

**DBMS LABORATORY WITH MINI PROJECT**  
**[As per Choice Based Credit System (CBCS) scheme]**  
**(Effective from the academic year 2018 -2019)**  
**SEMESTER – V**

Subject Code: **18CSL58**

Number of Contact Hours/Week: **0:2:2**

Total Number of Lab Contact Hours: **36**

IA Marks : **40**

Exam Marks: **60**

Exam Hours: **03**

**CREDITS – 02**

**Course objectives:** This course will enable students to

- Foundation knowledge in database concepts, technology and practice to groom students into well-informed database application developers.
- Strong practice in SQL programming through a variety of database problems.
- Develop database applications using front-end tools and back-end DBMS.

**Description (If any):**

**PART-A: SQL Programming (Max. Exam Mks. 50)**

- Design, develop, and implement the specified queries for the following problems using Oracle, MySQL, MS SQL Server, or any other DBMS under LINUX/Windows environment.
- Create Schema and insert at least 5 records for each table. Add appropriate database constraints.

**PART-B: Mini Project (Max. Exam Mks. 30)**

- Use Java, C#, PHP, Python, or any other similar front-end tool. All applications must be demonstrated on desktop/laptop as a stand-alone or web based application (Mobile apps on Android/IOS are not permitted.)
- Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

**Lab Experiments:**  
**Part A: SQL Programming**

1. Consider the following schema for a LibraryDatabase:

BOOK (Book\_id, Title, Publisher\_Name, Pub\_Year)

BOOK\_AUTHORS (Book\_id, Author\_Name)

PUBLISHER (Name, Address, Phone)

BOOK\_COPIES (Book\_id, Programme\_id, No-of\_Copies)

BOOK\_LENDING (Book\_id, Programme\_id, Card\_No, Date\_Out, Due\_Date)

LIBRARY\_PROGRAMME (Programme\_id, Programme\_Name, Address)

Write SQL queries to

1. Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each Programme, etc.
2. Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun 2017.
3. Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.
4. Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.
5. Create a view of all books and its number of copies that are currently available in the Library.

2. Consider the following schema for Order Database:
- SALESMAN**(Salesman\_id, Name, City, Commission)  
**CUSTOMER**(Customer\_id, Cust\_Name, City, Grade, Salesman\_id)  
**ORDERS**(Ord\_No, Purchase\_Amt, Ord\_Date, Customer\_id, Salesman\_id)
- Write SQL queries to
1. Count the customers with grades above Bangalore's average.
  2. Find the name and numbers of all salesman who had more than one customer.
  3. List all the salesman and indicate those who have and don't have customers in their cities (Use UNION operation.)
  4. Create a view that finds the salesman who has the customer with the highest order of a day.
  5. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.
3. Consider the schema for Movie Database:
- ACTOR**(Act\_id, Act\_Name, Act\_Gender)  
**DIRECTOR**(Dir\_id, Dir\_Name, Dir\_Phone)  
**MOVIES** (Mov\_id, Mov\_Title, Mov\_Year, Mov\_Lang, Dir\_id)  
**MOVIE\_CAST**(Act\_id, Mov\_id, Role)  
**RATING** (Mov\_id, Rev\_Stars)
- Write SQL queries to
1. List the titles of all movies directed by 'Hitchcock'.
  2. Find the movie names where one or more actors acted in two or more movies.
  3. List all actors who acted in a movie before 2000 and also in a movie after 2015 (use JOIN operation).
  4. Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.
  5. Update rating of all movies directed by 'Steven Spielberg' to 5.
4. Consider the schema for College Database:
- STUDENT**(USN, SName, Address, Phone, Gender)  
**SEMSEC**(SSID, Sem, Sec)  
**CLASS** (USN, SSID)  
**SUBJECT** (Subcode, Title, Sem, Credits)  
**IAMARKS** (USN, Subcode, SSID, Test1, Test2, Test3, FinalIA)
- Write SQL queries to
1. List all the student details studying in fourth semester 'C' section.
  2. Compute the total number of male and female students in each semester and in each section.
  3. Create a view of Test1 marks of student USN '1BI15CS101' in all subjects.
  4. Calculate the FinalIA (average of best two test marks) and update the corresponding table for all students.
  5. Categorize students based on the following criterion:  
 If FinalIA = 17 to 20 then CAT = 'Outstanding'  
 If FinalIA = 12 to 16 then CAT = 'Average'  
 If FinalIA < 12 then CAT = 'Weak'
- Give these details only for 8th semester A, B, and C section student

5. Consider the schema for Company Database:
- EMPLOYEE (SSN, Name, Address, Sex, Salary, SuperSSN, DNo)  
DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate)  
DLOCATION (DNo, DLoc)  
PROJECT (PNo, PName, PLocation, DNo)  
WORKS\_ON(SSN, PNo, Hours)
- Write SQL queries to
1. Make a list of all project numbers for projects that involve an employee whose last name is ‘Scott’, either as a worker or as a manager of the department that controls the project.
  2. Show the resulting salaries if every employee working on the ‘IoT’ project is given a 10 percent raise.
  3. Find the sum of the salaries of all employees of the ‘Accounts’ department, as well as the maximum salary, the minimum salary, and the average salary in this department
  4. Retrieve the name of each employee who works on all the projects controlled by department number 5 (use NOT EXISTS operator).
  5. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs. 6,00,000.

### **Part B: Mini project**

- For any problem selected
- Make sure that the application should have five or more tables
- Indicative areas include; health care

**Course outcomes:** The students should be able to:

- Create, Update and query on the database.
- Demonstrate the working of different concepts of DBMS
- Implement, analyze and evaluate the project developed for an application.

### **Conduction of Practical Examination:**

#### **Experiment distribution**

1. For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
2. For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
3. Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
4. Marks Distribution (Coursed to change in accordance with university regulations)
5. For laboratories having only one part – Procedure + Execution + Viva-Voce:  

$$15+70+15 = 100 \text{ Marks}$$
6. For laboratories having PART A and PART B
  - i. Part A – Procedure + Execution + Viva =  $6 + 28 + 6 = 40 \text{ Marks}$
  - ii. Part B – Procedure + Execution + Viva =  $9 + 42 + 9 = 60 \text{ Marks}$

## **CONTENTS**

<b>Sl.No.</b>	<b>Particulars</b>	<b>Page No</b>
1	INTRODUCTION TO DATABASE	1
2	BASIC QUERIES IN SQL	10
3	INTRODUCTION TO JOINS	22
4	INTRODUCTION TO SUBQUERY	27
5	VIEWS IN SQL	31
6	INTRODUCTION TO STORED PROCEDURES	33
7	INTRODUCTION TO MYSQL TRIGGERS	49
8	EXPERIMENT 1 - Library Database	65
9	EXPERIMENT 2 - Order Database	77
10	EXPERIMENT 3 - Movie Database	84
11	EXPERIMENT 4 - College Database	92
12	EXPERIMENT 5 - Company Database	103
13	VIVA QUESTIONS	112
14	ADDITIONAL QUERIES	121

## INTRODUCTION

### INTRODUCTION TO DATABASE

#### What is Database?

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching, and replicating the data it holds. now a days we use relational database management systems (RDBMS) to store and manager huge volume of data.

**A Relational DataBase Management System (RDBMS)** is a software that:

- Enables you to implement a database with tables, columns, and indexes.
- Guarantees the Referential Integrity between rows of various tables.
- Interprets an SQL query and combines information from various tables.

#### RDBMS Terminology:

**Database:** A database is a collection of tables, with related data.

**Table:** A table is a matrix with data. A table in a database looks like a *simple spreadsheet*.

**Column:** One column (data element) contains data of one and the same kind, for example the column postcode. or phone numbers

**Row:** A row (= tuple, entry or record) is a group of related data, for example the data of one subscription.

**Redundancy:** Storing data twice, redundantly to make the system faster.

**Primary Key:** A primary key is unique. A key value can not occur twice in one table. With a key you can find at most one row.

**Foreign Key:** A foreign key is the linking pin between two tables.

**Compound Key:** A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.

**Index:** An index in a database resembles an index at the back of a book.

**Referential Integrity:** Referential Integrity makes sure that a foreign key value always points to an existing row

**DDL or Data Definition Language** actually consists of the SQL commands that can be used to to create and modify the structure of database objects in a database. These database objects include views, schemas, tables, indexes, etc.

**Some examples:**

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database

**DML is Data Manipulation Language** statements: which are used to interact with a database by deleting, inserting, retrieving, or updating data in the database.

**Some examples:**

- SELECT - retrieve data from the a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain

**DCL is Data Control Language** statements: which includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

**Some examples:**

- **GRANT**-gives user's access privileges to database.
- **REVOKE**-withdraw user's access privileges given by using the GRANT command.

**TCL is Transaction Control Language** which deals with a transaction within a database.

**Some examples:**

- COMMIT - save work done
- SAVEPOINT - identify a point in a transaction to which you can later roll back
- ROLLBACK - restore database to original since the last COMMIT
- SET TRANSACTION - Change transaction options like what rollback segment to use.

## SQL Data Types

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

MySQL uses many different data types broken into three categories

- Numeric
- Date and Time
- String Type

**DATA TYPES****NUMERIC:**

- **INT** – A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
- **TINYINT** – A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
- **SMALLINT** – A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
- **MEDIUMINT** – A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
- **BIGINT** – A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.
- **FLOAT (M, D)** – A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10, 2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.
- **DOUBLE (M, D)** – A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16, 4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
- **DECIMAL (M, D)** – an unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

**DATE AND TIME TYPES**

The MySQL date and time data types are as follows –

- **DATE** – A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30<sup>th</sup>, 1973 would be stored as 1973-12-30.

- **DATETIME** – A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30<sup>th</sup>, 1973 would be stored as 1973-12-30 15:30:00.
- **TIMESTAMP** – A timestamp between midnight, January 1<sup>st</sup>, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30<sup>th</sup>, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).
- **TIME** – Stores the time in a HH:MM:SS format.
- **YEAR (M)** – Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2 (for example YEAR (2)), YEAR can be between 1970 to 2069 (70 to 69). If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

## STRING TYPES

This list describes the common string data types in MySQL.

- **CHAR (M)** – A fixed-length string between 1 and 255 characters in length (for example CHAR (5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
- **VARCHAR (M)** – A variable-length string between 1 and 255 characters in length. For example, VARCHAR (25). You must define a length when creating a VARCHAR field.
- **BLOB or TEXT** – A field with a maximum length of 65535 characters. BLOBS are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data. The difference between the two is that the sorts and comparisons on the stored data are **case sensitive** on BLOBS and are **not case sensitive** in TEXT fields. You do not specify a length with BLOB or TEXT.
- **TINYBLOB or TINYTEXT** – A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.
- **MEDIUMBLOB or MEDIUMTEXT** – A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.
- **LONGBLOB or LONGTEXT** – A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.

**CREATE TABLE**

Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types

Syntax of CREATE Command:

CREATE TABLE<table name>

(<AttributeA1><Data TypeD1> [<Constraints>],  
<Attribute A2><Data Type D2> [<Constraints>],  
.....  
<Attribute An><Data Type Dn> [<Constraints>],  
[<integrity-constraint1>, <integrity-constraint k> ]);

Specifying the unique, primary key attributes, secondary keys, and referential integrity constraints

**EXAMPLE OF CREATING TABLE**

CREATE TABLE ORDERS

(  
ORDER\_ID INT (6) PRIMARY KEY,  
ORDER\_DATE DATE  
);

**ALTER TABLE STATEMENT**

Once a table is created in the database, there are many occasions where one may wish to change the structure of the table. Typical cases include the following:

- Add a column
- Drop a column
- Change a column name
- Change the data type for a column
- add and drop various constraints on an existing table.including primarykey and foreign key

The SQL syntax for **ALTER TABLE** is

**ALTER TABLE "table\_name" [alter specification]**

[alter specification] is dependent on the type of alteration we wish to perform. alter specification is already mentioned above

**ADDING COLUMN IN TABLE**

To add a column in a table, use the following syntax:

```
ALTER TABLE TABLENAME ADD COLUMN_NAME DATETYPE;
```

**Example:** ALTER TABLE ORDERS ADD JOB VARCHAR (20);

**MODIFYING DATATYPE FOR COLUMN IN TABLE**

To modify a column data type in a table, use the following syntax:

```
ALTER TABLE ORDERS MODIFY COLUMN_NAME VARCHAR (50);
```

**Example:** ALTER TABLE ORDERS MODIFY JOB VARCHAR (50);

**RENAMING COLUMN IN TABLE**

You can rename a column in MySQL using the ALTER TABLE and CHANGE commands together to change an existing column.

For example, say the column is currently named JOB, but you decide that DESIGNATION is a more appropriate title. The column is located on the table entitled ORDERS.

Here is an example of how to change it:

```
ALTER TABLE TABLENAME CHANGE OLDNAME NEWNAME VARCHAR (20);
```

**Example:** ALTER TABLE ORDERS CHANGE JOB DESIGNATION VARCHAR (20);

**DELETING COLUMN**

To delete a column in a table, use the following syntax:

```
ALTER TABLE TABLENAME DROP COLUMN COLUMN_NAME
```

**Example:** ALTER TABLE ORDERS DROP COLUMN DESIGNATION;

**RENAMING A TABLE**

To rename a table, use the following syntax:

```
RENAME TABLE OLDTABLENAME TO NEWTABLENAME;
```

**Example:** RENAME TABLE ORDERS TO ORDERS\_TBL

**DROP TABLE**

It is very easy to drop an existing MySQL table, but you need to be very careful while deleting any existing table because the data lost will not be recovered after deleting a table.

The DROP TABLE statement is used to drop an existing table in a database

DROP TABLE TABLENAME;

**Example:** DROP TABLE ORDER

**TRUNCATE TABLE STATEMENT**

**if we wish to simply get rid of the data but not the table itself? For this, we can use the TRUNCATE TABLE command.**

The syntax for TRUNCATE TABLE is

TRUNCATE "table\_name"

So, if we wanted to truncate the table called customer that we created in MYSQL , we simply type,

**Example:** TRUNCATE customer

**CONSTRAINTS:**

Common types of constraints include the following:

**Primary Key:-**

- A primary key is used to uniquely identify each row in a table. It can either be part of the actual record itself, or it can be an artificial field (one that has nothing to do with the actual record).
- A primary key can consist of one or more fields on a table. When multiple fields are used as a primary key, they are called a composite key.
- Primary keys can be specified either when the table is created (using CREATE TABLE) or by changing the existing table structure (using ALTER TABLE).

Below are examples for specifying a primary key when creating a table:

**Example :**

CREATE TABLE ORDERS (ORDER\_ID INT (6) PRIMARY KEY, ORDER\_DATE DATE);

Below are examples for specifying a primary key by altering a table:

CREATE TABLE ORDERS (ORDER\_ID INT (6), ORDER\_DATE DATE);

---

---

**Example :**ALTER TABLE ORDERS ADD PRIMARY KEY (ORDER\_ID);

**Note:** - Before using the ALTER TABLE command to add a primary key, you'll need to make sure that the field is defined as 'NOT NULL' -- in other words, NULL cannot be an accepted value for that field. and column values must be unique

ALTER TABLE TABLENAME DROP PRIMARY KEY CONSTRAINT

To drop a PRIMARY KEY constraint in Table ORDERS, use the following MYSQL:syntax

**Example:**ALTER TABLE ORDERS DROP PRIMARY KEY

### FOREIGN KEY

- A foreign key is a field (or fields) that points to the primary key of another table.
- The purpose of the foreign key is to ensure referential integrity of the data. In other words, only values that are supposed to appear in the database are permitted
- For example, say we have two tables, a CUSTOMER table that includes all customer data, and an ORDERS table that includes all customer orders. The constraint here is that all orders must be associated with a customer that is already in the CUSTOMER table.
- In this case, we will place a foreign key on the ORDERS table and have it relate to the primary key of the CUSTOMER table. This way, we can ensure that all orders in the ORDERS table are related to a customer in the CUSTOMER table. .In other words, the ORDERS table cannot contain information on a customer that is not in the CUSTOMER table.

**The structure of these two tables will be as follows:**

**Table CUSTOMER**

column name	Characteristic
SID	Primary Key
Last_Name	varchar(50)
First_Name	varchar(50)

**Table ORDERS**

column name	characteristic
Order_ID	Primary Key
Order_Date	Date
Customer_SID	Foreign Key
Amount	Decimal(10,2)

---

```
CREATE TABLE CUSTOMER
(
SID INT PRIMARY KEY,
Last_Name varchar(50),
First_Name varchar(50)
);
```

In the below example, the Customer\_SID column in the ORDERS table is a foreign key pointing to the SID column which is primary key in the CUSTOMER table.

Below we show examples of how to specify the foreign key when creating the ORDERS table:

```
CREATE TABLE ORDERS
(
Order_ID int,
Order_Date date,
Customer_SID int,
Amount double,
Primary Key (Order_ID),
Foreign Key (Customer_SID) references CUSTOMER(SID)
);
```

Below are examples for specifying a foreign key by altering a table.

This assumes that the ORDERS table has been created, and the foreign key has not yet been put in

```
ALTER TABLE ORDERS ADD FOREIGN KEY (customer_sid) REFERENCES
CUSTOMER(SID);
```

We can drop a foreign key by using below syntax

```
ALTER TABLE ORDERS DROP FOREIGN KEY FOREIGNKEY_CONSTRAINT_NAME;
```

**NOT NULL Constraint:-** By default, a column can hold NULL. If you don't want to allow or store NULL value in a column, you will want to place a constraint on this column specifying that NULL is now not an allowable value.

**DEFAULT Constraint:-** The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

**UNIQUE Constraint:-** The UNIQUE constraint ensures that all values in a column are distinct.

**CHECK Constraint:-** The CHECK constraint ensures that all values in a column satisfy certain conditions. Once defined, the database will only insert a new row or update an existing row if the new value satisfies the CHECK constraint. The CHECK constraint is used to ensure data quality

## BASIC QUERIES IN SQL

- SQL has one basic statement for retrieving information from a database; the SELECT statement
- This is *not the same as* the SELECT operation of the relational algebra
- Important distinction between SQL and the formal relational model;
- SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
- Hence, an SQL relation (table) is a *multi-set* (sometimes called a bag) of tuples; it is *not* a set of tuples
- SQL relations can be constrained to be sets by using the CREATE UNIQUE INDEX command, or by using the DISTINCT option
- Basic form of the SQL SELECT statement is called a *mapping* of a *SELECT-FROM-WHERE block*

SELECT <attribute list> FROM <table list> WHERE <condition>

- <attribute list> is a list of attribute names whose values are to be retrieved by the query
- <table list> is a list of the relation names required to process the query
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

## SIMPLE SQL QUERIES

Basic SQL queries correspond to using the following operations of the relational algebra:

### SELECT PROJECT JOIN

All subsequent examples uses COMPANY database as shown below:

#### Example of a simple query on one relation

**Query 0: Retrieve the birth date and address of the employee whose name is 'John B. Smith'.**

```
Q0: SELECT BDATE, ADDRESS FROM EMPLOYEE
      WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith'
```

Similar to a SELECT-PROJECT pair of relational algebra operations: The SELECT-clause specifies the projection attributes and the WHERE-clause specifies the selection condition. However, the result of the query may contain duplicate tuples.

EMPLOYEE															
FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO						
DEPARTMENT															
DNAME		DNUMBER		MGRSSN		MGRSTARTDATE									
DEPT_LOCATIONS															
DNUMBER		DLOCATION													
PROJECT															
PNAME		PNUMBER		PLOCATION		DNUM									
WORKS_ON															
ESSN		PNO		HOURS											
DEPENDENT															
ESSN		DEPENDENT_NAME		SEX		BDATE		RELATIONSHIP							

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	566884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888888888	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
ProductX		1	Bethesda	5
ProductY		2	Sugartown	5
ProductZ		3	Houston	5
Computerization		10	Stratford	4
Reorganization		20	Houston	1
Newbenefits		30	Stratford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1968-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

**Example of a simple query on two relations**

**Query 1:** Retrieve the name and address of all employees who work for the 'Research' department.

Q1: SELECT FNAME, LNAME, ADDRESS FROM EMPLOYEE, DEPARTMENT  
WHERE DNAME='Research' AND DNUMBER=DNO

Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations (DNAME='Research') is a selection condition (corresponds to a SELECT operation in relational algebra) (DNUMBER=DNO) is a join condition (corresponds to a JOIN operation in relational algebra)

**Example of a simple query on three relations**

**Query 2:** For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

Q2: SELECT PNUMBER, DNUM, LNAME, BDATE, ADDRESS FROM PROJECT,  
DEPARTMENT, EMPLOYEE WHERE DNUM=DNUMBER AND MGRSSN=SSN  
AND PLOCATION='Stafford'

In Q2, there are two join conditions. The join condition DNUM=DNUMBER relates a project to its controlling department. The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

**ALIASES, \* AND DISTINCT, EMPTY WHERE-CLAUSE**

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in different relations
- A query that refers to two or more attributes with the same name must qualify the attribute name with the relation name by prefixing the relation name to the attribute name

**Example:** EMPLOYEE.LNAME, DEPARTMENT.DNAME

- Some queries need to refer to the same relation twice. In this case, aliases are given to the relation name.

**Example**

**Query 3:** For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

Q3: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME FROM EMPLOYEE S WHERE  
E.SUPERSSN=S.SSN

In Q3, the alternate relation names E and S are called aliases or tuple variables for the EMPLOYEE relation. We can think of E and S as two different copies of EMPLOYEE; E represents employees in role of supervisees and S represents employees in role of supervisors

Aliasing can also be used in any SQL query for convenience. Can also use the AS keyword to

specify aliases

Q3: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME FROM EMPLOYEE AS E,  
EMPLOYEE AS S WHERE E.SUPERSSN=S.SSN

### **UNSPECIFIED WHERE-clause**

A missing WHERE-clause indicates no condition; hence, all tuples of the relations in the FROM-clause are selected. This is equivalent to the condition WHERE TRUE

Example:

#### **Query 4: Retrieve the SSN values for all employees.**

Q4: SELECT SSN FROM EMPLOYEE

If more than one relation is specified in the FROM-clause and there is no join condition, then the CARTESIAN PRODUCT of tuples is selected

Example:

Q5: SELECT SSN, DNAME FROM EMPLOYEE, DEPARTMENT

**Note:** It is extremely important not to overlook specifying any selection and join conditions in the WHERE-clause; otherwise, incorrect and very large relations may result

### **USE OF \***

To retrieve all the attribute values of the selected tuples, a \* is used, which stands for all the attributes

Examples:

#### **Retrieve all the attribute values of EMPLOYEES who work in department 5.**

Q1a: SELECT \* FROM EMPLOYEE WHERE DNO=5

#### **Retrieve all the attributes of an employee and attributes of DEPARTMENT he works in for every employee of 'Research' department.**

Q1b: SELECT \* FROM EMPLOYEE, DEPARTMENT WHERE DNAME='Research'  
AND DNO=DNUMBER

### **USE OF DISTINCT**

SQL does not treat a relation as a set; duplicate tuples can appear. To eliminate duplicate tuples in a query result, the keyword DISTINCT is used

Example: the result of **Q1c** may have duplicate SALARY values whereas **Q1d** does not have any duplicate values

Q1c: SELECT SALARY FROM EMPLOYEE

Q1d: SELECT DISTINCT SALARY FROM EMPLOYEE

## SET OPERATIONS

SQL has directly incorporated some set operations such as union operation (UNION), set difference (MINUS) and intersection (INTERSECT) operations. The resulting relations of these set operations are sets of tuples; duplicate tuples are eliminated from the result. The set operations apply only to union compatible relations; the two relations must have the same attributes and the attributes must appear in the same order

**Query 5:** Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

Q5: (SELECT PNAME FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME='Smith')

## UNION

(SELECT PNAME FROM PROJECT, WORKS\_ON, EMPLOYEE WHERE PNUMBER=PNO AND ESSN=SSN AND NAME='Smith')

## NESTING OF QUERIES

A complete SELECT query, called a nested query, can be specified within the WHERE-clause of another query, called the outer query. Many of the previous queries can be specified in an alternative form using nesting

**Query 6:** Retrieve the name and address of all employees who work for the 'Research' department.

Q6: SELECT FNAME, LNAME, ADDRESS FROM EMPLOYEE WHERE DNO IN  
(SELECT DNUMBER FROM DEPARTMENT WHERE DNAME='Research' )

**Note:** The nested query selects the number of the 'Research' department. The outer query selects an EMPLOYEE tuple if its DNO value is in the result of either nested query. The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V

In general, we can have several levels of nested queries. A reference to an unqualified attribute refers to the relation declared in the innermost nested query. In this example, the nested query is not correlated with the outer query

## CORRELATED NESTED QUERIES

If a condition in the WHERE-clause of a nested query references an attribute of a relation declared in the outer query, the two queries are said to be correlated. The result of a correlated nested query is different for each tuple (or combination of tuples) of the relation(s) the outer query

**Query 7:** Retrieve the name of each employee who has a dependent with the same first name as the employee.

Q7: SELECT E.FNAME, E.LNAME FROM EMPLOYEE AS E WHERE E.SSN IN  
(SELECT ESSN FROM DEPENDENT WHERE ESSN=E.SSN AND

---

E.FNAME=DEPENDENT\_NAME)

In Q7, the nested query has a different result in the outer query. A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can *always* be expressed as a single block query. For example, Q7 may be written as in Q7a

Q7a: SELECT E.FNAME, E.LNAME FROM EMPLOYEE E, DEPENDENT D WHERE  
E.SSN=D.ESSN AND E.FNAME=D.DEPENDENT\_NAME

### THE EXISTS FUNCTION

EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not. We can formulate Query 7 in an alternative form that uses EXIST.

Q7b: SELECT FNAME, LNAME FROM EMPLOYEE  
WHERE EXISTS (SELECT \* FROM DEPENDENT WHERE SSN=ESSN  
AND FNAME=DEPENDENT\_NAME)

### Query 8: Retrieve the names of employees who have no dependents.

Q8: SELECT FNAME, LNAME FROM EMPLOYEE  
WHERE NOT EXISTS  
(SELECT \* FROM DEPENDENT WHERE SSN=ESSN)

**Note:** In Q8, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If none exist, the EMPLOYEE tuple is selected

### EXPLICIT SETS

It is also possible to use an explicit (enumerated) set of values in the WHERE-clause rather than a nested query

### Query 9: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

Q9: SELECT DISTINCT ESSN FROM WORKS\_ON WHERE PNO IN (1, 2, 3)

### NULLS IN SQL QUERIES

SQL allows queries that check if a value is NULL (missing or undefined or not applicable). SQL uses IS or IS NOT to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate.

### Query 10: Retrieve the names of all employees who do not have supervisors.

Q10: SELECT FNAME, LNAME FROM EMPLOYEE  
WHERE SUPERSSN IS NULL

**Note:** If a join condition is specified, tuples with NULL values for the join attributes are not

---

included in the result

### **AGGREGATE FUNCTIONS**

Include COUNT, SUM, MAX, MIN, and AVG

**Query 11: Find the maximum salary, the minimum salary, and the average salary among all employees.**

```
Q11: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
      FROM EMPLOYEE
```

**Note:** Some SQL implementations may not allow more than one function in the SELECT-clause

**Query 12: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.**

```
Q12: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY) FROM
      EMPLOYEE, DEPARTMENT WHERE DNO=DNUMBER AND DNAME='Research'
```

**Queries 13 and 14:** Retrieve the total number of employees in the company (Q13), and the number of employees in the 'Research' department (Q14).

```
Q13: SELECT COUNT(*) FROM EMPLOYEE
```

```
Q14: SELECT COUNT(*) FROM EMPLOYEE, DEPARTMENT
      WHERE DNO=DNUMBER AND DNAME='Research'
```

### **GROUPING**

- In many cases, we want to apply the aggregate functions to subgroups of tuples in a relation
- Each subgroup of tuples consists of the set of tuples that have the same value for the grouping attribute(s)
- The function is applied to each subgroup independently
- SQL has a GROUP BY-clause for specifying the grouping attributes, which must also appear in the SELECT-clause

**Query 15: For each department, retrieve the department number, the number of employees in the department, and their average salary.**

```
Q15: SELECT DNO, COUNT(*), AVG(SALARY)
      FROM EMPLOYEE GROUP BY DNO
```

- In Q15, the EMPLOYEE tuples are divided into groups. Each group having the same value for the grouping attribute DNO
- The COUNT and AVG functions are applied to each such group of tuples separately

- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
- A join condition can be used in conjunction with grouping

**Query 16: For each project, retrieve the project number, project name, and the number of employees who work on that project.**

```
Q16: SELECT PNUMBER, PNAME, COUNT (*)
      FROM PROJECT, WORKS_ON
      WHERE PNUMBER=PNO GROUP
      BY PNUMBER, PNAME
```

### THE HAVING-CLAUSE

Sometimes we want to retrieve the values of these functions for only those groups that satisfy certain conditions. The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples)

**Query 17: For each project on which more than two employees work, retrieve the project number, project name, and the number of employees who work on that project.**

```
Q17: SELECT PNUMBER, PNAME, COUNT (*)
      FROM PROJECT, WORKS_ON
      WHERE PNUMBER=PNO GROUP
      BY PNUMBER, PNAME HAVING
      COUNT (*) > 2
```

### SUBSTRING COMPARISON

The LIKE comparison operator is used to compare partial strings. Two reserved characters are used: '%' (or '\*' in some implementations) replaces an arbitrary number of characters, and '\_' replaces a single arbitrary character.

**Query 18: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring ‘Houston,TX’ in it.**

```
Q18: SELECT FNAME, LNAME
      FROM EMPLOYEE WHERE ADDRESS LIKE '%Houston,TX%'
```

**Query 19: Retrieve all employees who were born during the 1950s.**

Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is '\_\_\_\_\_5\_\_', with each underscore as a place holder for a single arbitrary character.

```
Q19: SELECT FNAME, LNAME
```

---

---

```
FROM EMPLOYEE WHERE BDATE LIKE '_____5_'
```

**Note:** The LIKE operator allows us to get around the fact that each value is considered atomic and indivisible. Hence, in SQL, character string attribute values are not atomic

### ARITHMETIC OPERATIONS

The standard arithmetic operators '+', '-'. '\*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result

**Query 20:** Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q20: SELECT FNAME, LNAME, 1.1*SALARY
      FROM EMPLOYEE, WORKS_ON, PROJECT
      WHERE SSN=ESSN
      AND PNO=PNUMBER AND PNAME='ProductX'
```

### ORDER BY

The ORDER BY clause is used to sort the tuples in a query result based on the values of some attribute(s)

**Query 21:** Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

```
Q21: SELECT DNAME, LNAME, FNAME, PNAME
      FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
      WHERE DNUMBER=DNO
      AND SSN=ESSN
      AND PNO=PNUMBER ORDER
      BY DNAME, LNAME
```

The default order is in ascending order of values. We can specify the keyword DESC if we want a descending order; the keyword ASC can be used to explicitly specify ascending order, even though it is the default

Ex: ORDER BY DNAME DESC, LNAME ASC, FNAME ASC

### MORE EXAMPLE QUERIES:

**Query 22:** Retrieve the names of all employees who have two or more dependents.

```
Q22: SELECT LNAME, FNAME FROM
      EMPLOYEE WHERE (SELECT COUNT (*) FROM DEPENDENT WHERE
```

$\text{SSN}=\text{ESSN}) \geq 2;$

**Query 23: List the names of managers who have least one dependent.**

```
Q23: SELECT FNAME, LNAME
      FROM EMPLOYEE
      WHERE EXISTS (SELECT * FROM DEPENDENT WHERE SSN=ESSN) AND
            EXISTS (SELECT * FROM DEPARTMENT WHERE SSN=MGRSSN );
```

### SPECIFYING UPDATES IN SQL

There are three SQL commands to modify the database: **INSERT**, **DELETE**, and **UPDATE**. **INSERT**

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the **CREATE TABLE** command

**Example:**

```
INSERT INTO EMPLOYEE VALUES ('Richard','K','Marini', '653298653', '30-DEC-52', '98 Oak
Forest,Katy,TX', 'M', 37000, '987654321', 4 )
```

- An alternate form of **INSERT** specifies explicitly the attribute names that correspond to the values in the new tuple. Attributes with **NULL** values can be left out.

**Example:** Insert a tuple for a new **EMPLOYEE** for whom we only know the **FNAME**, **LNAME**, and **SSN** attributes.

```
INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)VALUES ('Richard', 'Marini',
'653298653')
```

**Important Note:** Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database. Another variation of **INSERT** allows insertion of multiple tuples resulting from a **query** into a relation

**Example:** Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department. A table **DEPTS\_INFO** is created first, and is loaded with the summary information retrieved from the database by the query.

```
CREATE TABLE DEPTS_INFO(DEPT_NAME VARCHAR (10),NO_OF_EMPS INT,
TOTAL_SAL INT);
```

```
INSERT INTO DEPTS_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
```

```
SELECT DNAME, COUNT (*), SUM (SALARY) FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO GROUP BY DNAME ;
```

**Note:** The **DEPTS\_INFO** table may not be up-to-date if we change the tuples in either the

DEPARTMENT or the EMPLOYEE relations *after* issuing the above. We have to create a view (see later) to keep such a table up to date.

## **DELETE**

- Removes tuples from a relation. Includes a WHERE-clause to select the tuples to be deleted
- Referential integrity should be enforced
- Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
- A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
- The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause

### **Examples:**

1. DELETE FROM EMPLOYEE WHERE LNAME='Brown';
2. DELETE FROM EMPLOYEE WHERE SSN='123456789';
3. DELETE FROM EMPLOYEE WHERE DNO IN (SELECT DNUMBER FROM DEPARTMENT WHERE DNAME='Research');
4. DELETE FROM EMPLOYEE;

## **UPDATE**

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced

**Example1:** Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

UPDATE PROJECT

SET PLOCATION = 'Bellaire', DNUM = 5 WHERE PNUMBER=10;

**Example2:** Give all employees in the 'Research' department a 10% raise in salary.

UPDATE EMPLOYEE

SET SALARY = SALARY \*1.1

WHERE DNO IN (SELECT DNUMBER FROM DEPARTMENT

WHERE DNAME='Research');

## INTRODUCTION TO JOINS

- Joins help retrieving data from two or more database tables.
- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- Join establishes temporarily relationship between two or more tables. The tables are mutually related using primary and foreign keys.

```
CREATE TABLE MOVIES (MOVIE_ID INT (5) PRIMARY KEY,MOVIE_NAME
VARCHAR(50));
```

```
CREATE TABLE ACTORS(ACTOR_ID INT(5) PRIMARY KEY,ACTOR_NAME
VARCHAR(50), MOVIE_ID INT(5),FOREIGN KEY(MOVIE_ID) REFERENCES
MOVIES(MOVIE_ID));
```

### **HERE IS INSERT SCRIPTS FOR BOTH TABLES MOVIES AS WELL AS ACTORS**

```
INSERT INTO MOVIES VALUES(1000,'SHOLAY');
INSERT INTO MOVIES VALUES(1001,'ITTEFAQ');
INSERT INTO MOVIES VALUES(1002,'TEESRI MANZIL');
INSERT INTO MOVIES VALUES(1003,'JEWEL THIEF ');
INSERT INTO MOVIES VALUES(1004,'CARAVAN');
INSERT INTO MOVIES VALUES(1005,'GUMNAAM');
```

```
INSERT INTO ACTORS VALUES(1,'AMITABH BACHCHAN',1000);
INSERT INTO ACTORS VALUES(2,'RAJESH KHANNA',1001);
INSERT INTO ACTORS VALUES(3,'SHAMI KAPOOR',1002);
INSERT INTO ACTORS VALUES(4,'DEV ANAND',1003);
INSERT INTO ACTORS VALUES(5,'NULL',1004);
```

```
SELECT * FROM MOVIES;
```

<b>MOVIE_ID</b>	<b>MOVIE_NAME</b>
1000	SHOLAY
1001	ITTEFAQ
1002	TEESRI MANZIL
1003	JEWEL THIEF
1004	CARAVAN
1005	GUMNAAM

---

SELECT \* FROM ACTORS;

ACTOR_ID	ACTOR_NAME	MOVIE_ID
1	AMITABH BACHCHAN	1000
2	RAJESH KHANNA	1001
3	SHAMI KAPOOR	1002
4	DEV ANAND	1003
5	NULL	1004

### Types of JOINS

#### Cross JOIN

Cross JOIN is a simplest form of JOINS which matches each row from one database table to all rows of another.

In other words it gives us combinations of each row of first table with all records in second table.

**Select \* FROM TableA CROSS JOIN TableB;**

//OR//

**Select \* FROM Table1 A1,Table1 A2;**

SELECT \* FROM MOVIES CROSS JOIN ACTORS;

**Executing the above script in MySQL workbench gives us the following results.**

MOVIE_ID	MOVIE_NAME	ACTOR_ID	ACTOR_NAME	MOVIE_ID
1000	SHOLAY	1	AMITABH BACHCHAN	1000
1000	SHOLAY	2	RAJESH KHANNA	1001
1000	SHOLAY	3	SHAMI KAPOOR	1002
1000	SHOLAY	4	DEV ANAND	1003
1000	SHOLAY	5	NULL	1004
1001	ITTEFAQ	1	AMITABH BACHCHAN	1000
1001	ITTEFAQ	2	RAJESH KHANNA	1001
1001	ITTEFAQ	3	SHAMI KAPOOR	1002
1001	ITTEFAQ	4	DEV ANAND	1003
1001	ITTEFAQ	5	NULL	1004

1002	TEESRI MANZIL	1	AMITABH BACHCHAN	1000
1002	TEESRI MANZIL	2	RAJESH KHANNA	1001
1002	TEESRI MANZIL	3	SHAMI KAPOOR	1002
1002	TEESRI MANZIL	4	DEV ANAND	1003
1002	TEESRI MANZIL	5	NULL	1004
1003	JEWEL THIEF	1	AMITABH BACHCHAN	1000
1003	JEWEL THIEF	2	RAJESH KHANNA	1001
1003	JEWEL THIEF	3	SHAMI KAPOOR	1002
1003	JEWEL THIEF	4	DEV ANAND	1003
1003	JEWEL THIEF	5	NULL	1004
1004	CARAVAN	1	AMITABH BACHCHAN	1000
1004	CARAVAN	2	RAJESH KHANNA	1001
1004	CARAVAN	3	SHAMI KAPOOR	1002
1004	CARAVAN	4	DEV ANAND	1003
1004	CARAVAN	5	NULL	1004
1005	GUMNAAM	1	AMITABH BACHCHAN	1000
1005	GUMNAAM	2	RAJESH KHANNA	1001
1005	GUMNAAM	3	SHAMI KAPOOR	1002
1005	GUMNAAM	4	DEV ANAND	1003
1005	GUMNAAM	5	NULL	1004

### INNER JOIN

Technically, Join made by using equality-operator (=) to compare values of PrimaryKey of one table and Foreign Key values of another table, hence result set includes common(matched) records from both tables

The inner JOIN is used to return rows from both tables that satisfy the given condition.

**SELECT \* FROM Table1 A INNER JOIN Table2 B ON A.<PrimaryKey>=B.<ForeignKey>;**

**SELECT MOVIE\_NAME, ACTOR\_NAME, ACTOR\_ID FROM MOVIES M INNER JOIN ACTORS A ON M.MOVIE\_ID=A.MOVIE\_ID;**

**Executing the above script in MySQL workbench gives us the following result**

MOVIE_NAME	ACTOR_NAME	ACTOR_ID
SHOLAY	AMITABH BACHCHAN	1
ITTEFAQ	RAJESH KHANNA	2
TEESRI MANZIL	SHAMI KAPOOR	3

JEWEL THIEF	DEV ANAND	4
CARAVAN	NULL	5

**INNER JOIN CONSISTING OF WHERE CONDITION AND ACTOR NAME SHOULD NOT BE NULL**

SELECT MOVIE\_NAME, ACTOR\_NAME, ACTOR\_ID FROM MOVIES M INNER JOIN ACTORS A ON M.MOVIE\_ID=A.MOVIE\_ID WHERE ACTOR\_NAME!='NULL';

**OR**

SELECT MOVIE\_NAME, ACTOR\_NAME, ACTOR\_ID FROM MOVIES M INNER JOIN ACTORS A ON M.MOVIE\_ID=A.MOVIE\_ID WHERE ACTOR\_NAME<>'NULL';

Executing the above script in MySQL workbench gives us the following results.

MOVIE_NAME	ACTOR_NAME	ACTOR_ID
SHOLAY	AMITABH BACHCHAN	1
ITTEFAQ	RAJESH KHANNA	2
TEESRI MANZIL	SHAMI KAPOOR	3
JEWEL THIEF	DEV ANAND	4

**INNER JOIN CONSISTING OF WHERE CONDITION AND ACTOR NAME FIELDS ARE HAVING NULL VALUE**

SELECT MOVIE\_NAME, ACTOR\_NAME, ACTOR\_ID FROM MOVIES M INNER JOIN ACTORS A ON M.MOVIE\_ID=A.MOVIE\_ID WHERE ACTOR\_NAME='NULL';

Executing the above script in MySQL workbench gives us the following results.

MOVIE_NAME	ACTOR_NAME	ACTOR_ID
CARAVAN	NULL	5

**OUTER-JOIN**

A full outer join, or full join, which is not supported by the popular MySQL database management system, However, can customized selection of un-matched rows e.g, selecting unmatched row from first table or second table by sub-types: LEFT OUTER JOIN and RIGHT OUTER JOIN.

It can detect records having no match in joined table. It returns NULL values for records of joined table if no match is found.

### LEFT JOIN

The LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right. Where no matches have been found in the table on the right, NULL is returned.

```
Select * FROM Table1 A LEFT OUTER JOIN Table2 B On A.<PrimaryKey>=B.<ForeignKey>;
```

```
SELECT MOVIE_NAME, ACTOR_NAME, ACTOR_ID FROM MOVIES M LEFT  
OUTER JOIN ACTORS A ON M.MOVIE_ID=A.MOVIE_ID;
```

Executing the above script in MySQL workbench gives below result. You can see that in the returned result which is listed below that for movies which do not have a actor, actor name fields are having NULL values. That means no matching member found actor table for that particular movie.

MOVIE_NAME	ACTOR_NAME	ACTOR_ID
SHOLAY	AMITABH BACHCHAN	1
ITTEFAQ	RAJESH KHANNA	2
TEESRI MANZIL	SHAMI KAPOOR	3
JEWEL THIEF	DEV ANAND	4
CARAVAN	NULL	5
GUMNAAM	NULL	NULL

### RIGHT JOIN

RIGHT JOIN is obviously the opposite of LEFT JOIN. The RIGHT JOIN returns all the columns from the table on the right even if no matching rows have been found in the table on the left. Where no matches have been found in the table on the left, NULL is returned.

```
Select * FROM Table1 A RIGHT OUTER JOIN Table2 B on A.<PrimaryKey>=B.<ForeignKey>;
```

```
SELECT MOVIE_NAME, ACTOR_NAME, ACTOR_ID FROM MOVIES M RIGHT  
OUTER JOIN ACTORS A ON M.MOVIE_ID=A.MOVIE_ID;
```

Executing the above script in MySQL workbench gives the following results.

MOVIE_NAME	ACTOR_NAME	ACTOR_ID
SHOLAY	AMITABH BACHCHAN	1
ITTEFAQ	RAJESH KHANNA	2
TEESRI MANZIL	SHAMI KAPOOR	3
JEWEL THIEF	DEV ANAND	4
CARAVAN	NULL	5

## INTRODUCTION TO SUBQUERY

- A subquery is a query within another query. The outer query is called as main query and inner query is called as subquery.
- Subqueries are nested queries that provide data to the enclosing query.
- Subquery must be enclosed in parentheses.
- Subqueries can return individual values or a list of records
- You can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause.
- Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
- The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.
- Subqueries are on the right side of the comparison operator.
- ORDER BY command **cannot** be used in a Subquery. GROUPBY command can be used to perform same function as ORDER BY command.
- Use single-row operators with singlerow Subqueries. Use multiple-row operators with multiple-row Subqueries.

### Syntax:

- There is not any general syntax for Subqueries. However, Subqueries are seen to be used most frequently with SELECT statement as shown below:

```
SELECT column_name FROM table_name WHERE column_name expression operator
( SELECT COLUMN_NAME from TABLE_NAME WHERE ... );
```

Below is sample table StudentDetails and StudentSection we have created to demonstrate working of subquesry

```
create tab StudentDetails
(
Student_ID int primary key
NAME      varchar(100),
ROLL_NO   int,
LOCATION  varchar(100),
PHONE_NUMBER bigint
);
```

```
create table StudentSection
(
NAME varchar(100),
ROLL_NO int,
Section char(1)
);
```

Below is sample insert scripts for inserting information into StudentDetails

```
insert into StudentDetails values(1000,'Hemanth',101,'Mysore',9845113337);
insert into StudentDetails values(2000,'Nitin',102,'Banglore',8877665544);
insert into StudentDetails values(3000,'Sandeep',103,'Kodagu',9538945623);
insert into StudentDetails values(4000,'Sashank Hegde',104,'Udupi',8989898989);
insert into StudentDetails values(5000,'Nagendra',105,'Banglore',9901478945);
```

Below is sample insert scripts for inserting information into StudentSection

```
insert into StudentSection values('Sashank Hegde',104,'A');
insert into StudentSection values('Nagendra',105,'B');
insert into StudentSection values('Nitin',102,'A');
insert into StudentSection values('Hemanth',101,'B');
```

`select * from StudentDetails`

```
mysql> select * from StudentDetails;
+-----+-----+-----+-----+-----+
| Student_ID | NAME | ROLL_NO | LOCATION | PHONE_NUMBER |
+-----+-----+-----+-----+-----+
| 1000 | Hemanth | 101 | Mysore | 9845113337 |
| 2000 | Nitin | 102 | Banglore | 8877665544 |
| 3000 | Sandeep | 103 | Kodagu | 9538945623 |
| 4000 | Sashank Hegde | 104 | Udupi | 8989898989 |
| 5000 | Nagendra | 105 | Banglore | 9901478945 |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

`select * from StudentSection`

```
mysql> select * from StudentSection;
+-----+-----+-----+
| NAME | ROLL_NO | Section |
+-----+-----+-----+
| Sashank Hegde | 104 | A |
| Nagendra | 105 | B |
| Nitin | 102 | A |
| Hemanth | 101 | B |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

**Query1:** To display NAME, LOCATION, PHONE\_NUMBER of the StudentDetails table whose section is A

```
Select NAME, LOCATION, PHONE_NUMBER from StudentDetails WHERE ROLL_NO
IN(SELECT ROLL_NO from StudentSection where SECTION='A');
```

Explanation : First subquery executes “ SELECT ROLL\_NO from STUDENT where SECTION='A' ” returns ROLL\_NO from STUDENT table whose SECTION is ‘A’. Then outer-query executes it and return the NAME, LOCATION, PHONE\_NUMBER from the DATABASE table of the student whose ROLL\_NO is returned from inner subquery.

Below is snapshot of output of above executed subquery

```
mysql> Select NAME, LOCATION, PHONE_NUMBER from StudentDetails
-> WHERE ROLL_NO IN(SELECT ROLL_NO from StudentSection where SECTION='A');
+-----+-----+-----+
| NAME | LOCATION | PHONE_NUMBER |
+-----+-----+-----+
| Nitin | Banglore | 8877665544 |
| Sashank Hegde | Udupi | 8989898989 |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

**Query2:** To update name from StudentDetails table whose rollno is same as that in StudentSection table and having name as Nitin by Using subquery

```
UPDATE StudentDetails SET NAME='Nitin Jain'
```

```
WHERE ROLL_NO IN(SELECT ROLL_NO FROM StudentSection where NAME='Nitin');
```

```
mysql> select * from studentdetails;
+-----+-----+-----+-----+-----+
| Student_ID | NAME | ROLL_NO | LOCATION | PHONE_NUMBER |
+-----+-----+-----+-----+-----+
| 1000 | Hemanth | 101 | Mysore | 9845113337 |
| 2000 | Nitin | 102 | Banglore | 8877665544 |
| 3000 | Sandeep | 103 | Kodagu | 9538945623 |
| 4000 | Sashank Hegde | 104 | Udupi | 8989898989 |
| 5000 | Nagendra | 105 | Banglore | 9901478945 |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> UPDATE StudentDetails SET NAME='Nitin Jain'
-> WHERE ROLL_NO IN(SELECT ROLL_NO FROM StudentSection where NAME='Nitin');
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from studentdetails;
+-----+-----+-----+-----+-----+
| Student_ID | NAME | ROLL_NO | LOCATION | PHONE_NUMBER |
+-----+-----+-----+-----+-----+
| 1000 | Hemanth | 101 | Mysore | 9845113337 |
| 2000 | Nitin Jain | 102 | Banglore | 8877665544 |
| 3000 | Sandeep | 103 | Kodagu | 9538945623 |
| 4000 | Sashank Hegde | 104 | Udupi | 8989898989 |
| 5000 | Nagendra | 105 | Banglore | 9901478945 |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

**Query3: To delete students from Student2 table whose rollno is same as that in Student1 table and having location as chennai**

```
DELETE FROM StudentDetails WHERE ROLL_NO IN ( SELECT ROLL_NO
FROM StudentSection WHERE NAME ='Nagendra');
```

```
mysql> select * from StudentDetails;
+-----+-----+-----+-----+-----+
| Student_ID | NAME | ROLL_NO | LOCATION | PHONE_NUMBER |
+-----+-----+-----+-----+-----+
| 1000 | Hemanth S R | 101 | Mysore | 9845113337 |
| 2000 | Hemanth S R | 102 | Banglore | 8877665544 |
| 3000 | Sandeep | 103 | Kodagu | 9538945623 |
| 4000 | Sashank Hegde | 104 | Udupi | 8989898989 |
| 5000 | Nagendra | 105 | Banglore | 9901478945 |
+-----+-----+-----+-----+-----+
5 rows in set <0.00 sec>

mysql> DELETE FROM StudentDetails WHERE ROLL_NO IN (SELECT ROLL_NO
-> FROM StudentSection WHERE NAME = 'Nagendra');
Query OK, 1 row affected (0.00 sec)

mysql> select * from StudentDetails;
+-----+-----+-----+-----+-----+
| Student_ID | NAME | ROLL_NO | LOCATION | PHONE_NUMBER |
+-----+-----+-----+-----+-----+
| 1000 | Hemanth S R | 101 | Mysore | 9845113337 |
| 2000 | Hemanth S R | 102 | Banglore | 8877665544 |
| 3000 | Sandeep | 103 | Kodagu | 9538945623 |
| 4000 | Sashank Hegde | 104 | Udupi | 8989898989 |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> _
```

## Join vs. Subquery

- JOINS are faster than a subquery and it is very rare that the opposite.
- In JOINS the RDBMS calculates an execution plan, that can predict, what data should be loaded and how much it will take to process and as a result this process save some times, unlike the subquery there is no pre-process calculation and run all the queries and load all their data to do the processing.
- A JOIN is checked conditions first and then put it into table and displays; where as a subquery take separate temp table internally and checking condition.
- When joins are using, there should be connection between two or more than two tables and each table has a relation with other while subquery means query inside another query, has no need to relation, it works on columns and conditions

## VIEWS IN SQL

- A view is a single virtual table that is derived from other tables. The other tables could be base tables or previously defined view.
- Allows for limited update operations Since the table may not physically be stored
- Allows full query operations
- A convenience for expressing certain operations
- A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views.

### **Views syntax**

Let's now look at the basic syntax used to create a view in MySQL.

```
CREATE VIEW `view_name` AS SELECT statement;
```

#### **WHERE**

- "CREATE VIEW "view\_name"" tells MySQL server to create a view object in the database
- "AS SELECT statement" is the SQL statements to be packed in the views. It can be a SELECT statement can contain data from one table or multiple tables.

#### **Example1-Simple View consisting of only one tables**

```
CREATE VIEW VW_BOOKDETAILS AS SELECT BOOK_NAME, BOOK_AUTHOR,
PUBLISHER FROM BOOK_DETAILS;
```

#### **Example of Simple View consisting of two tables and using where condition**

```
CREATE VIEW VW_ORDERS AS SELECT BOOK_NAME, BOOK_AUTHOR,
PUBLISHER FROM ORDERS A, BOOK_DETAILS B WHERE
A.ORDER_ID=B.ORDER_ID;
```

#### **Example of View consisting of inner join**

```
CREATE VIEW VW_BOOK1 AS SELECT A.ORDER_ID, ORDER_DATE, BOOK_NAME,
BOOK_AUTHOR, PUBLISHER FROM ORDERS A INNER JOIN BOOK_DETAILS B ON
A.ORDER_ID=B.ORDER_ID;
```

#### **Example of View consisting of inner join and where condition**

```
CREATE VIEW VW_BOOK2 AS SELECT A.ORDER_ID, ORDER_DATE, BOOK_NAME,
BOOK_AUTHOR, PUBLISHER FROM ORDERS A INNER JOIN BOOK_DETAILS B ON
A.ORDER_ID=B.ORDER_ID AND PUBLISHER='Tata McGraw-Hill'
```

```
SELECT * FROM VW_BOOKDETAILS
SELECT * FROM VW_ORDERS
SELECT * FROM VW_BOOK1
SELECT * FROM VW_BOOK2
```

---

SHOW TABLES

Tables_in_naveen
VW_BOOKDETAILS
VW_ORDERS
VW_BOOK1
VW_BOOK2

### Example2-Simple View

```
CREATE VIEW VW_PUBLICATION AS SELECT PUB_YEAR FROM BOOK;
```

```
SELECT * FROM VW_PUBLICATION
```

```
mysql> SELECT * FROM VW_PUBLICATION;
+-----+
| PUB_YEAR |
+-----+
| JAN-2017 |
| JUN-2016 |
| SEP-2016 |
| SEP-2015 |
| MAY-2016 |
+-----+
5 rows in set (0.00 sec)
```

### DROPING VIEWS

The DROP command can be used to delete a view from the database that is no longer required.

The basic syntax to drop a view is as follows.

```
DROP VIEW VIEWNAME;
```

```
DROP VIEW V_PUBLICATION;
```

#### You may want to use views primarily for following 3 reasons

- Ultimately , you will use your SQL knowledge , to create applications , which will use a database for data requirements. It's recommended that you use VIEWS of the original table structure in your application instead of using the tables themselves. This ensures that when you refactor your DB, your legacy code will see the original schema via the view without breaking the application.
- **VIEWS increase re-usability.** You will not have to create complex queries involving joins repeatedly. All the complexity is converted into a single line of query use VIEWS. Such condensed code will be easier to integrate in your application. This will eliminates chances of typos and your code will be more readable.
- **VIEWS help in data security.** You can use views to show only authorized information to users and hide sensitive data like credit card numbers, pass

## INTRODUCTION TO STORED PROCEDURES

- A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again. So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
- You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.
- A procedure can return one or more than one value through parameters or may not return at all. The procedure can be used in SQL queries.

Creating a procedure

Syntax

```
CREATE PROCEDURE procedure_name
```

```
(
```

```
parameter datatype ,
```

```
parameter datatype
```

```
)
```

```
BEGIN
```

```
    Declaration_section
```

```
    Executable_section
```

```
END;
```

Parameter

**procedure\_name:** name of the stored procedure.

**Parameter:** number of parameters. It can be one or more than one.

**declaration\_section:** all variables are declared.

**executable\_section:** code is written here.

A variable is a named data object whose value can change during the stored procedure execution. We typically use the variables in stored procedures to hold the immediate results. These variables are local to the stored procedure. You must declare a variable before using it.

```
DELIMITER //
```

```
CREATE PROCEDURE sp_name
```

```
(
```

```
    p_1 INT
```

```
)
```

```
BEGIN
```

```
    ...code goes here...
```

```
END //
```

```
DELIMITER ;
```

- Replace procedure\_name with sp\_procedure\_name whatever name you'd like to use for the stored procedure. The parentheses are required — they enclose any parameters. If no parameters are required, the parentheses can be empty.
- The main body of the stored procedure goes in between the BEGIN and END keywords. These keywords are used for writing compound statements. A compound statement can contain multiple statements, and these can be nested if required. Therefore, you can nest BEGIN and END blocks.
- In most cases, you will also need to surround the CREATE PROCEDURE statement with DELIMITER commands and change END; to END //. Like this:

### About the DELIMITER Command

- The first command is DELIMITER // , which is not related to the stored procedure syntax. The DELIMITER statement changes the standard delimiter which is a semicolon ( ; ) to another.
- In this case, the delimiter is changed from the semicolon( ; ) to double-slashes(//) We need to change delimiter from ; to //. Because we want to pass the stored procedure to the server as a whole rather than letting mysql tool interpret each statement at a time.
- Following the END keyword, we use the delimiter // to indicate the end of the stored procedure. The last command ( DELIMITER; )changes the delimiter back to the semicolon (;).

### How to Execute a Stored Procedure

Call sp\_procedure\_name();

### Writing the first MySQL stored procedure

Here we are creating sample table named employee

```
create table employee
(
employee_id int primary key,
Name varchar(50),
Designation varchar(50),
Salary decimal(10,2)
)

insert into employee values(100,'vishwanath','clerk',20000.00);
insert into employee values(101,'shashikiran','instructor',20000.00);
insert into employee values(102,'nitin','assitant professor',25000.00);
insert into employee values(103,'deepak','associate professor',40000.00);
insert into employee values(104,'sanjay','professor',80000.00);
insert into employee values(105,'yogesh','system admin',30000.00);
insert into employee values(106,'anand','clerk',20000.00);
insert into employee values(107,'Hemanth','professor',80000.00);
insert into employee values(108,'Robert','cashier',15000.00);
insert into employee values(109,'amit','clerk',20000.00);
insert into employee values(110,'george','HR Manager',30000.00);
```

```
mysql> select * from employee;
+-----+-----+-----+-----+
| employee_id | Name | Designation | Salary |
+-----+-----+-----+-----+
| 100 | vishwanath | clerk | 20000.00 |
| 101 | shashikiran | instructor | 20000.00 |
| 102 | nitin | assitiant professor | 25000.00 |
| 103 | deepak | associate professor | 40000.00 |
| 104 | sanjay | professor | 80000.00 |
| 105 | yogesh | system admin | 30000.00 |
| 106 | anand | clerk | 20000.00 |
| 107 | Hemanth | professor | 80000.00 |
| 108 | Robert | cashier | 15000.00 |
| 109 | amit | clerk | 20000.00 |
| 110 | george | HR Manager | 30000.00 |
+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

**Example**

We are going to develop a simple stored procedure named SP\_getEmployee to help you get familiar with the syntax. The SP\_getEmployee() stored procedure selects all employee information from the employee table.:.

DELIMITER \$\$

DROP PROCEDURE IF EXISTS SP\_getEmployee \$\$

CREATE PROCEDURE SP\_getEmployee()

BEGIN

SELECT \* FROM employee;

END\$\$

**Execute the stored procedure above as follows:**

call SP\_getEmployee();

```
mysql> call SP_getEmployee();
+-----+-----+-----+-----+
| employee_id | Name | Designation | Salary |
+-----+-----+-----+-----+
| 100 | vishwanath | clerk | 20000.00 |
| 101 | shashikiran | instructor | 20000.00 |
| 102 | nitin | assitiant professor | 25000.00 |
| 103 | deepak | associate professor | 40000.00 |
| 104 | sanjay | professor | 80000.00 |
| 105 | yogesh | system admin | 30000.00 |
| 106 | anand | clerk | 20000.00 |
| 107 | Hemanth | professor | 80000.00 |
| 108 | Robert | cashier | 15000.00 |
| 109 | amit | clerk | 20000.00 |
| 110 | george | HR Manager | 30000.00 |
+-----+-----+-----+-----+
11 rows in set (0.00 sec)

Query OK, 0 rows affected (0.03 sec)

mysql> _
```

### Introduction to MySQL stored procedure parameters

Almost stored procedures that you develop require parameters. The parameters make the stored procedure more flexible and useful.

The syntax of defining a parameter in the stored procedures is as follows:

```
MODE param_name param_type(param_size)
```

**The MODE could be IN , OUT or INOUT , depending on the purpose of the parameter in the stored procedure.**

The param\_name is the name of the parameter. The name of the parameter must follow the naming rules of the column name in MySQL.

Followed the parameter name is its data type and size. Like a variable, the data type of the parameter can be any valid MySQL data type.

Each parameter is separated by a comma (,) if the stored procedure has more than one parameter

**IN** – is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure. In addition, the value of an IN parameter is protected. It means that even the value of the IN parameter is changed inside the stored procedure, its original value is retained after the stored procedure ends. In other words, the stored procedure only works on the copy of the IN parameter.

**OUT** – the value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program. Notice that the stored procedure cannot access the initial value of the OUT parameter when it starts.

**INOUT** – an INOUT parameter is a combination of IN and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter, and pass the new value back to the calling program.

### MySQL Procedure : Parameter IN example

#### Stored Procedure With One Parameter

The following SQL statement creates a stored procedure that selects employee information from a employee Table based on employee\_id from the " SP\_getEmployeeid .:

```
//SP_getEmployeeid
DELIMITER $$

DROP PROCEDURE IF EXISTS SP_getEmployeeid $$

CREATE PROCEDURE SP_getEmployeeid
(
IN emp_id INT(10)
)
BEGIN
```

```
select * FROM employee where employee_id=emp_id;
END$$
```

Execute the stored procedure below..... as follows:

```
mysql> call SP_getEmployeeid(110);
+-----+-----+-----+
| employee_id | Name    | Designation | Salary   |
+-----+-----+-----+
|      110    | george | HR Manager | 30000.00 |
+-----+-----+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> call SP_getEmployeeid(104);
+-----+-----+-----+
| employee_id | Name    | Designation | Salary   |
+-----+-----+-----+
|      104    | sanjay | professor   | 80000.00 |
+-----+-----+-----+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> _
```

### Stored Procedure With Multiple Parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects studentdetails from a particular USN with a particular NAME from the "studentmarks" table:

#### Example

```
DELIMITER $$

DROP PROCEDURE IF EXISTS SP_getStudentdetails $$

CREATE PROCEDURE SP_getStudentdetails
(
IN usn1 varchar(50),
IN name1 varchar(50)
)
BEGIN
select * FROM studentmarks where usn=usn1 and name=name1;
END$$
```

Execute the stored procedure above as follows:

```
mysql> select * from studentmarks;
+-----+-----+-----+-----+-----+
| USN | NAME | PHONE | GENDER | Marks |
+-----+-----+-----+-----+-----+
| 4AD17CS010 | ARVIND | 9900211201 | M | 21 |
| 4AD17CS011 | AJAY | 9845091341 | M | 17 |
| 4AD17CS020 | AKSHAY | 8877881122 | M | 25 |
| 4AD17CS025 | AKSHATHA | 7894737377 | F | 18 |
| 4AD17CS029 | CHANDANA | 7696772121 | F | 16 |
| 4AD17CS032 | BHASKAR | 9923211099 | M | 19 |
| 4AD17CS045 | JEEUAN | 9944850121 | M | 15 |
| 4AD17CS062 | SANDHYA | 7722829912 | F | 24 |
| 4AD17CS066 | VEENA | 877881122 | F | 22 |
| 4AD17CS091 | TARANATH | 7712312312 | M | 23 |
+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)

mysql> call SP_getStudentdetails('4AD17CS010','ARVIND');
+-----+-----+-----+-----+-----+
| USN | NAME | PHONE | GENDER | Marks |
+-----+-----+-----+-----+-----+
| 4AD17CS010 | ARVIND | 9900211201 | M | 21 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

### MySQL Procedure : Parameter OUT example

The following example shows a simple stored procedure that uses an OUT parameter. Within the procedure MySQL MAX() function retrieves maximum salary from MAX\_SALARY of employee . table.

```
DELIMITER $$

DROP PROCEDURE IF EXISTS sp_getemployemaxsalary()$$

CREATE PROCEDURE sp_getemployemaxsalary
(
    out max_salary float(10,2)
)
BEGIN
    SELECT max(Salary) into max_salary from employee;
END$$

DELIMITER ;
```

In the body of the procedure, the parameter max\_salary will get the highest salary from Salary column of Employee Table. After calling the procedure the word OUT tells the MYSQL that the value goes out from the procedure. Here max\_salary is the name of the output parameter and we have passed its value to a session variable named @ m, in the CALL statement.

```
mysql> select * from employee;
+-----+-----+-----+-----+
| employee_id | Name      | Designation | Salary   |
+-----+-----+-----+-----+
|    100 | vishwanath | clerk       | 20000.00 |
|    101 | shashikiran | instructor | 20000.00 |
|    102 | nitin      | assitiant professor | 25000.00 |
|    103 | deepak     | associate professor | 40000.00 |
|    104 | sanjay     | professor   | 80000.00 |
|    105 | yogesh     | system admin | 30000.00 |
|    106 | anand      | clerk       | 20000.00 |
|    107 | Hemanth    | professor   | 80000.00 |
|    108 | Robert     | cashier     | 15000.00 |
|    109 | amit       | clerk       | 20000.00 |
|    110 | george     | HR Manager  | 30000.00 |
|    111 | Jaishankar | Project Manager | 90000.00 |
|    112 | Kiran      | Insurance Manager | 35000.00 |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)

mysql>
```

```
mysql> call sp_getemployemaxsalary(@m);
Query OK, 0 rows affected (0.02 sec)

mysql> select @m;
+-----+
| @m  |
+-----+
| 90000 |
+-----+
1 row in set (0.00 sec)

mysql>
```

### MySQL Procedure : Parameter INOUT example

Here we are creating sample table STUDENT with USN as primary key

CREATE TABLE STUDENT

```
(  
USN VARCHAR (10) PRIMARY KEY,  
SNAME VARCHAR (25),  
ADDRESS VARCHAR (25),  
PHONE BIGINT (10),  
GENDER CHAR (1)  
);
```

Here we are inserting sample data into STUDENT Table

```
INSERT INTO STUDENT VALUES ('4AD16CS020','AKSHAY','BELAGAVI', 8877881122,'M');  
INSERT INTO STUDENT VALUES ('4AD16CS062','SANDHYA','BENGALURU',  
7722829912,'F');  
INSERT INTO STUDENT VALUES ('4AD16CS091','TARANATH','BENGALURU',  
7712312312,'M');  
INSERT INTO STUDENT VALUES ('4AD16CS066','SUPRIYA','MANGALURU',  
8877881122,'F');  
INSERT INTO STUDENT VALUES ('4AD16CS010','ABHAY','BENGALURU', 9900211201,'M');
```

```
INSERT INTO STUDENT VALUES ('4AD16CS032','BHASKAR','BENGALURU',
9923211099,'M');
INSERT INTO STUDENT VALUES ('4AD16CS025','AKSHATHA','BENGALURU',
7894737377,'F');
INSERT INTO STUDENT VALUES ('4AD16CS011','AJAY','TUMKUR', 9845091341,'M');
INSERT INTO STUDENT VALUES ('4AD16CS029','CHITRA','DAVANGERE', 7696772121,'F');
INSERT INTO STUDENT VALUES ('4AD16CS045','JEEVAN','BELLARY', 9944850121,'M');
```

mysql> SELECT * FROM STUDENT;				
USN	SNAME	ADDRESS	PHONE	GENDER
4AD16CS010	ABHAY	BENGALURU	9900211201	M
4AD16CS011	AJAY	TUMKUR	9845091341	M
4AD16CS020	AKSHAY	BELAGAVI	8877881122	M
4AD16CS025	AKSHATHA	BENGALURU	7894737377	F
4AD16CS029	CHITRA	DAVANGERE	7696772121	F
4AD16CS032	BHASKAR	BENGALURU	9923211099	M
4AD16CS045	JEEVAN	BELLARY	9944850121	M
4AD16CS062	SANDHYA	BENGALURU	7722829912	F
4AD16CS066	SUPRIYA	MANGALURU	8877881122	F
4AD16CS091	TARAMATH	BENGALURU	7712312312	M

10 rows in set <0.01 sec>

The following example shows a simple stored procedure that uses an INOUT parameter and an IN parameter. The user will supply 'M' or 'F' through IN parameter to count a number of male or female gender from STUDENT table. The INOUT parameter (countgender) will return the result to a user. Here is the code and output of the procedure

```
DELIMITER $$
```

```
DROP PROCEDURE IF EXISTS sp_studentcountgender $$

create procedure sp_studentcountgender
(
    IN studentgender char(1),
    OUT countgender int
)
begin
    select COUNT(GENDER) INTO countgender FROM STUDENT WHERE
    GENDER=studentgender;
END$$

DELIMITER ;
```

```
mysql> call sp_studentcountgender('M',@CountGender);
Query OK, 0 rows affected (0.00 sec)

mysql> select @CountGender;
+-----+
| @CountGender |
+-----+
|       6      |
+-----+
1 row in set (0.00 sec)

mysql> call sp_studentcountgender('F',@CountGender);
Query OK, 0 rows affected (0.00 sec)

mysql> select @CountGender;
+-----+
| @CountGender |
+-----+
|       4      |
+-----+
1 row in set (0.00 sec)

mysql>
```

Other examples of stored procedure are as follows

**This an example of inserting information into table employee through stored procedure**

**// INSERT STORED PROCEDURE**

DELIMITER \$\$

DROP PROCEDURE IF EXISTS sp\_InsertEmployee \$\$

CREATE PROCEDURE sp\_InsertEmployee

(

IN employee\_id INT(10),

IN Name VARCHAR(255),

IN Designation VARCHAR(255),

IN Salary float(10,2)

)

BEGIN

INSERT INTO employee values(employee\_id ,Name,Designation, Salary);

END\$\$

```

mysql> call sp_InsertEmployee(111,'Jagdish','System Engineer',40000);
Query OK, 1 row affected (0.02 sec)

mysql> call SP_getEmployee();
+-----+-----+-----+-----+
| employee_id | Name      | Designation | Salary   |
+-----+-----+-----+-----+
|    100 | vishwanath | clerk       | 20000.00 |
|    101 | shashikiran | instructor  | 20000.00 |
|    102 | nitin      | assitiant professor | 25000.00 |
|    103 | deepak     | associate professor | 40000.00 |
|    104 | sanjay     | professor   | 80000.00 |
|    105 | yogesh     | system admin | 30000.00 |
|    106 | anand      | clerk       | 20000.00 |
|    107 | Hemanth    | professor   | 80000.00 |
|    108 | Robert     | cashier     | 15000.00 |
|    109 | amit       | clerk       | 20000.00 |
|    110 | george     | HR Manager | 30000.00 |
|    111 | Jagdish    | System Engineer | 40000.00 |
+-----+-----+-----+-----+
12 rows in set (0.01 sec)

Query OK, 0 rows affected (0.06 sec)

mysql> _

```

This an example of updating information into table employee through stored procedure

```

// UPDATE STORED PROCEDURE
DELIMITER $$

DROP PROCEDURE IF EXISTS sp_UpdateEmployee $$

CREATE PROCEDURE sp_UpdateEmployee
(
    IN emp_id INT(10),
    IN Name VARCHAR(255),
    IN Designation VARCHAR(255),
    IN Salary float(10,2)
)
BEGIN
    UPDATE EMPLOYEE SET Name= Name, Designation= Designation, Salary= Salary WHERE
    employee_id = emp_id ;
END$$

```

```

mysql> call sp_getemployee;
+-----+-----+-----+-----+
| employee_id | Name | Designation | Salary |
+-----+-----+-----+-----+
| 100 | vishwanath | clerk | 20000.00 |
| 101 | shashikiran | instructor | 20000.00 |
| 102 | nitin | assitant professor | 25000.00 |
| 103 | deepak | associate professor | 40000.00 |
| 104 | sanjay | professor | 80000.00 |
| 105 | yogesh | system admin | 30000.00 |
| 106 | anand | clerk | 20000.00 |
| 107 | Hemanth | professor | 80000.00 |
| 108 | Robert | cashier | 15000.00 |
| 109 | amit | clerk | 20000.00 |
| 110 | george | HR Manager | 30000.00 |
| 111 | Jagdish | System Engineer | 40000.00 |
+-----+-----+-----+-----+
12 rows in set (0.00 sec)

Query OK, 0 rows affected (0.05 sec)

mysql> call sp_UpdateEmployee(111,'Jaishankar','Project Manager',90000);
Query OK, 1 row affected (0.04 sec)

mysql> call sp_getemployee;
+-----+-----+-----+-----+
| employee_id | Name | Designation | Salary |
+-----+-----+-----+-----+
| 100 | vishwanath | clerk | 20000.00 |
| 101 | shashikiran | instructor | 20000.00 |
| 102 | nitin | assitant professor | 25000.00 |
| 103 | deepak | associate professor | 40000.00 |
| 104 | sanjay | professor | 80000.00 |
| 105 | yogesh | system admin | 30000.00 |
| 106 | anand | clerk | 20000.00 |
| 107 | Hemanth | professor | 80000.00 |
| 108 | Robert | cashier | 15000.00 |
| 109 | amit | clerk | 20000.00 |
| 110 | george | HR Manager | 30000.00 |
| 111 | Jaishankar | Project Manager | 90000.00 |
+-----+-----+-----+-----+
12 rows in set (0.00 sec)

```

```

CREATE PROCEDURE SP_InsertUpdateemployee
(
    IN emp_id int,
    IN Name VARCHAR(255),
    IN Designation VARCHAR(255),
    IN Salary float(10,2)
)
BEGIN
    DECLARE counta int;
    SELECT count(*) INTO counta FROM employee WHERE employee_id=emp_id;
    IF counta=0

```

THEN

```
INSERT INTO employee values(emp_id,Name,Designation,Salary);
```

ELSEIF counta>0

THEN

```
UPDATE EMPLOYEE SET Name=Name,Designation=Designation,Salary=Salary
```

```
WHERE employee_id=emp_id;
```

END IF;

END \$\$

```
mysql> call SP_getEmployee;
+-----+-----+-----+-----+
| employee_id | Name      | Designation | Salary   |
+-----+-----+-----+-----+
|    100 | vishwanath | clerk       | 20000.00 |
|    101 | shashikiran | instructor  | 20000.00 |
|    102 | nitin      | assitant professor | 25000.00 |
|    103 | deepak     | associate professor | 40000.00 |
|    104 | sanjay     | professor   | 80000.00 |
|    105 | yogesh     | system admin | 30000.00 |
|    106 | anand      | clerk       | 20000.00 |
|    107 | Hemanth    | professor   | 80000.00 |
|    108 | Robert     | cashier     | 15000.00 |
|    109 | amit       | clerk       | 20000.00 |
|   110 | george     | HR Manager  | 30000.00 |
|   111 | Jaishankar | Project Manager | 90000.00 |
+-----+-----+-----+-----+
12 rows in set (0.00 sec)

Query OK, 0 rows affected (0.07 sec)

mysql> call sp_InsertUpdateEmployee(112,'Kishore','Insurance Agent',25000);
Query OK, 1 row affected (0.03 sec)

mysql> call SP_getEmployee;
+-----+-----+-----+-----+
| employee_id | Name      | Designation | Salary   |
+-----+-----+-----+-----+
|    100 | vishwanath | clerk       | 20000.00 |
|    101 | shashikiran | instructor  | 20000.00 |
|    102 | nitin      | assitant professor | 25000.00 |
|    103 | deepak     | associate professor | 40000.00 |
|    104 | sanjay     | professor   | 80000.00 |
|    105 | yogesh     | system admin | 30000.00 |
|    106 | anand      | clerk       | 20000.00 |
|    107 | Hemanth    | professor   | 80000.00 |
|    108 | Robert     | cashier     | 15000.00 |
|    109 | amit       | clerk       | 20000.00 |
|   110 | george     | HR Manager  | 30000.00 |
|   111 | Jaishankar | Project Manager | 90000.00 |
|   112 | Kishore    | Insurance Agent | 25000.00 |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)
```

```

mysql> call SP_getEmployee;
+-----+-----+-----+-----+
| employee_id | Name      | Designation | Salary |
+-----+-----+-----+-----+
|    100 | vishwanath | clerk       | 20000.00 |
|    101 | shashikiran | instructor | 20000.00 |
|    102 | nitin      | assitiant professor | 25000.00 |
|    103 | deepak     | associate professor | 40000.00 |
|    104 | sanjay     | professor   | 80000.00 |
|    105 | yogesh     | system admin | 30000.00 |
|    106 | anand      | clerk       | 20000.00 |
|    107 | Hemanth    | professor   | 80000.00 |
|    108 | Robert     | cashier     | 15000.00 |
|    109 | amit       | clerk       | 20000.00 |
|    110 | george     | HR Manager | 30000.00 |
|    111 | Jaishankar | Project Manager | 90000.00 |
|    112 | Kishore    | Insurance Agent | 25000.00 |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)

Query OK, 0 rows affected (0.08 sec)

mysql> call sp_InsertUpdateEmployee(112,'Kiran','Insurance Manager',35000);
Query OK, 1 row affected (0.03 sec)

mysql> call SP_getEmployee;
+-----+-----+-----+-----+
| employee_id | Name      | Designation | Salary |
+-----+-----+-----+-----+
|    100 | vishwanath | clerk       | 20000.00 |
|    101 | shashikiran | instructor | 20000.00 |
|    102 | nitin      | assitiant professor | 25000.00 |
|    103 | deepak     | associate professor | 40000.00 |
|    104 | sanjay     | professor   | 80000.00 |
|    105 | yogesh     | system admin | 30000.00 |
|    106 | anand      | clerk       | 20000.00 |
|    107 | Hemanth    | professor   | 80000.00 |
|    108 | Robert     | cashier     | 15000.00 |
|    109 | amit       | clerk       | 20000.00 |
|    110 | george     | HR Manager | 30000.00 |
|    111 | Jaishankar | Project Manager | 90000.00 |
|    112 | Kiran     | Insurance Manager | 35000.00 |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)

```

An example of deleting record in table named employee by using stored procedure

// DELETE STORED PROCEDURE

DELIMITER \$\$

DROP PROCEDURE IF EXISTS sp\_DeletetEmployee \$\$

CREATE PROCEDURE sp\_DeletetEmployee

(

IN emp\_id INT(10)

)

BEGIN

Delete from employee where employee\_id= emp\_id;

END\$\$

```

mysql> SELECT * FROM EMPLOYEE;
+-----+-----+-----+-----+
| employee_id | Name | Designation | Salary |
+-----+-----+-----+-----+
| 100 | vishwanath | clerk | 20000.00 |
| 101 | shashikiran | instructor | 20000.00 |
| 102 | nitin | assitiant professor | 25000.00 |
| 103 | deepak | associate professor | 40000.00 |
| 104 | sanjay | professor | 80000.00 |
| 105 | yogesh | system admin | 30000.00 |
| 106 | anand | clerk | 20000.00 |
| 107 | Hemanth | professor | 80000.00 |
| 108 | Robert | cashier | 15000.00 |
| 109 | amit | clerk | 20000.00 |
| 110 | george | HR Manager | 30000.00 |
| 111 | Jaishankar | project manager | 90000.00 |
| 112 | arun | Graphics Engineer | 35000.00 |
| 113 | Frank | Finance Manager | 40000.00 |
+-----+-----+-----+-----+
14 rows in set (0.00 sec)

mysql> CALL sp_DeleteEmployee(113);
Query OK, 1 row affected (0.13 sec)

mysql> SELECT * FROM EMPLOYEE;
+-----+-----+-----+-----+
| employee_id | Name | Designation | Salary |
+-----+-----+-----+-----+
| 100 | vishwanath | clerk | 20000.00 |
| 101 | shashikiran | instructor | 20000.00 |
| 102 | nitin | assitiant professor | 25000.00 |
| 103 | deepak | associate professor | 40000.00 |
| 104 | sanjay | professor | 80000.00 |
| 105 | yogesh | system admin | 30000.00 |
| 106 | anand | clerk | 20000.00 |
| 107 | Hemanth | professor | 80000.00 |
| 108 | Robert | cashier | 15000.00 |
| 109 | amit | clerk | 20000.00 |
| 110 | george | HR Manager | 30000.00 |
| 111 | Jaishankar | project manager | 90000.00 |
| 112 | arun | Graphics Engineer | 35000.00 |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)

mysql> _

```

### Variables scope

A variable has its own scope that defines its lifetime. If you declare a variable inside a stored procedure, it will be out of scope when the END statement of stored procedure reaches.

If you declare a variable inside BEGIN END block, it will be out of scope if the END is reached. You can declare two or more variables with the same name in different scopes because a variable is only effective in its own scope. However, declaring variables with the same name in different scopes is not good programming practice.

A variable whose name begins with the @ sign is a session variable. It is available and accessible until the session ends.

**Declare a Variable:**

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

To provide a default value for a variable, include a DEFAULT clause. The value can be specified as an expression; it need not be constant. If the DEFAULT clause is missing, the initial value is NULL.

**Example1: Local variables**

Local variables are declared within stored procedures and are only valid within the BEGIN...END block where they are declared. Local variables can have any SQL data type. The following example shows the use of local variables in a stored procedure.

Here below is an example of stored procedure which is used to insert as well both update contents of employee table

```
DELIMITER $$
```

```
DROP PROCEDURE IF EXISTS SP_InsertUpdateemployee $$
```

```
CREATE PROCEDURE SP_InsertUpdateemployee
```

```
(
```

```
IN emp_id int,
```

```
IN Name VARCHAR(255),
```

```
IN Designation VARCHAR(255),
```

```
IN Salary float(10,2)
```

```
)
```

```
BEGIN
```

```
DECLARE counta int;
```

```
SELECT count(*) INTO counta FROM employee WHERE employee_id=emp_id;
```

```
IF counta=0
```

```
THEN
```

```
INSERT INTO employee values(emp_id,Name,Designation,Salary);
```

```
ELSEIF counta>0
```

```
THEN
```

```
UPDATE EMPLOYEE SET Name=name,Designation=Designation,Salary=Salary
```

```
WHERE employee_id=emp_id;
```

```
END IF;
```

```
END $$
```

## TO DROP STORED PROCEDURE

Once you have created your procedure in MySQL, you might find that you need to remove it from the database. This statement is used to drop a stored procedure .

DROP procedure IF EXISTS procedure\_name;

**procedure\_name:** The name of the procedure that you wish to drop.'

### Example of Droping Stored Procedure

drop procedure IF EXISTS SP\_getcustomerid;

or

drop procedure SP\_getcustomerid;

### MySQL stored procedures advantages

- **Reduce Network Traffic :** Stored procedures help reduce the traffic between application and database server because instead of sending multiple lengthy SQL statements, the application has to send only the name and parameters of the stored procedure.
- **Faster Query Execution :** Since stored procedures are Parsed, Compiled at once, and the executable is cached in the Database. Therefore if same query is repeated multiple times then Database directly executes the executable and hence Time is saved in Parse,Compile etc.
- **Secure:** MySQL stored procedures are secure because the database administrator can grant appropriate permissions to applications that access stored procedures in the database without giving any permissions on the underlying database table

### MySQL stored procedures disadvantages

- Stored procedure's constructs are not designed for developing complex and flexible business logic.
- It is difficult to debug stored procedures. Only a few database management systems allow you to debug stored procedures. Unfortunately, MySQL does not provide facilities for debugging stored procedures.
- Memory usage increased: If we use many stored procedures, the memory usage of every connection that is using those stored procedures will increase substantially.

## INTRODUCTION TO MYSQL TRIGGER

- A trigger is a set of SQL Statements that are run automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a specified table.
- A SQL trigger is a special type of stored procedure. It is special because it is not called directly like a stored procedure. The main difference between a trigger and a stored procedure is that a trigger is called automatically when a data modification event is made against a table whereas a stored procedure must be called explicitly.
- A trigger can be defined to be invoked either before or after the data is changed by INSERT, UPDATE or DELETE statement
- Triggers are useful for tasks such as enforcing business rules, validating input data, and keeping an audit trail. A trigger can be set to activate either before or after the trigger event. For example, you can have a trigger activate before each row that is inserted into a table or after each row that is updated
- It is important to understand the SQL trigger's advantages and disadvantages so that you can use it appropriately. In the following sections, we will discuss the advantages and disadvantages of using SQL triggers.

### **Advantages of using SQL triggers**

- SQL triggers provide an alternative way to check the integrity of data.
- SQL triggers can catch errors in business logic in the database layer.
- SQL triggers provide an alternative way to run scheduled tasks. By using SQL triggers, you don't have to wait to run the scheduled tasks because the triggers are invoked automatically before or after a change is made to the data in the tables.
- SQL triggers are very useful to audit the changes of data in tables.

### **Disadvantages of using SQL triggers**

- SQL triggers only can provide an extended validation and they cannot replace all the validations. Some simple validations have to be done in the application layer. For example, you can validate user's inputs in the client side by using JavaScript or on the server side using server-side scripting languages such as JSP, PHP, ASP.NET, Perl.
- SQL triggers are invoked and executed invisible from the client applications, therefore, it is difficult to figure out what happens in the database layer.
- SQL triggers may increase the overhead of the database server.

In MySQL, trigger can also be created. There are 6 type of triggers that can be made they are:-

- 1.After/Before insert
- 2.After/Before update
- 3.After/Before delete

```
CREATE TRIGGER trigger_name
trigger_event ON table_name
FOR EACH ROW
BEGIN
...
END;
Here,
```

- Trigger\_name is the name of the trigger which must be put after CREATE TRIGGER statement.
- The naming convention for trigger\_name can be like [trigger time]\_[table name]\_[trigger event]. For example, before\_student\_update or after\_student\_insert can be a name of the trigger.
- Trigger\_time is the time of trigger activation and it can be BEFORE or AFTER. We must have to specify the activation time while defining a trigger. We must to use BEFORE if we want to process action prior to the change made on the table and AFTER if we want to process action post to the change made on the table.
- Trigger\_event can be INSERT, UPDATE or DELETE. This event causes the trigger to be invoked. A trigger only can be invoked by one event. To define a trigger that is invoked by multiple events, we have to define multiple triggers, one for each event.
- Table\_name is the name of the table. Actually, a trigger is always associated with a specific table. Without a table, a trigger would not exist hence we have to specify the table name after the 'ON' keyword.
- BEGIN...END is the block in which we will define the logic for the trigger.

### AFTER/BEFORE INSERT TRIGGER

```
CREATE TRIGGER trigger_name
AFTER/BEFORE INSERT ON table_name
FOR EACH ROW
BEGIN
--variable declarations
--trigger code
END;
```

**Parameter:**

- trigger\_name: name of the trigger to be created.
- AFTER/BEFORE INSERT: It points the trigger after or before insert query is executed.
- table\_name: name of the table in which a trigger is created.

### **AFTER/ BEFORE UPDATE Trigger**

- In MySQL, AFTER/BEFORE UPDATE trigger can also be created.
- AFTER/BEFORE UPDATE trigger means trigger will invoke after/before the record is updated.

Syntax:

```
CREATE TRIGGER trigger_name
    AFTER/BEFORE UPDATE ON table_name
FOR EACH ROW
    BEGIN
        --variable declarations
        --trigger code
    END;
```

Parameter:

- trigger\_name: name of the trigger to be created.
- AFTER UPDATE: It points the trigger update query is executed.
- table\_name: name of the table in which a trigger is created.

### **AFTER/BEFORE DELETE Trigger**

- In MySQL, AFTER/BEFORE DELETE trigger can also be created.
- AFTER/BEFORE DELETE trigger means trigger will invoke after/before the record is deleted.

Syntax:

```
CREATE TRIGGER trigger_name
    AFTER/BEFORE DELETE ON table_name
FOR EACH ROW
    BEGIN
        --variable declarations
        --trigger code
    END;
```

Parameter:

- trigger\_name: name of the trigger to be created.
- AFTER/BEFORE DELETE: It points the trigger after/before delete query is executed.
- table\_name: name of the table in which a trigger is created.

## AUDIT TRACKING

### EXAMPLES OF TRIGGERS FOR AUDIT PURPOSE

```
CREATE TABLE Employee_Details
(
    Emp_ID int primary key ,
    Emp_Name varchar(55),
    Emp_Sal decimal (10,2)
);
```

```
CREATE TABLE Employee_Details_Audit
(
    Emp_ID int,
    Emp_Name varchar(55),
    Emp_Sal decimal (10,2),
    Action varchar(55)
);
```

Above we have two scripts for creating two table Employee\_Details and Employee\_Details\_Audit. Both tables have same no of column names,same column name and same date type.In second table, Employee\_Details\_Audit keeps track of what kind of operations are performed on table Employee\_Details and any insert ,update and delete operation value are stored in Employee\_Details\_Audit

For the First table Employee\_Details we insert five records .

```
Insert into Employee_Details values (1000,'Amit',10000);
Insert into Employee_Details values (1001,'Hemanth',12000);
Insert into Employee_Details values (1002,'George',20000);
Insert into Employee_Details values (1003,'Nitin',30000);
Insert into Employee_Details values (1004,'Riyaz',40000);
```

```
mysql> select * from Employee_Details;
+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal |
+-----+-----+-----+
| 1000 | Amit | 10000.00 |
| 1001 | Hemanth | 12000.00 |
| 1002 | George | 20000.00 |
| 1003 | Nitin | 30000.00 |
| 1004 | Riyaz | 40000.00 |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

Below we have written three types of trigger meant for auditing purpose.what ever operations performed on table Employee\_Details gets automatically reflected in Employee\_Details\_Audit.For example we perform any insert,update or delete operations on Employee\_Details, its get automatically get reflected in Employee\_Details\_Audit table because we have written triggers on Employee\_Details table.

### **TRIGGER EXAMPLE 1**

#### **EXAMPLE OF CREATING AFTER INSERT TRIGGER FOR AUDIT PURPOSE**

```
CREATE TRIGGER TriggerAfterInsert
AFTER INSERT ON Employee_Details
FOR EACH ROW
insert into Employee_Details_Audit
values(new.Emp_ID,new.Emp_Name,new.Emp_Sal,'INSERT')
```

In above example,TriggerAfterInsert,an Insert Trigger written on table Employee\_Details.whenever we perform insert operation on Employee\_Details table, automatically TriggerAfterInsert is fired and whatever records we have inserted in Employee\_Details the same records gets inserted in Employee\_Details\_Audit along with what action we performed for example Insert action we have performed that too get inserted in Employee\_Details\_Audit.

```
mysql> select * from employee_details;
+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal |
+-----+-----+-----+
| 1000  | Amit     | 10000.00 |
| 1001  | Hemanth   | 12000.00 |
| 1002  | George    | 20000.00 |
| 1003  | Nitin    | 30000.00 |
| 1004  | Riyaz    | 40000.00 |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from employee_details_audit;
Empty set (0.00 sec)
```

Below is an screen shot of output of TriggerAfterInsert fired on table Employee\_Details

```

mysql> CREATE TRIGGER TriggerAfterInsert
-> AFTER INSERT ON Employee_Details
-> FOR EACH ROW
-> insert into Employee_Details_Audit
-> values(new.Emp_ID,new.Emp_Name,new.Emp_Sal,'INSERT')
-> ;
Query OK, 0 rows affected (0.05 sec)

mysql> insert into employee_details values(1005,'Sanjay',50000);
Query OK, 1 row affected (0.02 sec)

mysql> select * from employee_details;
+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal |
+-----+-----+-----+
| 1000   | Amit     | 10000.00 |
| 1001   | Hemanth  | 12000.00 |
| 1002   | George   | 20000.00 |
| 1003   | Nitin    | 30000.00 |
| 1004   | Riyaz    | 40000.00 |
| 1005   | Sanjay   | 50000.00 |
+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> select * from employee_details_audit;
+-----+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal | Action |
+-----+-----+-----+-----+
| 1005   | Sanjay   | 50000.00 | INSERT |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> -

```

## TRIGGER EXAMPLE 2

### EXAMPLE OF CREATING AFTER UPDATE TRIGGER FOR AUDIT PURPOSE

```

create trigger TriggerAfterUpdate
AFTER UPDATE ON Employee_Details
FOR EACH ROW
insert into Employee_Details_Audit
values(new.Emp_ID,new.Emp_Name,new.Emp_Sal,'UPDATE')

```

In above example, TriggerAfterUpdate, an Update Trigger written on table Employee\_Details. whenever we perform Update operation on Employee\_Details table, automatically TriggerAfterUpdate is fired and whatever records we have updated in Employee\_Details same records gets updated in Employee\_Details\_Audit along with what action we performed for example Update action we have performed that too get inserted in Employee\_Details\_Audit

```

mysql> select * from Employee_Details;
+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal |
+-----+-----+-----+
| 1000  | Amit     | 10000.00 |
| 1001  | Hemanth  | 12000.00 |
| 1002  | George   | 20000.00 |
| 1003  | Nitin    | 30000.00 |
| 1004  | Riyaz    | 40000.00 |
| 1005  | Sanjay   | 50000.00 |
+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> select * from Employee_Details_Audit;
+-----+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal | Action |
+-----+-----+-----+-----+
| 1005  | Sanjay   | 50000.00 | INSERT |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

Below is an screen shot of output of TriggerAfterUpdate fired on table Employee\_Details

```

mysql> create trigger TriggerAfterUpdate
-> AFTER UPDATE ON Employee_Details
-> FOR EACH ROW
-> insert into Employee_Details_Audit
-> values(new.Emp_ID,new.Emp_Name,new.Emp_Sal,'UPDATE')
->
->;
Query OK, 0 rows affected (0.05 sec)

mysql> update employee_details set Emp_Name='Ajay',Emp_Sal=50000 where Emp_ID=1000;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM employee_details;
+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal |
+-----+-----+-----+
| 1000  | Ajay     | 50000.00 |
| 1001  | Hemanth  | 12000.00 |
| 1002  | George   | 20000.00 |
| 1003  | Nitin    | 30000.00 |
| 1004  | Riyaz    | 40000.00 |
| 1005  | Sanjay   | 50000.00 |
+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> SELECT * FROM employee_details_audit;
+-----+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal | Action |
+-----+-----+-----+-----+
| 1005  | Sanjay   | 50000.00 | INSERT |
| 1000  | Ajay     | 50000.00 | UPDATE |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> _

```

**TRIGGER EXAMPLE 3****EXAMPLE OF CREATING AFTER DELETE TRIGGER FOR AUDIT PURPOSE**

```
create trigger TriggerAfterDelete
AFTER DELETE ON Employee_Details
FOR EACH ROW
insert into Employee_Details_Audit
values(old.Emp_ID, old.Emp_Name, old.Emp_Sal,'DELETE')
```

```
mysql> SELECT * FROM employee_details;
+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal |
+-----+-----+-----+
| 1000  | Ajay     | 50000.00 |
| 1001  | Hemanth   | 12000.00 |
| 1002  | George    | 20000.00 |
| 1003  | Nitin    | 30000.00 |
| 1004  | Riyaz    | 40000.00 |
| 1005  | Sanjay   | 50000.00 |
+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> SELECT * FROM employee_details_audit;
+-----+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal | Action  |
+-----+-----+-----+-----+
| 1005  | Sanjay   | 50000.00 | INSERT  |
| 1000  | Ajay     | 50000.00 | UPDATE  |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> _
```

```
mysql> create trigger TriggerAfterDelete
-> AFTER DELETE ON Employee_Details
-> FOR EACH ROW
-> insert into Employee_Details_Audit
-> values(old.Emp_ID, old.Emp_Name, old.Emp_Sal,'DELETE');
Query OK, 0 rows affected (0.05 sec)

mysql> select * from employee_details;
+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal |
+-----+-----+-----+
| 1000  | Ajay     | 50000.00 |
| 1001  | Hemanth   | 12000.00 |
| 1002  | George    | 20000.00 |
| 1003  | Nitin    | 30000.00 |
| 1004  | Riyaz    | 40000.00 |
| 1005  | Sanjay   | 50000.00 |
+-----+-----+-----+
6 rows in set (0.00 sec)
```

Below is an screen shot of output of TriggerAfterDelete fired on table Employee\_Details

```
mysql> delete from employee_details where Emp_ID=1005;
Query OK, 1 row affected (0.00 sec)

mysql> select * from employee_details;
+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal |
+-----+-----+-----+
| 1000  | Ajay     | 50000.00 |
| 1001  | Hemanth   | 12000.00 |
| 1002  | George    | 20000.00 |
| 1003  | Nitin    | 30000.00 |
| 1004  | Riyaz    | 40000.00 |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from employee_details_audit;
+-----+-----+-----+-----+
| Emp_ID | Emp_Name | Emp_Sal | Action |
+-----+-----+-----+-----+
| 1005  | Sanjay   | 50000.00 | INSERT |
| 1000  | Ajay     | 50000.00 | UPDATE |
| 1005  | Sanjay   | 50000.00 | DELETE |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

**Write a Trigger To Calculate the FinalIA (average of best two test marks) and update the corresponding table for all students By Using Trigger.**

```
CREATE TABLE IAMARKS
(
USN VARCHAR (10) PRIMARY KEY,
SUBCODE VARCHAR (8),
SSID VARCHAR (5),
TEST1 INT (2),
TEST2 INT (2),
TEST3 INT (2),
FINALIA INT (2)
);
```

Above is script for creating for IAMARKS table. Here is table named IAMARKS where we enter Marks of TEST1, TEST2 and TEST3 but FINALIA is calculated by taking best of two test marks divide by two.

Below is structure of IAMARKS Table.

```
mysql> DESC IAMARKS;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| USN  | varchar(10) | NO  | PRI  | NULL   |       |
| SUBCODE | varchar(8) | YES |       | NULL   |       |
| SSID | varchar(5) | YES |       | NULL   |       |
| TEST1 | int(2) | YES |       | NULL   |       |
| TEST2 | int(2) | YES |       | NULL   |       |
| TEST3 | int(2) | YES |       | NULL   |       |
| FINALIA | int(2) | YES |       | NULL   |       |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.01 sec)

mysql> SELECT * FROM IAMARKS;
Empty set (0.00 sec)

mysql> _
```

**TRIGGER EXAMPLE 4**

This is trigger is an example of Before Insert Trigger

```
DELIMITER $$  
DROP trigger IF EXISTS trg_insertbefore_IAMARKS $$  
CREATE TRIGGER trg_insertbefore_IAMARKS  
BEFORE INSERT ON IAMARKS  
FOR EACH ROW  
BEGIN  
SET NEW.FINALIA=GREATEST(NEW.TEST1+NEW.TEST2,NEW.TEST2+NEW.TEST3,  
NEW.TEST1+NEW.TEST3)/2;  
END$$
```

**Below is insert script for Student Info and marks into table IAMARKS**

Please note that we are inserting 0 marks into table IAMARKS containing columnname FINALIA.

```
INSERT INTO IAMARKS VALUES ('4AD18CS002','15CS51','CSE5A', 25, 16, 24,0);  
INSERT INTO IAMARKS VALUES ('4AD18CS072','15CS52','CSE5B', 22, 19, 14,0);  
INSERT INTO IAMARKS VALUES ('4AD18CS091','15CS53','CSE5C', 19, 15, 20,0);  
INSERT INTO IAMARKS VALUES ('4AD18CS011','15CS54','CSE5A', 20, 16, 19,0);  
INSERT INTO IAMARKS VALUES ('4AD18CS075','15CS55','CSE5B', 19, 18, 22,0);  
INSERT INTO IAMARKS VALUES ('4AD18CS095','15CS56','CSE5C', 23, 17, 24,0);
```

Above ,we have written trigger name before\_IAMARKS\_insert on IAMARKS table which automatically calculates FINALIA by using Greatest function and inserts into values into FINALIA column of table IAMARKS while trying to insert record on IAMARKS

```
mysql> DELIMITER $$  
mysql> DROP trigger IF EXISTS trg_insertbefore_IAMARKS $$  
Query OK, 0 rows affected, 1 warning (0.00 sec)  
  
mysql> CREATE TRIGGER trg_insertbefore_IAMARKS  
-> BEFORE INSERT ON IAMARKS  
-> FOR EACH ROW  
-> BEGIN  
-> SET NEW.FINALIA=GREATEST(NEW.TEST1+NEW.TEST2,NEW.TEST2+NEW.TEST3,NEW.TEST  
1+NEW.TEST3)/2;  
-> END$$  
Query OK, 0 rows affected (0.03 sec)  
  
mysql>
```

Below is snap shot of output generated by trigger before\_IAMARKS\_insert on table IAMARKS

```

mysql> INSERT INTO IAMARKS VALUES ('4AD18CS002','18CS51','CSE5A', 25, 16, 24,0);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO IAMARKS VALUES ('4AD18CS072','18CS52','CSE5B', 22, 19, 14,0);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO IAMARKS VALUES ('4AD18CS091','18CS53','CSE5C', 19, 15, 20,0);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO IAMARKS VALUES ('4AD18CS011','18CS54','CSE5A', 20, 16, 19,0);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO IAMARKS VALUES ('4AD18CS075','18CS55','CSE5B', 19, 18, 22,0);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO IAMARKS VALUES ('4AD18CS095','18CS56','CSE5C', 23, 17, 24,0);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM IAMARKS;
+-----+-----+-----+-----+-----+-----+-----+
| USN | SUBCODE | SSID | TEST1 | TEST2 | TEST3 | FINALIA |
+-----+-----+-----+-----+-----+-----+-----+
| 4AD18CS002 | 18CS51 | CSE5A | 25 | 16 | 24 | 25 |
| 4AD18CS072 | 18CS52 | CSE5B | 22 | 19 | 14 | 21 |
| 4AD18CS091 | 18CS53 | CSE5C | 19 | 15 | 20 | 20 |
| 4AD18CS011 | 18CS54 | CSE5A | 20 | 16 | 19 | 20 |
| 4AD18CS075 | 18CS55 | CSE5B | 19 | 18 | 22 | 21 |
| 4AD18CS095 | 18CS56 | CSE5C | 23 | 17 | 24 | 24 |
+-----+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> _

```

## OTHER TRIGGER EXAMPLES

Example Of Creating Before Insert Trigger Having The If Conditional Statements for incrementing salary of employee based on certain condition

### TRIGGER EXAMPLE 5

DELIMITER \$\$

DROP trigger IF EXISTS increment\_Salary\_Employee \$\$

CREATE TRIGGER increment\_Salary\_Employee

BEFORE insert ON employee

FOR EACH ROW

IF NEW.Salary>20000 and NEW.Salary<40000

THEN

SET NEW.salary=NEW.salary\*1.1;

ELSEIF NEW.Salary>=40000 and NEW.Salary<90000

THEN

SET NEW.Salary=NEW.Salary\*1.2;

END IF;

END\$\$

Above trigger increment\_Salary\_Employee is written on table employee.this trigger checks salary of employee table during insertion if salary is greater than 20000 and salary is lesser than 40000,it automatically increments salary by 10%.if salary is greater than or equal to 40000 and salary is lesser than 90000,salary is automatically incremented by 20%

Here is snapshot of output of trigger increment\_Salary\_Employee fired on Employee table when any records is inserted into table Employee

```
mysql> select * from employee;
+-----+-----+-----+-----+
| employee_id | Name | Designation | Salary |
+-----+-----+-----+-----+
| 100 | harsha | clerk | 15000.00 |
| 101 | shashikiran | instructor | 19000.00 |
| 102 | nitin | assitant professor | 25000.00 |
| 103 | deepak | associate professor | 40000.00 |
| 104 | sanjay | professor | 80000.00 |
| 105 | yogesh | system admin | 30000.00 |
| 106 | anand | clerk | 15000.00 |
| 107 | ARUN | TEST ENGINEER | 25000.00 |
| 108 | robert | hr manager | 35000.00 |
| 109 | amit | salesman | 40000.00 |
| 110 | kumar | salesmanager | 42000.00 |
+-----+-----+-----+-----+
11 rows in set (0.00 sec)

mysql> insert into employee values (111,'ashok','doctor',70000);
Query OK, 1 row affected (0.03 sec)

mysql> insert into employee values (112,'sunil','test engineer',30000);
Query OK, 1 row affected (0.02 sec)

mysql> select * from employee;
+-----+-----+-----+-----+
| employee_id | Name | Designation | Salary |
+-----+-----+-----+-----+
| 100 | harsha | clerk | 15000.00 |
| 101 | shashikiran | instructor | 19000.00 |
| 102 | nitin | assitant professor | 25000.00 |
| 103 | deepak | associate professor | 40000.00 |
| 104 | sanjay | professor | 80000.00 |
| 105 | yogesh | system admin | 30000.00 |
| 106 | anand | clerk | 15000.00 |
| 107 | ARUN | TEST ENGINEER | 25000.00 |
| 108 | robert | hr manager | 35000.00 |
| 109 | amit | salesman | 40000.00 |
| 110 | kumar | salesmanager | 42000.00 |
| 111 | ashok | doctor | 84000.00 |
| 112 | sunil | test engineer | 33000.00 |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)
```

### TRIGGER EXAMPLE 6

```
DELIMITER $$

DROP trigger IF EXISTS trg_insertbefore_trimupper $$

CREATE TRIGGER trg_insertbefore_trimupper
BEFORE INSERT ON employee
FOR EACH ROW
BEGIN
    SET NEW.Name = UPPER(NEW.Name);
    SET NEW.Designation = UPPER(NEW.Designation);
END IF;
END$$
```

Above trigger name trg\_insertbefore\_trimupper is create on table employee which automatically converts Name and Designation columns in table employee into upper case letters and inserts automatically into Table employee when any employee record is inserted in employee table.

Here is example of snapshot of output of trigger trg\_insertbefore\_trimupper

```
mysql> select * from employee;
+-----+-----+-----+
| employee_id | Name      | Designation | Salary |
+-----+-----+-----+
|    100 | harsha    | clerk       | 15000.00 |
|    101 | shashikiran | instructor | 19000.00 |
|    102 | nitin     | assitant professor | 25000.00 |
|    103 | deepak    | associate professor | 40000.00 |
|    104 | sanjay    | professor   | 80000.00 |
|    105 | yogesh    | system admin | 30000.00 |
|    106 | anand     | clerk       | 15000.00 |
+-----+-----+-----+
7 rows in set <0.00 sec>

mysql> insert into employee values(107,'deepak','system engineer',25000);
Query OK, 1 row affected <0.01 sec>

mysql> select * from employee;
+-----+-----+-----+
| employee_id | Name      | Designation | Salary |
+-----+-----+-----+
|    100 | harsha    | clerk       | 15000.00 |
|    101 | shashikiran | instructor | 19000.00 |
|    102 | nitin     | assitant professor | 25000.00 |
|    103 | deepak    | associate professor | 40000.00 |
|    104 | sanjay    | professor   | 80000.00 |
|    105 | yogesh    | system admin | 30000.00 |
|    106 | anand     | clerk       | 15000.00 |
|    107 | DEEPAK    | SYSTEM ENGINEER | 25000.00 |
+-----+-----+-----+
8 rows in set <0.00 sec>

mysql>
```

Below is screen shot for creating simple table people which has 2 columns age and name

```
mysql> use atme;
Database changed
mysql> CREATE TABLE people
  -> (
  -> age INT,
  -> name varchar(150)
  -> );
Query OK, 0 rows affected <0.05 sec>

mysql> insert into people values(20,'ajay');
Query OK, 1 row affected <0.02 sec>

mysql> insert into people values(50,'sanjay');
Query OK, 1 row affected <0.00 sec>

mysql> select * from people;
+-----+-----+
| age  | name   |
+-----+-----+
| 20   | ajay   |
| 50   | sanjay |
+-----+-----+
2 rows in set <0.00 sec>

mysql>
```

**TRIGGER EXAMPLE 7**

```

mysql> DELIMITER $$  

mysql> DROP trigger IF EXISTS beforeinsert_agecheck$$  

Query OK, 0 rows affected, 1 warning (0.00 sec)  
  

mysql> CREATE TRIGGER beforeinsert_agecheck  

-> BEFORE INSERT ON people  

-> FOR EACH ROW  

-> IF NEW.age < 0  

-> THEN SET NEW.age = 0;  

-> ELSEIF NEW.age > 100 THEN  

-> SET NEW.age = 60;  

-> END IF;  

-> END$$  

Query OK, 0 rows affected (0.05 sec)

```

Above trigger beforeinsert\_agecheck is created on table people which checks age being inserted into Table people if age is less than 0 or negative values ,it automatically insert age value has 0.if we are trying to insert into table people suppose age value is greater than 100,then automatically inserts 60 as default value in age column of people table.

Here is example of snapshot of output of beforeinsert\_agecheck

```

mysql> select * from people;  

+-----+-----+
| age | name |
+-----+-----+
| 20 | ajay |
| 50 | sanjay |
+-----+-----+
2 rows in set (0.00 sec)  
  

mysql> insert into people values(-70,'hemanth');  

Query OK, 1 row affected (0.00 sec)  
  

mysql> select * from people;  

+-----+-----+
| age | name |
+-----+-----+
| 20 | ajay |
| 50 | sanjay |
| 0 | hemanth |
+-----+-----+
3 rows in set (0.00 sec)  
  

mysql> insert into people values(1000,'nitin');  

Query OK, 1 row affected (0.00 sec)  
  

mysql> select * from people;  

+-----+-----+
| age | name |
+-----+-----+
| 20 | ajay |
| 50 | sanjay |
| 60 | nitin |
| 0 | hemanth |
+-----+-----+
4 rows in set (0.00 sec)  

mysql>

```

**TRIGGER EXAMPLE 8**

This is trigger is an example of Before Update Trigger

```
mysql>
mysql> DELIMITER $$ 
mysql> DROP trigger IF EXISTS beforeupdate_agecheck$$
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TRIGGER beforeupdate_agecheck
-> BEFORE update ON people
-> FOR EACH ROW
-> IF NEW.age < 0
-> THEN SET NEW.age = 0;
-> ELSEIF NEW.age > 100 THEN
-> SET NEW.age = 60;
-> END IF;
-> END$$
Query OK, 0 rows affected (0.03 sec)
```

Above trigger beforeupdate\_agecheck is created on table people which checks age being updated into Table people if age is less than 0 or negative values ,it automatically updates age value has 0. suppose if we are trying to update age value is greater than 100,then automatically updates 60 in age column of people table.

Below is snap shot of output of trigger beforeupdate\_agecheck

```
mysql> select * from people;
+-----+-----+
| age | name |
+-----+-----+
| 20 | ajay |
| 50 | sanjay |
| 60 | nitin |
| 0 | hemanth |
+-----+
4 rows in set (0.00 sec)

mysql> update people set age=1000 where name='hemanth';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from people;
+-----+-----+
| age | name |
+-----+-----+
| 20 | ajay |
| 50 | sanjay |
| 60 | nitin |
| 60 | hemanth |
+-----+
4 rows in set (0.00 sec)
```

```

mysql> update people set age=-55 where name='nitin';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from people;
+-----+-----+
| age | name |
+-----+-----+
| 20  | ajay  |
| 50  | sanjay |
| 0   | nitin |
| 60  | hemanth|
+-----+-----+
4 rows in set (0.01 sec)

mysql>

```

## How to DROP TRIGGER

**DROP TRIGGER IF EXISTS schema\_name.trigger\_name;**

In this syntax:

- First, specify the name of the trigger that you want to drop after the **DROP TRIGGER** keywords.
- Second, specify the name of the schema to which the trigger belongs. If you skip the schema name, the statement will drop the trigger in the current database.
- Third, use **IF EXISTS** option to conditionally drops the trigger if the trigger exists. The **IF EXISTS** clause is optional.

**DROP TRIGGER IF EXISTS employee.trg\_insertbefore\_IAMARKS;**

Or

**DROP TRIGGER trg\_insertbefore\_IAMARKS;**

Above statement drops trigger **trg\_insertbefore\_IAMARKS** created on table IAMARKS

**Note** that if you drop a table, MySQL will automatically drop all triggers associated with the table

**LAB EXPERIMENTS**  
**PART A: SQL PROGRAMMING**

A. Consider the following schema for a Library Database:

**BOOK (Book\_id, Title, Publisher\_Name, Pub\_Year)**

**BOOK\_AUTHORS (Book\_id, Author\_Name)**

**PUBLISHER (Name, Address, Phone)**

**BOOK\_COPIES(Book\_id, Programme\_id, No-of\_Copies)**

**BOOK\_LENDING(Book\_id, Programme\_id, Card\_No, Date\_Out, Due\_Date)**

**LIBRARY\_PROGRAMME(Programme\_id, Programme\_Name,Address)**

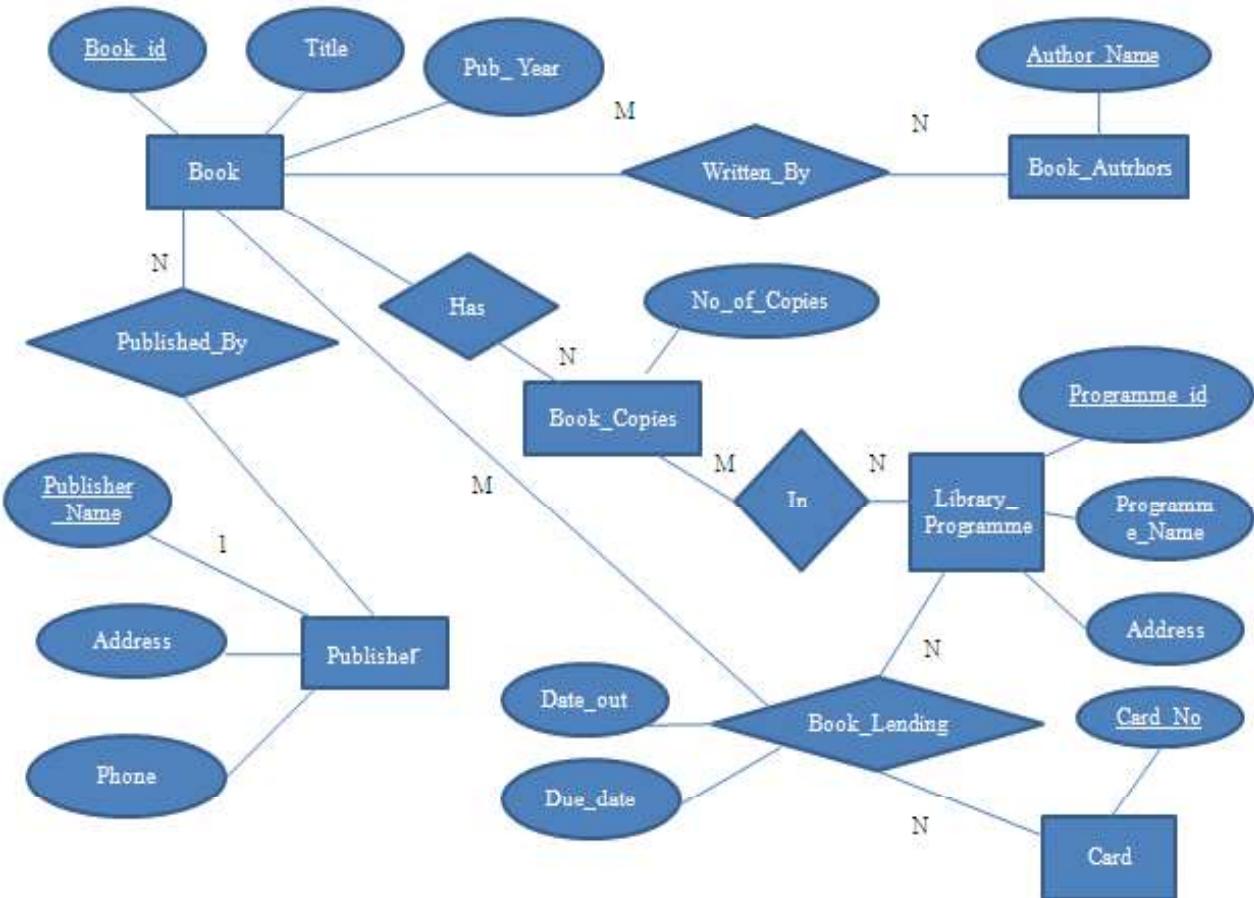
Write SQL queries to

1. Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each Programme, etc.
2. Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun 2017
3. Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.
4. Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.
5. Create a view of all books and its number of copies that are currently available in the Library.

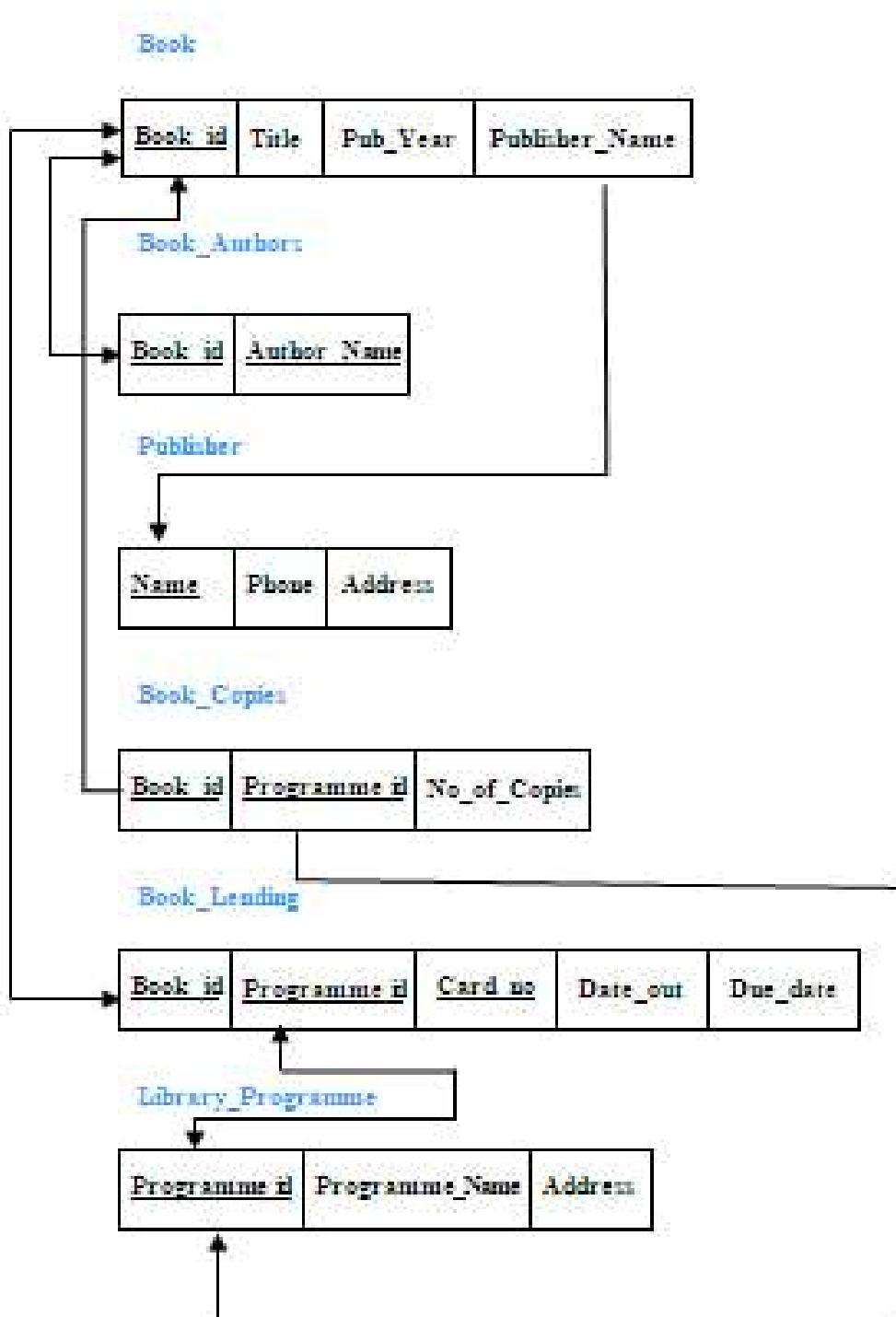
**Program Objectives:**

**This course will enable students to**

- Foundation knowledge in database concepts, technology and practice to groom students into well-informed database application developers.
- Strong practice in SQL programming through a variety of database problems.
- Develop database applications using front-end tools and back-end DBMS.

**Solution:****Entity-Relationship Diagram**

## Schema Diagram



### Table Creation

```
CREATE TABLE BOOK (
BOOK_ID INT (10) PRIMARY KEY,
TITLE VARCHAR (20),
PUB_YEAR VARCHAR (20),
PUBLISHER_NAME VARCHAR (20),
FOREIGN KEY (PUBLISHER_NAME) REFERENCES PUBLISHER (NAME) ON DELETE CASCADE);
```

```
CREATE TABLE BOOK_AUTHORS (
AUTHOR_NAME VARCHAR (20),
BOOK_ID INT (10),
PRIMARY KEY (BOOK_ID, AUTHOR_NAME),
FOREIGN KEY (BOOK_ID) REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE);
```

```
CREATE TABLE PUBLISHER (
NAME VARCHAR (20) PRIMARY KEY,
PHONE BIGINT (20),
ADDRESS VARCHAR (100));
```

```
CREATE TABLE BOOK_COPIES (
NO_OF_COPIES INT (5),
BOOK_ID INT (10),
PROGRAMME_ID INT (10),
PRIMARY KEY (BOOK_ID, PROGRAMME_ID),
FOREIGN KEY (BOOK_ID) REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE,
FOREIGN KEY (PROGRAMME_ID) REFERENCES LIBRARY_PROGRAMME
(PROGRAMME_ID) ON DELETE CASCADE);
```

```
CREATE TABLE BOOK_LENDING (
DATE_OUT DATE,
DUE_DATE DATE,
BOOK_ID INT (10),
PROGRAMME_ID INT (10),
CARD_NO INT (10),
PRIMARY KEY (BOOK_ID, PROGRAMME_ID, CARD_NO),
FOREIGN KEY (BOOK_ID) REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE,
FOREIGN KEY (PROGRAMME_ID) REFERENCES
LIBRARY_PROGRAMME (PROGRAMME_ID) ON DELETE CASCADE,
FOREIGN KEY (CARD_NO) REFERENCES CARD (CARD_NO) ON DELETE CASCADE);
```

```
CREATE TABLE CARD (
CARD_NO INT (10) PRIMARY KEY);
```

```
CREATE TABLE LIBRARY_PROGRAMME (
PROGRAMME_ID INT (10) PRIMARY KEY,
PROGRAMME_NAME VARCHAR (50),
ADDRESS VARCHAR (100));
```

### Table Descriptions

DESC BOOK

```
; mysql> DESC BOOK;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| BOOK_ID | int<10> | NO | PRI | NULL | |
| TITLE | varchar<20> | YES | | NULL | |
| PUB_YEAR | varchar<20> | YES | | NULL | |
| PUBLISHER_NAME | varchar<20> | YES | MUL | NULL | |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

DESC BOOK\_AUTHORS;

```
; mysql> DESC BOOK_AUTHORS ;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| AUTHOR_NAME | varchar<20> | NO | PRI | NULL | |
| BOOK_ID | int<10> | NO | PRI | 0 | |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

DESC PUBLISHER;

```
; mysql> DESC PUBLISHER;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| NAME | varchar<20> | NO | PRI | NULL | |
| PHONE | bigint<20> | YES | | NULL | |
| ADDRESS | varchar<100> | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

DESC BOOK\_COPIES

```
; mysql> DESC BOOK_COPIES;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| NO_OF_COPIES | int<5> | YES | | NULL | |
| BOOK_ID | int<10> | NO | PRI | NULL | |
| PROGRAMME_ID | int<10> | NO | PRI | NULL | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```

```
DESC BOOK_LENDING;
```

```
mysql> DESC BOOK_LENDING;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| DATE_OUT | date | YES |   | NULL |   |
| DUE_DATE | date | YES |   | NULL |   |
| BOOK_ID | int(10) | NO | PRI | NULL |   |
| PROGRAMME_ID | int(10) | NO | PRI | NULL |   |
| CARD_NO | int(10) | NO | PRI | NULL |   |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.03 sec)

mysql>
```

```
DESC CARD;
```

```
mysql> DESC CARD;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CARD_NO | int(10) | NO | PRI | NULL |   |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> _
```

```
DESC LIBRARY_PROGRAMME
```

```
mysql> DESC LIBRARY_PROGRAMME;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| PROGRAMME_ID | int(10) | NO | PRI | NULL |   |
| PROGRAMME_NAME | varchar(50) | YES |   | NULL |   |
| ADDRESS | varchar(100) | YES |   | NULL |   |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```

**Insertion of Values to Tables**

```
INSERT INTO BOOK VALUES (1,'DBMS','JAN-2017', 'MCGRAW-HILL');
INSERT INTO BOOK VALUES (2,'ADBMS','JUN-2016','MCGRAW-HILL');
INSERT INTO BOOK VALUES (3, 'CD','SEP-2016','PEARSON');
INSERT INTO BOOK VALUES (4, 'ALGORITHMS ','SEP-2015',' MIT');
INSERT INTO BOOK VALUES (5,'OS','MAY-2016','PEARSON');
```

```
INSERT INTO BOOK_AUTHORS VALUES ('NAVATHE', 1);
INSERT INTO BOOK_AUTHORS VALUES ('NAVATHE', 2);
INSERT INTO BOOK_AUTHORS VALUES ('ULLMAN',3);
INSERT INTO BOOK_AUTHORS VALUES ('CHARLES', 4);
INSERT INTO BOOK_AUTHORS VALUES('GALVIN', 5);
```

```
INSERT INTO PUBLISHER VALUES ('MCGRAW-HILL', 9989076587,'BANGALORE');
INSERT INTO PUBLISHER VALUES ('PEARSON', 9889076565,'NEWDELHI');
INSERT INTO PUBLISHER VALUES ('PRENTICE HALL', 7455679345,'HYEDRABAD');
INSERT INTO PUBLISHER VALUES ('WILEY', 8970862340,'CHENNAI');
INSERT INTO PUBLISHER VALUES ('MIT',7756120238,'BANGALORE');
```

```
INSERT INTO BOOK_COPIES VALUES (10, 1, 10);
INSERT INTO BOOK_COPIES VALUES (5, 1, 11);
INSERT INTO BOOK_COPIES VALUES (2, 2, 12);
INSERT INTO BOOK_COPIES VALUES (5, 2, 13);
INSERT INTO BOOK_COPIES VALUES (7, 3, 14);
INSERT INTO BOOK_COPIES VALUES (1, 5, 10);
INSERT INTO BOOK_COPIES VALUES (3, 4, 11);
```

```
INSERT INTO BOOK_LENDING VALUES ('2017-01-01','2017-06-01', 1, 10, 101);
INSERT INTO BOOK_LENDING VALUES ('2017-01-11 ','2017-03-11', 3, 14, 101);
INSERT INTO BOOK_LENDING VALUES ('2017-02-21','2017-04-21', 2, 13, 101);
INSERT INTO BOOK_LENDING VALUES ('2017-03-15 ','2017-07-15', 4, 11, 101);
INSERT INTO BOOK_LENDING VALUES ('2017-04-12','2017-05-12', 1, 11, 104);
```

```
INSERT INTO CARD VALUES (100);
INSERT INTO CARD VALUES (101);
INSERT INTO CARD VALUES (102);
INSERT INTO CARD VALUES (103);
INSERT INTO CARD VALUES (104);
```

```
INSERT INTO LIBRARY_PROGRAMME VALUES (10,'VIJAY NAGAR','MYSURU');
INSERT INTO LIBRARY_PROGRAMME VALUES (11,'VIDYANAGAR','HUBLI'); ;
INSERT INTO LIBRARY_PROGRAMME VALUES(12,'KUVEMPUNAGAR','MYSURU');
INSERT INTO LIBRARY_PROGRAMME VALUE(13,'RAJAJINAGAR','BANGALORE');
INSERT INTO LIBRARY_PROGRAMME VALUES (14,'MANIPAL','UDUPI');
```

SELECT \* FROM BOOK;

BOOK_ID	TITLE	PUB_YEAR	PUBLISHER_NAME
1	DBMS	Jan-2017	MCGRAW-HILL
2	ADBMS	Jun-2017	MCGRAW-HILL
3	CD	Sep-2016	PEARSON
4	ALGORITHMS	Sep-2015	MIT
5	OS	May-2016	PEARSON

SELECT \* FROM BOOK\_AUTHORS;

AUTHOR_NAME	BOOK_ID
NAVATHE	1
NAVATHE	2
ULLMAN	3
CHARLES	4
GALVIN	5

SELECT \* FROM PUBLISHER;

NAME	PHONE	ADDRESS
MCGRAW-HILL	9989076587	BANGALORE
MIT	7756120238	BANGALORE
PEARSON	9889076565	NEWDELHI
PRENTICE HALL	7455679345	HYEDRABAD
WILEY	8970862340	CHENNAI

SELECT \* FROM BOOK\_COPIES;

NO_OF_COPIES	BOOK_ID	PROGRAMME_ID
10	1	10
5	1	11
2	2	12
5	2	13
7	3	14
1	5	10
3	4	11

---

SELECT \* FROM BOOK\_LENDING;

DATEOUT	DUEDATE	BOOKID	PROGRAMME_ID	CARDNO
2017-01-01	2017-06-01	1	10	
2017-01-11	2017-03-11	3	4	101
2017-02-21	2017-04-21	2	13	101
2017-03-15	2017-07-15	4	11	101
2017-04-12	2017-05-12	1	11	104

SELECT \* FROM CARD;

CARDNO
101
102
103
104
105

SELECT \* FROM LIBRARY\_PROGRAMME;

PROGRAMME_ID	PROGRAMME_NAME	ADDRESS
10	VIJAY NAGAR	MYSURU
11	VIDYANAGAR	HUBLI
12	KUVEMPUNAGAR	MYSURU
13	RAJAJINAGAR	BANGALORE
14	MANIPAL	UDUPI

**Queries:**

- 1. Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each branch, etc.**

```
SELECT B.BOOK_ID, B.TITLE, B.PUBLISHER_NAME, A.AUTHOR_NAME,
C.NO_OF_COPIES, L.PROGRAMME_ID FROM BOOK B, BOOK_AUTHORS A, BOOK_COPIES
C, LIBRARY_PROGRAMME L WHERE B.BOOK_ID=A.BOOK_ID AND
B.BOOK_ID=C.BOOK_ID AND L.PROGRAMME_ID=C.PROGRAMME_ID;
```

BOOK_ID	TITLE	PUBLISHER_NAME	AUTHOR_NAME	NO_OF_COPIES	PROGRAMME_ID
1	DBMS	MCGRAW-HILL	NAVATHE	10	10
1	DBMS	MCGRAW-HILL	NAVATHE	5	11
2	ADBMS	MCGRAW-HILL	NAVATHE	2	12
2	ADBMS	MCGRAW-HILL	NAVATHE	5	13
3	CD	PEARSON	ULLMAN	7	14
4	ALGORITHMS	MIT	CHARLES	1	11
5	OS	PEARSON	GALVIN	3	10

- 2. Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun 2017.**

```
SELECT CARD_NO FROM BOOK_LENDING WHERE DATE_OUT
BETWEEN '2017-01-01'AND '2017-07-01' GROUP BY CARD_NO
HAVING COUNT(*)>3;
```

```
+-----+
| CARD_NO |
+-----+
| 101 |
+-----+
1 row in set (0.03 sec)

mysql> _
```

- 3. Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.**

```
DELETE FROM BOOK WHERE BOOK_ID=3;
```

```
mysql> SELECT * FROM BOOK;
+-----+-----+-----+-----+
| BOOK_ID | TITLE | PUB_YEAR | PUBLISHER_NAME |
+-----+-----+-----+-----+
| 1 | DBMS | JAN-2017 | MCGRAW-HILL |
| 2 | ADBMS | JUN-2016 | MCGRAW-HILL |
| 3 | CD | SEP-2016 | PEARSON |
| 4 | ALGORITHMS | SEP-2015 | MIT |
| 5 | OS | MAY-2016 | PEARSON |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> DELETE FROM BOOK WHERE BOOK_ID=3;
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM BOOK;
+-----+-----+-----+-----+
| BOOK_ID | TITLE | PUB_YEAR | PUBLISHER_NAME |
+-----+-----+-----+-----+
| 1 | DBMS | JAN-2017 | MCGRAW-HILL |
| 2 | ADBMS | JUN-2016 | MCGRAW-HILL |
| 4 | ALGORITHMS | SEP-2015 | MIT |
| 5 | OS | MAY-2016 | PEARSON |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

- 4. Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.**

```
CREATE VIEW VW_PUBLICATION AS SELECT PUB_YEAR FROM BOOK;
```

```
SELECT * FROM VW_PUBLICATION
```

```
mysql> SELECT * FROM VW_PUBLICATION;
+-----+
| PUB_YEAR |
+-----+
| JAN-2017 |
| JUN-2016 |
| SEP-2016 |
| SEP-2015 |
| MAY-2016 |
+-----+
5 rows in set (0.00 sec)
```

- 5. Create a view of all books and its number of copies that are currently available in the Library.**

```
CREATE VIEW VW_BOOKS AS SELECT B.BOOK_ID, B.TITLE, C.NO_OF_COPIES
FROM BOOK B, BOOK_COPIES C, LIBRARY_PROGRAMME L WHERE
B.BOOK_ID=C.BOOK_ID AND C.PROGRAMME_ID=L.PROGRAMME_ID;
```

```
SELECT * FROM VW_BOOKS;
```

```
mysql> SHOW TABLES;
+-----+
| Tables_in_library |
+-----+
| book
| book_authors
| book_copies
| book_lending
| card
| library_programme
| publisher
| vw_books
+-----+
8 rows in set <0.00 sec>

mysql> SELECT * FROM VW_BOOKS;
+-----+-----+-----+
| BOOK_ID | TITLE      | NO_OF_COPIES |
+-----+-----+-----+
| 1       | DBMS       | 10           |
| 1       | DBMS       | 5            |
| 2       | ADBMS      | 2            |
| 2       | ADBMS      | 5            |
| 3       | CD          | 7            |
| 5       | OS          | 1            |
| 4       | ALGORITHMS | 3            |
+-----+-----+-----+
7 rows in set <0.00 sec>

mysql> _
```

**Program Outcomes:**

The students are able to

- Create, Update and query on the database.
- Demonstrate the working of different concepts of DBMS
- Implement, analyze and evaluate the project developed for an application.

B. Consider the following schema for Order Database:

**SALESMAN** (Salesman\_id, Name, City, Commission)

**CUSTOMER** (Customer\_id, Cust\_Name, City, Grade, Salesman\_id)

**ORDERS** (Ord\_No, Purchase\_Amt, Ord\_Date, Customer\_id, Salesman\_id)

Write SQL queries to

1. Count the customers with grades above Bangalore's average.
2. Find the name and numbers of all salesmen who had more than one customer.
3. List all salesmen and indicate those who have and don't have customers in their cities (Use UNION operation.)
4. Create a view that finds the salesman who has the customer with the highest order of a day.
5. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.

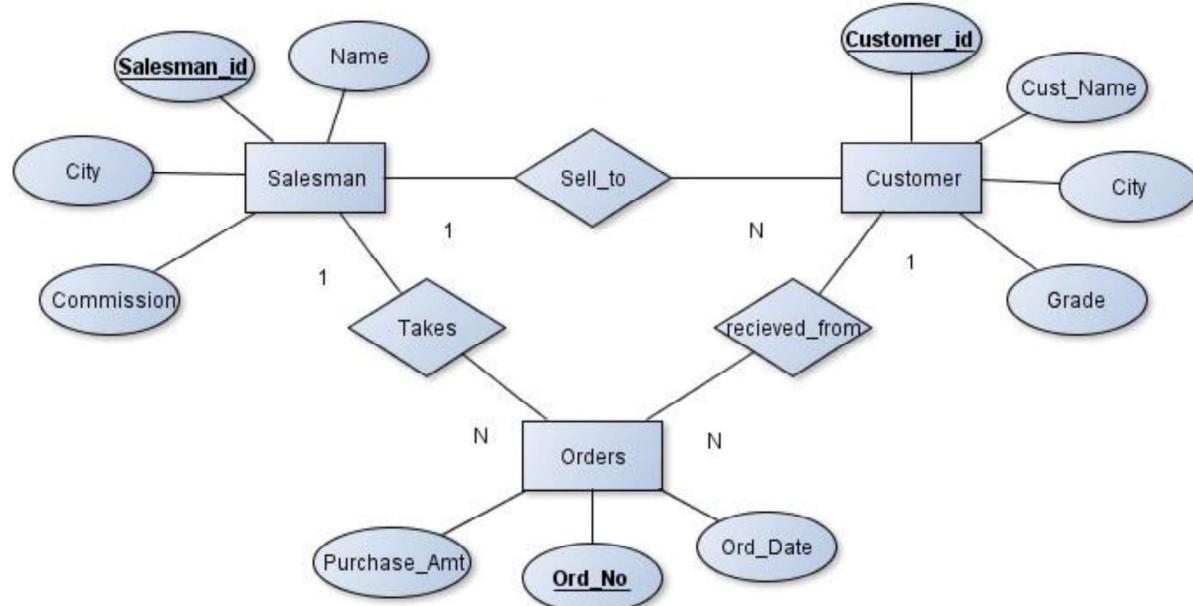
#### **Program Objectives:**

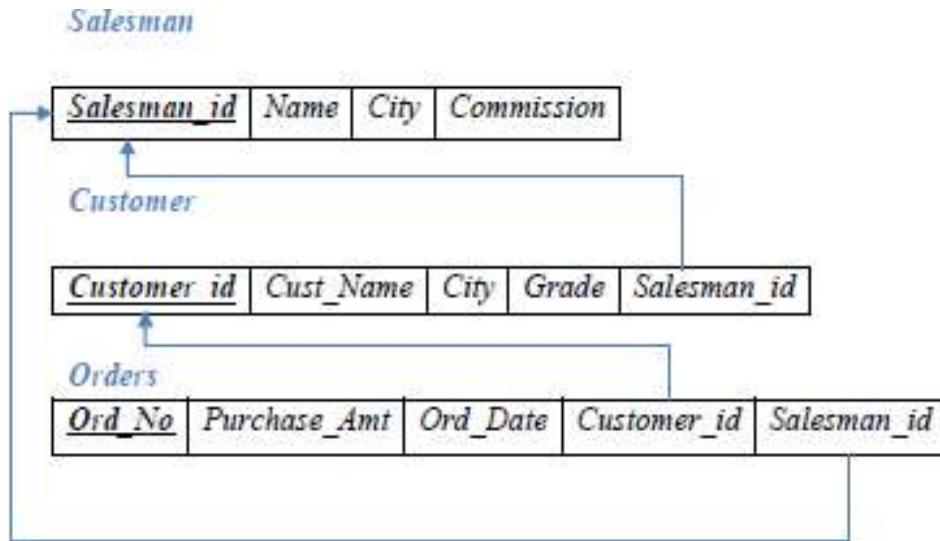
This course will enable students to

- Foundation knowledge in database concepts, technology and practice to groom students into well-informed database application developers.
- Strong practice in SQL programming through a variety of database problems.
- Develop database applications using front-end tools and back-end DBMS.

**Solution:**

**Entity-Relationship Diagram**



**Schema Diagram****Table Creation**

```
CREATE TABLE SALESMAN (
SALESMAN_ID INT (4) PRIMARY KEY,
NAME VARCHAR (20),
CITY VARCHAR (20),
COMMISSION VARCHAR (20));
```

```
CREATE TABLE CUSTOMER (
CUSTOMER_ID INT (5) PRIMARY KEY,
CUST_NAME VARCHAR (20),
CITY VARCHAR (20), GRADE INT (4),
SALESMAN_ID INT (6),
FOREIGN KEY (SALESMAN_ID) REFERENCES SALESMAN (SALESMAN_ID) ON DELETE
SET NULL);
```

```
CREATE TABLE ORDERS (
ORD_NO INT (5) PRIMARY KEY,
PURCHASE_AMT DECIMAL (10, 2),
ORD_DATE DATE,
CUSTOMER_ID INT (4),
SALESMAN_ID INT (4),
FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (CUSTOMER_ID) ON DELETE
CASCADE,
FOREIGN KEY (SALESMAN_ID) REFERENCES SALESMAN (SALESMAN_ID) ON DELETE
CASCADE);
```

### Table Descriptions

DESC SALESMAN;

```
mysql> DESC SALESMAN;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| SALESMAN_ID | int<4> | NO | PRI | NULL |       |
| NAME | varchar<20> | YES |       | NULL |       |
| CITY | varchar<20> | YES |       | NULL |       |
| COMMISSION | varchar<20> | YES |       | NULL |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

DESC CUSTOMER;

```
mysql> DESC CUSTOMER;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CUSTOMER_ID | int<5> | NO | PRI | NULL |       |
| CUST_NAME | varchar<20> | YES |       | NULL |       |
| CITY | varchar<20> | YES |       | NULL |       |
| GRADE | int<4> | YES |       | NULL |       |
| SALESMAN_ID | int<6> | YES | MUL | NULL |       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

DESC ORDERS;

```
mysql> DESC ORDERS;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ORD_NO | int<5> | NO | PRI | NULL |       |
| PURCHASE_AMT | decimal<10,2> | YES |       | NULL |       |
| ORD_DATE | date | YES |       | NULL |       |
| CUSTOMER_ID | int<4> | YES | MUL | NULL |       |
| SALESMAN_ID | int<4> | YES | MUL | NULL |       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

INSERT INTO SALESMAN VALUES(101,'RICHARD','LOS ANGELES','18%');

INSERT INTO SALESMAN VALUES(103,'GEORGE','NEWYORK','32%');

INSERT INTO SALESMAN VALUES(110,'CHARLES','BANGALORE','54%');

INSERT INTO SALESMAN VALUES(122,'ROWLING','PHILADELPHIA','46%');

INSERT INTO SALESMAN VALUES(126,'KURT','CHICAGO','52%');

INSERT INTO SALESMAN VALUES(132,'EDWIN','PHOENIX','41%');

INSERT INTO CUSTOMER VALUES(501,'SMITH','LOS ANGELES',10,103);

INSERT INTO CUSTOMER VALUES(510,'BROWN','ATLANTA',14,122);

INSERT INTO CUSTOMER VALUES(522,'LEWIS','BANGALORE',10,132);

INSERT INTO CUSTOMER VALUES(534,'PHILIPS','BOSTON',17,103);

INSERT INTO CUSTOMER VALUES(543,'EDWARD','BANGALORE',14,110);

INSERT INTO CUSTOMER VALUES(550,'PARKER','ATLANTA',19,126);

```
INSERT INTO ORDERS VALUES(1,1000,'2017-05-04',501,103);
INSERT INTO ORDERS VALUES(2,4000,'2017-01-20',522,132);
INSERT INTO ORDERS VALUES(3,2500,'2017-02-24',550,126);
INSERT INTO ORDERS VALUES(5,6000,'2017-04-13',522,103);
INSERT INTO ORDERS VALUES(6,7000,'2017-03-09',550,126);
INSERT INTO ORDERS VALUES(7,3400,'2017-01-20',501,122);
```

```
SELECT * FROM SALESMAN;
```

mysql> SELECT * FROM SALESMAN;			
SALESMAN_ID	NAME	CITY	COMMISSION
101	RICHARD	LOS ANGELES	18%
103	GEORGE	NEWYORK	32%
110	CHARLES	BANGALORE	54%
122	ROWLING	PHILADELPHIA	46%
126	KURT	CHICAGO	52%
132	EDWIN	PHOENIX	41%

6 rows in set (0.00 sec)

```
SELECT * FROM CUSTOMER;
```

mysql> SELECT * FROM CUSTOMER;				
CUSTOMER_ID	CUST_NAME	CITY	GRADE	SALESMAN_ID
501	SMITH	LOS ANGELES	10	103
510	BROWN	ATLANTA	14	122
522	LEWIS	BANGALORE	10	132
534	PHILIPS	BOSTON	17	103
543	EDWARD	BANGALORE	14	110
550	PARKER	ATLANTA	19	126

6 rows in set (0.00 sec)

```
SELECT * FROM ORDERS;
```

mysql> select * from orders;					
ORD_NO	PURCHASE_AMT	ORD_DATE	CUSTOMER_ID	SALESMAN_ID	
1	1000.00	2017-05-04	501	103	
2	4000.00	2017-01-20	522	132	
3	2500.00	2017-02-24	550	126	
5	6000.00	2017-04-13	522	103	
6	7000.00	2017-03-09	550	126	
7	3400.00	2017-01-20	501	122	

6 rows in set (0.00 sec)

**Queries**

1. Count the customers with grades above Bangalore's average.

```
SELECT GRADE, COUNT(CUSTOMER_ID) FROM
CUSTOMER GROUP BY GRADE
HAVING GRADE > (SELECT AVG(GRADE) FROM
CUSTOMER WHERE CITY='BANGALORE');
```

GRADE	COUNT(DISTINCT CUSTOMER_ID)
14	2
17	1
19	1

3 rows in set (0.03 sec)

2. Find the name and numbers of all salesmen who had more than one customer.

```
SELECT SALESMAN_ID,NAME
FROM SALESMAN A
WHERE 1 <(SELECT COUNT(*) FROM CUSTOMER
WHERE SALESMAN_ID=A.SALESMAN_ID)
```

OR

```
SELECT S.SALESMAN_ID,NAME, FROM CUSTOMER
C,SALESMAN S WHERE
S.SALESMAN_ID=C.SALESMAN_ID GROUP BY
C.SALESMAN_ID HAVING COUNT(*)>1
```

SALESMAN_ID	NAME
103	GEORGE

1 row in set (0.00 sec)

3. List all salesmen and indicate those who have and don't have customers in their cities  
(Use UNION operation.)

```
SELECT S.SALESMAN_ID,NAME,CUST_NAME,COMMISSION FROM SALESMAN
S,CUSTOMER C
WHERE S.CITY = C.CITY
UNION
SELECT SALESMAN_ID, NAME, 'NO MATCH',COMMISSION FROM SALESMAN
WHERE NOT CITY = ANY (SELECT CITY
FROM CUSTOMER) ORDER BY 2 DESC;
```

SALESMAN_ID	NAME	CUST_NAME	COMMISSION
122	ROWLING	NO MATCH	46%
101	RICHARD	SMITH	18%
126	KURT	NO MATCH	52%
103	GEORGE	NO MATCH	32%
132	EDWIN	NO MATCH	41%
110	CHARLES	LEWIS	54%
110	CHARLES	EDWARD	54%

7 rows in set (0.03 sec)

4. Create a view that finds the salesman who has the customer with the highest order of a day.

```
CREATE VIEW VW_ELITSALESMAN AS
SELECT B.ORD_DATE,A.SALESMAN_ID,A.NAME FROM SALESMAN A, ORDERS B
WHERE A.SALESMAN_ID = B.SALESMAN_ID AND B.PURCHASE_AMT=(SELECT
MAX(PURCHASE_AMT) FROM ORDERS C
WHERE C.ORD_DATE = B.ORD_DATE);
```

```
SELECT * FROM VW_ELITSALESMAN
```

mysql> SELECT * FROM VW_ELITSALESMAN;		
ORD_DATE	SALESMAN_ID	NAME
2017-05-04	103	GEORGE
2017-01-20	132	EDWIN
2017-02-24	126	KURT
2017-04-13	103	GEORGE
2017-03-09	126	KURT

5 rows in set (0.00 sec)

5. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.

Use ON DELETE CASCADE at the end of foreign key definitions while creating child table orders and then execute the following:

```
DELETE FROM SALESMAN WHERE SALESMAN_ID=101;
```

mysql> SELECT * FROM SALESMAN;			
SALESMAN_ID	NAME	CITY	COMMISSION
101	RICHARD	LOS ANGELES	18%
103	GEORGE	NEWYORK	32%
110	CHARLES	BANGALORE	54%
122	ROWLING	PHILADELPHIA	46%
126	KURT	CHICAGO	52%
132	EDWIN	PHOENIX	41%

6 rows in set (0.02 sec)

mysql> DELETE FROM SALESMAN WHERE SALESMAN_ID=101;			
Query OK, 1 row affected (0.02 sec)			
mysql> SELECT * FROM SALESMAN;			
SALESMAN_ID	NAME	CITY	COMMISSION
103	GEORGE	NEWYORK	32%
110	CHARLES	BANGALORE	54%
122	ROWLING	PHILADELPHIA	46%
126	KURT	CHICAGO	52%
132	EDWIN	PHOENIX	41%

5 rows in set (0.00 sec)

**Program Outcomes:**

**The students are able to**

- Create, Update and query on the database.
- Demonstrate the working of different concepts of DBMS
- Implement, analyze and evaluate the project developed for an application.

C. Consider the schema for Movie Database:

**ACTOR** (Act\_id, Act\_Name, Act\_Gender)

**DIRECTOR** (Dir\_id, Dir\_Name, Dir\_Phone)

**MOVIES** (Mov\_id, Mov\_Title, Mov\_Year, Mov\_Lang, Dir\_id)

**MOVIE\_CAST** (Act\_id, Mov\_id, Role)

**RATING** (Mov\_id, Rev\_Stars) Write SQL queries to

1. List the titles of all movies directed by ‘Hitchcock’.
2. Find the movie names where one or more actors acted in two or more movies.
3. List all actors who acted in a movie before 2000 and also in a movie after 2015  
(use JOIN operation).
4. Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.
5. Update rating of all movies directed by ‘Steven Spielberg’ to 5.

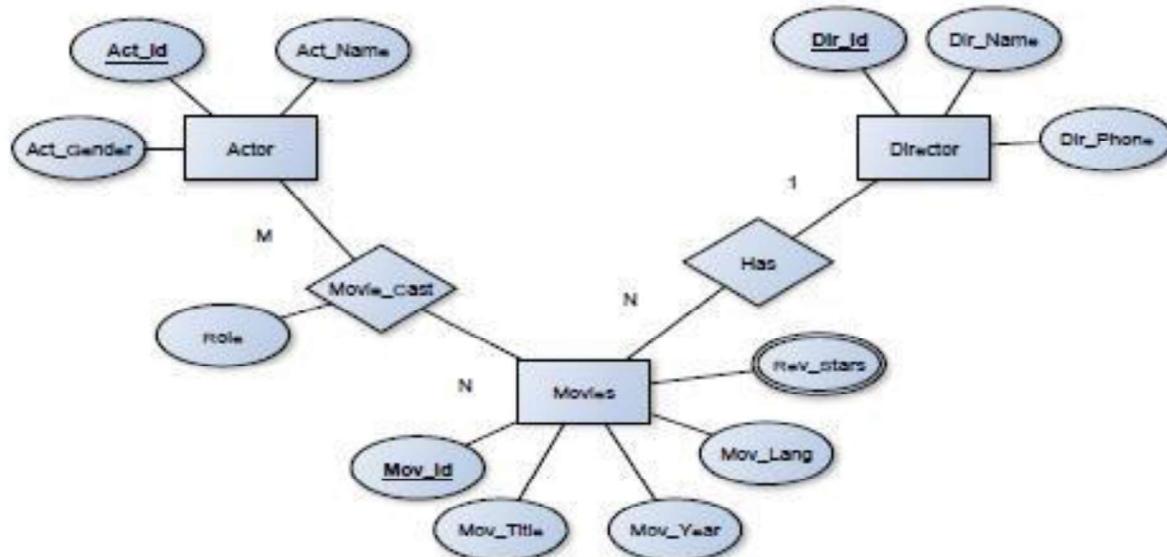
#### **Program Objectives:**

This course will enable students to

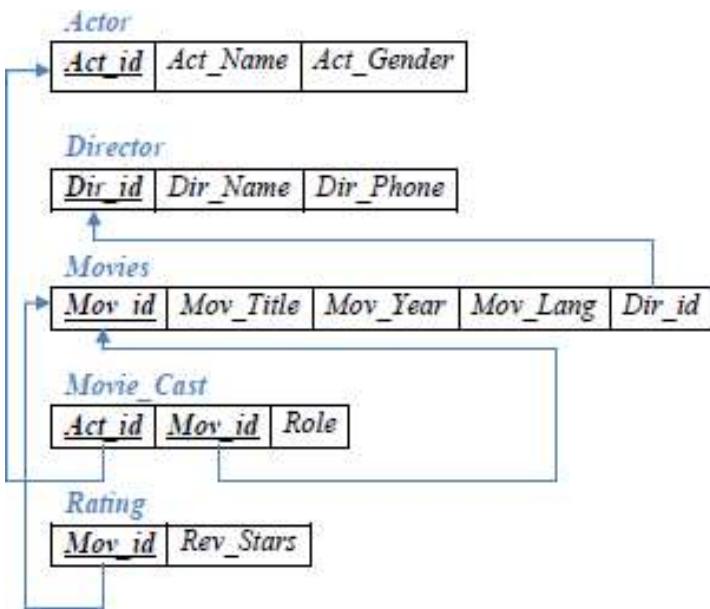
- Foundation knowledge in database concepts, technology and practice to groom students into well-informed database application developers.
- Strong practice in SQL programming through a variety of database problems.
- Develop database applications using front-end tools and back-end DBMS.

**Solution:**

**Entity-Relationship Diagram**



## Schema Diagram



## Table Creation

```
CREATE TABLE ACTOR (
ACT_ID INT (5) PRIMARY KEY,
ACT_NAME VARCHAR (20),
ACT_GENDER CHAR (1));
```

```
CREATE TABLE DIRECTOR (
DIR_ID INT (5) PRIMARY KEY,
DIR_NAME VARCHAR (20),
DIR_PHONE BIGINT);
```

```
CREATE TABLE MOVIES
(MOV_ID INT (4) PRIMARY KEY,
MOV_TITLE VARCHAR (50),
MOV_YEAR INT (4),
MOV_LANG VARCHAR (20),
DIR_ID INT (5),
FOREIGN KEY (DIR_ID) REFERENCES DIRECTOR(DIR_ID));
```

```
CREATE TABLE MOVIES_CAST (
ACT_ID INT (5),
MOV_ID INT (5),
ROLE VARCHAR (20),
PRIMARY KEY (ACT_ID, MOV_ID),
FOREIGN KEY (ACT_ID) REFERENCES ACTOR (ACT_ID),
FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));
```

```
CREATE TABLE RATING (
MOV_ID INT (5) PRIMARY KEY,
REV_STARS VARCHAR (25),
FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));
```

### Table Descriptions

DESC ACTOR;

```
mysql> DESC ACTOR;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ACT_ID | int(5) | NO | PRI | NULL |       |
| ACT_NAME | varchar(20) | YES |       | NULL |       |
| ACT_GENDER | char(1) | YES |       | NULL |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

DESC DIRECTOR;

```
mysql> DESC DIRECTOR;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| DIR_ID | int(5) | NO | PRI | NULL |       |
| DIR_NAME | varchar(20) | YES |       | NULL |       |
| DIR_PHONE | bigint(20) | YES |       | NULL |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

DESC MOVIES;

```
mysql> DESC MOVIES;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| MOU_ID | int(4) | NO | PRI | NULL |       |
| MOU_TITLE | varchar(50) | YES |       | NULL |       |
| MOU_YEAR | int(4) | YES |       | NULL |       |
| MOU_LANG | varchar(20) | YES |       | NULL |       |
| DIR_ID | int(5) | YES | MUL | NULL |       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

DESC MOVIES\_CAST;

```
mysql> DESC MOVIES_CAST;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ACT_ID | int(5) | NO | PRI | 0 |       |
| MOU_ID | int(5) | NO | PRI | 0 |       |
| ROLE | varchar(20) | YES |       | NULL |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

DESC RATING;

```
mysql> DESC RATING;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| MOU_ID | int<5> | NO | PRI | NULL |       |
| REV_STARS | varchar<25> | YES |     | NULL |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

### Insertion of Values to Tables

INSERT INTO ACTOR VALUES (1,'MADHURI DIXIT','F');

INSERT INTO ACTOR VALUES (2,'AAMIR KHAN','M');

INSERT INTO ACTOR VALUES (3,'JUHI CHAWLA','F');

INSERT INTO ACTOR VALUES (4,'SRIDEVI','F');

INSERT INTO DIRECTOR VALUES (100,'SUBHASH KAPOOR', 9563400156);

INSERT INTO DIRECTOR VALUES(102,'ALAN TAYLOR',9971960035);

INSERT INTO DIRECTOR VALUES (103,'SANTHOSH ANANDRAM', 9934611125);

INSERT INTO DIRECTOR VALUES (104,'IMTIAZ ALI', 8539920975);

INSERT INTO DIRECTOR VALUES (105,'HITCHCOCK',7766138911);

INSERT INTO DIRECTOR VALUES (106,'STEVEN SPIELBERG',9966138934);

INSERT INTO MOVIES VALUES (501,'JAB HARRY MET SEJAL',2017,'HINDI',104);

INSERT INTO MOVIES VALUES (502,'RAJAKUMARA',2017,'KANNADA',103);

INSERT INTO MOVIES VALUES (503,'JOLLY LLB 2', 2013,'HINDI', 100);

INSERT INTO MOVIES VALUES (504,'TERMINATOR GENESYS',2015,'ENGLISH',102);

INSERT INTO MOVIES VALUES (505,'JAWS',1975,'ENGLISH',106);

INSERT INTO MOVIES VALUES (506,'BRIDGE OF SPIES',2015,'ENGLISH', 106);

INSERT INTO MOVIES VALUES (507,'VERTIGO',1943,'ENGLISH',105);

INSERT INTO MOVIES VALUES (508,'SHADOW OF A DOUBT',1943,'ENGLISH', 105);

INSERT INTO MOVIES\_CAST VALUES (1, 501,'HEROINE');

INSERT INTO MOVIES\_CAST VALUES (1, 502,'HEROINE');

INSERT INTO MOVIES\_CAST VALUES (3, 503,'COMEDIAN');

INSERT INTO MOVIES\_CAST VALUES (4, 504,'GUEST');

INSERT INTO MOVIES\_CAST VALUES (4, 501,'HERO');

INSERT INTO RATING VALUES (501, 4);

INSERT INTO RATING VALUES (502, 2);

INSERT INTO RATING VALUES (503, 5);

INSERT INTO RATING VALUES (504, 4);

INSERT INTO RATING VALUES (505, 3);

INSERT INTO RATING VALUES (506, 2);

---

SELECT \* FROM ACTOR;

ACT_ID	ACT_NAME	ACT
1	MADHURI DIXIT	F
2	AAMIR KHAN	M
3	JUHI CHAWLA	F
4	SRIDEVI	F

SELECT \* FROM DIRECTOR;

DIR_ID	DIR_NAME	DIR_PHONE
100	SUBHASH KAPOOR	56340015
102	ALAN TAYLOR	719600310
103	SANTHOSH ANANDRAM	99346111
104	IMTIAZ ALI	85399209
105	HITCHCOCK	7766138911
106	STEVEN SPIELBERG	9966138934

SELECT \* FROM MOVIES;

MOV_ID	MOV_TITLE	MOV_YEAR	MOV_LANG	DIR_ID
501	JAB HARRY MET SEJAL	2017	HINDI	104
502	RAJAKUMARA	2017	KANNADA	103
503	JOLLY LLB 2	2013	HINDI	100
504	TERMINATOR GENESYS	2015	ENGLISH	102
505	JAWS	1975	ENGLISH	106
506	BRIDGE OF SPIES	2015	ENGLISH	106
507	VERTIGO	1958	ENGLISH	105
508	SHADOW OF A DOUBT	1943	ENGLISH	105

---

SELECT \* FROM MOVIE\_CAST;

MOV_ID	MOV_TITLE	MOV_YEAR	MOV_LANG	DIR_ID
501	JAB HARRY MET SEJAL	2017	HINDI	104
502	RAJAKUMARA	2017	KANNADA	103
503	JOLLY LLB 2	2013	HINDI	100
504	TERMINATOR GENESYS	2015	ENGLISH	102
505	JAWS	1975	ENGLISH	106
506	BRIDGE OF SPIES	2015	ENGLISH	106
507	VERTIGO	1958	ENGLISH	105
508	SHADOW OF A DOUBT	1943	ENGLISH	105

SELECT \* FROM RATING;

MOV_ID	REV_STARS
501	4
502	2
503	5
504	4
505	3
506	2
507	2
508	4

**Queries:**

1. List the titles of all movies directed by ‘Hitchcock’.

```
SELECT MOV_TITLE FROM MOVIES WHERE DIR_ID IN (SELECT DIR_ID FROM
DIRECTOR WHERE DIR_NAME = 'HITCHCOCK');
```

**OR**

```
SELECT MOV_TITLE FROM MOVIES M, DIRECTOR D WHERE M.DIR_ID=D.DIR_ID
AND DIR_NAME='HITCHCOCK';
```

MOV_TITLE
MOU_TITLE
VERTIGO
SHADOW OF A DOUBT

2 rows in set <0.00 sec>

2. Find the movie names where one or more actors acted in two or more movies.

```
SELECT MOV_TITLE FROM MOVIES M,MOVIES_CAST MV
WHERE M.MOV_ID=MV.MOV_ID AND ACT_ID IN(SELECT ACT_ID FROM
MOVIES_CAST GROUP BY ACT_ID HAVING COUNT(ACT_ID)>1) GROUP BY
MOV_TITLE HAVING COUNT(*)>1;
```

MOV_TITLE
JAB HARRY MET SEJAL

1 row in set <0.00 sec>

3. List all actors who acted in a movie before 2000 and also in a movie after 2015 (use JOIN operation).

```
SELECT ACT_NAME, MOV_TITLE, MOV_YEAR FROM ACTOR A JOIN
MOVIE_CAST C ON A.ACT_ID=C.ACT_ID INNER JOIN MOVIES M
ON C.MOV_ID=M.MOV_ID WHERE M.MOV_YEAR NOT BETWEEN 2000 AND 2015;
```

ACT_NAME	MOV_TITLE	MOV_YEAR
MADHURI DIXIT	JAB HARRY MET SEJAL	2017
MADHURI DIXIT	RAJAKUMARA	2017
Sridevi	JAB HARRY MET SEJAL	2017

3 rows in set <0.00 sec>

4. Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.

```
SELECT MOV_TITLE,MAX(REV_STARS) FROM MOVIES M ,RATING R WHERE
M.MOV_ID=R.MOV_ID GROUP BY MOV_TITLE HAVING MAX(REV_STARS)>0 ORDER
BY MOV_TITLE;
```

MOV_TITLE	MAX(REV_STARS)
BRIDGE OF SPIES	2
JAB HARRY MET SEJAL	4
JAWS	3
JOLLY LLB 2	5
RAJAKUMARA	2
TERMINATOR GENESYS	4

6 rows in set (0.00 sec)

5. Update rating of all movies directed by ‘Steven Spielberg’ to 5

```
UPDATE RATING SET REV_STARS=5 WHERE MOV_ID IN(SELECT MOV_ID FROM
MOVIES WHERE DIR_ID IN(SELECT DIR_ID FROM DIRECTOR
WHERE DIR_NAME='STEVEN SPIELBERG'));
```

OR

```
UPDATE RATING R, MOVIES M, DIRECTOR D SET REV_STARS=5 WHERE
R.MOV_ID=M.MOV_ID AND M.DIR_ID=D.DIR_ID AND DIR_NAME='STEVEN
SPIELBERG';
```

mysql> SELECT * FROM RATING;	
MOV_ID	REV_STARS
501	4
502	2
503	5
504	4
505	5
506	5

6 rows in set (0.00 sec)

#### Program Outcomes:

The students are able to

- Create, Update and query on the database.
- Demonstrate the working of different concepts of DBMS
- Implement, analyze and evaluate the project developed for an application.

**D. Consider the schema for College Database:****STUDENT (USN, SName, Address, Phone, Gender)****SEMSEC (SSID, Sem, Sec)****CLASS (USN, SSID)****SUBJECT (Subcode, Title, Sem, Credits)****IAMARKS (USN, Subcode, SSID, Test1, Test2, Test3, FinalIA)****Write SQL queries to**

1. List all the student details studying in fourth semester ‘C’ section.
2. Compute the total number of male and female students in each semester and in each section.
3. Create a view of Test1 marks of student USN ‘1BI15CS101’ in all subjects.
4. Calculate the FinalIA (average of best two test marks) and update the corresponding table for all students.
5. Categorize students based on the following criterion:

If FinalIA = 17 to 20 then CAT = ‘Outstanding’

If FinalIA = 12 to 16 then CAT = ‘Average’

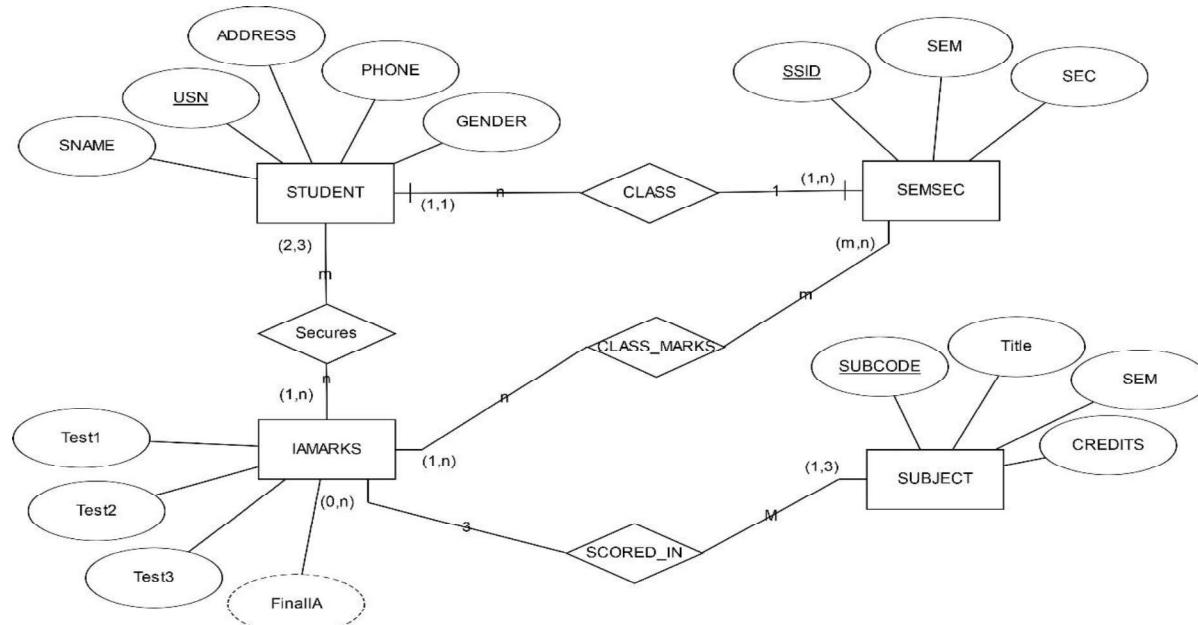
If FinalIA < 12 then CAT = ‘Weak’

Give these details only for 8th semester A, B, and C section students.

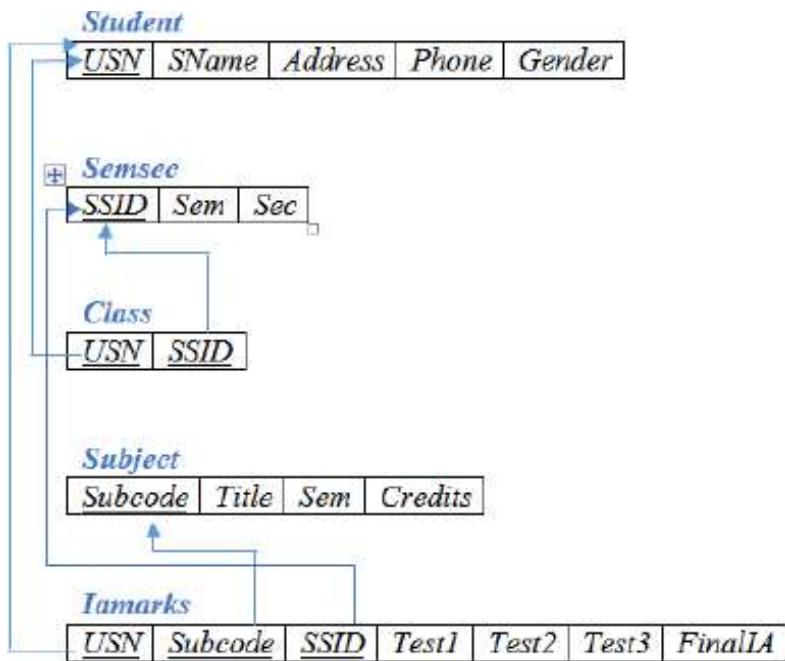
**Program Objectives:**

This course will enable students to

- Foundation knowledge in database concepts, technology and practice to groom students into well-informed database application developers.
- Strong practice in SQL programming through a variety of database problems.
- Develop database applications using front-end tools and back-end DBMS.

**Solution:****Entity - Relationship Diagram**

### Schema Diagram



### Table Creation

```
CREATE TABLE STUDENT (
    USN VARCHAR (10) PRIMARY KEY,
    SNAME VARCHAR (25),
    ADDRESS VARCHAR (25),
    PHONE BIGINT (10),
    GENDER CHAR (1));
```

```
CREATE TABLE SEMSEC (
    SSID VARCHAR (5) PRIMARY KEY,
    SEM INT (5),
    SEC CHAR (1));
```

```
CREATE TABLE CLASS (
    USN VARCHAR (10),
    SSID VARCHAR (5),
    PRIMARY KEY (USN, SSID),
    FOREIGN KEY (USN) REFERENCES STUDENT (USN),
    FOREIGN KEY (SSID) REFERENCES SEMSEC (SSID));
```

```
CREATE TABLE SUBJECT(
    SUBCODE VARCHAR(10)
    PRIMARY KEY,
    TITLE VARCHAR(20),
    SEM INT,
    CREDITS INT);
```

```
CREATE TABLE IAMARKS (
USN VARCHAR (10),
SUBCODE VARCHAR (8),
SSID VARCHAR (5),
TEST1 INT (2),
TEST2 INT (2),
TEST3 INT (2),
FINALIA INT (2),
PRIMARY KEY (USN, SUBCODE, SSID),
FOREIGN KEY (USN) REFERENCES STUDENT (USN),
FOREIGN KEY (SUBCODE) REFERENCES SUBJECT (SUBCODE), FOREIGN
KEY (SSID) REFERENCES SEMSEC (SSID));
```

### Table Descriptions

DESC STUDENT;

```
mysql> DESC STUDENT;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| USN   | varchar(10) | NO   | PRI  | NULL    |       |
| SNAME | varchar(25)  | YES  |       | NULL    |       |
| ADDRESS | varchar(25) | YES  |       | NULL    |       |
| PHONE  | bigint(10)   | YES  |       | NULL    |       |
| GENDER | char(1)      | YES  |       | NULL    |       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

DESC SEMSEC;

```
mysql> DESC SEMSEC;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| SSID  | varchar(5) | NO   | PRI  | NULL    |       |
| SEM   | int(5)    | YES  |       | NULL    |       |
| SEC   | char(1)   | YES  |       | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

DESC CLASS;

```
mysql> DESC CLASS;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| USN   | varchar(10) | NO   | PRI  |       |       |
| SSID  | varchar(5)  | NO   | PRI  |       |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

DESC SUBJECT;

```
mysql> DESC SUBJECT;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| SUBCODE | varchar(10) | NO | PRI | NULL |       |
| TITLE | varchar(20) | YES |       | NULL |       |
| SEM | int(11) | YES |       | NULL |       |
| CREDITS | int(11) | YES |       | NULL |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

DESC IAMARKS;

```
mysql> DESC IAMARKS;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| USN | varchar(10) | NO | PRI |       |       |
| SUBCODE | varchar(8) | NO | PRI |       |       |
| SSID | varchar(5) | NO | PRI |       |       |
| TEST1 | int(2) | YES |       | NULL |       |
| TEST2 | int(2) | YES |       | NULL |       |
| TEST3 | int(2) | YES |       | NULL |       |
| FINALIA | int(2) | YES |       | NULL |       |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

### Insertion of values to tables

INSERT INTO STUDENT VALUES ('4AD13CS020','AKSHAY','BELAGAVI', 8877881122,'M');  
 INSERT INTO STUDENT VALUES ('4AD13CS062','SANDHYA','BENGALURU', 7722829912,'F');  
 INSERT INTO STUDENT VALUES ('4AD13CS091','TEESHA','BENGALURU', 7712312312,'F');  
 INSERT INTO STUDENT VALUES ('4AD13CS066','SUPRIYA','MANGALURU', 8877881122,'F');  
 INSERT INTO STUDENT VALUES ('4AD14CS010','ABHAY','BENGALURU', 9900211201,'M');  
 INSERT INTO STUDENT VALUES ('4AD14CS032','BHASKAR','BENGALURU', 9923211099,'M');  
 INSERT INTO STUDENT VALUES ('4AD14CS025','ASMI','BENGALURU', 7894737377,'F');  
 INSERT INTO STUDENT VALUES ('4AD15CS011','AJAY','TUMKUR', 9845091341,'M');  
 INSERT INTO STUDENT VALUES ('4AD15CS029','CHITRA','DAVANGERE', 7696772121,'F');  
 INSERT INTO STUDENT VALUES ('4AD15CS045','JEEVA','BELLARY', 9944850121,'M');  
 INSERT INTO STUDENT VALUES ('4AD15CS091','SANTOSH','MANGALURU', 8812332201,'M');  
 INSERT INTO STUDENT VALUES ('4AD16CS045','ISMAIL','KABURGI', 9900232201,'M');  
 INSERT INTO STUDENT VALUES ('4AD16CS088','SAMEERA','SHIMOGA', 9905542212,'F');  
 INSERT INTO STUDENT VALUES ('4AD16CS122','VINAYAKA','CHIKAMAGALUR', 8800880011,'M');

```
INSERT INTO SEMSEC VALUES ('CSE8A', 8,'A');
INSERT INTO SEMSEC VALUES ('CSE8B', 8,'B');
INSERT INTO SEMSEC VALUES ('CSE8C', 8,'C');
INSERT INTO SEMSEC VALUES ('CSE7A', 7,'A');
INSERT INTO SEMSEC VALUES ('CSE7B', 7,'B');
INSERT INTO SEMSEC VALUES ('CSE7C', 7,'C');
INSERT INTO SEMSEC VALUES ('CSE6A', 6,'A');
INSERT INTO SEMSEC VALUES ('CSE6B', 6,'B');
INSERT INTO SEMSEC VALUES ('CSE6C', 6,'C');
INSERT INTO SEMSEC VALUES ('CSE5A', 5,'A');
INSERT INTO SEMSEC VALUES ('CSE5B', 5,'B');
INSERT INTO SEMSEC VALUES ('CSE5C', 5,'C');
INSERT INTO SEMSEC VALUES ('CSE4A', 4,'A');
INSERT INTO SEMSEC VALUES ('CSE4B', 4,'B');
INSERT INTO SEMSEC VALUES ('CSE4C', 4,'C');
INSERT INTO SEMSEC VALUES ('CSE3A', 3,'A');
INSERT INTO SEMSEC VALUES ('CSE3B', 3,'B');
INSERT INTO SEMSEC VALUES ('CSE3C', 3,'C');
INSERT INTO SEMSEC VALUES ('CSE2A', 2,'A');
INSERT INTO SEMSEC VALUES ('CSE2B', 2,'B');
INSERT INTO SEMSEC VALUES ('CSE2C', 2,'C');
INSERT INTO SEMSEC VALUES ('CSE1A', 1,'A');
INSERT INTO SEMSEC VALUES ('CSE1B', 1,'B');
INSERT INTO SEMSEC VALUES ('CSE1C', 1,'C');

INSERT INTO CLASS VALUES ('4AD13CS020','CSE8A');
INSERT INTO CLASS VALUES ('4AD13CS062','CSE8A');
INSERT INTO CLASS VALUES ('4AD13CS066','CSE8B');
INSERT INTO CLASS VALUES ('4AD13CS091','CSE8C');
INSERT INTO CLASS VALUES ('4AD14CS010','CSE7A');
INSERT INTO CLASS VALUES ('4AD14CS025','CSE7A');
INSERT INTO CLASS VALUES ('4AD14CS032','CSE7A');
INSERT INTO CLASS VALUES ('4AD15CS011','CSE4A');
INSERT INTO CLASS VALUES ('4AD15CS029','CSE4A');
INSERT INTO CLASS VALUES ('4AD15CS045','CSE4B');
INSERT INTO CLASS VALUES ('4AD15CS091','CSE4C');
INSERT INTO CLASS VALUES ('4AD16CS045','CSE3A');
INSERT INTO CLASS VALUES ('4AD16CS088','CSE3B');
INSERT INTO CLASS VALUES ('4AD16CS122','CSE3C');

INSERT INTO SUBJECT VALUES ('10CS81','ACA', 8, 4);
INSERT INTO SUBJECT VALUES ('10CS82','SSM', 8, 4);
INSERT INTO SUBJECT VALUES ('10CS83','NM', 8, 4);
INSERT INTO SUBJECT VALUES ('10CS84','CC', 8, 4);
INSERT INTO SUBJECT VALUES ('10CS85','PW', 8, 4);
INSERT INTO SUBJECT VALUES ('10CS71','OOAD', 7, 4);
```

```

INSERT INTO SUBJECT VALUES ('10CS72','ECS', 7, 4);
INSERT INTO SUBJECT VALUES ('10CS73','PTW', 7, 4);
INSERT INTO SUBJECT VALUES ('10CS74','DWDM', 7, 4); I
INSERT INTO SUBJECT VALUES ('10CS75','JAVA', 7, 4);
INSERT INTO SUBJECT VALUES ('10CS76','SAN', 7, 4);
INSERT INTO SUBJECT VALUES ('15CS51', 'ME', 5, 4);
INSERT INTO SUBJECT VALUES ('15CS52','CN', 5, 4);
INSERT INTO SUBJECT VALUES ('15CS53','DBMS', 5, 4);
INSERT INTO SUBJECT VALUES ('15CS54','ATC', 5, 4);
INSERT INTO SUBJECT VALUES ('15CS55','JAVA', 5, 3);
INSERT INTO SUBJECT VALUES ('15CS56','AI', 5, 3);
INSERT INTO SUBJECT VALUES ('15CS41','M4', 4, 4);
INSERT INTO SUBJECT VALUES ('15CS42','SE', 4, 4);
INSERT INTO SUBJECT VALUES ('15CS43','DAA', 4, 4);
INSERT INTO SUBJECT VALUES ('15CS44','MPMC', 4, 4);
INSERT INTO SUBJECT VALUES ('15CS45','OOC', 4, 3);
INSERT INTO SUBJECT VALUES ('15CS46','DC', 4, 3);
INSERT INTO SUBJECT VALUES ('15CS31','M3', 3, 4);
INSERT INTO SUBJECT VALUES ('15CS32','ADE', 3, 4);
INSERT INTO SUBJECT VALUES ('15CS33','DSA', 3, 4);
INSERT INTO SUBJECT VALUES ('15CS34','CO', 3, 4);
INSERT INTO SUBJECT VALUES ('15CS35','USP', 3, 3);
INSERT INTO SUBJECT VALUES ('15CS36','DMS', 3, 3);

```

```

INSERT INTO IAMARKS VALUES ('4AD13CS091','10CS81','CSE8C', 15, 16, 18,0);
INSERT INTO IAMARKS VALUES ('4AD13CS091','10CS82','CSE8C', 12, 19, 14,0);
INSERT INTO IAMARKS VALUES ('4AD13CS091','10CS83','CSE8C', 19, 15, 20,0);
INSERT INTO IAMARKS VALUES ('4AD13CS091','10CS84','CSE8C', 20, 16, 19,0);
INSERT INTO IAMARKS VALUES ('4AD13CS091','10CS85','CSE8C', 15, 15, 12,0);

```

SELECT \* FROM STUDENT;

USN	SNAME	ADDRESS	PHONE	GENDER
4AD13CS020	AKSHAY	BELAGAVI	8877881122	M
4AD13CS062	SANDHYA	BENGALURU	7722829912	F
4AD13CS066	SUPRIYA	MANGALURU	8877881122	F
4AD13CS091	TEESHA	BENGALURU	7712312312	F
4AD14CS010	ABHAY	BENGALURU	9900211201	M
4AD14CS025	ASMI	BENGALURU	7894737377	F
4AD14CS032	BHASKAR	BENGALURU	9923211099	M
4AD15CS011	AJAY	TUMKUR	9845091341	M
4AD15CS029	CHITRA	DAVANGERE	7696772121	F
4AD15CS045	JEEVA	BELLARY	9944850121	M
4AD15CS091	SANTOSH	MANGALURU	8812332201	M
4AD16CS045	ISMAIL	KABURGI	9900232201	M
4AD16CS088	SAMEERA	SHIMOGA	9905542212	F
4AD16CS122	VINAYAKA	CHIKAMAGALUR	8800880011	M

14 rows in set (0.00 sec)

```
SELECT * FROM SEMSEC;
```

```
mysql> SELECT * FROM SEMSEC;
+-----+-----+-----+
| SSID | SEM | SEC |
+-----+-----+-----+
| CSE1A | 1   | A   |
| CSE1B | 1   | B   |
| CSE1C | 1   | C   |
| CSE2A | 2   | A   |
| CSE2B | 2   | B   |
| CSE2C | 2   | C   |
| CSE3A | 3   | A   |
| CSE3B | 3   | B   |
| CSE3C | 3   | C   |
| CSE4A | 4   | A   |
| CSE4B | 4   | B   |
| CSE4C | 4   | C   |
| CSE5A | 5   | A   |
| CSE5B | 5   | B   |
| CSE5C | 5   | C   |
| CSE6A | 6   | A   |
| CSE6B | 6   | B   |
| CSE6C | 6   | C   |
| CSE7A | 7   | A   |
| CSE7B | 7   | B   |
| CSE7C | 7   | C   |
| CSE8A | 8   | A   |
| CSE8B | 8   | B   |
| CSE8C | 8   | C   |
+-----+-----+-----+
24 rows in set (0.00 sec)
```

```
SELECT * FROM CLASS;
```

```
mysql> SELECT * FROM CLASS;
+-----+-----+
| USN | SSID |
+-----+-----+
| 4AD16CS045 | CSE3A |
| 4AD16CS088 | CSE3B |
| 4AD16CS122 | CSE3C |
| 4AD15CS011 | CSE4A |
| 4AD15CS029 | CSE4A |
| 4AD15CS045 | CSE4B |
| 4AD15CS091 | CSE4C |
| 4AD14CS010 | CSE7A |
| 4AD14CS025 | CSE7A |
| 4AD14CS032 | CSE7A |
| 4AD13CS020 | CSE8A |
| 4AD13CS062 | CSE8A |
| 4AD13CS066 | CSE8B |
| 4AD13CS091 | CSE8C |
+-----+-----+
14 rows in set (0.00 sec)
```

SELECT \* FROM SUBJECT;

```
mysql> SELECT * FROM SUBJECT;
+-----+-----+-----+-----+
| SUBCODE | TITLE | SEM | CREDITS |
+-----+-----+-----+-----+
| 10CS71 | OOAD | 7 | 4 |
| 10CS72 | ECS | 7 | 4 |
| 10CS73 | PTW | 7 | 4 |
| 10CS74 | DWDM | 7 | 4 |
| 10CS75 | JAVA | 7 | 4 |
| 10CS76 | SAN | 7 | 4 |
| 10CS81 | ACA | 8 | 4 |
| 10CS82 | SSM | 8 | 4 |
| 10CS83 | NM | 8 | 4 |
| 10CS84 | CC | 8 | 4 |
| 10CS85 | PW | 8 | 4 |
| 15CS31 | M3 | 3 | 4 |
| 15CS32 | ADE | 3 | 4 |
| 15CS33 | DSA | 3 | 4 |
| 15CS34 | CO | 3 | 4 |
| 15CS35 | USP | 3 | 3 |
| 15CS36 | DMS | 3 | 3 |
| 15CS41 | M4 | 4 | 4 |
| 15CS42 | SE | 4 | 4 |
| 15CS43 | DAA | 4 | 4 |
| 15CS44 | MPMC | 4 | 4 |
| 15CS45 | OOC | 4 | 3 |
| 15CS46 | DC | 4 | 3 |
| 15CS51 | ME | 5 | 4 |
| 15CS52 | CN | 5 | 4 |
| 15CS53 | DBMS | 5 | 4 |
| 15CS54 | ATC | 5 | 4 |
| 15CS55 | JAVA | 5 | 3 |
| 15CS56 | AI | 5 | 3 |
+-----+-----+-----+-----+
29 rows in set (0.00 sec)
```

SELECT \* FROM IAMARKS;

```
mysql> SELECT * FROM IAMARKS;
+-----+-----+-----+-----+-----+-----+-----+-----+
| USN | SUBCODE | SSID | TEST1 | TEST2 | TEST3 | FINALIA |
+-----+-----+-----+-----+-----+-----+-----+
| 4AD13CS091 | 10CS81 | CSE8C | 15 | 16 | 18 | 0 |
| 4AD13CS091 | 10CS82 | CSE8C | 12 | 19 | 14 | 0 |
| 4AD13CS091 | 10CS83 | CSE8C | 19 | 15 | 20 | 0 |
| 4AD13CS091 | 10CS84 | CSE8C | 20 | 16 | 19 | 0 |
| 4AD13CS091 | 10CS85 | CSE8C | 15 | 15 | 12 | 0 |
+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

**Queries:**

1. List all the student details studying in fourth semester ‘C’ section.

```
SELECT S.*, SS.SEM, SS.SEC FROM STUDENT S, SEMSEC SS, CLASS C WHERE
S.USN = C.USN AND SS.SSID = C.SSID AND SS.SEM = 4 AND SS.SEC='C'
```

USN	SNAME	ADDRESS	PHONE	GENDER	SEM	SEC
4AD15CS091	SANTOSH	MANGALURU	8812332201	M	4	C

1 row in set (0.00 sec)

2. Compute the total number of male and female students in each semester and in each section.

```
SELECT SS.SEM, SS.SEC, S.GENDER, COUNT (S.GENDER) AS COUNT FROM
STUDENT S, SEMSEC SS, CLASS C
WHERE S.USN = C.USN AND SS.SSID = C.SSID
GROUP BY SS.SEM, SS.SEC, S.GENDER ORDER BY SEM;
```

SEM	SEC	GENDER	COUNT
3	A	M	1
3	B	F	1
3	C	M	1
4	A	F	1
4	A	M	1
4	B	M	1
4	C	M	1
7	A	F	1
7	A	M	2
8	A	F	1
8	A	M	1
8	B	F	1
8	C	F	1

13 rows in set (0.00 sec)

3. Create a view of Test1 marks of student USN ‘1BI15CS101’ in all subjects.

```
CREATE VIEW VW_STUDENT_TEST AS SELECT TEST1,SUBCODE FROM
IAMARKS WHERE USN='4AD13CS091';
```

```
SELECT * FROM VW_STUDENT_TEST
```

TEST1	SUBCODE
15	10CS81
12	10CS82
19	10CS83
20	10CS84
15	10CS85

5 rows in set (0.00 sec)

4. Calculate the FinalIA (average of best two test marks) and update the corresponding table for all students.

UPDATE IAMARKS

SET FINALIA=GREATEST(TEST1+TEST2,TEST2+TEST3,TEST1+TEST3)/2;

**Note:** Before execution above SQL statement, IAMARKS table contents are:

SELECT \* FROM IAMARKS;

USN	SUBCODE	SSID	TEST1	TEST2	TEST3	FINALIA
4AD13CS091	10CS81	CSE8C	15	16	18	0
4AD13CS091	10CS82	CSE8C	12	19	14	0
4AD13CS091	10CS83	CSE8C	19	15	20	0
4AD13CS091	10CS84	CSE8C	20	16	19	0
4AD13CS091	10CS85	CSE8C	15	15	12	0

5 rows in set (0.00 sec)

UPDATE IAMARKS

SET FINALIA=GREATEST(TEST1+TEST2,TEST2+TEST3,TEST1+TEST3)/2;

After executing above SQL statement, IAMARKS table contents are:

USN	SUBCODE	SSID	TEST1	TEST2	TEST3	FINALIA
4AD13CS091	10CS81	CSE8C	15	16	18	17
4AD13CS091	10CS82	CSE8C	12	19	14	17
4AD13CS091	10CS83	CSE8C	19	15	20	20
4AD13CS091	10CS84	CSE8C	20	16	19	20
4AD13CS091	10CS85	CSE8C	15	15	12	15

5 rows in set (0.00 sec)

5. Categorize students based on the following criterion:

If FinalIA = 17 to 20 then CAT = 'Outstanding'

If FinalIA = 12 to 16 then CAT = 'Average'

If FinalIA < 12 then CAT = 'Weak'

Give these details only for 8th semester A, B, and C section students.

```
SELECT S.USN,S.SNAME,S.ADDRESS,S.PHONE,S.GENDER,
(CASE
WHEN IA.FINALIA BETWEEN 17 AND 20 THEN 'OUTSTANDING'
WHEN IA.FINALIA BETWEEN 12 AND 16 THEN 'AVERAGE'
ELSE 'WEAK'
END) AS CAT
FROM STUDENT S, SEMSEC SS, IAMARKS IA, SUBJECT SUB WHERE S.USN = IA.USN
AND SS.SSID = IA.SSID AND SUB.SUBCODE = IA.SUBCODE AND SUB.SEM = 8;
```

USN	SNAME	ADDRESS	PHONE	GENDER	CAT
4AD13CS091	TEESHA	BENGALURU	7712312312	F	OUTSTANDING
4AD13CS091	TEESHA	BENGALURU	7712312312	F	OUTSTANDING
4AD13CS091	TEESHA	BENGALURU	7712312312	F	OUTSTANDING
4AD13CS091	TEESHA	BENGALURU	7712312312	F	OUTSTANDING
4AD13CS091	TEESHA	BENGALURU	7712312312	F	AVERAGE

5 rows in set (0.00 sec)

**Program Outcomes:**

**The students are able to**

- Create, Update and query on the database.
- Demonstrate the working of different concepts of DBMS
- Implement, analyze and evaluate the project developed for an application.

**E.Consider the schema for Company Database:**

**EMPLOYEE (SSN, Name, Address, Sex, Salary, SuperSSN,**

**DNo) DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate)**

**DLOCATION (DNo,DLoc)**

**PROJECT (PNo, PName, PLocation, DNo)**

**WORKS\_ON (SSN, PNo, Hours)**

Write SQL queries to

1. Make a list of all project numbers for projects that involve an employee whose last name is ‘Scott’, either as a worker or as a manager of the department that controls the project.
2. Show the resulting salaries if every employee working on the ‘IoT’ project is given a 10 percent raise.
3. Find the sum of the salaries of all employees of the ‘Accounts’ department, as well as the maximum salary, the minimum salary, and the average salary in this department
4. Retrieve the name of each employee who works on all the projects controlled by department number 5 (use NOT EXISTS operator).
5. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs. 6,00,000.

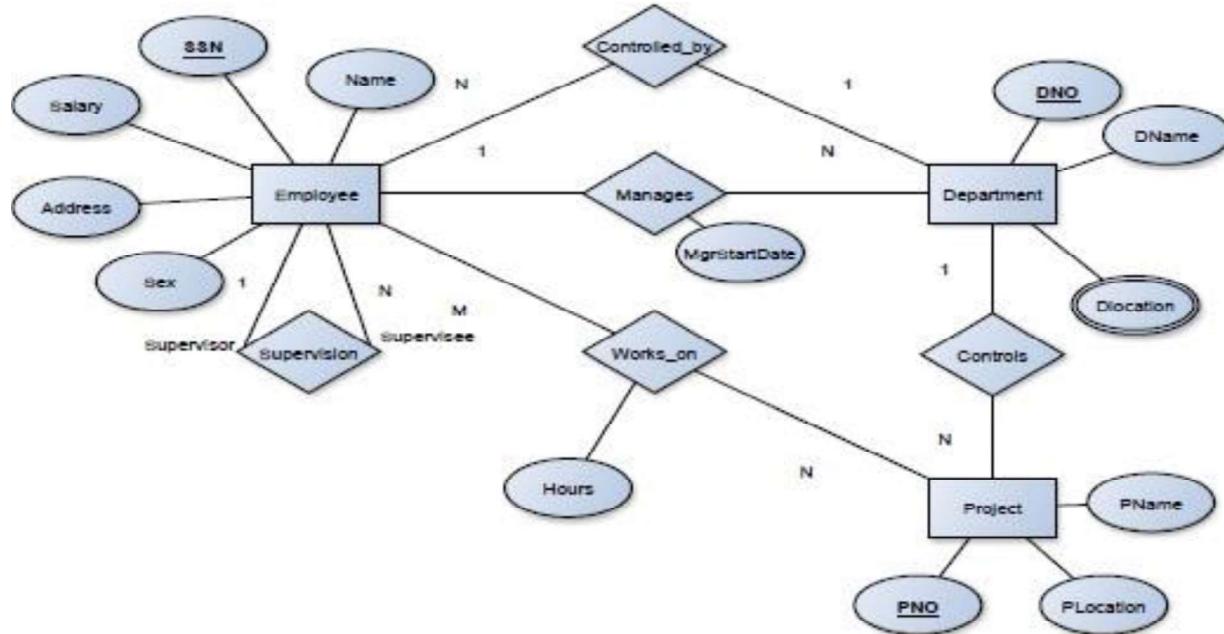
**Program Objectives:**

**This course will enable students to**

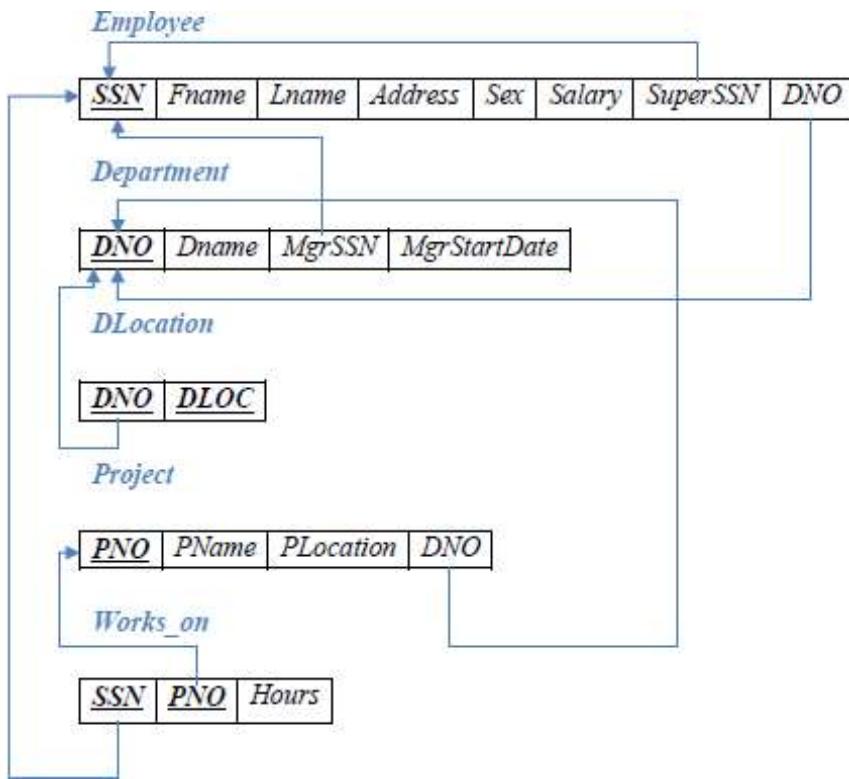
- Foundation knowledge in database concepts, technology and practice to groom students into well-informed database application developers.
- Strong practice in SQL programming through a variety of database problems.
- Develop database applications using front-end tools and back-end DBMS.

**Solution:**

**Entity-Relationship Diagram**



## Schema Diagram



## Table Creation

```
CREATE TABLE DEPARTMENT (
DNO VARCHAR (20) PRIMARY KEY,
DNAME VARCHAR (20),
MGRSTARTDATE DATE,
MGRSSN VARCHAR (20));
```

```
CREATE TABLE EMPLOYEE (
SSN VARCHAR (20) PRIMARY KEY,
FNAME VARCHAR (20),
LNAME VARCHAR (20),
ADDRESS VARCHAR (100),
SEX CHAR (1),
SALARY INT (10),
SUPERSSN VARCHAR (20),
DNO VARCHAR (20),
FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE (SSN),
FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNO));
```

**NOTE:** Once DEPARTMENT and EMPLOYEE tables are created we must alter department table to add foreign constraint MGRSSN using sql command

```
ALTER TABLE DEPARTMENT ADD FOREIGN KEY(MGRSSN) REFERENCES
EMPLOYEE(SSN);
```

```
CREATE TABLE DLOCATION (
DLOC VARCHAR (20),
DNO VARCHAR (20),
PRIMARY KEY (DNO, DLOC),
FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNO));
```

```
CREATE TABLE PROJECT (
PNO INT (10) PRIMARY KEY,
PNAME VARCHAR (20),
PLOCATION VARCHAR (20),
DNO VARCHAR (20),
FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNO));
```

```
CREATE TABLE WORKS_ON (
HOURS INT (4),
SSN VARCHAR (20),
PNO INT (10),
PRIMARY KEY (SSN, PNO),
FOREIGN KEY (SSN) REFERENCES EMPLOYEE (SSN),
FOREIGN KEY (PNO) REFERENCES PROJECT (PNO));
```

### Table Descriptions

DESC EMPLOYEE;

```
mysql> DESC EMPLOYEE;
```

Field	Type	Null	Key	Default	Extra
SSN	varchar(20)	NO	PRI	NULL	
FNAME	varchar(20)	YES		NULL	
LNAME	varchar(20)	YES		NULL	
ADDRESS	varchar(100)	YES		NULL	
SEX	char(1)	YES		NULL	
SALARY	int(10)	YES		NULL	
SUPERSSN	varchar(20)	YES	MUL	NULL	
DNO	varchar(20)	YES	MUL	NULL	

```
3 rows in set (0.00 sec)
```

DESC DEPARTMENT;

```
mysql> DESC DEPARTMENT;
```

Field	Type	Null	Key	Default	Extra
DNO	varchar(20)	NO	PRI	NULL	
DNAME	varchar(20)	YES		NULL	
MGRSTARTDATE	date	YES		NULL	
MGRSSN	varchar(20)	YES	MUL	NULL	

```
4 rows in set (0.00 sec)
```

DESC DLOCATION;

```
mysql> DESC DLOCATION;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| DLOC  | varchar(20) | NO   | PRI   |          |       |
| DNO   | varchar(20)  | NO   | PRI   |          |       |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

DESC PROJECT;

```
mysql> DESC PROJECT;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| PNO   | int<10> | NO   | PRI   | NULL    |       |
| PNAME  | varchar(20) | YES  |        | NULL    |       |
| PLOCATION | varchar(20) | YES  |        | NULL    |       |
| DNO   | varchar(20)  | YES  | MUL   | NULL    |       |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

DESC WORKS\_ON;

```
mysql> DESC WORKS_ON;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| HOURS | int<4> | YES  |        | NULL    |       |
| SSN   | varchar(20) | NO   | PRI   |          |       |
| PNO   | int<10>  | NO   | PRI   | 0       |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

#### **Insertion of values to tables**

```
INSERT INTO EMPLOYEE VALUES ('ATMEECE01','JOHN','SCOTT','BANGALORE','M',  
450000,NULL,NULL);  
INSERT INTO EMPLOYEE VALUES ('ATMECSE01','JAMES','SMITH','BANGALORE','M',  
500000,NULL,NULL);  
INSERT INTO EMPLOYEE VALUES ('ATMECSE02','HEARN','BAKER','BANGALORE','M',  
700000,NULL,NULL);  
INSERT INTO EMPLOYEE VALUES ('ATMECSE03','EDWARD','SCOTT','MYSORE','M',  
500000,NULL,NULL);  
INSERT INTO EMPLOYEE VALUES ('ATMECSE04','PAVAN','HEGDE','MANGALORE','M',  
650000,NULL,NULL);  
INSERT INTO EMPLOYEE VALUES ('ATMECSE05','GIRISH','MALYA','MYSORE','M',  
450000,NULL,NULL);  
INSERT INTO EMPLOYEE VALUES ('ATMECSE06','NEHA','SN','BANGALORE','F',  
800000,NULL,NULL);  
INSERT INTO EMPLOYEE VALUES ('ATMEACC01','AHANA','K','MANGALORE','F',  
350000,NULL,NULL);  
INSERT INTO EMPLOYEE VALUES  
('ATMEACC02','SANTHOSH','KUMAR','MANGALORE','M', 300000,NULL,NULL);  
INSERT INTO EMPLOYEE VALUES ('ATMEISE01','VEENA','M','MYSORE','F',  
600000,NULL,NULL);
```

---

INSERT INTO EMPLOYEE VALUES ('ATMEIT01','NAGESH','HR','BANGALORE','M',  
500000,NULL,NULL);

INSERT INTO DEPARTMENT VALUES ('1','ACCOUNTS','2001-01-01','ATMEACC02');  
 INSERT INTO DEPARTMENT VALUES ('2','IT','2016-08-01','ATMEIT01');  
 INSERT INTO DEPARTMENT VALUES ('3','ECE','2008-6-01','ATMEECE01');  
 INSERT INTO DEPARTMENT VALUES ('4','ISE','2015-06-01','ATMEISE01');  
 INSERT INTO DEPARTMENT VALUES ('5','CSE','2002-06-01','ATMECSE05');

**Note: update entries of employee table to fill missing fields SUPERSSN and DNO**

UPDATE EMPLOYEE SET SUPERSSN='ATMECSE02', DNO='5' WHERE  
SSN='ATMECSE01';

UPDATE EMPLOYEE SET SUPERSSN='ATMECSE03', DNO='5' WHERE SSN='ATMECSE02';

UPDATE EMPLOYEE SET SUPERSSN='ATMECSE04', DNO='5' WHERE SSN='ATMECSE03';

UPDATE EMPLOYEE SET DNO='5', SUPERSSN='ATMECSE05' WHERE SSN='ATMECSE04';

UPDATE EMPLOYEE SET DNO='5', SUPERSSN='ATMECSE06' WHERE SSN='ATMECSE05';

UPDATE EMPLOYEE SET DNO='5', SUPERSSN=NULL WHERE SSN='ATMECSE06';

UPDATE EMPLOYEE SET DNO='1', SUPERSSN='ATMEACC02' WHERE SSN='ATMEACC01';

UPDATE EMPLOYEE SET DNO='1', SUPERSSN=NULL WHERE  
SSN='ATMEACC02';

UPDATE EMPLOYEE SET DNO='4', SUPERSSN=NULL WHERE  
SSN='ATMEISE01';

UPDATE EMPLOYEE SET DNO='2', SUPERSSN=NULL WHERE  
SSN='ATMEIT01';

INSERT INTO DLOCATION VALUES ('BANGALORE', '1');  
 INSERT INTO DLOCATION VALUES ('BANGALORE', '2');  
 INSERT INTO DLOCATION VALUES ('BANGALORE', '3');  
 INSERT INTO DLOCATION VALUES ('MANGALORE', '4');  
 INSERT INTO DLOCATION VALUES ('MANGALORE', '5');

INSERT INTO PROJECT VALUES (100,'IOT','BANGALORE','5');  
 INSERT INTO PROJECT VALUES (101,'CLOUD','BANGALORE','5');  
 INSERT INTO PROJECT VALUES (102,'BIGDATA','BANGALORE','5');  
 INSERT INTO PROJECT VALUES (103,'SENSORS','BANGALORE','3');  
 INSERT INTO PROJECT VALUES (104,'BANK MANAGEMENT','BANGALORE','1');  
 INSERT INTO PROJECT VALUES (105,'SALARY MANAGEMENT','BANGALORE','1');

```
INSERT INTO PROJECT VALUES (106,'OPENSTACK','BANGALORE','4');
INSERT INTO PROJECT VALUES (107,'SMART CITY','BANGALORE','2');
```

```
INSERT INTO WORKS_ON VALUES (4, 'ATMECSE01', 100);
INSERT INTO WORKS_ON VALUES (6, 'ATMECSE01', 101);
INSERT INTO WORKS_ON VALUES (8, 'ATMECSE01', 102);
INSERT INTO WORKS_ON VALUES (10, 'ATMECSE02', 100);
INSERT INTO WORKS_ON VALUES (3, 'ATMECSE04', 100);
INSERT INTO WORKS_ON VALUES (4, 'ATMECSE05', 101);
INSERT INTO WORKS_ON VALUES (5, 'ATMECSE06', 102);
INSERT INTO WORKS_ON VALUES (6, 'ATMECSE03', 102);
INSERT INTO WORKS_ON VALUES (7, 'ATMEECE01', 103);
INSERT INTO WORKS_ON VALUES (5, 'ATMEACC01', 104);
INSERT INTO WORKS_ON VALUES (6, 'ATMEACC02', 105);
INSERT INTO WORKS_ON VALUES (4, 'ATMEISE01', 106);
INSERT INTO WORKS_ON VALUES (10, 'ATMEIT01', 107);
```

```
SELECT * FROM EMPLOYEE;
```

mysql> SELECT * FROM EMPLOYEE;								
SSN	FNAME	LNAME	ADDRESS	SEX	SALARY	SUPERSSN	DNO	
ATMEACC01	AHANA	K	MANGALORE	F	350000	ATMEACC02	1	
ATMEACC02	SANTHOSH	KUMAR	MANGALORE	M	300000	NULL	1	
ATMECSE01	JAMES	SMITH	BANGALORE	M	500000	ATMECSE02	5	
ATMECSE02	HEARN	BAKER	BANGALORE	M	700000	ATMECSE03	5	
ATMECSE03	EDWARD	SCOTT	mysore	M	500000	ATMECSE04	5	
ATMECSE04	PAVAN	HEGDE	MANGALORE	M	650000	ATMECSE05	5	
ATMECSE05	GIRISH	MALYA	mysore	M	450000	ATMECSE06	5	
ATMECSE06	NEHA	SN	BANGALORE	F	800000	NULL	5	
ATMEECE01	JOHN	SCOTT	BANGALORE	M	450000	NULL	3	
ATMEISE01	VEENA	M	mysore	F	600000	NULL	4	
ATMEIT01	NAGESH	HR	BANGALORE	M	500000	NULL	2	

11 rows in set (0.00 sec)

mysql>

```
SELECT * FROM DEPARTMENT ;
```

mysql> SELECT * FROM DEPARTMENT;			
DNO	DNAME	MGRSTARTDATE	MGRSSN
1	ACCOUNTS	2001-01-01	ATMEACC02
2	IT	2016-08-01	ATMEIT01
3	ECE	2008-06-01	ATMEECE01
4	ISE	2015-06-01	ATMEISE01
5	CSE	2002-06-01	ATMECSE05

5 rows in set (0.02 sec)

---

```
SELECT * FROM DLOCATION;
```

DLOC	DNO
BANGALORE	1
BANGALORE	2
BANGALORE	3
MANGALORE	4
MANGALORE	5

5 rows in set (0.03 sec)

```
SELECT * FROM PROJECT;
```

PNO	PNAME	PLOCATION	DNO
100	IOT	BANGALORE	5
101	CLOUD	BANGALORE	5
102	BIGDATA	BANGALORE	5
103	SENSORS	BANGALORE	3
104	BANK MANAGEMENT	BANGALORE	1
105	SALARY MANAGEMENT	BANGALORE	1
106	OPENSTACK	BANGALORE	4
107	SMART CITY	BANGALORE	2

8 rows in set (0.03 sec)

```
SELECT * FROM WORKS_ON
```

HOURS	SSN	PNO
5	ATMEACC01	104
6	ATMEACC02	105
4	ATMECSE01	100
6	ATMECSE01	101
8	ATMECSE01	102
10	ATMECSE02	100
6	ATMECSE03	102
3	ATMECSE04	100
4	ATMECSE05	101
5	ATMECSE06	102
7	ATMEECE01	103
4	ATMEISE01	106
10	ATMEIT01	107

13 rows in set (0.00 sec)

**Queries:**

1. Make a list of all project numbers for projects that involve an employee whose last name is ‘Scott’, either as a worker or as a manager of the department that controls the project.

```
(SELECT DISTINCT P.PNO FROM PROJECT P, DEPARTMENT D, EMPLOYEE E
 WHERE E.DNO=D.DNO AND D.MGRSSN=E.SSN AND E.LNAME='SCOTT')
UNION
(SELECT DISTINCT P1.PNO FROM PROJECT P1, WORKS_ON W, EMPLOYEE E1 WHERE
P1.PNO=W.PNO AND E1.SSN=W.SSN AND E1.LNAME='SCOTT')
```

PNO
100
101
102
103
104
105
106
107

8 rows in set (0.00 sec)

2. Show the resulting salaries if every employee working on the ‘IoT’ project is given a 10 percent raise.

```
SELECT E.FNAME, E.LNAME, 1.1*E.SALARY AS INCR_SAL FROM EMPLOYEE E,
WORKS_ON W, PROJECT P WHERE E.SSN=W.SSN AND W.PNO=P.PNO AND
P.PNAME='IOT';
```

FNAME	LNAME	INCR_SAL
JAMES	SMITH	550000.0
HEARN	BAKER	770000.0
PAVAN	HEGDE	715000.0

3 rows in set (0.01 sec)

3. Find the sum of the salaries of all employees of the ‘Accounts’ department, as well as the maximum salary, the minimum salary, and the average salary in this department

```
SELECT SUM(E.SALARY), MAX(E.SALARY), MIN(E.SALARY), AVG(E.SALARY)
FROM EMPLOYEE E, DEPARTMENT D WHERE E.DNO=D.DNO AND
D.DNAME='ACCOUNTS';
```

SUM(E.SALARY)	MAX(E.SALARY)	MIN(E.SALARY)	AVG(E.SALARY)
650000	350000	300000	325000.0000

1 row in set (0.00 sec)

4. Retrieve the name of each employee who works on all the projects Controlled by department number 5 (use NOT EXISTS operator).

```
SELECT E.FNAME,E.LNAME FROM EMPLOYEE E WHERE NOT EXISTS
(SELECT PNO FROM PROJECT P WHERE DNO=5 AND PNO NOT IN
(SELECT PNO FROM WORKS_ON W WHERE E.SSN=SSN));
```

FNAME	LNAME
JAMES	SMITH

1 row in set (0.00 sec)

5. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs. 6, 00,000.

```
SELECT D.DNO, COUNT (*)
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.DNO=E.DNO
AND E.SALARY>600000
AND D.DNO IN (SELECT E1.DNO
FROM EMPLOYEE E1
GROUP BY E1.DNO
HAVING COUNT (*)>5)
GROUP BY D.DNO;
```

DNO	COUNT(*)
5	3

1 row in set (0.00 sec)

#### Program Outcomes:

The students are able to

- Create, Update and query on the database.
- Demonstrate the working of different concepts of DBMS
- Implement, analyze and evaluate the project developed for an application.

Viva Questions**1. What is SQL?**

Structured Query Language

**2. What is database?**

A database is a logically coherent collection of data with some inherent meaning, representing some aspect of real world and which is designed, built and populated with data for a specific purpose.

**3. What is DBMS?**

It is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of defining, constructing and manipulating the database for various applications.

**4. What is a Database system?**

The database and DBMS software together is called as Database system.

**5. Advantages of DBMS?**

- Redundancy is controlled.
- Unauthorized access is restricted.
- Providing multiple user interfaces.
- Enforcing integrity constraints.
- Providing backup and recovery.

**6. Disadvantage in File Processing System?**

- Data redundancy & inconsistency.
- Difficult in accessing data.
- Data isolation.
- Data integrity.
- Concurrent access is not possible.
- Security Problems.

**7. Define the "integrity rules"**

There are two Integrity rules.

- Entity Integrity: States that “Primary key cannot have NULL value”
- Referential Integrity: States that “Foreign Key can be either a NULL value or should be Primary Key value of other relation.

**8. What is a view? How it is related to data independence?**

A view may be thought of as a virtual table, that is, a table that does not really exist in its own right but is instead derived from one or more underlying base table. In other words, there is no stored file that directly represents the view instead a definition of view is stored in data dictionary. Growth and restructuring of base tables is not reflected in views. Thus the view can insulate users from the effects of restructuring and growth in the database. Hence accounts for logical data independence.

**9. What is Data Model?**

A collection of conceptual tools for describing data, data relationships, data semantics and constraints.

**10. What is E-R model?**

This data model is based on real world that consists of basic objects called entities and of relationship among these objects. Entities are described in a database by a set of attributes.

**11. What is Object Oriented model?**

This model is based on collection of objects. An object contains values stored in instance variables within the object. An object also contains bodies of code that operate on the object. These bodies of code are called methods. Objects that contain same types of values and the same methods are grouped together into classes.

**12. What is an Entity?**

It is an 'object' in the real world with an independent existence.

**13. What is an Entity type?**

It is a collection (set) of entities that have same attributes.

**14. What is an attribute?**

It is a particular property, which describes the entity.

**15. What is degree of a Relation?**

It is the number of attribute of its relation schema.

**16. What is Relationship?**

It is an association among two or more entities.

**17. What is DDL (Data Definition Language)?**

A data base schema is specified by a set of definitions expressed by a special language called DDL.

**18. What is DML (Data Manipulation Language)?**

This language that enable user to access or manipulate data as organized by appropriate

data model.

**19. What is normalization?**

It is a process of analyzing the given relation schemas based on their Functional Dependencies (FDs) and primary key to achieve the properties

- Minimizing redundancy
- Minimizing insertion, deletion and update anomalies.

**20. What is 1 NF (Normal Form)?**

The domain of attribute must include only atomic (simple, indivisible) values.

**21. What is 2NF?**

A relation schema R is in 2NF if it is in 1NF and every non-prime attribute A in R is fully functionally dependent on primary key.

**22. What is 3NF?**

A relation schema R is in 3NF if it is in 2NF and for every FD X A either of the following is true

- X is a Super-key of R.
- A is a prime attribute of R.

In other words, if every non prime attribute is non-transitively dependent on primary key.

**23. What is BCNF (Boyce-Codd Normal Form)?**

A relation schema R is in BCNF if it is in 3NF and satisfies additional constraints that for every FD X A, X must be a candidate key.

**24. What is 4NF?**

A relation schema R is said to be in 4NF if for every Multivalued dependency X Y that holds over R, one of following is true

- X is subset or equal to (or) XY = R.
- X is a super key.

**25. What is 5NF?**

A Relation schema R is said to be 5NF if for every join dependency {R1, R2, ..., Rn} that holds R, one the following is true

- $R_i = R$  for some i.
- The join dependency is implied by the set of FD, over R in which the left side is key of R.

**26. What are partial, alternate,, artificial, compound and natural key?****Partial Key:**

It is a set of attributes that can uniquely identify weak entities and that are related to same owner entity. It is sometime called as Discriminator.

**Alternate Key:**

All Candidate Keys excluding the Primary Key are known as Alternate Keys.

**Artificial Key:**

If no obvious key, either standalone or compound is available, then the last resort is to simply create a key, by assigning a unique number to each record or occurrence. Then this is known as developing an artificial key.

**Compound Key:**

If no single data element uniquely identifies occurrences within a construct, then combining multiple elements to create a unique identifier for the construct is known as creating a compound key.

**Natural Key:**

When one of the data elements stored within a construct is utilized as the primary key, then it is called the natural key.

**27. What is meant by query optimization?**

The phase that identifies an efficient execution plan for evaluating a query that has the least estimated cost is referred to as query optimization.

**28. What do you mean by atomicity and aggregation?****Atomicity:**

Either all actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions. DBMS ensures this by undoing the actions of incomplete transactions.

**Aggregation:**

A concept which is used to model a relationship between a collection of entities and relationships. It is used when we need to express a relationship among relationships.

**29. What is a checkpoint and when does it occur?**

A Checkpoint is like a snapshot of the DBMS state. By taking checkpoints, the DBMS can reduce the amount of work to be done during restart in the event of subsequent crashes.

**30. What do you mean by flat file database?**

It is a database in which there are no programs or user access languages. It has no cross-file

capabilities but is user-friendly and provides user-interface management.

**31. Brief theory of Network, Hierarchical schemas and their properties**

Network schema uses a graph data structure to organize records example for such a database management system is CTCG while a hierarchical schema uses a tree data structure example for such a system is IMS.

**32. What is a query?**

A query with respect to DBMS relates to user commands that are used to interact with a data base. The query language can be classified into data definition language and data manipulation language.

**33. What do you mean by Correlated subquery?**

Subqueries, or nested queries, are used to bring back a set of rows to be used by the parent query. Depending on how the subquery is written, it can be executed once for the parent query or it can be executed once for each row returned by the parent query. If the subquery is executed for each row of the parent, this is called a *correlated subquery*.

A correlated subquery can be easily identified if it contains any references to the parent subquery columns in its WHERE clause. Columns from the subquery cannot be referenced anywhere else in the parent query. The following example demonstrates a non-correlated subquery.

E.g. Select \* From CUST Where '2019/03/05' IN (Select ODATE From ORDER Where CUST.CNUM = ORDER.CNUM)

**34. What are the primitive operations common to all record management systems?**

Addition, deletion and modification

**35. How do you communicate with an RDBMS?**

You communicate with an RDBMS using Structured Query Language (SQL)

**36. Define SQL and state the differences between SQL and other conventional programming Languages**

SQL is a nonprocedural language that is designed specifically for data access operations on normalized relational database structures. The primary difference between SQL and other conventional programming languages is that SQL statements specify what data operations should be performed rather than how to perform them.

**37. What is database Trigger?**

A database trigger is a PL/SQL block that can be defined to automatically execute for insert, update, and delete statements against a table. The trigger can be defined to execute once for the entire statement or once for every row that is inserted, updated, or deleted.

**38. What are stored-procedures? And what are the advantages of using them.**

Stored procedures are database objects that perform a user defined operation. A stored procedure can have a set of compound SQL statements. A stored procedure executes the SQL commands and return the result to the client. Stored procedures are used to reduce network traffic.

**39. Which is the subset of SQL commands used to manipulate Database structures, including tables?**

Data Definition Language (DDL)

**40. What operator performs pattern matching?**

LIKE operator

**41. What operator tests column for the absence of data?**

IS NULL operator

**42. What are the wildcards used for pattern matching?**

For single character substitution and % for multi-character substitution

**43. What are the difference between TRUNCATE and DELETE commands?**

TRUNCATE	DELETE
• TRUNCATE is a DDL command	• DELETE is a DML command
• TRUNCATE operation cannot be rolled back	• DELETE operation can be rolled back
• TRUNCATE does not invoke trigger	• DELETE does invoke trigger
• TRUNCATE resets auto_increment value to 0	• DELETE does not resets auto_increment value to 0

**44. What is the use of the ADD OR DROP option in the ALTER TABLE command?**

It is used to add/drop columns or add/drop constraints specified on the table

**45. What is the use of DESC in SQL?**

DESC has two purposes. It is used to describe a schema as well as to retrieve rows from table in descending order.

The query SELECT \* FROM EMP ORDER BY ENAME DESC will display the output sorted on ENAME in descending order

**46. What is the use of ON DELETE CASCADE?**

Whenever rows in the master (referenced) table are deleted ,the respective rows of the child (referencing) table with a matching foreign key column will get deleted as well. This is called a cascade delete

**Example Tables:**

```
CREATE TABLE Customer
(
customer_id INT (6) PRIMARY KEY,
cname VARCHAR (100),
caddress VARCHAR (100)
);
```

```
CREATE TABLE Order
```

```
(order_id INT (6) PRIMARY KEY,
products VARCHAR (100),
payment DECIMAL(10,2),
customer_id INT (6),
FOREIGN KEY (customer_id ) REFERENCES Customer(customer_id) ON DELETE CASCADE
);
```

Customer is the master table and Order is the child table, where 'customer\_id' is primary key in customer table and customer\_id is the foreign key in Order table and represents the customer who placed the order. When a row of Customer is deleted, any Order row matching the deleted Customer's customer\_id will also be deleted.

**47. What is the use of Floor()?**

The FLOOR() function returns the largest integer value that is smaller than or equal to a number.

**EXAMPLE;**

```
SELECT FLOOR(25.75);
```

**OUTPUT**

25

**48. What is the use of Truncate()?**

The TRUNCATE() function truncates a number to the specified number of decimal places.

**EXAMPLE;**

```
SELECT TRUNCATE(135.375, 2);
```

**OUTPUT**

135.37

**49. What is the use of CEILING?**

Return the smallest integer value that is greater than or equal to 25.75:

**EXAMPLE;**

```
SELECT CEILING(25.75)
```

**OUTPUT**

26

---

**50. What you mean by SQL UNIQUE Constraint?**

- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

**51. How to add and drop UNIQUE Constraint in table in mysql?**

ALTER TABLE contacts ADD CONSTRAINT UNC\_name\_email UNIQUE(name,email)

ALTER TABLE contacts DROP INDEX UNC\_name\_email;

**52. What is the Group by Clause?**

- The GROUP BY clause is a SQL command that is used to group rows that have the same values.
- The GROUP BY clause is used in the SELECT statement .Optionally it is used in conjunction with aggregate functions to produce summary reports from the database. That's what it does, summarizing data from the database.
- The queries that contain the GROUP BY clause are called grouped queries and only return single row for every grouped item.

**Example:**SELECT COUNT(CustomerID), Country FROM Customers

GROUP BY Country

**53. What is use of having clause in mysql**

- The HAVING clause is used in the SELECT statement to specify filter conditions for a group of rows or aggregates.
- The HAVING clause is often used with the GROUP BY clause to filter groups based on a specified condition. If the GROUP BY clause is omitted, the HAVING clause behaves like the WHERE clause.
- Notice that the HAVING clause applies a filter condition to each group of rows, while the WHERE clause applies the filter condition to each individual row.

**Example:**SELECT COUNT(CustomerID), Country FROM Customers

GROUP BY Country HAVING COUNT(CustomerID) > 5;

**54. What is distinct clause in SQL?**

When querying data from a table, you may get duplicate rows. In order to remove these duplicate rows, you use the DISTINCT clause in the SELECT statement.

**Example:**SELECT DISTINCT columns FROM table\_name WHERE where\_conditions;

**55. What is a union?**

Unions combine the results from multiple SELECT queries into a consolidated result set.

The only requirements for this to work is that the number of columns should be the same from all the SELECT queries which needs to be combined

**56. What is use of MySQL Aggregate Functions?**

- The data that you need is not always directly stored in the tables. However, you can get it by performing the calculations of the stored data when you select it.
- By definition, an aggregate function performs a calculation on a set of values and returns a single value.
- MySQL provides many aggregate functions that include AVG, COUNT, SUM, MIN, MAX, etc. An aggregate function ignores NULL values when it performs calculation except for the COUNT function.
- Often, aggregate functions are accompanied by the GROUP BY clause of the SELECT statement
- Below are some of aggregate functions used in sql query

**AVG function**

The AVG function calculates the average value of a set of values. It ignores NULL values in the calculation.

**COUNT function**

The COUNT function returns the number of the rows in a table. For example, you can use below query .below query return number of employees in Employee table

```
SELECT COUNT (Empname) FROM Employee
```

**SUM function**

The SUM function returns the sum of a set of values. The SUM function ignores NULL values. If no matching row found, the SUM function returns a NULL value.

The **SUM** function to get the sum of salary of employees in the Employee table

```
SELECT SUM(Salary) FROM Employee
```

**MAX function**

The MAX function returns the maximum value in a set of values.

Below query gets maximum salary of table Employee

```
SELECT MAX(Salary) FROM Employee
```

**MIN function**

The MIN function returns the minimum value in a set of values.

Below query returns minimum salary of table Employee

```
SELECT MIN(Salary) FROM Employee
```

## Additional Queries

### CREATE command

```
CREATE TABLE Employee
(
    Empno int(4) primary key,
    Empname varchar(50),
    job varchar(40),
    Hiredate date,
    Salary decimal(10,2),
    Deptno int(7),
    Age int(10)
);
```

### DESC command

DESC Employee;

Field	Type	Null	Key	Default	Extra
Empno	int(4)	NO	PRI	NULL	
Empname	varchar(50)	YES		NULL	
job	varchar(40)	YES		NULL	
Hiredate	date	YES		NULL	
Salary	decimal(10,2)	YES		NULL	
Deptno	int(7)	YES		NULL	
Age	int(10)	YES		NULL	

7 rows in set <0.00 sec>

### INSERT command

Insert the values into the table as specified.

- 1) Insert into Employee values(1000,'Hemanth','Manager','2018-11-17',35000, 30, 38);
- 2) Insert into Employee values(1001, 'Nitin','Manager','2018-05-01',45000, 10, 42);
- 3) Insert into Employee values(1002, 'Sachin','Salesman','2018-01-09',18000, 20, 28);
- 4) Insert into Employee values(1003, 'Deepak','Clerk','2018-05-15',15000, 40, 34);
- 5) Insert into Employee values(1004, 'Ajay','Analyst','2018-10-22',60000, 50, 45);
- 6) Insert into Employee values(1005, 'Arun','Programmer','2018-07-24',25000, 60,25);

mysql> select * from Employee;							
Empno	Empname	job	Hiredate	Salary	Deptno	Age	
1000	Hemanth	Manager	2018-11-17	35000.00	30	38	
1001	Nitin	Manager	2018-05-01	45000.00	10	42	
1002	Sachin	Salesman	2018-01-09	18000.00	20	28	
1003	Deepak	Clerk	2018-05-15	15000.00	40	34	
1004	Ajay	Analyst	2018-10-22	60000.00	50	45	
1005	Arun	Programmer	2018-07-24	25000.00	60	25	

6 rows in set <0.00 sec>

**Queries:**  
**Problems on select command:**

**1) Display the details of all managers of Employee Table**

```
SELECT * FROM Employee WHERE job='Manager';
```

**2) Display the details of all employees getting salary less than 30,000.**

```
SELECT * FROM Employee WHERE salary<30000;
```

**3) Display the details of employees who age is between 35 and 45**

```
SELECT * FROM Employee WHERE age BETWEEN 35 AND 45;
```

**4) Display the details of Clerks who have joined after 01-MAR-05.**

```
SELECT * FROM Employee WHERE job='Clerk' AND hiredate>'2018-03-05';
```

**5) Sort the details in descending order of Empno.**

```
SELECT * FROM Employee ORDER BY Empno DESC;
```

**6) Sort the details of employees in ascending order of name**

```
SELECT * FROM Employee ORDER BY Empname
```

**7) Display the details of employees whose names contain ‘i’ in them.**

```
SELECT * FROM Employee WHERE Empname LIKE '%i%';
```

**8) Display the details of employees whose names starts with ‘a’ in them.**

```
SELECT * FROM Employee WHERE Empname LIKE 'a%';
```

**9) Display the details of employees whose names does not starts with ‘a’ in them.**

```
SELECT * FROM Employee WHERE Empname NOT LIKE 'a%';
```

**10) Display the employee details whose names have exactly 4 characters.**

```
SELECT * FROM Employee WHERE length(Empname)=4
```

**11)Copy all the records of their from employee table and insert the records into a temp table with column names same as in Employee table**

```
CREATE TABLE TEMP SELECT * FROM Employee;
```

**Problems on update command:**

1. **Update the salary by 10% hike to Manager working in department number 20 and 30**

SOL: UPDATE EMP SET SAL = SAL \* 1.1 WHERE Deptno IN (20,30) AND JOB = 'Manager';

2. **Give 5% raise in salary to all the Salesman**

SOL1:UPDATE EMPLOYEE SET Salary=Salary\*1.15 WHERE JOB='Salesman';

**OR**

SOL2 :UPDATE EMPLOYEE SET Salary=Salary+(Salary \* 15/100) WHERE JOB='Salesman';

3. **Change the department no of Sachin to 40**

SOL: UPDATE EMP SET DEPTNO = 40 WHERE Empname = 'Sachin';

4. **Update all employee name to uppercase**

SOL: UPDATE EMPLOYEE SET Empname=upper(Empname);

**Problems on delete command:**

1. **Delete all the records of employees**

SOL: DELETE FROM Employee;

2. **Delete the records of employee name Ajay's only**

SOL: DELETE FROM EMP WHERE ENAME = 'Ajay';

3. **Delete the record of employee table whose Empno is 1005**

SOL: DELETE FROM EMP WHERE Empno =1005;

4. **Delete the first five records of employee table**

SOL: DELETE FROM EMPLOYEE LIMIT 5;

**ALTER command**

1. **How to create database name COLLEGE ?**

CREATE DATABASE COLLEGE;

2. **How Modify datatype of age column in Employee table**

ALTER TABLE Employee MODIFY age int(3);

3. **How to rename column name of job to Designation in Employee table?**

ALTER TABLE Employee CHANGE job Designation varchar(40);

**4. How to add column Commission in Employee table?**

ALTER TABLE Employee add Commission varchar(40);

**5. How to drop column Commission in Employee table?**

ALTER TABLE Employee DROP column Commission;

**6. How to add primary key to Employee table?**

ALTER TABLE Employee add primary key(Empno);

**7. How to drop primary key to Employee table?**

ALTER TABLE Employee DROP primary key;

**8. How to rename employee table?**

RENAME TABLE Employee to Employee\_Details

**9. How to delete contents of Employee table?**

DELETE FROM Employee;

OR

TRUNCATE Employee;

**10. How to drop Employee table?**

DROP TABLE Employee;

**11. How to drop database name COLLEGE?**

DROP DATABASE COLLEGE

