

# **ATME COLLEGE OF ENGINEERING**

**13<sup>th</sup> KM Stone, Bannur Road, Mysore - 570 028**



**A T M E**  
**College of Engineering**

## **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (ACADEMIC YEAR 2023-24)**

### **LABORATORY MANUAL**

**SUBJECT: DATABASE MANAGEMENT SYSTEM  
(BCS403)**

**SEMESTER: IV  
SCHEME: 2022**

**Composed by**

**Ms. Pragathi B**

**PROGRAMMER**

**Verified by**

**Mr. Sandesh R & Mr. Kiran B**

**FACULTY CO-ORDINATORS**

**Approved by**

**Dr. Puttegowda D**

**HOD, CSE**

# **INSTITUTIONAL MISSION AND VISION**

## **Objectives**

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

## **Vision**

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

## **Mission**

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

## **Department of Computer Science & Engineering**

### **Vision of the Department**

- To develop highly talented individuals in Computer Science and Engineering to deal with real world challenges in industry, education, research and society.

### **Mission of the Department**

- To inculcate professional behavior, strong ethical values, innovative research capabilities and leadership abilities in the young minds & to provide a teaching environment that emphasizes depth, originality and critical thinking.
- Motivate students to put their thoughts and ideas adoptable by industry or to pursue higher studies leading to research.

### **Program outcomes (POs)**

**Engineering Graduates will be able to:**

- **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

- **PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **PO12. Life-long learning:** Recognize the need for, and have the preparation and ability to engage

**Program Educational Objectives (PEO'S):**

1. Empower students with a strong basis in the mathematical, scientific and engineering fundamentals to solve computational problems and to prepare them for employment, higher learning and R&D.
2. Gain technical knowledge, skills and awareness of current technologies of computer science engineering and to develop an ability to design and provide novel engineering solutions for software/hardware problems through entrepreneurial skills.
3. Exposure to emerging technologies and work in teams on interdisciplinary projects with effective communication skills and leadership qualities.
4. Ability to function ethically and responsibly in a rapidly changing environment by applying innovative ideas in the latest technology, to become effective professionals in Computer Science to bear a life-long career in related areas.

## **Program Specific Outcomes**

### **Program Specific Outcomes (PSOs)**

1. **PSO1:** Ability to apply skills in the field of algorithms, database design, web design, cloud computing and data analytics.
2. **PSO2:** Apply knowledge in the field of computer networks for building network and internet based applications.

## **TABLE OF CONTENTS**

<b>Sl.NO</b>	<b>Experiments</b>
1	<p>Create a table called Employee &amp; execute the following.</p> <p><b>Employee(EMPNO,ENAME,JOB, MANAGER_NO, SAL, COMMISSION)</b></p> <ol style="list-style-type: none"> <li>1. Create a user and grant all permissions to the user.</li> <li>2. Insert the any three records in the employee table contains attributes EMPNO,ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback. Check the result.</li> <li>3. Add primary key constraint and not null constraint to the employee table.</li> <li>4. Insert null values to the employee table and verify the result.</li> </ol>
2	<p>Create a table called Employee that contain attributes EMPNO,ENAME,JOB, MGR,SAL &amp; execute the following.</p> <ol style="list-style-type: none"> <li>1. Add a column commission with domain to the Employee table.</li> <li>2. Insert any five records into the table.</li> <li>3. Update the column details of job</li> <li>4. Rename the column of Employee table using alter command.</li> <li>5. Delete the employee whose Empno is 105.</li> </ol>
3	<p>Queries using aggregate functions (COUNT,AVG,MIN,MAX,SUM),Group by,Order by.</p> <p><b>Employee (E_id, E_name, Age, Salary)</b></p> <ol style="list-style-type: none"> <li>1. Create Employee table containing all Records E_id, E_name, Age, Salary.</li> <li>2. Count number of employee names from employee table</li> <li>3. Find the Maximum age from employee table.</li> <li>4. Find the Minimum age from employee table.</li> <li>5. Find salaries of employee in Ascending Order.</li> <li>6. Find grouped salaries of employees.</li> </ol>
4	<p>Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old &amp; new Salary.</p> <p><b>CUSTOMERS(ID,NAME,AGE,ADDRESS,SALARY)</b></p>
5	<p>Create cursor for Employee table &amp; extract the values from the table. Declare the variables , Open the cursor &amp; extract the values from the cursor. Close the cursor.</p> <p><b>Employee (E_id, E_name, Age, Salary)</b></p>
6	<p>Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.</p>
7	<p>Install an Open Source NoSQL Data base MongoDB &amp; perform basic CRUD(Create, Read, Update &amp; Delete) operations. Execute MongoDB basic Queries using CRUD operations.</p>
<p><b>Course outcomes (Course Skill Set):</b></p> <p>At the end of the course, the student will be able to:</p> <ul style="list-style-type: none"> <li>● Describe the basic elements of a relational database management system</li> <li>● Design entity relationship for the given scenario.</li> <li>● Apply various Structured Query Language (SQL) statements for database manipulation.</li> <li>● Analyze various normalization forms for the given application.</li> <li>● Develop database applications for the given real-world problem.</li> <li>● Understand the concepts related to NoSQL databases.</li> </ul>	
<p><b>Assessment Details (both CIE and SEE)</b></p> <p>The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum</p>	

passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE(Semester End Examination) taken together.

#### **CIE for the theory component of the IPCC (maximum marks 50)**

- IPCC means practical portion integrated with the theory of the course.
- CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**.
- 25 marks for the theory component are split into **15 marks** for two Internal Assessment Tests (Two Tests, each of 15 Marks with 01-hour duration, are to be conducted) and **10 marks** for other assessment methods mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.
- Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theory component of IPCC (that is for **25 marks**).
- The student has to secure 40% of 25 marks to qualify in the CIE of the theory component of IPCC.

#### **CIE for the practical component of the IPCC**

- **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks**.
- The laboratory test (**duration 02/03 hours**) after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks**.
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

<b>SI NO</b>	<b>Particulars</b>	<b>Page No</b>
1.	INTRODUCTION TO DATABASE	01
2.	EXPERITMENT 1	23
3.	EXPERITMENT 2	27
4.	EXPERITMENT 3	30
5.	EXPERITMENT 4	32
6.	EXPERITMENT 5	36
7.	EXPERITMENT 6	38
8.	EXPERITMENT 7	42
9.	VIVA QUESTIONS	51



---

## DBMS LABORATORY

**Aim:** Study of Open-Source Relational Databases: MySQL

**Objective:** To learn and understand open-source relational databases.

**Hardware requirements:** Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

**Software requirements:** Ubuntu 14 Operating System, MySQL WorkBench

### THEORY

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

A relational database management system (RDBMS) is a program that lets you create, update, and administer a relational database. Most commercial RDBMS's use the Structured Query Language (SQL) to access the database, although SQL was invented after the development of the relational model and is not necessary for its use.

MySQL open-source RDBMS overview:

MySQL is a popular open-source relational database management system (RDBMS) choice for web-based applications. Developers, database administrators use MySQL to build and manage next-generation web- and cloud-based applications. With most open-source RDBMS options, MySQL is available in several different editions and runs on Windows, OS X, Solaris, FreeBSD and other variants of Linux and Unix.

### MySQL Workbench

MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more. MySQL Workbench is available on Windows, Linux and Mac OS X.

### Design

MySQL Workbench enables a DBA, developer, or data architect to visually design, model, generate, and manage databases. It includes everything a data modeler needs for creating complex ER models, forward and reverse engineering, and also delivers key features for performing difficult change management and documentation tasks that normally require much time and effort.

**Develop**

MySQL Workbench delivers visual tools for creating, executing, and optimizing SQL queries. The SQL Editor provides color syntax highlighting, auto-complete, reuse of SQL snippets, and execution history of SQL. The Database Connections Panel enables developers to easily manage standard database connections, including MySQL Fabric. The Object Browser provides instant access to database schema and objects.

**Database Migration**

MySQL Workbench now provides a complete, easy to use solution for migrating Microsoft SQL Server, Microsoft Access, Sybase ASE, PostgreSQL, and other RDBMS tables, objects and data to MySQL. Developers and DBAs can quickly and easily convert existing applications to run on MySQL both on Windows and other platforms. Migration also supports migrating from earlier versions of MySQL to the latest releases

**Administer**

MySQL Workbench provides a visual console to easily administer MySQL environments and gain better visibility into databases. Developers and DBAs can use the visual tools for configuring servers, administering users, performing backup and recovery, inspecting audit data, and viewing database health.

**Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as**

Table View Index

Introduction to SQL:

SQL stands for Structured Query Language SQL

lets you access and manipulate databases

SQL is an ANSI (American National Standards Institute) standard

Commands of SQL are grouped into four languages.

1>DDL

DDL is abbreviation of Data Definition Language. It is used to create and modify the structure of database objects in database.

Examples: CREATE, ALTER, DROP, RENAME, TRUNCATE statements

2>DML

DML is abbreviation of Data Manipulation Language. It is used to retrieve, store, modify, delete, insert and update data in database.

Examples: SELECT, UPDATE, INSERT, DELETE statements

3>DCL

DCL is abbreviation of Data Control Language. It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

Examples: GRANT, REVOKE statements

4>TCL

TCL is abbreviation of Transactional Control Language. It is used to manage different transactions occurring within a database.

Examples: COMMIT, ROLLBACK statements

**Data Definition Language (DDL)**

1. Data definition Language (DDL) is used to create, rename, alter, modify, drop, replace, and delete tables, Indexes, Views, and comment on database objects; and establish a default database.

2. The DDL part of SQL permits database tables to be created or deleted. It also defines indexes (keys), specify links between tables, and impose constraints between tables. The most important DDL statements in SQL are:

□CREATE TABLE- Creates a new table

□ALTER TABLE- Modifies a table

□DROP TABLE- Deletes a table

□TRUNCATE -Use to truncate (delete all rows) a table.

□CREATE INDEX- Creates an index (search key)

□DROP INDEX- Deletes an index

**1.The CREATE TABLE Statement**

The CREATE TABLE statement is used to create a table in a database.

Syntax

```
CREATE TABLE table_name  
(attr1_name attr1_datatype(size) attr1_constraint,  
attr2_name attr2_datatype(size) attr2_constraint,...);
```

**SQL Constraints**

Constraints are used to limit the type of data that can go into a table.

Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

We will focus on the following constraints:

- NOT NULL
- UNIQUE PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

Add constraint after table creation using alter table option

Syntax - Alter table add constraint constraint\_name constraint\_type(Attr\_name) Example -

Alter table stud add constraint

prk1 primary key(rollno);

Drop constraint:

Syntax- Drop Constraint Constraint\_name;

Example - Drop constraint prk1;

**2.The Drop TABLE Statement**

Removes the table from the database

Syntax

```
DROP TABLE table_name;
```

**3. The ALTER TABLE Statement**

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

Syntax

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

```
ALTER TABLE table_name
```

```
ADD column_name datatype;
```

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name DROP COLUMN column_name;
```

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

```
ALTER TABLE table_name
```

```
MODIFY COLUMN column_name datatype;
```

**4. The RENAME TABLE Statement**

Rename the old table to new table;

**5. Syntax**

Rename old\_tabname to new\_tabname;

**6. The TRUNCATE TABLE Statement**

The ALTER TABLE Statement is used to truncate (delete all rows) a table.

Syntax

To truncate a table, use following syntax: TRUNCATE TABLE table\_name;

**7. CREATE VIEW Statement**

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

Syntax

CREATE VIEW view\_name AS

SELECT column\_name(s)

FROM table\_name

WHERE condition;

SQL Dropping a View

You can delete a view with the DROP VIEW command.

Syntax

DROP VIEW view\_name;

**8. Create Index Statement**

1. Index in SQL is created on existing tables to retrieve the rows quickly. When there are thousands of records in a table, retrieving information will take a long time.

2. Therefore indexes are created on columns which are accessed frequently, so that the information can be retrieved quickly.

3. Indexes can be created on a single column or a group of columns. When an index is created, it first sorts the data and then it assigns a ROWID for each row.

Syntax

CREATE INDEX index\_name

ON table\_name (column\_name1, column\_name2...);

index\_name is the name of the INDEX.

table\_name is the name of the table to which the indexed column belongs.

column\_name1, column\_name2. is the list of columns which make up the INDEX.

**9. Drop Index Statement**

Syntax: DROP INDEX index\_name;

**10. Create Synonym statement**

1. Use the CREATE SYNONYM statement to create a synonym, which is an alternative name

for a table, view,  
sequence, procedure, stored function, package, materialized view.

2. Synonyms provide both data independence and location transparency. Synonyms permit applications to function without modification regardless of which user owns the table or view and regardless of which database holds the table or view.

3. You can refer to synonyms in the following DML statements: SELECT, INSERT, UPDATE, DELETE

Syntax - Create synonym synonym-name for object-name;

Example-Create synonym synonym\_name for table\_name

DML command

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

1) INSERT command

Insert command is used to insert data into a table. Following is its general syntax,

INSERT into table-name values(data1,data2,..)

Lets see an example,

Consider a table Student with following fields.

S\_id S\_Name age

INSERT into Student values(101,'Adam',15);

The above command will insert a record into Student table.

S\_id S\_Name age

101 Adam 15

2) UPDATE command

Update command is used to update a row of a table. Following is its general syntax,

UPDATE table-name set column-name = value where condition;

Let's see an example,

update Student set age=18 where s\_id=102;

Example to Update multiple columns

UPDATE Student set s\_name='Abhi',age=17 where s\_id=103;

3) Delete command

Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row. Following is its general syntax,

DELETE from table-name;

Example to Delete all Records from a Table

DELETE from Student;

The above command will delete all the records from Student table.

Example to Delete a particular Record from a Table

Consider Student table

DELETE from Student where s\_id=103;

## SQL Functions

SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc. SQL functions are divided into two categories,

- **Aggregate Functions**
- **Scalar Functions**

### **Aggregate Functions**

These functions return a single value after calculating from a group of values. Following are some frequently used Aggregate functions.

#### 1) AVG ()

Average returns average value after calculating from values in a numeric column.

Its general Syntax is,

SELECT AVG (column\_name) from table\_name

e.g. SELECT avg(salary) from Emp;

#### 2) COUNT ()

Count returns the number of rows present in the table either based on some condition or without condition. Its general Syntax is,

SELECT COUNT (column\_name) from table-name;

Example using COUNT ()

Consider following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000

SQL query to count employees, satisfying specified condition is,

SELECT COUNT (name) from Emp where salary = 8000;

#### 3) FIRST ()

First function returns first value of a selected column

Syntax for FIRST function is,

SELECT FIRST (column\_name) from table-name

SQL query

SELECT FIRST (salary) from Emp;

#### 4) LAST ()

LAST return the return last value from selected column

Syntax of LAST function is,

SELECT LAST(column\_name) from table-name

SQL query will be,

SELECT LAST(salary) from emp;

#### 5) MAX()

MAX function returns maximum value from selected column of the table.

Syntax of MAX function is,

SELECT MAX(column\_name) from table-name

SQL query to find Maximum salary is,

SELECT MAX(salary) from emp;

#### 6) MIN()

MIN function returns minimum value from a selected column of the table.

Syntax for MIN function is,

SELECT MIN(column\_name) from table-name

SQL query to find minimum salary is,

SELECT MIN(salary) from emp;

**7) SUM()**

SUM function returns total sum of a selected columns numeric values.

Syntax for SUM is,

SELECT SUM(column\_name) from table-name

SQL query to find sum of salaries will be,

SELECT SUM(salary) from emp;

**Scalar Functions**

Scalar functions return a single value from an input value. Following are some frequently used Scalar Functions.

**1) UCASE()**

UCASE function is used to convert value of string column to Uppercase character.

Syntax of UCASE,

SELECT UCASE(column\_name) from table-name

Example of UCASE()

SQL query for using UCASE is,

SELECT UCASE(name) from emp;

**2) LCASE()**

LCASE function is used to convert value of string column to Lowecase character.

Syntax for LCASE is:

SELECT LCASE(column\_name) from table-name

**3) MID()**

MID function is used to extract substrings from column values of string type in a table.

Syntax for MID function is:

SELECT MID(column\_name, start, length) from table-name

**4) ROUND()**

ROUND function is used to round a numeric field to number of nearest integer. It is used on Decimal point values. Syntax of Round function is,

SELECT ROUND(column\_name, decimals) from table-name

**Operators:**

AND and OR operators are used with Where clause to make more precise conditions for fetching data from database by combining more than one condition together.

**1) AND operator**

AND operator is used to set multiple conditions with Where clause.

Example of AND

SELECT \* from Emp WHERE salary < 10000 AND age > 25

**2) OR operator**

OR operator is also used to combine multiple conditions with Where clause. The only difference between AND and OR is their behavior. When we use AND to combine two or more than two conditions, records satisfying all the condition will be in the result. But in case of OR, at least one condition from the conditions specified must be satisfied by any record to be in the result



Example of OR

```
SELECT * from Emp WHERE salary > 10000 OR age  
> 25
```

Set Operation in SQL

SQL supports few Set operations to be performed on table data. These are used to get meaningful

results from data, under different special conditions.

### 3) Union

UNION is used to combine the results of two or more Select statements. However, it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables.

Example of UNION

```
select * from First  
UNION  
select * from second
```

### 4) Union All

This operation is similar to Union. But it also shows the duplicate rows. Union All query will be like,

```
select * from  
First UNION  
ALL  
select * from second
```

### 5) Intersect

Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of Intersect the number of columns and datatype must be same. MySQL does not support INTERSECT operator.

Intersect query will be,

```
select * from First  
INTERSECT  
select * from second
```

### 6) Minus

Minus operation combines result of two Select statements and return only those result which belongs to first set of result. MySQL does not support INTERSECT operator.

Minus query will be,

```
select * from First  
MINUS  
select * from second
```

## 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 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 tablename 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  
ROWBEGIN  
--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  
ROWBEGIN  
--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 data 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. whatever 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> 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> _
```

Below is screen shot of output of Trigger After Insert fired on table Employee\_Detail

```
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
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 a 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> _

```

## 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','assitiant 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','systemadmin',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);
```

**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> _
```

**Concept of Normalization**

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

**What is Normalization?**

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

**Why do we need Normalization?**

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that help to guide you in creating a good database structure.

**Data modification anomalies can be categorized into three types:**

- Insertion Anomaly: Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- Deletion Anomaly: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- Update Anomaly: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

**Advantages of Normalization**

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

**Disadvantages of Normalization**

- You cannot start building the database before knowing what the user needs.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems

## Concepts of MongoDB

**MongoDB**, the most popular NoSQL database, is an open-source document-oriented database. The term 'NoSQL' means 'non-relational'. It means that MongoDB isn't based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data. This format of storage is called BSON (similar to JSON format).

SQL databases store data in tabular format. This data is stored in a predefined data model which is not very much flexible for today's real-world highly growing applications. **Modern applications are more networked, social and interactive than ever.** Applications are storing more and more data and are accessing it at higher rates.

Relational Database Management System(RDBMS) is **not the correct choice when it comes to handling big data by the virtue of their design since they are not horizontally scalable.** If the database runs on a single server, then it will reach a scaling limit. NoSQL databases are more scalable and provide superior performance. MongoDB is such a NoSQL database that scales by adding more and more servers and increases productivity with its flexible document model.

### **RDBMS vs MongoDB:**

- RDBMS has a typical schema design that shows number of tables and the relationship between these tables whereas MongoDB is document-oriented. There is no concept of schema or relationship.
- Complex transactions are not supported in MongoDB because complex join operations are not available.
- MongoDB allows a highly flexible and scalable document structure. For example, one data document of a collection in MongoDB can have two fields whereas the other document in the same collection can have four.
- MongoDB is faster as compared to RDBMS due to efficient indexing and storage techniques.
- There are a few terms that are related in both databases. What's called Table in RDBMS is called a Collection in MongoDB. Similarly, a Row is called a Document and a Column is called a Field. MongoDB provides a default '\_id' (if not provided explicitly) which is a 12-byte hexadecimal number that assures the uniqueness of every document. It is similar to the Primary key in RDBMS.

### **Features of MongoDB:**

- **Document Oriented:** MongoDB stores the main subject in the minimal number of documents and not by breaking it up into multiple relational structures like RDBMS. For example, it stores all the information of a computer in a single document called Computer and not in distinct relational structures like CPU, RAM, Hard disk, etc.
- **Indexing:** Without indexing, a database would have to scan every document of a collection to select those that match the query which would be inefficient. So, for efficient searching Indexing is a must and MongoDB uses it to process huge volumes of data in very less time.

- **Scalability:** MongoDB scales horizontally using sharing (partitioning data across various servers). Data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers. Also, new machines can be added to a running database.
- **Replication and High Availability:** MongoDB increases the data availability with multiple copies of data on different servers. By providing redundancy, it protects the database from hardware failures. If one server goes down, the data can be retrieved easily from other active servers which also had the data stored on them.
- **Aggregation:** Aggregation operations process data records and return the computed results. It is similar to the GROUPBY clause in SQL. A few aggregation expressions are sum, avg, min, max, etc

### **Where do we use MongoDB?**

MongoDB is preferred over RDBMS in the following scenarios:

- **Big Data:** If you have huge amount of data to be stored in tables, think of MongoDB before RDBMS databases. MongoDB has built-in solution for partitioning and sharing your database.
- **Unstable Schema:** Adding a new column in RDBMS is hard whereas MongoDB is schema-less. Adding a new field does not affect old documents and will be very easy.
- **Distributed data** Since multiple copies of data are stored across different servers, recovery of data is instant and safe even if there is a hardware failure.

## **Lab Experiments:**

### **Program 1:**

**Create a table called Employee & execute the following.**

**Employee (EMPNO, ENAME, JOB, MANAGER\_NO, SAL, COMMISSION)**

#### **1. Create a user and grant all permissions to the user**

This step can differ significantly based on the database system being used. Below is an example.

```
drop user abc@localhost;
```

```
flush privileges;
```

```
create user abc@localhost identified by 'abc'
```

```
GRANT ALL PRIVILEGES ON *.* TO 'abc'@'localhost' WITH GRANT OPTION;
```

```
FLUSH PRIVILEGES;
```

**2. Insert any three records in the employee table contains attributes EMPNO, ENAME JOB, MANAGER\_NO, SAL, COMMISSION and use rollback. Check the result.**

```
CREATE DATABASE Employee
```

```
USE Employee
```

```
CREATE TABLE Employee (
```

```
EMPNO INT ,
```

```
ENAME VARCHAR(255) NOT NULL,
```

```
JOB VARCHAR(255) NOT NULL,
```

```
MANAGER_NO INT NOT NULL, SAL
```

```
DECIMAL(10, 2) NOT NULL,
```

```
COMMISSION DECIMAL(10, 2) NOT NULL);
```

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (1, 'Ajay Kumar', 'Manager', 2, 5000.00, 1000.00);
```

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (2, 'Amith Kumar', 'HRManager', 4, 7000.00, 3000.00);
```

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (3, 'Aman Kumar', 'Tester', 8, 3000.00, 2000.00);
```

```
SELECT * FROM Employee;
```

364 • `SELECT * FROM Employee;`

365 `START A NEW TRANSACTION`

Result Grid

	EMPNO	ENAME	JOB	MANAGER_NO	SAL	COMMISSION
▶	1	Ajay Kumar	Manager	2	5000.00	1000.00
	2	Amith Kumar	HRManager	4	7000.00	3000.00
	3	Aman Kumar	Tester	8	3000.00	2000.00
*	NULL	NULL	NULL	NULL	NULL	NULL

```
alter table Employee engine=innodb;
```

```
-- START A TRANSACTION
```

```
START TRANSACTION;
```

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (1, 'Ajay Kumar', 'Manager', 2, 5000.00, 1000.00);
```

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (2, 'Amith Kumar', 'HRManager', 4, 7000.00, 3000.00);
```

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (3, 'Aman Kumar', 'Tester', 8, 3000.00, 2000.00);
```

```
COMMIT;
```

```
SELECT * FROM Employee;
```

364 • `SELECT * FROM Employee;`

365 `START A NEW TRANSACTION`

Result Grid

	EMPNO	ENAME	JOB	MANAGER_NO	SAL	COMMISSION
▶	1	Ajay Kumar	Manager	2	5000.00	1000.00
	2	Amith Kumar	HRManager	4	7000.00	3000.00
	3	Aman Kumar	Tester	8	3000.00	2000.00
*	NULL	NULL	NULL	NULL	NULL	NULL



```
-- START ANOTHER TRANSACTION
START TRANSACTION;
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES (4, 'Pradee Kumar', 'FinanceManager', 5, 8000.00, 4000.00);
UPDATE Employee SET ENAME='Sanjay Kumar' where EMPNO=2;
DELETE FROM Employee where ENAME = 'Ajay Kumar';
-- ROLLBACK 2 INSERTS AND 1 DELETE OPERATIONS
ROLLBACK;
```

```
SELECT * FROM Employee;
```

EMPNO	ENAME	JOB	MANAGER_NO	SAL	COMMISSION
1	Ajay Kumar	Manager	2	5000.00	1000.00
2	Amith Kumar	HRManager	4	7000.00	3000.00
3	Aman Kumar	Tester	8	3000.00	2000.00
NULL	NULL	NULL	NULL	NULL	NULL

### 3. Add primary key constraint and not null constraint to the employee table.

Field	Type	Null	Key	Default	Extra
EMPNO	int	YES		NULL	
ENAME	varchar(255)	YES		NULL	
JOB	varchar(255)	YES		NULL	
MANAGER_NO	int	YES		NULL	
SAL	decimal(10,2)	YES		NULL	
COMMISSION	decimal(10,2)	YES		NULL	

```
ALTER TABLE Employee ADD PRIMARY KEY( EMPNO)
```

Field	Type	Null	Key	Default	Extra
EMPNO	int	NO	PRI	NULL	
ENAME	varchar(255)	YES		NULL	
JOB	varchar(255)	YES		NULL	
MANAGER_NO	int	YES		NULL	
SAL	decimal(10,2)	YES		NULL	
COMMISSION	decimal(10,2)	YES		NULL	

## NOT NULL CONSTRAINT

**ALTER TABLE Employee MODIFY ENAME VARCHAR(255) NOT NULL**

Result Grid						
		Filter Rows:			Export:	Wrap Cell Content: <a href="#">IA</a>
	Field	Type	Null	Key	Default	Extra
▶	EMPNO	int	NO	PRI	NULL	
	ENAME	varchar(255)	NO		NULL	
	JOB	varchar(255)	YES		NULL	
	MANAGER_NO	int	YES		NULL	
	SAL	decimal(10,2)	YES		NULL	
	COMMISSION	decimal(10,2)	YES		NULL	

INSERT INTO Employee (EMPNO,ENAME,JOB,MANAGER\_NO,SAL,COMMISSION)

VALUES (4, NULL, 'clerk,3, 50000, 20)

**Error Code: 1048. Column 'ENAME' cannot be null 0.000 sec**

### 4.Insert null values to the employee table and verify the result.

INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER\_NO, SAL, COMMISSION)

VALUES (5, 'Hemath', NULL,NULL, NULL, null);

SELECT \* FROM Employee

Result Grid						
		Filter Rows:			Edit:	Export/Import:
	EMPNO	ENAME	JOB	MANAGER_NO	SAL	COMMISSION
▶	4	Nitin	NULL	3	50000.00	20.00
	5	Hemath	NULL	NULL	NULL	NULL
✱	NULL	NULL	NULL	NULL	NULL	NULL

**Program 2**

**Create a table called Employee that contain attributes EMPNO, ENAME, JOB, MGR, SAL & execute the following.**

**Create the Employee table:**

```
CREATE TABLE Employee (  
    EMPNO INT,  
    ENAME VARCHAR(100),  
    JOB VARCHAR(50),  
    MGR INT,  
    SAL DECIMAL(10, 2)  
);
```

**1. Add a column commission with domain to the Employee table.**

Assuming that you want to set a domain for the COMMISSION which typically means defining a specific range or type, standard doesn't directly support the "DOMAIN" keyword outside of Postgre. However, you can ensure only positive values are entered by using a CHECK constraint:

```
ALTER TABLE Employee  
ADD COMMISSION DECIMAL(10, 2) CHECK (COMMISSION >= 0);
```

**2. Insert any five records into the table.**

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MGR, SAL, COMMISSION) VALUES  
(101, 'John Doe', 'Manager', NULL, 50000.00, 1500.00),  
(102, 'Jane Smith', 'Developer', 101, 45000.00, 1000.00),  
(103, 'Eric Johnson', 'Analyst', 101, 40000.00, 800.00),  
(104, 'Mary Jane', 'Designer', 103, 35000.00, 500.00),  
(105, 'Alice Brown', 'Tester', 104, 30000.00, 300.00);
```

22

23 • `SELECT * FROM Employee`

24

25

26

27

Result Grid						
		Filter Rows:		Export:		Wrap Cell Content: <a href="#">IA</a>
	EMPNO	ENAME	JOB	MGR	SAL	COMMISSION
▶	101	John Doe	Manager	NULL	50000.00	1500.00
	102	Jane Smith	Developer	101	45000.00	1000.00
	103	Eric Johnson	Analyst	101	40000.00	800.00
	104	Mary Jane	Designer	103	35000.00	500.00
	105	Alice Brown	Tester	104	30000.00	300.00

### 3. Update the column details of the job

To update the job title for an employee, you would use an UPDATE statement. For instance, if you wanted to change the job title of all testers to 'Quality Analyst':

UPDATE Employee

SET JOB = 'Quality Analyst'

WHERE EMPNO = '105';

36 `UPDATE Employee SET JOB = 'Quality Analyst' WHERE EMPNO = 105;`

37 • `SET SQL_SAFE_UPDATES=0;`

38 • `SELECT * FROM Employee`

Result Grid						
		Filter Rows:		Export:		Wrap Cell Content: <a href="#">IA</a>
	EMPNO	ENAME	JOB	MGR	SAL	COMMISSION
▶	101	John Doe	Manager	NULL	50000.00	1500.00
	102	Jane Smith	Developer	101	45000.00	1000.00
	103	Eric Johnson	Analyst	101	40000.00	800.00
	104	Mary Jane	Designer	103	35000.00	500.00
	105	Alice Brown	Quality Analyst	104	30000.00	300.00

### 4. Rename a column of the Employee table using the ALTER command

Suppose we want to rename the column MGR to MANAGER\_ID:

```

ALTER TABLE Employee
RENAME COLUMN MGR TO MANAGER_ID;
    
```

31

•

ALTER TABLE Employee

32

RENAME COLUMN MGR TO MANAGER\_ID;

33

34

•

SELECT \* FROM Employee

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	EMPNO	ENAME	JOB	MANAGER_ID	SAL	COMMISSION
▶	101	John Doe	Manager	NULL	50000.00	1500.00
	102	Jane Smith	Developer	101	45000.00	1000.00
	103	Eric Johnson	Analyst	Analyst	40000.00	800.00
	104	Mary Jane	Designer	103	35000.00	500.00
	105	Alice Brown	Quality Analyst	104	30000.00	300.00

5. Delete the employee whose EMPNO is 105

```

DELETE FROM Employee
WHERE EMPNO = 105;
    
```

This will remove the record for the employee with EMPNO 105 from the Employee table.

41

✖

DELETE FROM Employee

42

WHERE EMPNO = 105;

43

44

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	EMPNO	ENAME	JOB	MGR	SAL	COMMISSION
▶	101	John Doe	Manager	NULL	50000.00	1500.00
	102	Jane Smith	Developer	101	45000.00	1000.00
	103	Eric Johnson	Analyst	101	40000.00	800.00
	104	Mary Jane	Designer	103	35000.00	500.00

### Lab Program 3

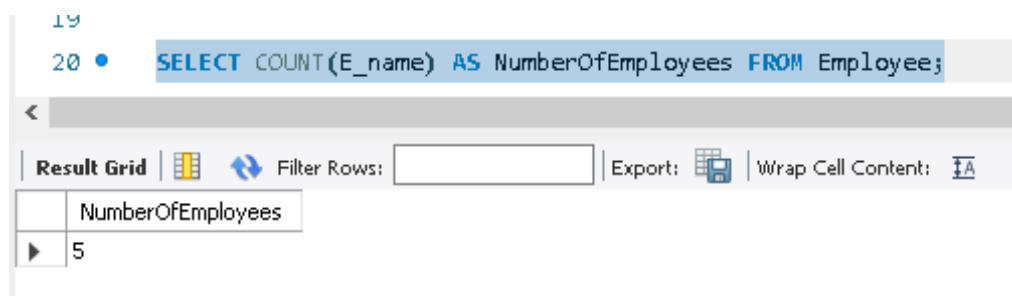
Queries using aggregate functions(COUNT,AVG,MIN,MAX,SUM),Group by,Orderby.  
Employee(E\_id, E\_name, Age, Salary)

#### 1.Create Employee table containing all Records E\_id, E\_name, Age, Salary.

```
CREATE TABLE Employee  
(E_id INT,  
E_name VARCHAR(100),Age INT,  
Salary DECIMAL(10, 2)  
);
```

#### 2. Count number of employee names from employeetable

```
SELECT COUNT(E_name) AS NumberOfEmployees FROM Employee;
```

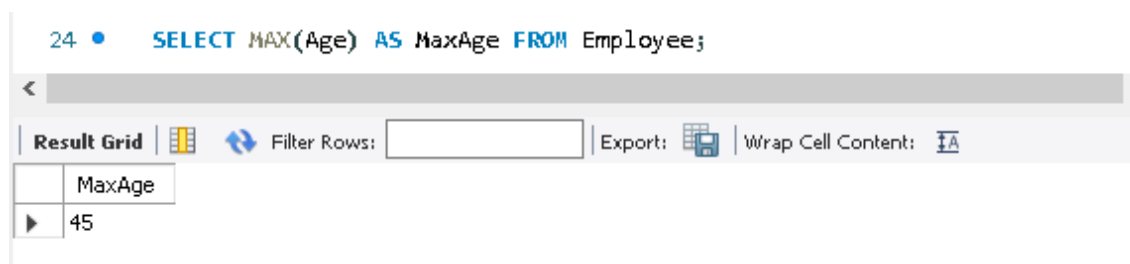


The screenshot shows a database query interface. The query editor displays the SQL statement: `SELECT COUNT(E_name) AS NumberOfEmployees FROM Employee;`. Below the editor, the 'Result Grid' tab is active, showing a single row with the column header 'NumberOfEmployees' and the value '5'. The interface includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox.

NumberOfEmployees
5

#### 3. Find the Maximum age from employee table.

```
SELECT MAX(Age) AS MaxAge FROM Employee;
```



The screenshot shows a database query interface. The query editor displays the SQL statement: `SELECT MAX(Age) AS MaxAge FROM Employee;`. Below the editor, the 'Result Grid' tab is active, showing a single row with the column header 'MaxAge' and the value '45'. The interface includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox.




MaxAge
45

#### 4. Find the Minimum age from employeetable.

SELECT MIN(Age) AS MinAge FROM Employee;

25 • SELECT MIN(Age) AS MinAge FROM Employee;

<

Result Grid  Filter Rows:  Export:  Wrap Cell Content: 




	MinAge
▶	25

#### 5. Find salaries of employee in Ascending Order.

SELECT Salary FROM Employee ORDER BY Salary ASC;

27 • SELECT Salary FROM Employee ORDER BY Salary ASC;

<

Result Grid  Filter Rows:  Export:  Wrap Cell Content: 




	Salary
▶	45000.00
	50000.00
	60000.00
	70000.00
	80000.00

#### 6. Find Grouped Salaries of Employees

SELECT Salary, COUNT(\*) AS  
NumberOfEmployees FROM Employee  
GROUP BY Salary;

29 • SELECT Salary, COUNT(\*) AS NumberOfEmployees  
30 FROM Employee  
31 GROUP BY Salary;  
32

<

Result Grid  Filter Rows:  Export:  Wrap Cell Content: 

	Salary	NumberOfEmployees
▶	50000.00	1
	60000.00	1
	70000.00	1
	45000.00	1
	80000.00	1

**Insertion Commands**

```
INSERT INTO Employee (E_id, E_name, Age, Salary) VALUES
(1, 'John', 30, 50000.00),
(2, 'Jane', 35, 60000.00),
(3, 'David', 40, 70000.00),
(4, 'Emily', 25, 45000.00),
(5, 'Michael', 45, 80000.00);
```

---

**Lab Program 4**

**Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary.**

**CUSTOMERS(ID,NAME,AGE,ADDRESS,SALARY)**

```
create database CUSTOMERS
USE CUSTOMERS;
```

```
CREATE TABLE CUSTOMERS (
    ID INT PRIMARY KEY ,
    NAME VARCHAR(255),
    AGE INT,
    ADDRESS VARCHAR(255),
    SALARY DECIMAL(10, 2)
);
```

```
INSERT INTO CUSTOMERS VALUES (1, 'Ajay Hegde', 50, 'Mysuru',50000.00),
INSERT INTO CUSTOMERS VALUES (2, 'Satynaryana Rao ', 65,'Mandya', 45000.00),
INSERT INTO CUSTOMERS VALUES (3, 'Ravishankar', 35,'Shivmogga', 60000.00) ,
INSERT INTO CUSTOMERS VALUES (4, 'Anish Behl', 28,'Manglore', 80000.00) ,
INSERT INTO CUSTOMERS VALUES (5, 'Hari das Upadaya', 28,'Udupi', 90000.00);
```



SELECT \* FROM CUSTOMERS

17 SELECT \* FROM CUSTOMERS

ID	NAME	AGE	ADDRESS	SALARY
1	Ajay Hegde	50	Mysuru	50000.00
2	Satynaryana Rao	65	Mandya	45000.00
3	Ravishankar	35	Shivmogga	60000.00
4	Anish Behl	28	Manglore	80000.00
5	Hari das Upadaya	28	Udupi	90000.00
NULL	NULL	NULL	NULL	NULL

DELIMITER //

CREATE TRIGGER after\_insert\_salary\_difference

AFTER INSERT ON CUSTOMERS

FOR EACH ROW

BEGIN

SET @my\_sal\_diff = CONCAT('salary inserted is ', NEW.SALARY);

END; //

DELIMITER ;

INSERT INTO CUSTOMERS VALUES (6, 'amit kumar', 38, 'bijapur', 90000.00)

select @my\_sal\_diff as SALARY\_INSERTED

## OUTPUT

SALARY_INSERTED
salary inserted is 90000.00

```
DELIMITER //
CREATE TRIGGER after_update_salary_difference
AFTER UPDATE ON CUSTOMERS
FOR EACH ROW
BEGIN
    DECLARE old_salary DECIMAL(10, 2);
    DECLARE new_salary DECIMAL(10, 2);
    SET old_salary = OLD.SALARY;
    SET new_salary = NEW.SALARY;
    SET @my_sal_diff = CONCAT('salary difference after update is ', NEW.SALARY -
    OLD.SALARY);
END;//
DELIMITER ;
```

**OUTPUT:**

```
INSERT INTO CUSTOMERS VALUES (6, 'amit kumar', 38, 'bijapur', 90000.00)
UPDATE CUSTOMERS SET SALARY=95000 WHERE ID=6
SELECT @my_sal_diff AS salary_diffrence
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	salary_diffrence			
▶	salary difference after update is 5000.00			

```
-- DELETE TRIGGER
```

```
DELIMITER //
```

```
CREATE TRIGGER after_delete_salary_difference
```

```
AFTER DELETE ON CUSTOMERS
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    SET @my_sal_diff = CONCAT('salary deleted is ', OLD.SALARY);
```

```
END; //
```

```
DELIMITER ;
```

```
select * from CUSTOMERS
```

Result Grid					
		Filter Rows:			
		Edit:			
		Export/Import:			
		Wrap Cell Content:			
ID	NAME	AGE	ADDRESS	SALARY	
1	Ajay Hegde	50	Mysuru	50000.00	
2	Satynaryana Rao	65	Mandya	45000.00	
3	Ravishankar	35	Shivmogga	60000.00	
4	Anish Behl	28	Manglore	80000.00	
5	Hari das Upadaya	28	Udupi	90000.00	
6	amit kumar	38	bijapur	95000.00	
NULL	NULL	NULL	NULL	NULL	

```
delete from CUSTOMERS where ID=1
```

```
SELECT @my_sal_diff AS Deleted_Salary
```

Result Grid	
Filter Rows:	
Export:	
Wrap Cell Content:	
Deleted_Salary	
salary deleted is 50000.00	

**Lab Program 5**

**Create cursor for Employee table & extract the values from the table. Declare the variables ,Open the cursor & extract the values from the cursor. Close the cursor.**

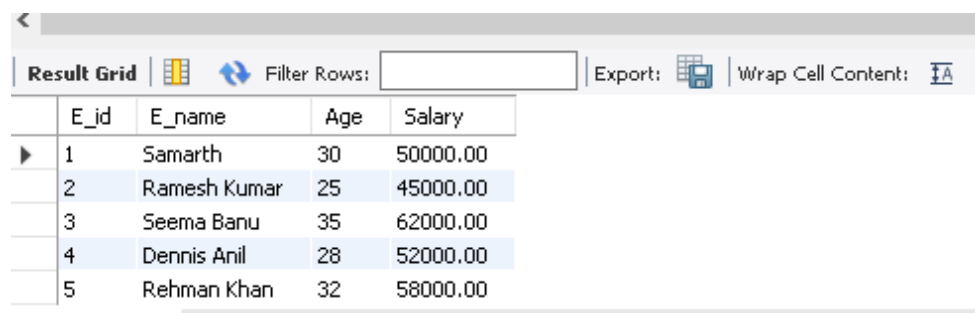
**Employee(E\_id, E\_name, Age, Salary)**

```
CREATE TABLE Employee (  
    E_id INT,  
    E_name VARCHAR(255),  
    Age INT,  
    Salary DECIMAL(10, 2)  
);
```

```
INSERT INTO Employee (E_id, E_name, Age, Salary)  
VALUES
```

```
(1, 'Samarth', 30, 50000.00),  
(2, 'Ramesh Kumar', 25, 45000.00),  
(3, 'Seema Banu', 35, 62000.00),  
(4, 'Dennis Anil', 28, 52000.00),  
(5, 'Rehman Khan', 32, 58000.00);
```

```
SELECT * FROM Employee
```



The screenshot shows a database interface with a 'Result Grid' tab. It displays the data from the 'Employee' table. The grid has columns for E\_id, E\_name, Age, and Salary. There are five rows of data. The interface also includes a 'Filter Rows' search bar, an 'Export' button, and a 'Wrap Cell Content' checkbox.

	E_id	E_name	Age	Salary
▶	1	Samarth	30	50000.00
	2	Ramesh Kumar	25	45000.00
	3	Seema Banu	35	62000.00
	4	Dennis Anil	28	52000.00
	5	Rehman Khan	32	58000.00

```
DELIMITER $$
CREATE PROCEDURE gettable (
INOUT Tableontents varchar(4000)
)
BEGIN
DECLARE finished INTEGER DEFAULT 0;
DECLARE content varchar(100) DEFAULT "";
#Cursor declaration
DECLARE curName
CURSOR FOR
SELECT concat(E_id , ' , ' , E_name, Age , ' , ' , Salary) FROM Employee order by E_id desc;

#declare NOT FOUND handler
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
#Open cursor
OPEN curName;#fetch
records
getName: LOOP
FETCH curName INTO content;
IF finished = 1 THEN LEAVE getName;END IF;
SET Tableontents = CONCAT(content,";",Tableontents);END LOOP
getName;
CLOSE curName;
END$$
DELIMITER ;
SET @Tableontents = "";
CALL gettable(@Tableontents);
SELECT @Tableontents;
```

**Output**

```
'1 , Samarth30 , 50000.00;
2 , Ramesh Kumar25 , 45000.00;
3 , Seema Banu35 , 62000.00;
4 , Dennis Anil28 , 52000.00;
5 , Rehman Khan32 , 58000.00;'
```

**Lab Program 6**

**Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N\_RollCall with the data available in the table O\_RollCall. If the data in the first table already exist in the second table then that data should be skipped.**

Below is an example of a PL/SQL block of code that uses a parameterized cursor to merge data from the N\_RollCall table into the O\_RollCall table, skipping records that already exist in the second table.

Let's start with the table schema and insertion commands:

Table Schema

```
CREATE DATABASE ROLLCALL;  
USE ROLLCALL;
```

```
-- Create N_RollCall table
```

```
CREATE TABLE N_RollCall (  
  student_id INT PRIMARY KEY,  
  student_name VARCHAR(255),  
  birth_date DATE  
);
```

```
-- Create O_RollCall table with common data
```

```
CREATE TABLE O_RollCall (  
  student_id INT PRIMARY KEY,  
  student_name VARCHAR(255),  
  birth_date DATE  
);
```

```
-- Insert common data into O_RollCall
```

```
INSERT INTO O_RollCall (student_id, student_name, birth_date) VALUES  
(1, Amith, '1995-08-15'),  
(2, Ajay, '1990-12-10');
```

SELECT \* FROM O\_RollCall

66  
67 **SELECT \* FROM O\_RollCall**

**Result Grid** | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	student_id	student_name	birth_date
▶	1	Amith	1995-08-15
	2	Ajay	1990-12-10
*	NULL	NULL	NULL

INSERT INTO N\_RollCall (student\_id, student\_name,  
birth\_date)VALUES  
(1, 'Amith', '1995-08-15'), -- Common record with O\_RollCall  
(2, 'Ajay', '1990-12-10'),  
(3, 'Ravi', '1990-12-10'), -- Common record with O\_RollCall  
(4, 'Rakesh', '2000-05-18'),  
(5, 'Vinay', '1997-09-03');

SELECT \* FROM N\_RollCall

68 **SELECT \* FROM N\_RollCall**  
69  
70

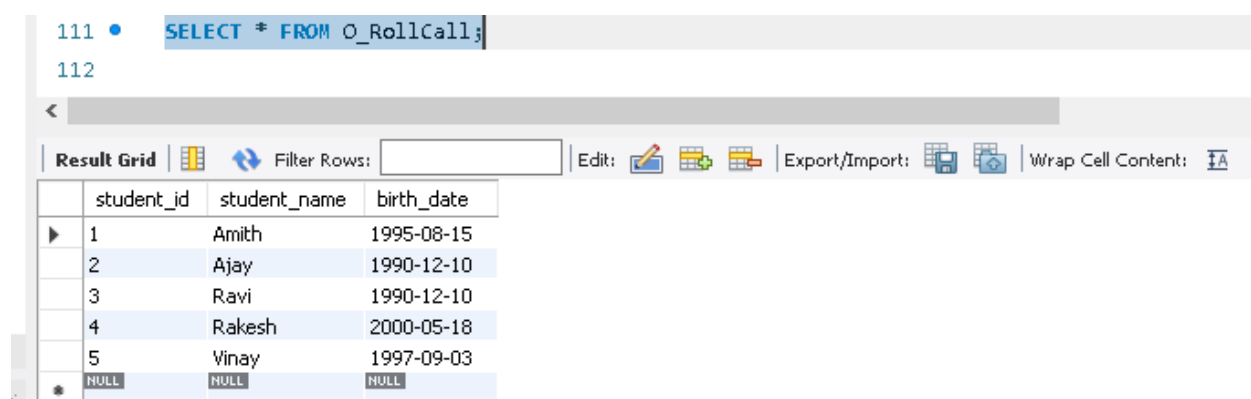
**Result Grid** | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	student_id	student_name	birth_date
▶	1	Amith	1995-08-15
	2	Ajay	1990-12-10
	3	Ravi	1990-12-10
	4	Rakesh	2000-05-18
	5	Vinay	1997-09-03
*	NULL	NULL	NULL

```
DELIMITER //
CREATE PROCEDURE merge_rollcall_data()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE n_id INT;
    DECLARE n_name VARCHAR(255);
    DECLARE n_birth_date DATE;
    -- Declare cursor for N_RollCall table
    DECLARE n_cursor CURSOR FOR
        SELECT student_id, student_name, birth_date
        FROM N_RollCall;
    -- Declare handler for cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET done = TRUE;
    -- Open the cursor
    OPEN n_cursor;
    -- Start looping through cursor results
    cursor_loop: LOOP
        -- Fetch data from cursor into variables
        FETCH n_cursor INTO n_id, n_name, n_birth_date;
        -- Check if no more rows to fetch
        IF done THEN
            LEAVE cursor_loop;
        END IF;
        -- Check if the data already exists in O_RollCall
        IF NOT EXISTS (
            SELECT 1
            FROM O_RollCall
            WHERE student_id = n_id
        ) THEN
            -- Insert the record into O_RollCall
            INSERT INTO O_RollCall (student_id, student_name, birth_date)
            VALUES (n_id, n_name, n_birth_date);
        END IF;
    END LOOP;
END LOOP;
```



```
-- Close the cursor  
CLOSE n_cursor;  
END//  
DELIMITER ;  
  
CALL merge_rollcall_data();  
SELECT * FROM O_RollCall;
```



The screenshot shows a database query result grid. The query executed is `SELECT * FROM O_RollCall;`. The result is displayed in a table with the following columns: `student_id`, `student_name`, and `birth_date`. The table contains five rows of data, followed by a row with three NULL values. The interface includes a toolbar with options like 'Filter Rows', 'Edit', 'Export/Import', and 'Wrap Cell Content'.

	student_id	student_name	birth_date
▶	1	Amith	1995-08-15
	2	Ajay	1990-12-10
	3	Ravi	1990-12-10
	4	Rakesh	2000-05-18
	5	Vinay	1997-09-03
*	NULL	NULL	NULL

This PL/SQL block iterates through each record in the N\_RollCall table, checks if the record exists in the O\_RollCall table based on the ID, and inserts it into O\_RollCall if it doesn't already exist.

## Lab Program 7

**Install an Open Source NoSQL Data base MangoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MangoDB basic Queries using CRUD operations.**

**let's go through the steps to install MongoDB and perform basic CRUD operations.**

### 1. Install MongoDB

Ubuntu

bash

sudo apt update

sudo apt install -y mongodb

CentOS/RHEL

bash

sudo yum install -y mongodb-org

macOS

You can use Homebrew:

bash

brew tap mongodb/brew

brew install mongodb-community

### 2. Start MongoDB Service

Ubuntu

bash

sudo systemctl start mongodb

CentOS/RHEL

bash

sudo systemctl start mongod

macOS

bash

brew services start mongodb-community

### 3. Connect to MongoDB

bash

mongo

### 4. Basic CRUD Operations

Launch the MongoDB shell to perform basic CRUD operations.

mongosh

```
C:\Users\Atme>mongosh
Current Mongosh Log ID: 664c2a444b885bdb56cdcdf5
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.2.6
Using MongoDB:      7.0.9
Using Mongosh:      2.2.6

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
The server generated these startup warnings when booting
2024-05-21T10:07:51.896+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----
```

#### Switch to a Database (Optional):

If you want to use a specific database, switch to that database using the **use** command. If the database doesn't exist, MongoDB will create it implicitly when you insert data into it:

**use bookDB**

**switched to db bookDB**

```
test> use bookDB
switched to db bookDB
bookDB> db.createCollection("ProgrammingBooks")
```

### Create the ProgrammingBooks Collection:

To create the **ProgrammingBooks** collection, use the **createCollection()** method. This step is optional because MongoDB will automatically create the collection when you insert data into it, but you can explicitly create it if needed:

```
bookDB> db.createCollection("ProgrammingBooks")
{ ok: 1 }
```

### INSERT operations

#### a. Insert 5 Documents into the ProgrammingBooks Collection :

Now, insert 5 documents representing programming books into the **ProgrammingBooks** collection using the **insertMany()** method:

```
bookDB> db.ProgrammingBooks.insertMany([
...   {
...     title: "Clean Code: A Handbook of Agile Software Craftsmanship",
...     author: "Robert C. Martin",
...     category: "Software Development",
...     year: 2008
...   },
...   {
...     title: "JavaScript: The Good Parts",
...     author: "Douglas Crockford",
...     category: "JavaScript",
...     year: 2008
...   },
...   {
...     title: "Design Patterns: Elements of Reusable Object-Oriented Software",
...     author: "Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides",
...     category: "Software Design",
...     year: 1994
...   },
...   {
...     title: "Introduction to Algorithms",
...     author: "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein",
...     category: "Algorithms",
...     year: 1990
...   },
...   {
...     title: "Python Crash Course: A Hands-On, Project-Based Introduction to Programming",
...     author: "Eric Matthes",
...     category: "Python",
...     year: 2015
...   }
... ])
bookDB>
```

### Insert a Single Document into ProgrammingBooks:

Use the **insertOne()** method to insert a new document into the **ProgrammingBooks** collection:

```
db.ProgrammingBooks.insertOne({  
  title: "The Pragmatic Programmer: Your Journey to Mastery",  
  author: "David Thomas, Andrew Hunt",  
  category: "Software Development",  
  year: 1999  
})
```

```
bookDB> db.ProgrammingBooks.insertOne({  
...   title: "The Pragmatic Programmer: Your Journey to Mastery",  
...   author: "David Thomas, Andrew Hunt",  
...   category: "Software Development",  
...   year: 1999  
... })  
{  
  acknowledged: true,  
  insertedId: ObjectId('664c2b2d4b885bdb56cdcdfb')  
}  
bookDB>
```

## Read (Query) Operations

### a. Find All Documents

To retrieve all documents from the **ProgrammingBooks** collection:

```
db.ProgrammingBooks.find().pretty()
```

```
bookDB> db.ProgrammingBooks.find().pretty()
[
  {
    _id: ObjectId('664c2acc4b885bdb56cdcd46'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('664c2acc4b885bdb56cdcd47'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: ObjectId('664c2acc4b885bdb56cdcd48'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Software Design',
    year: 1994
  },
  {
    _id: ObjectId('664c2acc4b885bdb56cdcd49'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein',
    category: 'Algorithms',
    year: 1990
  },
  {
    _id: ObjectId('664c2acc4b885bdb56cdcd4a'),
    title: 'Python Crash Course: A Hands On, Project Based Introduction to Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  },
  {
    _id: ObjectId('664c2b2c4b885bdb56cdcd4b'),
    title: 'The Pragmatic Programmer: Your Journey to Mastery',
    author: 'David Thomas, Andrew Hunt',
    category: 'Software Development',
    year: 1999
  }
]
```

### Find Documents Matching a Condition

To find books published after the year 2000:

```
db.ProgrammingBooks.find({ year: { $gt: 2000 } }).pretty()
```

```
bookDB> db.ProgrammingBooks.find({ year: { $gt: 2000 } }).pretty()
[
  {
    _id: ObjectId('664c2acd4b885bdb56cdcdf6'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('664c2acd4b885bdb56cdcdf7'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: ObjectId('664c2acd4b885bdb56cdcdfa'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
```

### Update Operations

#### a. Update a Single Document

To update a specific book (e.g., change the author of a book):

```
db.ProgrammingBooks.updateOne(
  { title: "Clean Code: A Handbook of Agile Software Craftsmanship" },
  { $set: { author: "Robert C. Martin (Uncle Bob)" } }
)
```

//verify by displaying books published in year 2008

```
db.ProgrammingBooks.find({ year: { $eq: 2008 } }).pretty()
```

```
bookDB> db.ProgrammingBooks.updateOne(
...   { title: "Clean Code: A Handbook of Agile Software Craftsmanship" },
...   { $set: { author: "Robert C. Martin (Uncle Bob)" } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
bookDB> db.ProgrammingBooks.find({ year: { $eq: 2008 } }).pretty()
[
  {
    _id: ObjectId('664c2acd4b885bdb56cdcdf6'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('664c2acd4b885bdb56cdcdf7'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  }
]
```

### Update Multiple Documents

To update multiple books (e.g., update the category of books published before 2010):

**db.ProgrammingBooks.updateMany(**



```

bookDB> db.ProgrammingBooks.updateMany(
...   { year: { $lt: 2010 } },
...   { $set: { category: "Classic Programming Books" } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
}
bookDB> db.ProgrammingBooks.find({ year: { $lt: 2010 } }).pretty()
[
  {
    _id: ObjectId('664c2acd4b885bdb56cdcdf6'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('664c2acd4b885bdb56cdcdf7'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('664c2acd4b885bdb56cdcdf8'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Classic Programming Books',
    year: 1994
  },
  {
    _id: ObjectId('664c2acd4b885bdb56cdcdf9'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein',
    category: 'Classic Programming Books',
    year: 1990
  },
  {
    _id: ObjectId('664c2b2d4b885bdb56cdcdfb'),
    title: 'The Pragmatic Programmer: Your Journey to Mastery',
    author: 'David Thomas, Andrew Hunt',
    category: 'Classic Programming Books',
    year: 1999
  }
]
bookDB>

```

## Delete Operations

### a. Delete a Single Document

To delete a specific book from the collection (e.g., delete a book by title):

```
db.ProgrammingBooks.deleteOne({ title: "JavaScript: The Good Parts" })
{ acknowledged: true, deletedCount: 1 }
```

```

bookDB> db.ProgrammingBooks.deleteOne({ title: "JavaScript: The Good Parts" })
{ acknowledged: true, deletedCount: 0 }
bookDB>

```

You can check whether the specified document is deleted by displaying the contents of the collection.

### b. Delete Multiple Documents

To delete multiple books based on a condition (e.g., delete all books published before 1995):

```
db.ProgrammingBooks.deleteMany({ year: { $lt: 1995 } })
{ acknowledged: true, deletedCount: 2 }
```

```
bookDB> db.ProgrammingBooks.deleteMany({ year: { $lt: 1995 } })
{ acknowledged: true, deletedCount: 2 }
bookDB>
```

You can check whether the specified documents were deleted by displaying the contents of the collection.

### c. Delete All Documents in the Collection:

To delete all documents in a collection (e.g., **ProgrammingBooks**), use the **deleteMany()** method with an empty filter {}:

```
db.ProgrammingBooks.deleteMany({})
```

```
bookDB> db.ProgrammingBooks.deleteMany({})
{ acknowledged: true, deletedCount: 3 }
bookDB>
```

//verify by displaying the collection

```
db.ProgrammingBooks.find().pretty()
```

Delete the Collection Using drop():

To delete a collection named ProgrammingBooks, use the drop() method with the name of the collection:

```
bookDB> db.ProgrammingBooks.drop()
true
bookDB>

bookDB> show collections
bookDB>
```

### **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. What are Advantages of DBMS?**

- Redundancy is controlled.
- Unauthorized access is restricted.
- Providing multiple user interfaces.
- Enforcing integrity constraints.
- Providing backup and recovery.

**6. What are Disadvantages 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 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.

**21. 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.

**22. What is a query?**

A query with respect to DBMS relates to user commands that are used to interact with a database. The query language can be classified into data definition language and data manipulation language.

**23. What is MongoDB?**

MongoDB is a cross-platform document-based database. Categorized as a NoSQL database, MongoDB avoids the conventional table-oriented relational database structure in support of the JSON-like documents with the dynamic schemas, making the data integration in specific kinds of applications quicker and simpler. MongoDB was developed by a software company “10gen”, in October 2007 as an element of the planned platform as the service product. After that, the

company was shifted to a freeware deployment model in 2009, providing sales assistance and other services.

**24. What are the features of MongoDB?**

Following are the important features of MongoDB:

- A compliant data model in the format of documents.
- Agile and extremely scalable database.
- Quicker than traditional databases.
- Demonstrative query language

**25. What is the use of MongoDB?**

- Generally, we use MongoDB as the main data store for the operational requirements with live needs. Generally, MongoDB is suitable for 80% of the applications which we develop today. MongoDB is simple to operate and extent in ways that are tough if they are not possible with the relational databases.
- MongoDB stands out in various use cases where the relational databases are not suitable, like applications with semi-structured, structured, along with the big scalability needs or the multi-datacenter deployments.
- MongoDB cannot be suitable for some applications. For instance, applications that need complex transactions and scan-based applications that access huge subsets of the data largely cannot be suitable for MongoDB.
- Some general uses of MongoDB comprise product catalogs, mobile apps, content management, real-time personalization, and applications providing individual views throughout several systems.