Primary Key in another table. The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

```
mysql> ALTER TABLE student ADD FOREIGN KEY (branch) REFERENCES course(branch_name);
Query OK, 4 rows affected (0.09 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> describe course;
+-------------+------------+------+-----+---------+-------+
| Field       | Type       | Null | Key | Default | Extra |
+-------------+------------+------+-----+---------+-------+
| course_id   | varchar(3) | NO   |     | NULL    |       |
| branch_name | varchar(4) | NO   | PRI | NULL    |       |
| strength    | int        | YES  |     | NULL    |       |
+-------------+------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Screenshot of output when constraint foreign key is given to a field.

3. NOT NULL:

The NOT NULL constraint enforces a column to NOT accept NULL values, and thus enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

```
mysql> create table students(id int primary key not null, name varchar(10), branch varchar(4));
Query OK, 0 rows affected (0.02 sec)

mysql> describe students;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | NO   | PRI | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
| branch | varchar(4)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Screenshot of output when constraint NOT NULL is given to a field.

4. UNIQUE:

Ensures that all values in a column are different.

```
mysql> alter table students add unique (id);
Query OK, 0 rows affected, 1 warning (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

Screenshot of output when UNIQUE constraint is given to a field.

5. DEFAULT:

Sets a default value for a column if no value is specified.

```
mysql> alter table course alter branch_name set default 'ECE';
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Screenshot of output when DEFAULT constraint is given to a field.

6. CHECK:

Ensures that the values in a column satisfies a specific condition.

Our table ‘Students’ is as follows.

```
mysql> select * from student;
+------+------+--------+------+------+
| id   | name | branch | age  | cgpa |
+------+------+--------+------+------+
| 101 | A    | ECE    |   19 |  8.9 |
| 102 | B    | EEE    |   21 |  7.6 |
| 103 | C    | ECE    |   19 |  8.8 |
| 104 | D    | CSE    |   20 |  5.5 |
+------+------+--------+------+------+
4 rows in set (0.00 sec)
```

Screenshot shows our table Student, holding student details.

Adding CHECK Constraint,

```
mysql> alter table student add check (age>=18);
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

Screenshot of output when CHECK constraint is given to a field, and it satisfies the condition.

Since condition of age being greater than 18 is satisfied, we get no error. On the other hand,

```
mysql> alter table student add check (age>=20);
ERROR 3819 (HY000): Check constraint 'student_chk_3' is violated.
mysql>
```

Screenshot of output when CHECK constraint is given to a field, and it does not satisfy the condition.

We get an error when a constraint of age being greater than 20 is given, since, not all records in the table satisfy this condition.

Experiment 7: Implementation of Views

What is a View?
A view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

Syntax:
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;

Example 1:
Creating a view with IDs and Names of ECE students.

```
mysql> create view ece_stud as select id, name from student where branch='ECE';
Query OK, 0 rows affected (0.01 sec)

mysql> select * from ece_stud;
+-----+------+
| id  | name |
+-----+------+
| 101 | A    |
| 103 | C    |
+-----+------+
2 rows in set (0.00 sec)
```

Screenshot of output when 'CREATE VIEW…' statement is used on a table with a condition.

Example 2:
Creating a view with ID, Name, Branch and CGPA of students having a CGPA greater than 8.

```
mysql> create view top_rankers as select id, name, branch, cgpa from student where cgpa>8;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from top_rankers;
+-----+------+--------+------+
| id  | name | branch | cgpa |
+-----+------+--------+------+
| 101 | A    | ECE    |  8.9 |
| 103 | C    | ECE    |  8.8 |
+-----+------+--------+------+
2 rows in set (0.00 sec)
```

Screenshot of output when 'CREATE VIEW…' statement is used on a table with another condition.

Experiment 8: Implementation of Triggers

What is a Trigger?

A Trigger is a block of SQL code that is automatically executed (fired) when a specified event occurs on a table, such as INSERT, UPDATE, or DELETE.

Purpose of Triggers:
· Enforce business rules.
· Maintain audit logs.
· Automatically update dependent data.

Syntax:
CREATE TRIGGER trigger_name
BEFORE | AFTER INSERT | UPDATE | DELETE
ON table_name
FOR EACH ROW
BEGIN
  -- SQL statements
END;
Example 1:
Creating a Trigger:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER after_student_insert
    -> AFTER INSERT ON Student
    -> FOR EACH ROW
    -> BEGIN
    ->   INSERT INTO Student_Audit (student_id, action_type)
    ->   VALUES (NEW.id, 'INSERT');
    -> END;
    -> //
Query OK, 0 rows affected (0.485 sec)

mysql> DELIMITER ;
mysql> INSERT INTO Student (id, name, branch, age)
    -> VALUES (105, 'E', 'ECE', 22);
Query OK, 1 row affected (0.135 sec)

mysql>
mysql> SELECT * FROM Student_Audit;
+----------+------------+-------------+---------------------+
| audit_id | student_id | action_type | action_time         |
+----------+------------+-------------+---------------------+
|        1 |        105 | INSERT      | 2025-11-03 09:27:33 |
+----------+------------+-------------+---------------------+
1 row in set (0.026 sec)
```

Creating an Update trigger:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER after_student_update
    -> AFTER UPDATE ON Student
    -> FOR EACH ROW
    -> BEGIN
    ->   INSERT INTO Student_Audit (student_id, action_type)
    ->   VALUES (NEW.id, 'UPDATE');
    -> END;
    -> //
Query OK, 0 rows affected (0.172 sec)

mysql> DELIMITER ;
mysql> UPDATE Student SET age = 23 WHERE id = 105;
Query OK, 1 row affected (0.147 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Student_Audit;
+----------+------------+-------------+---------------------+
| audit_id | student_id | action_type | action_time         |
+----------+------------+-------------+---------------------+
|        1 |        105 | INSERT      | 2025-11-03 09:27:33 |
|        2 |        105 | UPDATE      | 2025-11-03 09:27:50 |
+----------+------------+-------------+---------------------+
2 rows in set (0.009 sec)
```

Creating a Delete Trigger:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER after_student_delete
    -> AFTER DELETE ON Student
    -> FOR EACH ROW
    -> BEGIN
    ->   INSERT INTO Student_Audit (student_id, action_type)
    ->   VALUES (OLD.id, 'DELETE');
    -> END;
    -> //
Query OK, 0 rows affected (0.180 sec)

mysql> DELIMITER ;
mysql> DELETE FROM Student WHERE id = 103;
Query OK, 1 row affected (0.141 sec)

mysql> SELECT * FROM Student_Audit;
+----------+------------+-------------+---------------------+
| audit_id | student_id | action_type | action_time         |
+----------+------------+-------------+---------------------+
|        1 |        105 | INSERT      | 2025-11-03 09:27:33 |
|        2 |        105 | UPDATE      | 2025-11-03 09:27:50 |
|        3 |        103 | DELETE      | 2025-11-03 09:28:03 |
+----------+------------+-------------+---------------------+
3 rows in set (0.019 sec)
```

Experiment 9: Referential Integrity

What is a REFERENTIAL INTEGRITY ?

Referential Integrity ensures that the relationship between tables remains consistent. It is maintained using the FOREIGN KEY constraint, which enforces a link between data in two tables. The child table depends on the parent table for valid entries.

For example, in a college database, the 'Student' table (child) depends on the 'Department' table (parent). A student's department ID must match an existing department ID in the Department table.

Key options in Referential Integrity:
· ON DELETE RESTRICT – Prevents deletion of parent records if dependent child records exist.
· ON DELETE CASCADE – Automatically deletes child records when the parent is deleted.

· ON UPDATE CASCADE – Updates the child record if the parent key is modified.

DELETE CASCADE:

```
mysql> DELETE FROM Course WHERE branch_name = 'EEE';
Query OK, 1 row affected (0.049 sec)

mysql> SELECT * FROM Course;
+-----------+-------------+----------+-------+
| course_id | branch_name | strength | scope |
+-----------+-------------+----------+-------+
| E01       | ECE         |       55 | mid   |
| E03       | CSE         |       70 | high  |
+-----------+-------------+----------+-------+
2 rows in set (0.010 sec)

mysql> SELECT * FROM Student;
+-----+------+--------+------+
| id  | name | branch | age  |
+-----+------+--------+------+
| 101 | A    | ECE    |   19 |
| 103 | C    | CSE    |   21 |
+-----+------+--------+------+
2 rows in set (0.008 sec)
```

UPDATE CASCADE:

```
mysql> UPDATE Course SET branch_name = 'ECE_NEW' WHERE branch_name = 'ECE';
Query OK, 1 row affected (0.076 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Course;
+-----------+-------------+----------+-------+
| course_id | branch_name | strength | scope |
+-----------+-------------+----------+-------+
| E01       | ECE_NEW     |       55 | mid   |
| E03       | CSE         |       70 | high  |
+-----------+-------------+----------+-------+
2 rows in set (0.010 sec)

mysql> SELECT * FROM Student;
+-----+------+---------+------+
| id  | name | branch  | age  |
+-----+------+---------+------+
| 101 | A    | ECE_NEW |   19 |
| 103 | C    | CSE     |   21 |
+-----+------+---------+------+
2 rows in set (0.012 sec)
```

Experiment 10 : Creating Users And Assigning Roles

A Database User is an entity (person or process) that connects to the database to perform operations like querying, inserting, or modifying data. Each user has a username and password for authentication.

Database Roles are collections of privileges that can be assigned to one or more users. Instead of granting privileges individually to each user, roles simplify the process by grouping permissions together.

Privileges are permissions granted to users or roles to perform specific actions on database objects. They can be classified into:
· System Privileges – Allow actions such as creating tables or databases.
· Object Privileges – Allow actions like SELECT, INSERT, UPDATE, DELETE on specific tables.

Common Commands:
· CREATE USER – Creates a new database user.
· CREATE ROLE – Creates a reusable set of privileges.
· GRANT – Assigns privileges or roles to users.
· REVOKE – Removes privileges or roles from users.

Create New Database Users:

```
mysql> CREATE USER 'admin_user'@'localhost' IDENTIFIED BY 'Admin123';
ERROR 1396 (HY000): Operation CREATE USER failed for 'admin_user'@'localhost'
mysql> CREATE USER 'faculty_user'@'localhost' IDENTIFIED BY 'Faculty123';
Query OK, 0 rows affected (0.097 sec)

mysql> CREATE USER 'student_user'@'localhost' IDENTIFIED BY 'Student123';
Query OK, 0 rows affected (0.095 sec)
```

Grant Privileges to Users:

```
mysql> -- Full privileges for admin
Query OK, 0 rows affected (0.006 sec)

mysql> GRANT ALL PRIVILEGES ON CollegeDB.* TO 'admin_user'@'localhost';
Query OK, 0 rows affected (0.109 sec)

mysql>
mysql> -- Faculty can view and modify student data
Query OK, 0 rows affected (0.005 sec)

mysql> GRANT SELECT, INSERT, UPDATE ON CollegeDB.Student TO 'faculty_user'@'localhost';
Query OK, 0 rows affected (0.074 sec)

mysql>
mysql> -- Student can only view data
Query OK, 0 rows affected (0.004 sec)

mysql> GRANT SELECT ON CollegeDB.Course TO 'student_user'@'localhost';
Query OK, 0 rows affected (0.041 sec)
```

Verify Privileges to Users:

```
mysql> SHOW GRANTS FOR 'admin_user'@'localhost';
+-----------------------------------------------------------------------+
| Grants for admin_user@localhost                                       |
+-----------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `admin_user`@`localhost`                        |
| GRANT ALL PRIVILEGES ON `collegedb`.* TO `admin_user`@`localhost`     |
| GRANT ALL PRIVILEGES ON `onlineshopping`.* TO `admin_user`@`localhost` |
+-----------------------------------------------------------------------+
3 rows in set (0.015 sec)

mysql> SHOW GRANTS FOR 'faculty_user'@'localhost';
+-----------------------------------------------------------------------------------+
| Grants for faculty_user@localhost                                                 |
+-----------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `faculty_user`@`localhost`                                  |
| GRANT SELECT, INSERT, UPDATE ON `collegedb`.`student` TO `faculty_user`@`localhost` |
+-----------------------------------------------------------------------------------+
2 rows in set (0.005 sec)

mysql> SHOW GRANTS FOR 'student_user'@'localhost';
+-----------------------------------------------------------------+
| Grants for student_user@localhost                               |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO `student_user`@`localhost`                |
| GRANT SELECT ON `collegedb`.`course` TO `student_user`@`localhost` |
+-----------------------------------------------------------------+
2 rows in set (0.003 sec)
```

Create roles:

```
mysql> CREATE ROLE 'FacultyRole', 'StudentRole';
Query OK, 0 rows affected (0.106 sec)

mysql> GRANT SELECT, INSERT, UPDATE ON CollegeDB.Student TO 'FacultyRole';
Query OK, 0 rows affected (0.067 sec)

mysql> GRANT SELECT ON CollegeDB.Course TO 'StudentRole';
Query OK, 0 rows affected (0.050 sec)

mysql> GRANT 'FacultyRole' TO 'faculty_user'@'localhost';
Query OK, 0 rows affected (0.039 sec)

mysql> GRANT 'StudentRole' TO 'student_user'@'localhost';
Query OK, 0 rows affected (0.020 sec)

mysql> REVOKE UPDATE ON CollegeDB.Student FROM 'faculty_user'@'localhost';
Query OK, 0 rows affected (0.062 sec)
```

Conclusion

SQL is a powerful tool for managing relational databases. This report covered essential concepts like table creation, constraints (Primary Key, Foreign Key, NOT NULL), and operations (ALTER, UPDATE, DELETE) to ensure data integrity and consistency. Joins enable combining data from multiple tables, while functions like MAX(), MIN(), AVG(), and COUNT() simplify data analysis. Integrity constraints and views further enhance data quality, security, and performance. These SQL features collectively empower developers to create efficient and robust database systems.