

ATME COLLEGE OF ENGINEERING

13th KM Stone, Bannur Road, Mysore - 560 028



A T M E

College of Engineering

DEPARTMENT OF CSE-CYBERSECURITY

(ACADEMIC YEAR 2024-25)

LABORATORY MANUAL

SUBJECT: DATABASE MANAGEMENT SYSTEM LABORATORY

SUB CODE: BCS403

SEMESTER: IV-2022 CBCS Scheme

Composed By

Verified By

Approved By

**Mr.MAHADEVASWAMY D.M
PROGRAMMER**

**Mr.Basavaraj G
FACULTY INCHARGE**

**Dr. NASREEN FATHIMA
HOD, CS&DESIGN**

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 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.

Sl.NO	Experiments
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. 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. 3. Add primary key constraint and not null constraint to the employee table. 4. Insert null values to the employee table and verify the result.
	Create a table called Employee that contain attributes EMPNO,ENAME,JOB, MGR,SAL & execute the following. 1. Add a column commission with domain to the Employee table. 2. Insert any five records into the table. 3. Update the column details of job 4. Rename the column of Employ table using alter command. 5. Delete the employee whose Empno is 105
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. 2. Count number of employee names from employeetable 3. Find the Maximum age from employee table. 4. Find the Minimum age from employeetable. 5. Find salaries of employee in Ascending Order. 6. Find grouped salaries of employees.
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)
5	Create cursor for Employee table & extract the values from the table. Declare the variables ,Open the cursor & extrct the values from the cursor. Close the cursor. Employee(E_id, E_name, Age, Salary)
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.
7	Install an Open Source NoSQL Data base MangoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MangoDB basic Queries using CRUD operations.

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

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)

DEPARTMENT OF CSE-CY ATMECE

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;
```

The RENAME TABLE Statement Rename the old table to new table; Syntax
Rename old_tabname to new_tabname;

4. 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;

5. 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;
```

6. 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 a 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;

Lets 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, atleast 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
```

Experiments

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.
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.
3. Add primary key constraint and not null constraint to the employee table.
4. Insert null values to the employee table and verify the result.

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

```
CREATE USER myuser IDENTIFIED BY mypassword;  
GRANT ALL PRIVILEGES ON Employee TO myuser;
```

2. Create the Employee table

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

3. Insert three records into the employee table and rollback

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)  
VALUES  
  (1, 'John Doe', 'Manager', NULL, 5000.00, 1000.00),  
  (2, 'Jane Smith', 'Developer', 1, 4000.00, 500.00),  
  (3, 'Bob Johnson', 'Analyst', 1, 3500.00, NULL);
```

```
ROLLBACK;
```

Check if the records were inserted

```
SELECT * FROM Employee;
```

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

```
ALTER TABLE Employee
ADD CONSTRAINT PK_Employee PRIMARY KEY (EMPNO);
```

```
ALTER TABLE Employee
MODIFY (EMPNO INT NOT NULL,
        ENAME VARCHAR(50) NOT NULL,
        JOB VARCHAR(50) NOT NULL,
        SAL DECIMAL(10, 2) NOT NULL);
```

4. Insert null values into the employee table and verify the result

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL,
COMMISSION) VALUES (4, NULL, 'Intern', NULL, NULL, NULL);
```

```
SELECT * FROM Employee;
```

EMPNO	ENAME	JOB	MANAGER_NO	SAL	COMMISSION
1	John Doe	Manager	NULL	5000	1000
2	Jane Smith	Developer	1	4000	500
3	Bob Johnson	Analyst	1	3500	NULL
4	NULL	Intern	NULL	NULL	NULL

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

- 1. Add a column commission with domain to the Employee table.**
- 2. Insert any five records into the table.**
- 3. Update the column details of job**
- 4. Rename the column of Employ table using alter command.**
- 5. Delete the employee whose Empno is 105.**

1. Create the Employee table with attributes EMPNO, ENAME, JOB, MGR, SAL

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

2. Add a column 'commission' to the Employee table

```
ALTER TABLE Employee  
ADD commission DECIMAL (10, 2);
```

3. Insert five records into the table

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MGR, SAL, commission)  
VALUES  
    (101, 'John Doe', 'Manager', NULL, 5000.00, 1000.00),  
    (102, 'Jane Smith', 'Developer', 101, 4000.00, 500.00),  
    (103, 'Bob Johnson', 'Analyst', 101, 3500.00, NULL),  
    (104, 'Alice Jones', 'Designer', 101, 3800.00, 800.00),  
    (105, 'Michael Brown', 'Engineer', 101, 4200.00, 700.00);
```

4. Update the column details of 'JOB'

```
ALTER TABLE Employee  
MODIFY (JOB VARCHAR(50) NOT NULL);
```

5. Rename the column 'MGR' to 'MANAGER' in the Employee table

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

6. Delete the employee whose Empno is 105

```
DELETE FROM Employee WHERE EMPNO = 105;
```

```
Select*from Employee;
```

OUTPUT:

EMPNO	ENAME	JOB	MANAGER	SAL	commission
101	John Doe	Manager	NULL	5000	1000
102	Jane Smith	Developer	101	4000	500
103	Bob Johnson	Analyst	101	3500	NULL
104	Alice Jones	Designer	101	3800	800

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.**
- 2. Count number of employee names from employee table**
- 3. Find the Maximum age from employee table.**
- 4. Find the Minimum age from employee table.**
- 5. Find salaries of employee in Ascending Order.**
- 6. Find grouped salaries of employees.**

1. Create the Employee table

```
CREATE TABLE Employee (E_id INT, E_name VARCHAR(50), Age INT, Salary  
DECIMAL(10, 2));
```

2. Insert records into the Employee table

```
INSERT INTO Employee (E_id, E_name, Age, Salary) VALUES  
(1, 'John Doe', 30, 50000.00),  
(2, 'Jane Smith', 25, 45000.00),  
(3, 'Bob Johnson', 35, 60000.00),  
(4, 'Alice Jones', 28, 52000.00),  
(5, 'Michael Brown', 32, 55000.00);
```

```
Select * from Employee;
```

3. Count number of employee names from employee table

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

5

4. Find the Maximum age from employee table

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

35

5. Find the Minimum age from employee table

```
SELECT MIN(Age) AS min_age FROM Employee;
```

25

6. Find salaries of employee in Ascending Order

```
SELECT E_name, Salary FROM Employee ORDER BY Salary ASC;
```

Jane Smith	45000.00
John Doe	50000.00
Alice Jones	52000.00
Michael Brown	55000.00
Bob Johnson	60000.00

7. Find grouped salaries of employees

```
SELECT Salary, COUNT(*) AS num_employees FROM Employee GROUP BY Salary;
```

50000.00	1
45000.00	1
60000.00	1
52000.00	1
55000.00	1

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 the customers table

```
CREATE TABLE customers (  
    ID INT PRIMARY KEY,  
    NAME VARCHAR(50),  
    AGE INT,  
    ADDRESS VARCHAR(100),  
    SALARY DECIMAL(10, 2)  
);
```

```
-- Create a sequence for trigger  
CREATE SEQUENCE salary_diff_seq;
```

```
-- Create the trigger  
CREATE OR REPLACE TRIGGER salary_diff_trigger  
AFTER INSERT OR UPDATE OR DELETE ON customers  
FOR EACH ROW
```

```
DECLARE
```

```
    old_salary DECIMAL(10, 2);  
    new_salary DECIMAL(10, 2);  
    salary_diff DECIMAL(10, 2);
```

```
BEGIN
```

```
-- Get the old and new salary values  
IF INSERTING OR UPDATING THEN  
    old_salary := NVL(:OLD.SALARY, 0);  
    new_salary := NVL(:NEW.SALARY, 0);  
END IF;
```

```
IF DELETING THEN  
    old_salary := NVL(:OLD.SALARY, 0);  
    new_salary := 0;  
END IF;
```

```
-- Calculate the salary difference  
salary_diff := new_salary - old_salary;
```

```
-- Display the salary difference  
DBMS_OUTPUT.PUT_LINE('Salary difference for ID ' || :OLD.ID || ' : ' || salary_diff);
```

```
-- Increment sequence
```

```
SELECT salary_diff_seq.NEXTVAL INTO NULL FROM DUAL;  
END;  
/
```

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).

DECLARE

-- Declare variables to hold values from the cursor

v_E_id Employee.E_id%TYPE;

v_E_name Employee.E_name%TYPE;

v_Age Employee.Age%TYPE;

v_Salary Employee.Salary%TYPE;

-- Declare cursor for the Employee table

CURSOR emp_cursor IS

SELECT E_id, E_name, Age, Salary

FROM Employee;

BEGIN

-- Open the cursor

OPEN emp_cursor;

-- Fetch and process each row from the cursor

LOOP

FETCH emp_cursor INTO v_E_id, v_E_name, v_Age, v_Salary;

EXIT WHEN emp_cursor%NOTFOUND;

-- Process the values, for example, you can print them

DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_E_id || ', Name: ' || v_E_name || ', Age: ' || v_Age || ', Salary: ' || v_Salary);

END LOOP;

-- Close the cursor

CLOSE emp_cursor;

END;

/

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.

```
create database sql6;
use sql6;

create table o_rollcall(roll_no int,name varchar(20),address varchar(20));
create table n_rollcall(roll_no int,name varchar(20),address varchar(20));
insert into o_rollcall values('1','Hitesh','Nandura');
insert into o_rollcall values('2','Piyush','MP');
insert into o_rollcall values('3','Ashley','Nsk');
insert into o_rollcall values('4','Kalpesh','Dhule');
insert into o_rollcall values('5','Abhi','Satara');

delimiter //
create procedure p3(in r1 int)
begin
declare r2 int;
declare exit_loop boolean;
declare c1 cursor for select roll_no from o_rollcall
where roll_no>r1;
declare continue handler for not found set
exit_loop=true;
open c1;
e_loop:loop
fetch c1 into r2;
if not exists(select * from n_rollcall where
roll_no=r2)
then
insert into n_rollcall select * from o_rollcall where
roll_no=r2;
end if;
if exit_loop
then
close c1;
leave e_loop;
end if;
end loop e_loop;
end
//

call p3(3);
select * from n_rollcall;
call p3(0);
select * from n_rollcall;
insert into o_rollcall values('6','Patil','Kolhapur');
call p3(4);

select * from n_rollcall;
```

OUTPUT:

Roll no	Name	Address
4	Kalpesh	Dhule
5	Abhi	Satara
1	Hitesh	Nandura
1	Hitesh	Nandura
2	Piyush	MP
3	Ashley	Nsk
6	Patil	Kolhapur

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

Installation:

1. Download the MongoDB Installer:

- Visit the official MongoDB download center:
- URL: <https://www.mongodb.com/try/download/community>
- Select the appropriate version (latest stable is recommended for most cases).
- Under "Platform," choose "Windows."
- For "Package," select "MSI."
- Click "Download."

2. Run the Installer:

- Locate the downloaded .msi file (usually in your Downloads folder).
- Double-click the file to launch the MongoDB Setup Wizard.

3. Follow the Installation Wizard:

- Click "Next" on the initial screen.
- Accept the license agreement and click "Next."
- Choose the installation type:
 - Complete: Installs all MongoDB components (recommended for most users).
 - Custom: Allows you to select specific features (advanced users).
- Click "Next."

4. Configure Service Options (Optional):

- In the "Service Configuration" step (available in newer versions), you can choose how MongoDB runs as a service:
 - Run service as Network Service user: This is the default and recommended option.
 - Run service as a custom user account: If you have specific security requirements, you can create a dedicated service account.
- Click "Next."

5. Review and Install:

- Review the installation summary.
- Click "Install" to begin the installation process.

6. Verify Installation

Basic CRUD(Create, Read, Update & Delete) operations

Now that you've installed MongoDB Compass on your local machine, you can connect it to the MongoDB instance running on your remote server.

Create new database "mflix"

Create a collection "movies"

Create Operation

Insert data into the collection.

Click on "Add Data". There will be two options:

1. Import JSON/CSV file
2. Insert Document

Click on "Insert Document" and enter the record details:

```

{
  "_id": {},
  "title": "Jurassic World: Fallen Kingdom",
  "genres": [],
  "runtime": 130,
  "rated": "PG-13",
  "year": 2018,
  "directors": [],
  "cast": [],
  "type": "movie"
}

```

movies						
	_id ObjectId	title String	genres Array	runtime Int32	rated String	
1	ObjectId('66164b31992bb9...')	"Jurassic World: Fallen ..."	[] 2 elements	130	"PG-13"	  
2	ObjectId('66164c54992bb9...')	No field	No field	No field	No field	  
3	ObjectId('66164d6e992bb9...')	"Avatar"	[] 3 elements	162	"PG-13"	  
4	ObjectId('66164e12992bb9...')	"Friends"	[] 1 elements	10	"TV-PG"	  
5	ObjectId('66164f6a992bb9...')	No field	No field	No field	No field	  
6	ObjectId('66165024992bb9...')	No field	No field	No field	No field	  

Read Operation

Click on a specific collection to view its contents.

To filter the contents, enter the query in the filter box.





➕ ADD DATA
📄 EXPORT DATA
✎ UPDATE
🗑 DELETE

```

_id: ObjectId('66164f6a992bb9d7db5e675e')
Title: "Frozen"
▸ Genres: Array (4)
Runtime: 102
Rated: "PG"
Year: "2013"
▸ Director: Array (2)
▸ Cast: Array (2)

```

Update Operation

Select the desired collection and click on Update button to modify the values of a record. Set new values for the desired fields.

Update 1 document

mflix.movies

Filter ⓘ

{ Title: 'Frozen' }

Update

[Learn more about Update syntax](#)

```
1 {
2   $set: { Runtime: 103
3
4   },
5 }
```

★ Save

Cancel

Update 1 document

Delete Operation

Click on the Delete button to delete records of a specific collection



Delete 6 documents

mflix.movies

Filter ⓘ

[Export](#)

Preview (sample of 5 documents)

```
{
  _id: ObjectId('66164b31992bb9d7db5e6755'),
  title: "Jurassic World: Fallen Kingdom",
  genres: Array (2),
  runtime: 130,
  rated: "PG-13",
  year: 2018,
  directors: Array (1),
  cast: Array (3),
  type: "movie"
}
```

```
{
  _id: ObjectId('66164c54992bb9d7db5e675a'),
  Title: "Home Alone",
  Genres: Array (2)
}
```

Cancel

Delete 6 documents

To delete a specific record, filter the record and then delete

