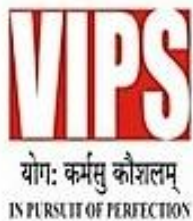


**Practical file submitted in partial fulfillment
for the evaluation of**

“Database Management System Lab (AIDS-254)”



**VIVEKANANDA SCHOOL
OF ENGINEERING AND
TECHNOLOGY**

Submitted By:

Student Name: Vikram Ranjan

Enrolment no: 09917711922

Branch & Section: AI-DS (B)

Submitted To:

- Dr. Deepika Bhatia

Index

S.No	Experiment Title (GGSIPU)	Page No.	Date	Grade/ Evaluation	Sign
1	Study and practice various database management systems like MySQL/Oracle/PostgreSQL/SQL Server and others.				
2	Implement simple queries of DDL and DML.				
3	Implement basic queries to Create, Insert, Update, Delete and Select Statements for two different scenarios (For instance: Bank, College etc.)				
4	Implement queries including various functions- mathematical, string, date etc.				
5	Implement queries including Sorting, Grouping and Subqueries-like any, all, exists, not exists.				
6	Implement queries including various Set operations (Union, Intersection, Except etc.).				
7	Implement various JOIN operations- (Inner, Outer).				
8	Write a PL/SQL program using FOR loop to insert ten rows into a database table.				
9	Given the table EMPLOYEE (Emp No, Name, Salary, Designation, Dept_ID), write a cursor to select the five highest-paid employees from the table.				
10	Illustrate how you can embed PL/SQL in a high-level host language such as C/Java And demonstrates how a banking debit transaction might be done.				

Index

S.No	Experiment Title (Beyond Curriculum)	Page No.	Date	Grade/ Evaluation	Sign
1	Write the steps to install and implement NOSQL databases-MongoDB				
2	Study and implement basic commands of MongoDB				
3	Implement any one real-time project using MySQL/MongoDB such as Library Database Management System etc..				

SECTION 1
Database Management System Lab
GGSIPIU

EXPERIMENT-1

AIM: study and practice various databases management systems like my sql/oracle/postgre sql / sql server/mongo db.

THEORY:

MY SQL

MySQL is an open-source relational database management system (RDBMS) that enables users to store, manage, and Retrieve structured data efficiently. It is widely used for various applications, from small-scale projects to large-scale websites and enterprise-level solutions. MySQL offers various features and solutions for different scenarios, such as data warehousing, data lakes, machine learning, and cloud applications. MySQL is available in different editions, including MySQL Heat Wave, Enterprise Edition, Cluster CGE, and other products and services.

ADVANTAGE

Faster Query Processing –

Large amount of data is retrieved quickly and efficiently. Operations like Insertion, deletion, manipulation of data is also done in almost no time.

No Coding Skills –

For data retrieval, large number of lines of code is not required. All basic keywords such as SELECT, INSERT INTO, UPDATE are used and also the syntactical rules are not complex in SQL, which makes it a user-friendly language.

Standardized Language –

Due to documentation and long establishment over years, it provides a uniform platform worldwide to all its users.

Portable –

It can be used in programs in PCs, server, laptops independent of any platform (Operating System,). Also, it can be embedded with other applications as per need/requirement/use.

Interactive Language –

Easy to learn and understand, answers to complex queries can be received in seconds.

Multiple data views –

Scalability: SQL databases can handle large volumes of data and can be scaled up or down as per the requirements of the application.

Security: SQL databases have built-in security features that help protect data from unauthorized access, such as user authentication, encryption, and access control.

Data Integrity: SQL databases enforce data integrity by enforcing constraints such as unique keys, primary keys, and foreign keys, which help prevent data duplication and maintain data accuracy.

Backup and Recovery: SQL databases have built-in backup and recovery tools that help recover data in case of system failures, crashes, or other disasters.

Data Consistency: SQL databases ensure consistency of data across multiple tables through the use of transactions, which ensure that changes made to one table are reflected in all related tables.



POSTGRE SQL

PostgreSQL is an open-source object-relational database system that has been under active development for over 35 years ¹. It is known for its reliability, feature robustness, and performance. PostgreSQL is available for download on various platforms, including Linux, mac OS, Windows, BSD, and Solaris.

PostgreSQL is a powerful database management system that supports a wide range of features, including ACID transactions, multi-version concurrency control, table inheritance, foreign keys, triggers, and stored procedures. It also supports a variety of programming languages, including C/C++, Java, Perl, Python, Ruby, and TCL.

PostgreSQL is widely used in various industries, including finance, healthcare, telecommunications, government, and e-commerce. It is also used by many popular websites and applications, such as Apple, Cisco, Fujitsu, IMDb, Instagram, Nokia, Skype, and Uber ⁴.

ADVANTAGES

Reliability and Performance: Users have consistently praised PostgreSQL for its reliability and performance, with many reviewers stating that they have experienced no downtime or issues related to the database. Some users also mentioned that PostgreSQL's performance is exceptionally fast, providing them with great speed in their operations.

Ease of Use and Flexibility: Many users find PostgreSQL easy to use and appreciate the availability of good open-source tools to work with it. Reviewers have highlighted that constructing queries in PostgreSQL is straightforward and that it integrates well with all development languages, making migration easy. The flexibility of PostgreSQL's user/role management system has also been praised by users, as it allows for easy control over access to tables.

Wide Industry Adoption and Community Support: Several reviewers acknowledge that PostgreSQL has achieved wide industry adoption, making it easier to integrate into a stack and hire knowledgeable developers. The availability of a huge online community for support was highly appreciated by users. Additionally, many users mentioned the extensive documentation

available for PostgreSQL, along with the ease of finding examples, which further contributes to community support.

DISADVANTAGES

Complicated Installation and Setup: Many users have found the installation and setup process of PostgreSQL to be complicated, especially for Mac users. They have mentioned the need to learn new commands and have recommended blog posts for guidance.

Difficult Syntax of SQL: Users have expressed difficulty in understanding the syntax of SQL in PostgreSQL, which they find different and hard to grasp. This may be a reason why the software is not widely adopted.

Lack of Clear Benefits: Users have mentioned the lack of clear benefits for choosing PostgreSQL over other products. They feel that there are better alternatives available with more extensive features, documentation, and community support.



ORACLE

Oracle Database, commonly referred to as Oracle DBMS or simply Oracle, is a multi-model database management system produced and marketed by Oracle Corporation. It serves various purposes, including:

Online Transaction Processing (OLTP): Managing real-time transactional data.

Data Warehousing (DW): Storing and analysing large volumes of historical data.

Mixed (OLTP & DW) workloads: Handling a combination of transactional and analytical tasks.

ADVANTAGE

In memory feature - very fast. Earlier queries used to run in fetching data used to take time now it is very fast. We can even select which table should be used in the memory feature. So complex and big tables are utilized properly.

Restore and Point of Time Recovery - In my entire career of 10 years, Oracle Database has been the most consistent and reliable database. The RMAN backup concept is the best example. If any disaster happens in another database, there is a chance of loss of data or getting corrupted but Oracle is best.

DISADVANTAGE

Oracle Database restoration and recovery is complex. RMAN backup recovery has quite a lot of steps and is complex. In other databases, it is very much simplified.

Oracle user extend validity - this is tricky as users are assigned to profiles and if you want to extend the validity of a particular user, it becomes difficult. That is where it can be improved.



MONGO DB

MongoDB is a source-available, cross-platform, document-oriented database program. Classified as a NoSQL database product, MongoDB utilizes JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and current versions are licensed under the Server Side Public License (SSPL). MongoDB is a member of the MACH Alliance.

ADVANTAGES

MongoDB **stores data in flexible, JSON-like documents**, meaning fields can vary from document to document and data structure can be changed over time

The document model **maps to the objects in your application code**, making data easy to work with

Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyse your data

MongoDB is a **distributed database at its core**, so high availability, horizontal scaling, and geographic distribution are built in and easy to use

MongoDB is **free to use**.

DISADVANTAGES

Despite MongoDB's many strengths, it also carries a few drawbacks that warrant consideration. Some drawbacks are:

Limited Transactions Scope: In MongoDB, transactions work within each piece of data (called a document), but they don't fully cover situations where you need to do multiple things at once across lots of data. This might be tricky for applications that really need everything to happen perfectly together.

Lacks in Full ACID Compliance: While MongoDB offers Atomicity, Consistency, Isolation, and Durability (ACID) at the document level, it doesn't provide full ACID compliance across multiple documents or collections. This limitation can be challenging for applications requiring strict and complex transactional guarantees.

Limited Join Capabilities: Unlike traditional relational databases, MongoDB doesn't support joins in the same way. While it's possible to manually perform join-like operations using code, it can slow down execution and affect performance.

Data Redundancy and Memory Usage: MongoDB stores key names with each value pair, causing some data redundancy due to the limitations of joins. This redundancy might lead to increased memory usage compared to what's strictly necessary.

Document Size Limit: MongoDB imposes a maximum document size limit of 16 MB. Larger documents might need to be handled differently or divided into smaller documents to fit within this constraint.

Nested Document Levels: Document nesting in MongoDB is possible but limited to a maximum of 100 levels. This restriction can impact how deeply you can organize and structure your data within documents.



SQL SERVER

SQL Server, also known as MS SQL Server or Microsoft SQL Server, is a relational database management system (RDBMS). It's a program that saves database information and runs SQL commands and queries to alter a relational database. Furthermore, it manages and executes all database activities.

Microsoft created SQL Server in 1989 for business purposes. It is proprietary software written in the C and C++ programming languages. T-SQL (Transact Structured Query Language) is a different type of SQL that is nearly identical to SQL, with minor differences in query syntax.

The SQL Server is platform-specific, with separate software available for various platforms. The Microsoft Windows and Linux operating systems both support SQL Server. The most recent SQL Server version is 15.0, which was published in 2019.

ADVANTAGES

As compared to other RDBMS, SQL Server is reported to be simpler to use; it offers more functionality and user-friendly procedures. For easier user operations, it provides both command-line and GUI (Graphical User Interface) options. SQL Server also receives frequent security and operational updates, which contribute to its popularity.

DISADVANTAGE

Database backup and recovery functionality need improvement. Sometimes I have observed that when you try to restore a backup to a previous date/state, it does not work as expected, and restore fails.

Cost gets higher on integrating with Azure SQL



EXPERIMENT-2

AIM: Implement simple queries of DDL and DML.

Theory:

1. DDL commands:

1.a Create a table with 7 attributes with constraints

Command:

CREATE TABLE STUDENT

-> (S_ID INT NOT NULL UNIQUE PRIMARY KEY,

-> ROLL_NO INT NOT NULL UNIQUE,

-> NAME CHAR(30) NOT NULL,

-> SEM INT,

-> ADDRESS VARCHAR(30),

-> DOB DATE,

-> PHONE_NO INT UNIQUE);

DESCRIBE STUDENT;

OUTPUT:

Field	Type	Null	Key	Default	Extra
S_ID	int	NO	PRI	NULL	
ROLL_NO	int	NO	UNI	NULL	
NAME	char(30)	NO		NULL	
SEM	int	YES		NULL	
ADDRESS	varchar(30)	YES		NULL	
DOB	date	YES		NULL	
PHONE_NO	int	YES	UNI	NULL	

7 rows in set (0.01 sec)

1.b Create a new table with same attributes as in query 1 and add constraints after table creation(Primary key, Unique key, Not Null, check, Default).

Command:

```
CREATE TABLE CUSTOMER (ID INT,NAME VARCHAR(20),AGE INT, SEX CHAR(1));
```

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);
```

```
ALTER TABLE CUSTOMER ADD CHECK (Age>=18);
```

```
ALTER TABLE CUSTOMER MODIFY Age int NOT NULL;
```

```
ALTER TABLE CUSTOMER ADD ADDRESS VARCHAR(20) DEFAULT "DELHI";
```

DESCRIBE CUSTOMER;

Field	Type	Null	Key	Default	Extra
ID	int	NO	PRI	NULL	
NAME	varchar(20)	YES		NULL	
Age	int	NO		NULL	
SEX	char(1)	YES		NULL	
ADDRESS	varchar(20)	YES		DELHI	

5 rows in set (0.00 sec)

1.c.a. Add columns Commission and Phone to the table

Command:

ALTER TABLE CUSTOMER ADD PHONE INT UNIQUE NOT NULL;

ALTER TABLE CUSTOMER ADD COMMISSION VARCHAR(20);

SELECT * FROM CUSTOMER;

ID	NAME	Age	SEX	ADDRESS	PHONE	COMMISSION
123	ARSHAD	18	M	DELHI	0	NULL

1 row in set (0.00 sec)

1.c.b. Drop column phone.

ALTER TABLE CUSTOMER DROP COLUMN PHONE;

1.c.c. Change the datatype of any column.

ALTER TABLE CUSTOMER MODIFY COMMISSION INT;

1.d. Rename one of the columns in table.

Command: RENAME TABLE CUSTOMER TO NEW_CUSTOMER;

SELECT * FROM CUSTOMER;

ID	NAME	Age	SEX	ADDRESS	COMISSION
123	ARSHAD	18	M	DELHI	NULL

1 row in set (0.00 sec)

1.e. Rename the table name:

Command: RENAME TABLE CUSTOMER TO NEW_CUSTOMER;

1.f. Create a table VIPS1 (ID, name, state, sal, dno).

Command:

CREATE TABLE VIPS1(ID INT PRIMARY KEY,NAME VARCHAR(20),STATE VARCHAR(20) DEFAULT 'DELHI',SAL INT,DNO INT);

1.g. Create a table VIPS-TC1 (ID, name, state, sal, dno).

Command:

CREATE TABLE VIPS_TC1(ID INT PRIMARY KEY,NAME VARCHAR(20),STATE VARCHAR(20) DEFAULT 'DELHI',SAL INT,DNO INT);

1.g. Drop the table VIPS1 and truncate VIPS_TC1 and show difference.

Command : DROP TABLE VIPS1;

```
TRUNCATE TABLE VIPS_TC1;
```

```
DESC VIPS1;
```

Field	Type	Null	Key	Default	Extra
ID	int	NO	PRI	NULL	
NAME	varchar(20)	YES		NULL	
STATE	varchar(20)	YES		DELHI	
SAL	int	YES		NULL	
DNO	int	YES		NULL	

5 rows in set (0.00 sec)

```
SELECT * FROM VIPS1;
```

ERROR 1146 (42S02): Table 'collage.vips1' does not exist

```
SELECT * FROM VIPS_TC1;
```

Empty set (0.00 sec)

Difference between Drop and Truncate is that drop command delete the whole table but truncate deletes all the rows from table.

2. DML commands :

2.a. Insert values in tables .

Commands:

```
DESC EMPLOYEES:
```

```
INSERT INTO EMPLOYEES VALUES
```

- > (1, 'RAHUL', 60000.00, 'ABC', '1989-05-15','8833445566'),
- > (2, 'SURAJ', 55000.00, 'ABC', '1992-10-25','8822113344'),
- > (3, 'RAJESH', 50000.00, 'ABC', '1980-03-08','7788669955'),
- > (4, 'SAMEER', 48000.00, 'ABC', '1986-11-30','7755443322'),
- > (5, 'VIKRAM', 70000.00, 'ABC', '1975-08-20','7788996655'),
- > (6, 'ROHAN', 62000.00, 'ABC', '1991-04-12','8877996655'),
- > (7, 'JUMAID', 65000.00, 'ABC', '1983-07-02','9988776655');

2.b. Show entire table:

Command:

```
SELECT * FROM EMPLOYEES;
```

E_ID	NAME	SALARY	ADDRESS	DOB	PHONE_NO
1	RAHUL	60000	ABC	1989-05-15	8833445566
2	SURAJ	55000	ABC	1992-10-25	8822113344
3	RAJESH	50000	ABC	1980-03-08	7788669955
4	SAMEER	48000	ABC	1986-11-30	7755443322
5	VIKRAM	70000	ABC	1975-08-20	7788996655
6	ROHAN	62000	ABC	1991-04-12	8877996655
7	JUMAID	65000	ABC	1983-07-02	9988776655

7 rows in set (0.00 sec)

2.c. Select only 3 columns from table with and without where clause.

Command: `SELECT NAME, SALARY, PHONE_NO FROM EMPLOYEES;`

NAME	SALARY	PHONE_NO
RAHUL	60000	8833445566
SURAJ	55000	8822113344
RAJESH	50000	7788669955
SAMEER	48000	7755443322
VIKRAM	70000	7788996655
ROHAN	62000	8877996655
JUMAID	65000	9988776655

7 rows in set (0.00 sec)

`SELECT NAME, SALARY, PHONE_NO FROM EMPLOYEES WHERE NAME='RAHUL';`

NAME	SALARY	PHONE_NO
RAHUL	60000	8833445566

1 row in set (0.00 sec)

2.d. Update the salary of the employees by 3000.

Command; `UPDATE EMPLOYEES SET SALARY=SALARY+3000;`

2.e. Update the salary of the employees by 10% whose ID is 7 or salary is less than 10000.

Command:

```
UPDATE EMPLOYEES SET SALARY=SALARY+SALARY*0.1 WHERE E_ID=7 OR SALARY>10000;
```

```
SELECT* FROM EMPLOYEES;
```

E_ID	NAME	SALARY	ADDRESS	DOB	PHONE_NO
1	RAHUL	63100	ABC	1989-05-15	8833445566
2	SURAJ	58100	ABC	1992-10-25	8822113344
3	RAJESH	53100	ABC	1980-03-08	7788669955
4	SAMEER	51100	ABC	1986-11-30	7755443322
5	VIKRAM	73100	ABC	1975-08-20	7788996655
6	ROHAN	65100	ABC	1991-04-12	8877996655
7	JUMAID	68100	ABC	1983-07-02	9988776655

7 rows in set (0.00 sec)

2.e. Delete the salary of the employees with salary between 50000 and 60000 and id is less than 2.

Command:

```
DELETE FROM employees WHERE salary BETWEEN 50000 AND 60000 AND E_ID < 2;
```

```
SELECT* FROM EMPLOYEES;
```

E_ID	NAME	SALARY	ADDRESS	DOB	PHONE_NO
2	SURAJ	55000	ABC	1992-10-25	8822113344
3	RAJESH	50000	ABC	1980-03-08	7788669955
4	SAMEER	48000	ABC	1986-11-30	7755443322
5	VIKRAM	70000	ABC	1975-08-20	7788996655
6	ROHAN	62000	ABC	1991-04-12	8877996655
7	JUMAID	65000	ABC	1983-07-02	9988776655

6 rows in set (0.00 sec)

2.f. Delete the employee whose designation is nurse.

Command:

```
ALTER TABLE employees ADD designation VARCHAR(255);
```

```
UPDATE employees SET designation = 'manager' WHERE E_ID=2;
```


UPDATE employees SET designation = 'manager' WHERE E_ID=3;

UPDATE employees SET designation = 'nurse' WHERE E_ID=4;

UPDATE employees SET designation = 'nurse' WHERE E_ID=5;

UPDATE employees SET designation = 'manager' WHERE E_ID=6;

UPDATE employees SET designation = 'manager' WHERE E_ID=7;

DELETE FROM employees WHERE designation = 'nurse';

E_ID	NAME	SALARY	ADDRESS	DOB	PHONE_NO	designation
2	SURAJ	55000	ABC	1992-10-25	8822113344	manager
3	RAJESH	50000	ABC	1980-03-08	7788669955	manager
6	ROHAN	62000	ABC	1991-04-12	8877996655	manager
7	JUMAID	65000	ABC	1983-07-02	9988776655	manager

4 rows in set (0.00 sec)

EXPERIMENT-3

AIM: Implement basic queries to Create, Insert, Update, Delete and Select Statements for two different scenarios (For instance: College, Bank).

Theory:

SQL Commands:

- **For College:**

1. Creating tables with constraints:

- a. faculty (fid, name, *dno*, sal, address, phone, dob, exp)
- b. dept (dno, dname, budget)
- c. student (sid, name, *dno*, phone, dob, address, sem)
- d. society (sc_id, name)
- e. std_soc(*sid*, *sc_id*)
- f. course (cid, cname)
- g. fac_course (*fid*, *cid*)

1.a. Faculty table

Command:

```
CREATE TABLE FACULTY(
```

```
-> F_ID INT PRIMARY KEY NOT NULL,  
-> NAME VARCHAR(20),  
-> D_NO INT,  
-> SAL INT ,  
-> ADDRESS VARCHAR(30),  
-> PHONE VARCHAR(10) UNIQUE,  
-> DOB DATE,  
-> EXP INT);
```

```
ALTER TABLE FACULTY ADD FOREIGN KEY (D_NO) REFERENCES DEPT(D_NO);
```

1.b. Dept table:

Command:

```
CREATE TABLE DEPT( D_NO INT PRIMARY KEY NOT NULL,
```

```
-> D_NAME VARCHAR(20) NOT NULL,  
-> BUDGET INT);
```

1.c. Student table:

Command: CREATE TABLE STUDENT

```
-> (S_ID INT NOT NULL UNIQUE PRIMARY KEY,  
-> D_NO INT NOT NULL,  
-> NAME CHAR(30) NOT NULL,  
-> SEM INT,  
-> ADDRESS VARCHAR(30),  
-> DOB DATE,  
-> PHONE_NO INT UNIQUE);
```

```
ALTER TABLE STUDENT ADD FOREIGN KEY (D_NO) REFERENCES DEPT(D_NO);
```

1.d. Society table:

```
CREATE TABLE SOCIETY(
```

```
-> SC_ID INT PRIMARY KEY NOT NULL,
```

```
-> NAME VARCHAR(20));
```

1.e. STD_SOC table:

```
CREATE TABLE STD_SOC(
```

```
-> S_ID INT PRIMARY KEY NOT NULL, SC_ID INT NOT NULL,
```

```
-> FOREIGN KEY (S_ID) REFERENCES STUDENT(S_ID),
```

```
-> FOREIGN KEY (SC_ID) REFERENCES SOCIETY(SC_ID));
```

1.f. Course table:

```
CREATE TABLE COURSE(
```

```
-> C_ID INT PRIMARY KEY NOT NULL,
```

```
-> C_NAME VARCHAR(20));
```

1.g. FAC_COURSE:

```
CREATE TABLE FAC_COURSE(
```

```
-> F_ID INT PRIMARY KEY NOT NULL,
```

```
-> C_ID INT NOT NULL,
```

```
-> FOREIGN KEY (C_ID) REFERENCES COURSE(C_ID),
```

```
-> FOREIGN KEY (F_ID) REFERENCES FACULTY(F_ID));
```

2. Add a column no_of_awards to faculty table and upload the values in the column.

Command:

```
ALTER TABLE FACULTY
```

```
-> ADD COLUMN NO_OF_AWARDS INT;
```

```
DESC FACULTY;
```

Field	Type	Null	Key	Default	Extra
F_ID	int	NO	PRI	NULL	
NAME	varchar(20)	YES		NULL	
D_NO	int	YES	MUL	NULL	
SAL	int	YES		NULL	
ADDRESS	varchar(30)	YES		NULL	
PHONE	varchar(10)	YES	UNI	NULL	
DOB	date	YES		NULL	
EXP	int	YES		NULL	
NO_OF_AWARDS	int	YES		NULL	

9 rows in set (0.00 sec)

3. Insert rows in each table (atleast 5).

Command:

INSERT INTO FACULTY VALUES

-> (1,"SONAM",1,100000,"ABC","9980776655","1991-12-02",4,2);

INSERT INTO FACULTY VALUES

-> (2,"SAMEER",1,120000,"ABCD","9980776659","1990-03-07",5,6);

INSERT INTO FACULTY VALUES

-> (3,"SAMEER",3,110000,"ABFD","9980776859","1993-04-11",3,5);

INSERT INTO FACULTY VALUES

-> (4,"RAJEEV",4,140000,"ABFDF","9980776879","1989-04-11",4,3);

INSERT INTO FACULTY VALUES

-> (5,"RAJEEV",5,120000,"ABF","9980776379","1988-05-10",4,4);

SELECT * FROM FACULTY;

F_ID	NAME	D_NO	SAL	ADDRESS	PHONE	DOB	EXP	NO_OF_AWARDS
1	SONAM	1	100000	ABC	9980776655	1991-12-02	4	2
2	SAMEER	2	120000	ABCD	9980776659	1990-03-07	5	6
3	SAMEER	3	110000	ABFD	9980776859	1993-04-11	3	5
4	RAJEEV	4	140000	ABFDF	9980776879	1989-04-11	4	3
5	RAJEEV	5	120000	ABF	9980776379	1988-05-10	4	4

5 rows in set (0.00 sec)

```

INSERT INTO DEPT VALUES(1,'CSE A',1000);
INSERT INTO DEPT VALUES(2,'CSE B',2000);
INSERT INTO DEPT VALUES(3,'AIDS A',2000);
INSERT INTO DEPT VALUES(4,'AIDS B',3000);

SELECT * FROM DEPT;

```

D_NO	D_NAME	BUDGET
1	CSE A	1000
2	CSE B	2000
3	AIDS A	2000
4	AIDS B	3000
5	IOT	1500

5 rows in set (0.00 sec)

```

INSERT INTO COURSE VALUES(101,'CSE A');
INSERT INTO COURSE VALUES(102,'CSE B');
INSERT INTO COURSE VALUES(103,'AIDS A');
INSERT INTO COURSE VALUES(104,'AIDS B');
INSERT INTO COURSE VALUES(105,'IOT');

SELECT * FROM COURSE;

```

C_ID	C_NAME
101	CSE A
102	CSE B
103	AIDS A
104	AIDS B
105	IOT

5 rows in set (0.00 sec)

```

INSERT INTO fac_course VALUES(1,101);
INSERT INTO fac_course VALUES(2,102);
INSERT INTO fac_course VALUES(3,103);

```

```
INSERT INTO fac_course VALUES(4,104);
```

```
INSERT INTO fac_course VALUES(5,105);
```

```
SELECT * FROM FAC_COURSE;
```

F_ID	C_ID
1	101
2	102
3	103
4	104
5	105

5 rows in set (0.00 sec)

```
INSERT INTO STUDENT VALUES(301,1,'SOURAV',4,'ABC','1999-12-03','9980775533');
```

```
INSERT INTO STUDENT VALUES(302,2,'RAHUL',4,'ABCD','2003-02-04','9980775539');
```

```
INSERT INTO STUDENT VALUES (303,3,'RANVIJAY',4,'ABCD','2003-07-06','9980775239');
```

```
INSERT INTO STUDENT VALUES (304,4,'SANJAY',4,'ABCDE','2003-02-02','9980775139');
```

```
INSERT INTO STUDENT VALUES(305,5,'SANYOG',4,'ABCDE','2003-02-05','9980775138');
```

```
SELECT * FROM STUDENTS;
```

S_ID	D_NO	NAME	SEM	ADDRESS	DOB	PHONE_NO
301	1	SOURAV	4	ABC	2002-12-03	9980775533
302	2	RAHUL	4	ABCD	2003-02-04	9980775539
303	3	RANVIJAY	4	ABCD	2003-07-06	9980775239
304	4	SANJAY	4	ABCDE	2003-02-02	9980775139
305	5	SANYOG	4	ABCDE	2003-02-05	9980775138

5 rows in set (0.00 sec)

```
INSERT INTO SOCIETY VALUES(91,"DFGH");
```

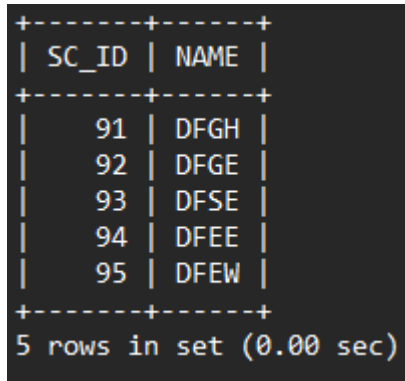
```
INSERT INTO SOCIETY VALUES(92,"DFGE");
```

```
INSERT INTO SOCIETY VALUES(93,"DFSE");
```

```
INSERT INTO SOCIETY VALUES(94,"DFEE");
```

```
INSERT INTO SOCIETY VALUES(95,"DFEW");
```

```
SELECT * FROM SOCIETY;
```



```
+-----+-----+
| SC_ID | NAME |
+-----+-----+
|    91 | DFGH |
|    92 | DFGE |
|    93 | DFSE |
|    94 | DFEE |
|    95 | DFEW |
+-----+-----+
5 rows in set (0.00 sec)
```

```
INSERT INTO STD_SOC VALUES(301,91);
```

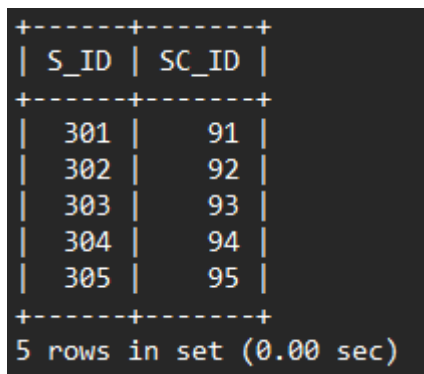
```
INSERT INTO STD_SOC VALUES(302,92);
```

```
INSERT INTO STD_SOC VALUES(303,93);
```

```
INSERT INTO STD_SOC VALUES(304,94);
```

```
INSERT INTO STD_SOC VALUES(305,95);
```

```
SELECT * FROM STD_SOC;
```



```
+-----+-----+
| S_ID | SC_ID |
+-----+-----+
|   301 |    91 |
|   302 |    92 |
|   303 |    93 |
|   304 |    94 |
|   305 |    95 |
+-----+-----+
5 rows in set (0.00 sec)
```

4. Retrieve the faculties with experience greater than 1 year and salary greater than 1000;

Command: `SELECT NAME FROM FACULTY WHERE EXP>1 AND SAL>1000;`


```

+-----+
| NAME   |
+-----+
| SONAM  |
| SAMEER |
| SAMEER |
| RAJEEV |
| RAJEEV |
+-----+
5 rows in set (0.00 sec)

```

5. Retrieve the student enrolled in department 2 or semester 4.

Command: `SELECT NAME FROM STUDENT WHERE D_NO=2 OR SEM=4;`

```

+-----+
| NAME   |
+-----+
| SOURAV |
| RAHUL  |
| RANVIJAY |
| SANJAY |
| SANYOG |
+-----+
5 rows in set (0.00 sec)

```

6. Increase the faculty salary whose dept_no is 1.

Command: `UPDATE FACULTY SET SAL=SAL+1000 WHERE D_NO=1;`

7. Increase the budget of CSE department by 1000.

Command: `UPDATE DEPT SET BUDGET=BUDGET+1000 WHERE D_NAME='CSE A' OR D_NAME='CSE B';`

8. Update the phone_no of std_id 2.

Command: `UPDATE STUDENT SET PHONE_NO='9988776655' WHERE S_ID=2;`

- **For College:**

2. Creating tables with constraints:

- a. ACC(acc_no, name, phone)
- b. Trans(acc_no, C_D, amount)
- c. Loan(acc_no, interest, time, type)
- d. FD(acc_no, amount, interest, time)
- e. Locker(acc_no, L_no)

Commands:

- 2.a. create table ACC(

- > acc_no int primary key not null,
- > name varchar(10),
- > phone varchar(10));

DESC ACC;

Field	Type	Null	Key	Default	Extra
acc_no	int	NO	PRI	NULL	
name	varchar(10)	YES		NULL	
phone	varchar(10)	YES		NULL	

3 rows in set (0.01 sec)

- 2.b. CREATE TABLE TRANS(

- > ACC_NO INT NOT NULL UNIQUE,
- > C_D INT ,
- > AMOUNT INT,
- > FOREIGN KEY(ACC_NO) REFERENCES ACC(ACC_NO));

DESC TRANS;

Field	Type	Null	Key	Default	Extra
ACC_NO	int	NO	PRI	NULL	
C_D	int	YES		NULL	
AMOUNT	int	YES		NULL	

3 rows in set (0.01 sec)

2.c CREATE TABLE LOAN(

- > ACC_NO INT UNIQUE,
- > INTEREST INT NOT NULL ,
- > TIME INT ,
- > TYPE VARCHAR(20),
- > FOREIGN KEY(ACC_NO) REFERENCES ACC(ACC_NO));

DESC LOAN;

Field	Type	Null	Key	Default	Extra
ACC_NO	int	YES	UNI	NULL	
INTEREST	int	NO		NULL	
TIME	int	YES		NULL	
TYPE	varchar(20)	YES		NULL	

4 rows in set (0.00 sec)

2.d. CREATE TABLE FD(

- > ACC_NO INT UNIQUE,
- > AMOUNT INT NOT NULL,
- > INTEREST INT NOT NULL,
- > TIME INT,
- > FOREIGN KEY(ACC_NO) REFERENCES ACC(ACC_NO));

DESC FD;

Field	Type	Null	Key	Default	Extra
ACC_NO	int	YES	UNI	NULL	
AMOUNT	int	NO		NULL	
INTEREST	int	NO		NULL	
TIME	int	YES		NULL	

4 rows in set (0.00 sec)

2.e. CREATE TABLE LOCKER(

- > ACC_NO INT UNIQUE NOT NULL,
- > L_NO INT NOT NULL UNIQUE,
- > FOREIGN KEY(ACC_NO) REFERENCES ACC(ACC_NO));

DESC LOCKER;

Field	Type	Null	Key	Default	Extra
ACC_NO	int	NO	PRI	NULL	
L_NO	int	NO	UNI	NULL	

2 rows in set (0.00 sec)

3. Insert values in each table and show content.

3.a. INSERT INTO ACC VALUES(2345,'RAHUL','9988776655');

INSERT INTO ACC VALUES(23458,'RAJAN','9888776655');

INSERT INTO ACC VALUES(23456,'SUMIT','9888776659');

INSERT INTO ACC VALUES(23416,'ROHAN','9888776859');

SELECT * FROM ACC;

acc_no	name	phone
2345	RAHUL	9988776655
23416	ROHAN	9888776859
23456	SUMIT	9888776659
23458	RAJAN	9888776655

4 rows in set (0.00 sec)

3.b. INSERT INTO TRANS VALUES(2345,'C',15000),

-> (23458,'C',150000),

-> (23456,'D',150000),

-> (23416,'D',17000);

SELECT * FROM TRANS;

ACC_NO	C_D	AMOUNT
2345	C	15000
23416	D	17000
23456	D	150000
23458	C	150000

4 rows in set (0.00 sec)

3.c. INSERT INTO LOAN VALUES(2345, 5,2,'CAR LOAN'),

-> (23458, 3,1,'HOME LOAN'),

-> (23456, 7,1,'PERSONAL LOAN'),

-> (23416, 5,1,'PERSONAL LOAN');

SELECT * FROM LOAN;

ACC_NO	INTEREST	TIME	TYPE	LOAN_NO
2345	5	2	CAR LOAN	101
23458	3	1	HOME LOAN	102
23456	7	1	PERSONAL LOAN	103
23416	5	1	PERSONAL LOAN	103

4 rows in set (0.00 sec)

3.d. INSERT INTO FD VALUES(2345,50000,10,3),

-> (23458,50000,8,2),

-> (23456,10000,9,3),

-> (23416,100000,6,3);

SELECT * FROM FD;

ACC_NO	AMOUNT	INTEREST	TIME
2345	50000	10	3
23458	50000	8	2
23456	10000	9	3
23416	100000	6	3

4 rows in set (0.00 sec)

3.e. INSERT INTO LOCKER VALUES(2345,1), (23458,2), (23456,3), (23416,4);

SELECT * FROM LOCKER;

ACC_NO	L_NO
2345	1
23458	2
23456	3
23416	4

4 rows in set (0.00 sec)

4. Add a column loan_no in loan table and update values for this column.

Command: ALTER TABLE LOAN ADD LOAN_NO VARCHAR(20);

UPDATE LOAN SET LOAN_NO=101 WHERE ACC_NO=2345;

UPDATE LOAN SET LOAN_NO=102 WHERE ACC_NO=23458;

UPDATE LOAN SET LOAN_NO=103 WHERE ACC_NO=23456;

UPDATE LOAN SET LOAN_NO=103 WHERE ACC_NO=23416;

SELECT * FROM LOAN;

ACC_NO	INTEREST	TIME	TYPE	LOAN_NO
2345	5	2	CAR LOAN	101
23458	3	1	HOME LOAN	102
23456	7	1	PERSONAL LOAN	103
23416	5	1	PERSONAL LOAN	103

4 rows in set (0.00 sec)

5. Insert a row in account table without phone number;

Command: INSERT INTO ACC VALUES(23457,'SURAJ',NULL);

SELECT * FROM ACC;

acc_no	name	PHONE
2345	RAHUL	9988776655
23416	ROHAN	9888776859
23456	SUMIT	9888776659
23457	SURAJ	NULL
23458	RAJAN	9888776655

5 rows in set (0.00 sec)

6. Delete account with no phone number.

Command: DELETE FROM ACC WHERE PHONE IS NULL;

SELECT * FROM ACC;

acc_no	name	PHONE
2345	RAHUL	9988776655
23416	ROHAN	9888776859
23456	SUMIT	9888776659
23458	RAJAN	9888776655

4 rows in set (0.00 sec)

EXPERIMENT-4

AIM: Implement queries include various function such as math, string, date etc.

Theory:

1. Create table from given schema and insert values in table.

FAC(FID, NAME, DOB, EXP, NO_OF_AWARDS, SAL, ADDRESS, PHONE).

Command:

```
CREATE TABLE FAC(
```

```
-> FID INT PRIMARY KEY NOT NULL,
```

```
-> NAME VARCHAR(20) NOT NULL,
```

```
-> DOB DATE NOT NULL,
```

```
-> EXP INT,
```

```
-> NO_OF_AWARDS INT NOT NULL,
```

```
-> DNO INT NOT NULL,
```

```
-> SAL INT NOT NULL,
```

```
-> ADDRESS VARCHAR(30),
```

```
-> PHONE VARCHAR(20));
```

DESC FAC;

Field	Type	Null	Key	Default	Extra
FID	int	NO	PRI	NULL	
NAME	varchar(20)	NO		NULL	
DOB	date	NO		NULL	
EXP	int	YES		NULL	
NO_OF_AWARDS	int	NO		NULL	
DNO	int	NO		NULL	
SAL	int	NO		NULL	
ADDRESS	varchar(30)	YES		NULL	
PHONE	varchar(20)	YES		NULL	

9 rows in set (0.00 sec)

```
INSERT INTO FAC VALUES(101, "RAM", "2002-01-01", 3, 3, 1, 200000, "DELHI",  
8937293643);
```

```
INSERT INTO FAC VALUES(102, "SHYAM", "2002-06-05", 4, 5, 2, 30000
```

```
0, "DELHI", 9876543212);
```

```
INSERT INTO FAC VALUES(103, "KRISHNA", "2003-09-18", 4, 5, 3, 400000,
"RAJASTHAN", 6789631579);
```

```
INSERT INTO FAC VALUES(104, "SASHANK", "2003-11-25", 5, 5, 5, 400000,
"RAJASTHAN", 9867535617);
```

```
SELECT * FROM FAC;
```

FID	NAME	DOB	EXP	NO_OF_AWARDS	DNO	SAL	ADDRESS	PHONE
101	RAM	2002-01-01	3	3	1	200000	DELHI	8937293643
102	SHYAM	2002-06-05	4	5	2	300000	DELHI	9876543212
103	KRISHNA	2003-09-18	4	5	3	400000	RAJASTHAN	6789631579
104	SASHANK	2003-11-25	5	5	5	400000	RAJASTHAN	9867535617

4 rows in set (0.00 sec)

2. Retrieve the sum of salary of all the faculties.

Command:

```
SELECT SUM(SAL) FROM FAC;
```

SUM(SAL)
1300000

3. Retrieve the sum of salary of all the faculties of department number 1.

Command: SELECT SUM(SAL) FROM FAC WHERE DNO=1;

SUM(SAL)
200000

4. Retrieve average, minimum, maximum salary of all the faculty.

Commands: SELECT AVG(SAL) FROM FAC;

AVG(SAL)
325000.0000

```
SELECT MAX(SAL) FROM FAC;
```

MAX(SAL)
400000

SELECT MIN(SAL) FROM FAC;

MIN(SAL)
200000

5. Retrieve average, minimum, maximum salary of all the faculty of department number 5.

Commands: SELECT MIN(SAL) FROM FAC WHERE DNO=5;

MIN(SAL)
400000

SELECT MAX(SAL) FROM FAC WHERE DNO=5;

MAX(SAL)
400000

SELECT AVG(SAL) FROM FAC WHERE DNO=5;

AVG(SAL)
400000.0000

6. Retrieve average experience of all the faculty of department number 1.

Commands: SELECT AVG(EXP) FROM FAC WHERE DNO=1;

AVG(EXP)
3.0000

7. Retrieve the number of faculties in college.

Command: SELECT COUNT(FID) FROM FAC;

```
+-----+
| COUNT(FID) |
+-----+
|          4 |
+-----+
```

8. Retrieve the number of awards of faculties in department number 5.

```
+-----+
| COUNT(FID) |
+-----+
|          3 |
+-----+
```

9. Demonstrate the use of : ASCII, ABS, CONCAT, SUBSTR, TRIM, UPPER, LOWER, COS, SIN, TAN, LOG, POWER, ROUND, FLOOR, CEIL, SQRT.

Commands:

SELECT ASCII(NAME) FROM FAC;

```
+-----+
| ASCII(NAME) |
+-----+
|          82 |
|          83 |
|          75 |
|          83 |
+-----+
```

SELECT AVG(NO_OF_AWARDS) FROM FAC;

```
+-----+
| AVG(NO_OF_AWARDS) |
+-----+
|          4.5000 |
+-----+
```

SELECT COS(AVG(NO_OF_AWARDS)) FROM FAC;

```
+-----+
| COS(AVG(NO_OF_AWARDS)) |
+-----+
| -0.2107957994307797 |
+-----+
```

SELECT ABS(COS(AVG(NO_OF_AWARDS))) FROM FAC;

```

+-----+
| ABS(COS(AVG(NO_OF_AWARDS))) |
+-----+
|          0.2107957994307797 |
+-----+

```

SELECT CONCAT(NAME,PHONE) FROM FAC;

```

+-----+
| CONCAT(NAME,PHONE) |
+-----+
| RAM8937293643      |
| SHYAM9876543212    |
| KRISHNA6789631579  |
| SASHANK9867535617  |
+-----+

```

SELECT SUBSTRING('VIKRAM RANJAN', 5,10) AS ExtractString;

```

+-----+
| ExtractString |
+-----+
| AM RANJAN |
+-----+

```

SELECT TRIM(' VIKRAM RANJAN ') AS TRIMMED_STRING;

```

+-----+
| TRIMMED_STRING |
+-----+
| VIKRAM RANJAN |
+-----+

```

SELECT LOWER(NAME) FROM FAC;

```

+-----+
| LOWER(NAME) |
+-----+
| ram         |
| shyam       |
| krishna     |
| sashank     |
+-----+

```

SELECT UPPER(ADDRESS) FROM FAC;

```

+-----+
| UPPER(ADDRESS) |
+-----+
| DELHI          |
| DELHI          |
| RAJASTHAN      |
| RAJASTHAN      |
+-----+

```

SELECT SIN(AVG(SAL)) FROM FAC;

```
+-----+  
| SIN(AVG(SAL)) |  
+-----+  
| 0.7843245257709923 |  
+-----+
```

SELECT FLOOR(SIN(AVG(SAL))) FROM FAC;

```
+-----+  
| FLOOR(SIN(AVG(SAL))) |  
+-----+  
| 0 |  
+-----+
```

SELECT TAN(67) AS TAN_67;

```
+-----+  
| TAN_67 |  
+-----+  
| 1.6523172640102353 |  
+-----+
```

SELECT ROUND(TAN(67)) AS TAN_67;

```
+-----+  
| TAN_67 |  
+-----+  
| 2 |  
+-----+
```

SELECT SQRT(SAL) FROM FACULTY;

```
+-----+  
| SQRT(SAL) |  
+-----+  
| 317.80497164141406 |  
| 346.41016151377545 |  
| 331.66247903554 |  
| 374.16573867739413 |  
| 346.41016151377545 |  
+-----+
```

SELECT CEIL(SQRT(SAL)) FROM FACULTY;

CEIL(SQRT(SAL))
318
347
332
375
347

10. Demonstrate the use of any 5 date function.

Commands:

SELECT CURDATE();

CURDATE()
2024-03-02

SELECT DATE_FORMAT(NOW(), '%Y-%m-%d %H:%i:%s');

DATE_FORMAT(NOW(), '%Y-%m-%d %H:%i:%s')
2024-03-02 20:04:18

SELECT DATE_FORMAT(DOB, '%Y-%m-%d') AS formatted_date FROM FAC;

formatted_date
2002-01-01
2002-06-05
2003-09-18
2003-11-25

SELECT TIMESTAMPDIFF(YEAR, DOB, CURDATE()) AS AGE FROM FAC;

AGE
22
21
20
20

EXPERIMENT-5

AIM: Implement queries include various set of operation.

Theory:

CREATE A TABLE FACULTY WITH THE COLUMNS (FID, NAME, DNO, SAL, ADDRESS, PHONE, DOB, EXP).

CREATE TABLE FACULTY (

2 FID INT,

3 NAME VARCHAR(10),

4 DNO INT,

5 SAL INT,

6 ADDRESS VARCHAR(10),

7 PHONE VARCHAR(10),

8 DOB DATE,

9 EXP INT);

INSERT INTO FACULTY VALUES(1,'VIKRAM',101,10000,'ABC','9988776655','2001-03-2',4);

INSERT INTO FACULTY VALUES(2,'RAJ',102,15000,'ABC','9988776645','2002-12-3',5);

INSERT INTO FACULTY VALUES(3,'HGJ',103,150000,'ABC','9988576645','2004-12-27',2);

INSERT INTO FACULTY VALUES(4,'HSF',104,110000,'ABC','9988576642','2004-03-4',3);

INSERT INTO FACULTY VALUES(5,'HSF',105,11000,'ABC','9988576632','2004-05-12',2);

fid	name	dno	sal	address	phone	dob	exp
1	Vikram	101	10000	ABC	9988776655	2001-03-02	4
2	raj	102	15000	ABC	9988776645	2002-12-03	5
3	hgj	103	150000	ABC	9988576645	2004-12-27	2
4	hsf	104	110000	ABC	9988576642	2004-03-04	3
5	hsf	105	11000	ABC	9988576632	2004-05-12	2

5 rows in set (0.00 sec)

CREATE A TABLE FACULTY WITH THE DEPT (BUDGET, DNAME, DNO).

CREATE TABLE DEPT (

-> DNO INT,

-> DNAME VARCHAR (10),

-> BUDGET INT);

INSERT INTO DEPT VALUES(101,'ACCOUNT',1300000);

INSERT INTO DEPT VALUES(102,'HR',1400000);

INSERT INTO DEPT VALUES(103,'RECEPTION',400000);

INSERT INTO DEPT VALUES(104,'RECEPTION',400000);

INSERT INTO DEPT VALUES(105,'HR',4000000);

dno	dname	budget
101	account	1300000
102	HR	1400000
103	reception	400000
104	reception	400000
105	HR	4000000

5 rows in set (0.00 sec)

1. RETRIEVE THE ID, NAME AND SALARY OF FACULTY IN INCREASING ORDER OF SALARY.

COMMAND: SELECT FID,NAME,SAL FROM FACULTY ORDER BY SAL;

fid	name	sal
1	Vikram	10000
5	hsf	11000
2	raj	15000
4	hsf	110000
3	hgj	150000

5 rows in set (0.00 sec)

2. RETRIEVE THE ID, NAME AND SALARY PF FACULTY IN DECREASING ORDER OF EXPERIENCE.

COMMAND: SELECT FID,NAME,SAL FROM FACULTY ORDER BY EXP DESC;

fid	name	sal
2	raj	15000
1	Vikram	10000
4	hsf	110000
3	hgj	150000
5	hsf	11000

5 rows in set (0.00 sec)

3. RETRIEVE THE ID, NAME AND SALARY OF FACULTY IN INCREASING ORDER OF SALARY AND DECREASING ORDER OF EXPERIENCE.

COMMAND: SELECT FID,NAME,SAL FROM FACULTY ORDER BY SAL ASC, EXP DESC;

```

+-----+-----+-----+
| fid | name | sal |
+-----+-----+-----+
| 1 | Vikram | 10000 |
| 5 | hsf | 11000 |
| 2 | raj | 15000 |
| 4 | hsf | 110000 |
| 3 | hgj | 150000 |
+-----+-----+-----+
5 rows in set (0.00 sec)

```

4.SHOW THE AVERAGE EXPERIENCE DEPARTMENT WISE.

COMMAND: SELECT AVG(EXP) FROM FACULTY GROUP BY DNO;

```

+-----+
| avg(exp) |
+-----+
| 4.0000 |
| 5.0000 |
| 2.0000 |
| 3.0000 |
| 2.0000 |
+-----+
5 rows in set (0.00 sec)

```

5. RETRIEVE THE MAXIMUM SALARY DEPARTMENT WISE.

COMMAND: SELECT MAX(SAL),DNO FROM FACULTY GROUP BY DNO;

```

+-----+-----+
| max(sal) | dno |
+-----+-----+
| 10000 | 101 |
| 15000 | 102 |
| 150000 | 103 |
| 110000 | 104 |
| 11000 | 105 |
+-----+-----+
5 rows in set (0.00 sec)

```

6. RETRIEVE THE AVERAGE SALARY DEPARTMENT WISE HAVING AVERAGE SALARY GREATER THAN 15000.

COMMAND: SELECT AVG(SAL) DNO FROM FACULTY WHERE SAL>15000 GROUP BY DNO;

```

+-----+-----+
| avg(sal) | dno |
+-----+-----+
| 150000.0000 | 103 |
| 110000.0000 | 104 |
+-----+-----+
2 rows in set (0.00 sec)

```

7. RETRIEVE FACULTIES DETAILS WHERE NAME (TABLE SHOULD HAVE NAMES ACCORDINGLY).

1. STARTING WITH V.

COMMAND: SELECT * FROM FACULTY WHERE NAME LIKE 'V%';

fid	name	dno	sal	address	phone	dob	exp
1	Vikram	101	10000	ABC	9988776655	2001-03-02	4

1 row in set (0.00 sec)

2. STARTING WITH R.

COMMAND: SELECT * FROM FACULTY WHERE NAME LIKE 'R%';

fid	name	dno	sal	address	phone	dob	exp
2	raj	102	15000	ABC	9988776645	2002-12-03	5

1 row in set (0.00 sec)

3. STARTING WITH A AND ENDING WITH J.

COMMAND: SELECT * FROM FACULTY WHERE NAME LIKE '%A%J';

fid	name	dno	sal	address	phone	dob	exp
2	raj	102	15000	ABC	9988776645	2002-12-03	5

1 row in set (0.00 sec)

4. MUST CONTAIN LETTER M.

COMMAND: SELECT * FROM FACULTY WHERE NAME LIKE '%M%';

fid	name	dno	sal	address	phone	dob	exp
1	Vikram	101	10000	ABC	9988776655	2001-03-02	4

1 row in set (0.00 sec)

5. SHOULD HAVE 3 LETTER ONLY STATING WITH H.

COMMAND: SELECT * FROM FACULTY WHERE NAME LIKE 'H__';

fid	name	dno	sal	address	phone	dob	exp
3	hgj	103	150000	ABC	9988576645	2004-12-27	2
4	hsf	104	110000	ABC	9988576642	2004-03-04	3
5	hsf	105	11000	ABC	9988576632	2004-05-12	2

3 rows in set (0.00 sec)

8. SHOW THE USE OF ANY/ALL CLAUSE. (WRITE QUERIES YOURSELF).

COMMAND:

SELECT NAME

-> FROM FACULTY

-> WHERE SAL < ANY (SELECT MIN(BUDGET) FROM DEPT);

name
Vikram
raj
hgj
hsf
hsf

5 rows in set (0.00 sec)

SELECT DNAME

FROM DEPT

WHERE 0 < ALL (SELECT EXP FROM FACULTY WHERE SAL < 150000);

dname
account
HR
reception
reception
HR

5 rows in set (0.00 sec)

9. SHOW THE USE OF EXIST/ NOT EXIST CLAUSES.

COMMAND:

SELECT FID

FROM FACULTY

WHERE EXISTS (SELECT DNAME FROM DEPT WHERE DEPT.DNO = FACULTY.DNO AND EXP > 2);

```

+-----+
|  fid  |
+-----+
|    1  |
|    2  |
|    4  |
+-----+
3 rows in set (0.00 sec)

```

SELECT FID FROM FACULTY

WHERE NOT EXISTS (SELECT DNAME FROM DEPT WHERE DEPT.DNO =
FACULTY.DNO AND EXP > 2);

```

+-----+
|  fid  |
+-----+
|    3  |
|    5  |
+-----+
2 rows in set (0.00 sec)

```

10. FIND THE NAMES OF FACULTY OF FACULTY WHO WORK IN DEPARTMENT
HR (USING SUBQUERY).

COMMAND:

SELECT NAME

FROM FACULTY

WHERE FID IN (

SELECT F.FID

FROM FACULTY F

JOIN DEPT D ON F.DNO = D.DNO

WHERE D.DNAME = 'HR');

```

+-----+
| name |
+-----+
| raj  |
| hsf  |
+-----+
2 rows in set (0.00 sec)

```

EXPERIMENT-6

AIM: Implement queries include various set of operation.

Theory:

CREATE A TABLE SECTION WITH THE COLUMNS (COURSEID, SECTION_ID, SEMESTER, YEAR, BUILDING, ROOM NO).

FIND COURSES THAT RAN IN FALL 2017

FIND COURSES THAT RAN IN SPRING 2018

FIND COURSES THAT RAN IN FALL 2017 OR IN SPRING 2018

FIND COURSES THAT RAN IN FALL 2017 AND IN SPRING 2018

FIND COURSES THAT RAN IN FALL 2017 BUT NOT IN SPRING 2018

FIND COURSES THAT RAN IN SPRING 2018 BUT NOT IN SPRING 2017

1. Create table with above given attributes and insert values in it.

Command: CREATE TABLE SECTION(

COURSE_ID VARCHAR(10),

SEC_ID VARCHAR(10),

SEM VARCHAR(10),

YEAR INT,

BUILDING VARCHAR(10),

ROOM_NO INT);

INSERT INTO SECTION VALUES ('B10-101', 1,'SUMMER', '2017', 'PAINTER',514);

INSERT INTO SECTION VALUES ('B10-102', 1,'FALL', '2018', 'PAINTER',5134),

INSERT INTO SECTION VALUES('C011', 2,'SUMMER', '2017','PACKERED', 101);

INSERT INTO SECTION VALUES('C011', 1,'FALL', '2018','TAYLOR', 1023);

INSERT INTO SECTION VALUES('FU-34',1,'SPRING', '2017','TAYLOR',242);

INSERT INTO SECTION VALUES('FU-34',1,'SUMMER','2018','PARKERED',2345);

INSERT INTO SECTION VALUES('FU-354',2,'SUMMER','2017','TAYLOR',232); etc.

SELECT * FROM SECTION

COURSE_ID	SEC_ID	SEM	YEAR	BUILDING	ROOM_NO
B10-101	1	SUMMER	2017	PAINTER	514
B10-102	1	FALL	2018	PAINTER	5134
B10-102	1	FALL	2018	PAINTER	5134
C011	2	SUMMER	2017	PACKERED	101
C011	1	FALL	2018	TAYLOR	1023
FU-34	1	SPRING	2017	TAYLOR	242
FU-34	1	SUMMER	2018	PARKERED	2345
FU-354	2	SUMMER	2017	TAYLOR	232
FU-354	2	FALL	2017	TAYLOR	232
KU-324	2	FALL	2018	PAINTER	986
KU-324	2	FALL	2017	PAINTER	986
KU24	1	SPRING	2018	TAYLOR	644
JH78	2	SPRING	2018	PAINTER	632
FU-34	1	SPRING	2018	TAYLOR	242
AI01-150	2	FALL	2017	PAINTER	46
AI01-150	2	SPRING	2018	PAINTER	46

16 rows in set (0.02 sec)

2. FIND COURSES THAT RAN IN FALL 2017

COMMAND: SELECT * FROM SECTION WHERE SEM="FALL" AND YEAR=2017;

COURSE_ID	SEC_ID	SEM	YEAR	BUILDING	ROOM_NO
FU-354	2	FALL	2017	TAYLOR	232
KU-324	2	FALL	2017	PAINTER	986
AI01-150	2	FALL	2017	PAINTER	46

3 rows in set (0.00 sec)

3. FIND COURSES THAT RAN IN SPRING 2018

COMMAND: SELECT * FROM SECTION WHERE SEM="SPRING" AND YEAR=2018;

COURSE_ID	SEC_ID	SEM	YEAR	BUILDING	ROOM_NO
KU24	1	SPRING	2018	TAYLOR	644
JH78	2	SPRING	2018	PAINTER	632
FU-34	1	SPRING	2018	TAYLOR	242
AI01-150	2	SPRING	2018	PAINTER	46

4 rows in set (0.00 sec)

4. FIND COURSES THAT RAN IN FALL 2017 OR IN SPRING 2018

COMMAND: SELECT COURSE_ID FROM SECTION WHERE SEM="FALL" AND YEAR=2017 UNION SELECT COURSE_ID FROM SECTION WHERE SEM="SPRING" AND YEAR=2018;

COURSE_ID
FU-354
KU-324
KU24
JH78
FU-34
AI01-150
AI01-150

7 rows in set (0.00 sec)

5. FIND COURSES THAT RAN IN FALL 2017 AND IN SPRING 2018

COMMAND: SELECT COURSE_ID FROM SECTION WHERE SEM="FALL" AND YEAR=2017 UNION SELECT COURSE_ID FROM SECTION WHERE SEM="SPRING" AND YEAR=2018;

```

+-----+
| COURSE_ID |
+-----+
| AI01-150  |
| B01-101   |
+-----+
2 rows in set (0.00 sec)

```

6. FIND COURSES THAT RAN IN FALL 2017 BUT NOT IN SPRING 2018

COMMAND: (SELECT COURSE_ID FROM SECTION WHERE SEM='FALL' AND YEAR=2017) EXCEPT (SELECT COURSE_ID FROM SECTION WHERE SEM='SPRING' AND YEAR=2018);

```

+-----+
| COURSE_ID |
+-----+
| FU-354     |
| KU-324     |
+-----+
2 rows in set (0.00 sec)

```

7. FIND COURSES THAT RAN IN SPRING 2018 BUT NOT IN SPRING 2017

COMMAND: (SELECT COURSE_ID FROM SECTION WHERE SEM='SPRING' AND YEAR=2018) EXCEPT (SELECT COURSE_ID FROM SECTION WHERE SEM='SPRING' AND YEAR=2017);

```

+-----+
| COURSE_ID |
+-----+
| KU24       |
| JH78       |
| AI01-150   |
+-----+
3 rows in set (0.00 sec)

```

EXPERIMENT-7

AIM: Implement various join operations - inner, outer.

Theory:

CREATE A TABLE STUDENTS WITH THE COLUMNS (ROLL_NO, NAME, ADDDRES, PHONE, AGE).

CREATE A TABLE STUDENT_COURSE WITH THE COLUMNS (ROLL_NO, COURSE_ID).

CREATE A TABLE COURSE WITH THE COLUMNS (COURSE_ID, C_NAME).

Commands:

```
1.CREATE TABLE STUDENTS(  
  ROLL_NO INT PRIMARY KEY,  
  NAME VARCHAR(10) UNIQUE,  
  ADDRESS VARCHAR(10),  
  PHONE INT,  
  AGE INT NOT NULL);
```

```
INSERT INTO STUDENTS VALUES (1, 'John', 'ABC', 1234567890, 20);
```

```
INSERT INTO STUDENTS VALUES (2, 'Alice', 'EDS', 9876543210, 22);
```

```
INSERT INTO STUDENTS VALUES (3, 'Bob', 'YTG', 5551234, 21);
```

```
INSERT INTO STUDENTS VALUES (4, 'Emily', 'YHT', 8889990, 19);
```

```
INSERT INTO STUDENTS VALUES (5, 'Michael', 'HGS', 1112223, 20);
```

```

2. CREATE TABLE COURSE (
COURSE_ID INT PRIMARY KEY,
C_NAME VARCHAR (10));
INSERT INTO COURSE VALUES (1, 'Math');
INSERT INTO COURSE VALUES (2, 'Science');
INSERT INTO COURSE VALUES (3, 'History');
INSERT INTO COURSE VALUES (4, 'English');
INSERT INTO COURSE VALUES (5, 'CS');

```

```

3. CREATE TABLE STUDENT_COURSE (COURSE_ID INT,ROLL_NO INT);
INSERT INTO STUDENT_COURSE VALUES (1, 1);
INSERT INTO STUDENT_COURSE VALUES (2, 2);
INSERT INTO STUDENT_COURSE VALUES (3, 3);
INSERT INTO STUDENT_COURSE VALUES (4, 4);
INSERT INTO STUDENT_COURSE VALUES (5, 5);

```

Queries

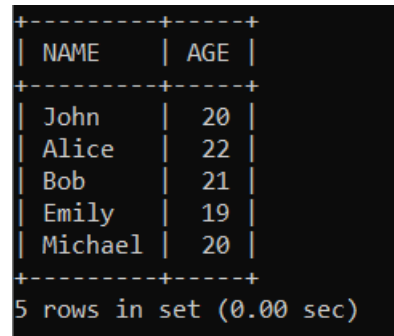
1- RETRIEVE THE NAMES AND AGE OF STUDENTS ENROLLED IN DIFFERENT COURSES.

Command:

```

SELECT S.NAME, S.AGE, C.C_NAME
FROM STUDENTS S
LEFT OUTER JOIN STUDENT_COURSE SC ON S.ROLL_NO = SC.ROLL_NO
LEFT OUTER JOIN COURSE C ON SC.COURSE_ID = C.COURSE_ID;

```



NAME	AGE
John	20
Alice	22
Bob	21
Emily	19
Michael	20

5 rows in set (0.00 sec)

2- APPLY LEFT OUTER JOIN, RIGHT OUTER JOIN AND FULL OUTER JOIN ON TABLES.

Command:

```

SELECT S.NAME, S.AGE, C.C_NAME
FROM STUDENT_COURSE SC
RIGHT OUTER JOIN STUDENTS S ON S.ROLL_NO = SC.ROLL_NO
RIGHT OUTER JOIN COURSE C ON SC.COURSE_ID = C.COURSE_ID;
SELECT S.NAME, S.AGE, C.C_NAME

```

FROM STUDENTS S

```

LEFT OUTER JOIN STUDENT_COURSE SC ON S.ROLL_NO = SC.ROLL_NO
LEFT OUTER JOIN COURSE C ON SC.COURSE_ID = C.COURSE_ID;

```

```

SELECT S.NAME, S.AGE, C.C_NAME
FROM STUDENTS S
FULL OUTER JOIN STUDENT_COURSE SC ON S.ROLL_NO = SC.ROLL_NO
FULL OUTER JOIN COURSE C ON SC.COURSE_ID = C.COURSE_ID;

```

NAME	AGE	C_NAME
John	20	Math
Alice	22	Science
Bob	21	History
Emily	19	English
Michael	20	CS

5 rows in set (0.00 sec)

3- RETRIEVE THE NAMES AND ADDRESS OF STUDENTS ENROLLED IN COURSES WITH COURSE_ID=3.

```

SELECT S.NAME, S.ADDRESS
FROM STUDENTS S
JOIN STUDENT_COURSE SC ON S.ROLL_NO = SC.ROLL_NO
JOIN COURSE C ON SC.COURSE_ID = C.COURSE_ID
WHERE C.COURSE_ID = 3;

```

NAME	ADDRESS
Bob	YTG

1 row in set (0.00 sec)

4- RETRIEVE THE NAMES AND PHONE OF STUDENTS ENROLLED IN COURSES WITH COURSE_ID=2 AND AGE=22.

```

SELECT S.NAME, S.PHONE
FROM STUDENTS S
JOIN STUDENT_COURSE SC ON S.ROLL_NO = SC.ROLL_NO
JOIN COURSE C ON SC.COURSE_ID = C.COURSE_ID
WHERE C.COURSE_ID = 2 AND S.AGE = 22;

```

NAME	PHONE
Alice	9876543210

1 row in set (0.00 sec)

5- RETRIEVE THE NAMES, AGE, COURSE_ID AND COURSE_NAME OF STUDENTS ENROLLED IN DIFFERENT COURSE.

```

SELECT S.NAME, S.AGE, C.COURSE_ID, C.C_NAME
FROM STUDENTS S

```

```
JOIN STUDENT_COURSE SC ON S.ROLL_NO = SC.ROLL_NO
JOIN COURSE C ON SC.COURSE_ID = C.COURSE_ID;
```

NAME	AGE	COURSE_ID	C_NAME
John	20	1	Math
Alice	22	2	Science
Bob	21	3	History
Emily	19	4	English
Michael	20	5	CS

5 rows in set (0.00 sec)

6- RETRIEVE THE NAMES OF STUDENTS ENROLLED IN CSE.

```
SELECT S.NAME
FROM STUDENTS S
JOIN STUDENT_COURSE SC ON S.ROLL_NO = SC.ROLL_NO
JOIN COURSE C ON SC.COURSE_ID = C.COURSE_ID
WHERE C.C_NAME = 'CS';
```

NAME
Michael

1 row in set (0.00 sec)

EXPERIMENT-8

AIM: Write a PL/SQL program using FOR loop to insert ten rows into a database table.

Theory:

CREATE A TABLE AND INSERT VALUES INTO IT USING FOR LOOP.

COMMAND:

```
1. CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY,  
    FirstName VARCHAR(10),  
    LastName VARCHAR(10),  
    Department VARCHAR(10),  
    Salary INT);  
DECLARE  
    v_counter NUMBER := 1;  
BEGIN  
    FOR v_counter IN 1..10 LOOP  
        INSERT INTO Employees (EmployeeID, FirstName, LastName, Department, Salary)  
        VALUES (v_counter, 'First' || v_counter, 'Last' || v_counter, 'IT', v_counter * 1000);  
    END LOOP;  
    COMMIT;  
END;  
/  
  
SELECT * FROM Employees;
```

EMPLOYEEID	FIRSTNAME	LASTNAME	DEPARTMENT	SALARY
1	First1	Last1	IT	1000
2	First2	Last2	IT	2000
3	First3	Last3	IT	3000
4	First4	Last4	IT	4000
5	First5	Last5	IT	5000
6	First6	Last6	IT	6000
7	First7	Last7	IT	7000
8	First8	Last8	IT	8000
9	First9	Last9	IT	9000
10	First10	Last10	IT	10000

EXPERIMENT-9

AIM: Given the table EMPLOYEE (Emp No, Name, Salary, Designation, Dept_ID), write a cursor to select the five highest-paid employees from the table.

Theory:

CREATE A TABLE EMPLOYEE (Emp No, Name, Salary, Designation, Dept_ID). AND INSERT VALUES TO IT.

```
CREATE TABLE EMPLOYEE (  
    Emp_No INT PRIMARY KEY,  
    Name VARCHAR(20),  
    Salary INT,  
    Designation VARCHAR(20),  
    Dept_ID INT );  
INSERT INTO EMPLOYEE (Emp_No, Name, Salary, Designation, Dept_ID) VALUES  
(1, 'John Doe', 60000, 'Manager', 101),  
(2, 'Jane Smith', 55000, 'Senior Engineer', 102),  
(3, 'Michael Johnson', 58000, 'Senior Developer', 103),  
(4, 'Emily Davis', 52000, 'Software Engineer', 102),  
(5, 'David Wilson', 62000, 'Project Manager', 101),  
(6, 'Sarah Brown', 53000, 'QA Analyst', 104),  
(7, 'Robert Jones', 57000, 'Database Administrator', 105),  
(8, 'Jennifer Taylor', 54000, 'Business Analyst', 106),  
(9, 'William Martinez', 59000, 'System Analyst', 107),  
(10, 'Amanda Anderson', 56000, 'Network Engineer', 108);
```

EMP_NO	NAME	SALARY	DESIGNATION	DEPT_ID
1	John Doe	60000	Manager	101
2	Jane Smith	55000	Senior Engineer	102
3	Michael Johnson	58000	Senior Developer	103
4	Emily Davis	52000	Software Engineer	102
5	David Wilson	62000	Project Manager	101
6	Sarah Brown	53000	QA Analyst	104
7	Robert Jones	57000	Database Administrator	105
8	Jennifer Taylor	54000	Business Analyst	106
9	William Martinez	59000	System Analyst	107
10	Amanda Anderson	56000	Network Engineer	108

DECLARE

CURSOR highest_paid_cursor IS

SELECT Emp_No, Name, Salary, Designation, Dept_ID

FROM EMPLOYEE

ORDER BY Salary DESC

FETCH FIRST 5 ROWS ONLY; -- Select only the top 5 highest-paid employees

v_Emp_No EMPLOYEE.Emp_No%TYPE;

v_Name EMPLOYEE.Name%TYPE;

v_Salary EMPLOYEE.Salary%TYPE;

v_Designation EMPLOYEE.Designation%TYPE;

v_Dept_ID EMPLOYEE.Dept_ID%TYPE;

BEGIN

OPEN highest_paid_cursor;

DBMS_OUTPUT.PUT_LINE('-----
-----');

DBMS_OUTPUT.PUT_LINE(' | Employee No | Name | Salary | Designation |
Dept ID |');

DBMS_OUTPUT.PUT_LINE('-----
-----');

LOOP

```
    FETCH highest_paid_cursor INTO v_Emp_No, v_Name, v_Salary, v_Designation,
    v_Dept_ID;
```

```
    EXIT WHEN highest_paid_cursor%NOTFOUND;
```

```
    DBMS_OUTPUT.PUT_LINE(' ' || RPAD(v_Emp_No, 12) || ' ' || RPAD(v_Name, 15) || ' ' ||
    RPAD(v_Salary, 7) || ' ' || RPAD(v_Designation, 22) || ' ' || RPAD(v_Dept_ID, 7) || '');
```

```
    END LOOP;
```

```
    DBMS_OUTPUT.PUT_LINE('-----
    -----');
```

```
    CLOSE highest_paid_cursor;
```

```
END;
```

```
/
```

Employee No	Name	Salary	Designation	Dept ID

5	David Wilson	62000	Project Manager	101
1	John Doe	60000	Manager	101
9	William Martine	59000	System Analyst	107
3	Michael Johnson	58000	Senior Developer	103
7	Robert Jones	57000	Database Administrator	105

SECTION 2

BEYOND CURRICULUM

EXPERIMENT-1

AIM: Write the steps to install and implement NOSQL databases-MongoDB.

1. Download MongoDB:

Visit the official MongoDB website: <https://www.mongodb.com/try/download/community>

Choose the appropriate version for your operating system (Windows, macOS, Linux) and download it.

2. Install MongoDB:

Follow the installation instructions provided for your operating system.

For Windows:

Run the downloaded `.msi` file and follow the installation wizard.

MongoDB by default installs in `C:\Program Files\MongoDB\Server<version>` and the data directory in `C:\data\db`.

3. Start MongoDB:

For Windows:

MongoDB should start automatically after installation. If not, you can start it from the Services panel.

4. Connect to MongoDB:

Use the MongoDB shell (`mongo`) to connect to the MongoDB server:

`mongo`. By default, the shell connects to the MongoDB instance running on `localhost` on port `27017`.

5. Create and Manage Databases and Collections:

In MongoDB, databases and collections are created implicitly when data is inserted.

To create a new database explicitly, use the `use` command:

`use mydatabase`. To create a new collection, you can simply insert a document into it:

`db.myCollection.insertOne({ key: "value" })`

6. Perform CRUD Operations:

MongoDB supports CRUD operations (Create, Read, Update, Delete) using its shell or various programming language drivers.

Use `insertOne`, `insertMany`, `find`, `updateOne`, `updateMany`, `deleteOne`, `deleteMany`, etc., methods to perform operations on documents.

7. Security and Authentication:

Secure your MongoDB installation by enabling authentication.

Create users with appropriate roles and permissions.

Modify the MongoDB configuration file (`mongod.conf`) to enable authentication and set up user authentication.

8. Backup and Restore:

MongoDB provides utilities like `mongodump` and `mongorestore` for backup and restore operations. Use `mongodump` to create backups of databases and collections, and `mongorestore` to restore them.

9. Explore MongoDB Documentation and Resources:

MongoDB has extensive documentation available online, covering installation, configuration, usage, and best practices. Explore tutorials, guides, and forums to learn more about MongoDB and its features.

10. Practice and Experiment:

Practice writing queries, performing CRUD operations, and managing databases and collections. Experiment with different features and options to understand MongoDB's capabilities fully.



Experiment 2

Experiment 2: Study and implement basic commands of MongoDB.

Theory:

SQL Commands:

1. Creating and using database

```
test> use mydatabase  
switched to db mydatabase
```

2. Creating Collection

```
mydatabase> db.createCollection("mycollection")  
{ ok: 1 }
```

3. Getting list of all collections

```
mydatabase> show collections  
mycollection
```

4. Inserting document in collection

```
mydatabase> db.mycollection.insertOne({ name: "John", age: 30 })
{
  acknowledged: true,
  insertedId: ObjectId('662a881e677a8cd27d46b79d')
}
```

```
mydatabase> db.mycollection.insertMany([
... { name: "Alice", age: 25 },
... { name: "Bob", age: 35 }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('662a883b677a8cd27d46b79e'),
    '1': ObjectId('662a883b677a8cd27d46b79f')
  }
}
```

5. Get collection document

```
mydatabase> db.mycollection.find()
[
  { _id: ObjectId('662a881e677a8cd27d46b79d'), name: 'John', age: 30 },
  { _id: ObjectId('662a883b677a8cd27d46b79e'), name: 'Alice', age: 25 },
  { _id: ObjectId('662a883b677a8cd27d46b79f'), name: 'Bob', age: 35 }
]
```

```
mydatabase> db.mycollection.find({ name: "John" })
[
  { _id: ObjectId('662a881e677a8cd27d46b79d'), name: 'John', age: 30 }
]
```

6. Updating Document

```
mydatabase> db.mycollection.updateOne({ name: "John" }, { $set: { age: 32 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```



```

mydatabase> db.mycollection.find()
[
  { _id: ObjectId('662a881e677a8cd27d46b79d'), name: 'John', age: 32 },
  { _id: ObjectId('662a883b677a8cd27d46b79e'), name: 'Alice', age: 25 },
  { _id: ObjectId('662a883b677a8cd27d46b79f'), name: 'Bob', age: 35 }
]

mydatabase> db.mycollection.updateMany({ age: { $gt: 30 } }, { $set: { status:
"adult" } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
mydatabase> db.mycollection.find()
[
  {
    _id: ObjectId('662a881e677a8cd27d46b79d'),
    name: 'John',
    age: 32,
    status: 'adult'
  },
  { _id: ObjectId('662a883b677a8cd27d46b79e'), name: 'Alice', age: 25 },
  {
    _id: ObjectId('662a883b677a8cd27d46b79f'),
    name: 'Bob',
    age: 35,
    status: 'adult'
  }
]

```

7. Deleting Document

```

mydatabase> db.mycollection.deleteOne({ name: "John" })
{ acknowledged: true, deletedCount: 1 }

mydatabase> db.mycollection.find()
[
  { _id: ObjectId('662a883b677a8cd27d46b79e'), name: 'Alice', age: 25 },
  {
    _id: ObjectId('662a883b677a8cd27d46b79f'),
    name: 'Bob',
    age: 35,
    status: 'adult'
  }
]

mydatabase> db.mycollection.deleteMany({ age: { $lt: 30 } })
{ acknowledged: true, deletedCount: 1 }

```

```
mydatabase> db.mycollection.find()
[
  {
    _id: ObjectId('662a883b677a8cd27d46b79f'),
    name: 'Bob',
    age: 35,
    status: 'adult'
  }
]
```

8. Dropping Collection

```
mydatabase> db.mycollection.drop()
true
mydatabase> db.mycollection.find()
```

9. Dropping Database

```
mydatabase> db.dropDatabase()
{ ok: 1, dropped: 'mydatabase' }
```

Learning outcome

Experiment 3

Experiment 3: Implement any one real-time project using MySQL/MongoDB such as Library Database Management System etc.

Theory:

Library Management System

The library management system is a database-driven application designed to efficiently manage the operations of a library. It allows librarians to keep track of books, users, book loans, fines, and more. Here's a brief description of the project:

Features:

User Management: Allows librarians to manage users, including adding new users, updating user information, and deleting users.

Book Management: Provides functionalities for managing books in the library, including adding new books, updating book information, and removing books from the collection.

Book Issuance: Enables librarians to issue books to users, record the issue date, and set return dates. It also automatically updates the availability of books in the inventory.

Fine Management: Calculates fines for late returns and records fines in the database. It also provides functionalities for viewing and managing fines.

Authentication and Authorization: Supports user authentication and authorization to ensure that only authorized users can perform certain actions, such as issuing books or managing user accounts.

Database Schema:

The database schema includes tables for users, books, book issues, and fines, along with appropriate relationships between them.

Triggers are implemented to enforce business rules, automate tasks, and maintain data integrity.

Benefits:

Efficiently manages library operations, reducing manual effort and errors.

Provides real-time access to information about books, users, and transactions.

Enhances user experience by automating fine calculations and providing timely notifications.

Use Cases:

Librarians can easily add, update, or remove books from the library collection.

Users can borrow books, return them on time to avoid fines, and view their borrowing history.

Administrators can monitor library activities, manage user accounts, and generate reports for analysis.

Entity Descriptions:

Users: Represents users of the library system. Each user has a unique user_id, username, password, and role (e.g., admin, librarian, member).

Books: Represents books available in the library. Each book has a unique book_id, title, author, isbn, publication_year, and quantity_available.

BookIssues: Represents the issuance of books to users. Each issue has a unique issue_id, references a user_id from the Users table, and a book_id from the Books table. It also includes issue_date, return_date, and status (issued or returned).

Fines: Represents fines imposed on users for late returns. Each fine has a unique fine_id, references a user_id and issue_id, and includes fine_amount, fine_reason, and fine_date.

Relationship Descriptions:

Entity Relations:

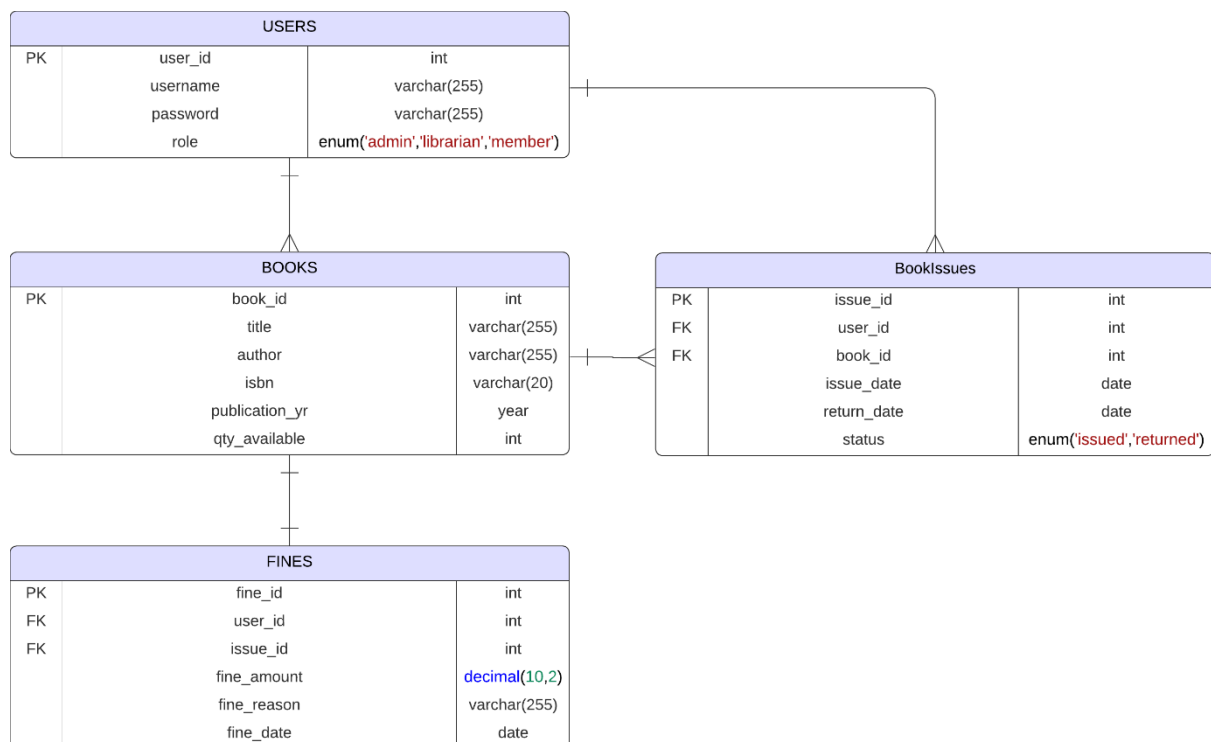
Users - Books: One-to-Many relationship where a user can issue multiple books, but each book can be issued to multiple users.

Books - BookIssues: One-to-Many relationship where each book can have multiple issues, but each issue is associated with only one book.

Users - BookIssues: One-to-Many relationship where each user can have multiple issued books, but each issue is associated with only one user.

BookIssues - Fines: One-to-One relationship where each issue can have only one fine, but each fine is associated with only one issue.

ER Diagram:



Triggers:

Trigger 1: This trigger updates the quantity_available column in the Books table after a book is issued (inserted into the BookIssues table).

Trigger 2: This trigger updates the quantity_available column in the Books table before a book is returned (deleted from the BookIssues table).

Trigger 3: This trigger calculates fines for late returns. It calculates the number of days between the issue date and return date, and if the return date is more than 15 days after the issue date, it inserts a fine record into the Fines table.

SQL Commands:

1. Creating and Using Database:

```
mysql> CREATE DATABASE LibraryDB;  
Query OK, 1 row affected (0.01 sec)
```

```
mysql> USE LibraryDB;  
Database changed
```

2. Creating table with constraints:

- a. Users:
 - i. user_id (Primary Key)
 - ii. username
 - iii. password
 - iv. role (e.g., admin, librarian, member)
- b. Books:
 - i. book_id (Primary Key)
 - ii. title
 - iii. author
 - iv. isbn
 - v. publication_year
 - vi. quantity_available
- c. BookIssues:
 - i. issue_id (Primary Key)
 - ii. user_id (Foreign Key referencing Users Table)
 - iii. book_id (Foreign Key referencing Books Table)
 - iv. issue_date
 - v. return_date
 - vi. status (e.g., issued, returned)
- d. Fines:
 - i. fine_id (Primary Key)
 - ii. user_id (Foreign Key referencing Users Table)
 - iii. issue_id (Foreign Key referencing Book Issues Table)
 - iv. fine_amount
 - v. fine_reason
 - vi. fine_date

```
mysql> -- Create Users Table
```

```
mysql> CREATE TABLE Users (
->     user_id INT AUTO_INCREMENT PRIMARY KEY,
->     username VARCHAR(255) NOT NULL UNIQUE,
->     password VARCHAR(255) NOT NULL,
->     role ENUM('admin', 'librarian', 'member') NOT NULL
-> );
```

Query OK, 0 rows affected (0.16 sec)

```
mysql> -- Create Books Table
```

```
mysql> CREATE TABLE Books (
->     book_id INT AUTO_INCREMENT PRIMARY KEY,
->     title VARCHAR(255) NOT NULL,
->     author VARCHAR(255) NOT NULL,
->     isbn VARCHAR(20) NOT NULL UNIQUE,
->     publication_year YEAR NOT NULL,
->     quantity_available INT NOT NULL
-> );
```

Query OK, 0 rows affected (0.14 sec)

```
mysql> -- Create Book Issues Table
```

```
mysql> CREATE TABLE BookIssues (
->     issue_id INT AUTO_INCREMENT PRIMARY KEY,
->     user_id INT,
->     book_id INT,
->     issue_date DATE NOT NULL,
->     return_date DATE,
->     status ENUM('issued', 'returned') NOT NULL,
->     FOREIGN KEY (user_id) REFERENCES Users(user_id),
->     FOREIGN KEY (book_id) REFERENCES Books(book_id)
-> );
```

Query OK, 0 rows affected (0.19 sec)

```
mysql> -- Create Fines Table
```

```
mysql> CREATE TABLE Fines (
->     fine_id INT AUTO_INCREMENT PRIMARY KEY,
->     user_id INT,
->     issue_id INT,
->     fine_amount DECIMAL(10, 2) NOT NULL,
->     fine_reason VARCHAR(255) NOT NULL,
->     fine_date DATE NOT NULL,
->     FOREIGN KEY (user_id) REFERENCES Users(user_id),
->     FOREIGN KEY (issue_id) REFERENCES BookIssues(issue_id)
-> );
```

Query OK, 0 rows affected (0.19 sec)

```
mysql> DESC users;
```

Field	Type	Null	Key	Default	Extra
user_id	int	NO	PRI	NULL	auto_increment
username	varchar(255)	NO	UNI	NULL	

password	varchar(255)	NO		NULL	
role	enum('admin','librarian','member')	NO		NULL	

4 rows in set (0.01 sec)

mysql> DESC books;

Field	Type	Null	Key	Default	Extra
book_id	int	NO	PRI	NULL	auto_increment
title	varchar(255)	NO		NULL	
author	varchar(255)	NO		NULL	
isbn	varchar(20)	NO	UNI	NULL	
publication_year	year	NO		NULL	
quantity_available	int	NO		NULL	

6 rows in set (0.00 sec)

mysql> DESC BookIssues;

Field	Type	Null	Key	Default	Extra
issue_id	int	NO	PRI	NULL	auto_increment
user_id	int	YES	MUL	NULL	
book_id	int	YES	MUL	NULL	
issue_date	date	NO		NULL	
return_date	date	YES		NULL	
status	enum('issued','returned')	NO		NULL	

6 rows in set (0.00 sec)

mysql> DESC Fines;

Field	Type	Null	Key	Default	Extra
fine_id	int	NO	PRI	NULL	auto_increment
user_id	int	YES	MUL	NULL	
issue_id	int	YES	MUL	NULL	
fine_amount	decimal(10,2)	NO		NULL	
fine_reason	varchar(255)	NO		NULL	
fine_date	date	NO		NULL	

6 rows in set (0.00 sec)

3. Creating Triggers:

a. Issue a Book to a User

```
mysql> CREATE TRIGGER update_quantity_available AFTER INSERT ON BookIssues
-> FOR EACH ROW
-> BEGIN
->     UPDATE Books
->     SET quantity_available = quantity_available - 1
->     WHERE book_id = NEW.book_id;
-> END;
-> //
```

Query OK, 0 rows affected (0.03 sec)

b. Return a Book from a User

```
mysql> CREATE TRIGGER update_quantity_available_return BEFORE DELETE ON
BookIssues
-> FOR EACH ROW
-> BEGIN
->     UPDATE Books
->     SET quantity_available = quantity_available + 1
->     WHERE book_id = OLD.book_id;
-> END;
-> //
```

Query OK, 0 rows affected (0.04 sec)

c. Calculate Fines for Late Returns

```
mysql> CREATE TRIGGER calculate_fine AFTER INSERT ON BookIssues
-> FOR EACH ROW
-> BEGIN
->     DECLARE fine_days INT;
->     DECLARE fine_amount DECIMAL(10, 2);
->
->     IF NEW.return_date IS NOT NULL
        AND NEW.return_date > NEW.issue_date THEN
->         SET fine_days = DATEDIFF(NEW.return_date, NEW.issue_date);
->         IF fine_days > 15 THEN
->             SET fine_amount = (fine_days - 15) * 0.5;
->             INSERT INTO Fines (user_id, issue_id, fine_amount,
                                fine_reason, fine_date)
->             VALUES (NEW.user_id, NEW.issue_id, fine_amount,
                        'Late return', NOW());
->         END IF;
->     END IF;
-> END;
-> //
```

Query OK, 0 rows affected (0.13 sec)

4. Inserting Sample Data:

```
mysql> -- Sample data for Users Table
mysql> INSERT INTO Users (username, password, role) VALUES
-> ('admin', 'admin123', 'admin'),
-> ('librarian', 'librarian123', 'librarian'),
-> ('user1', 'user123', 'member'),
-> ('user2', 'user456', 'member');
```

Query OK, 4 rows affected (0.02 sec)

Records: 4 Duplicates: 0 Warnings: 0

```
mysql> -- Sample data for Books Table
mysql> INSERT INTO Books (title, author, isbn, publication_year,
    quantity_available) VALUES
-> ('Book1', 'Author1', '978-3-16-148410-0', '2020', 6),
```



```

-> ('Book2', 'Author2', '978-3-16-148411-0', '2019', 4),
-> ('Book3', 'Author3', '978-3-16-148412-0', '2021', 7);
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

```

5. Accessing table data:

```
mysql> select * from users;
```

user_id	username	password	role
1	admin	admin123	admin
2	librarian	librarian123	librarian
3	user1	user123	member
4	user2	user456	member

```

4 rows in set (0.00 sec)

```

```
mysql> select * from books;
```

book_id	title	author	isbn	publication_year	quantity_available
1	Book1	Author1	978-3-16-148410-0	2020	6
2	Book2	Author2	978-3-16-148411-0	2019	4
3	Book3	Author3	978-3-16-148412-0	2021	7

```

3 rows in set (0.00 sec)

```

```
mysql> select * from bookissues;
```

```
Empty set (0.01 sec)
```

```
mysql> select * from fines;
```

```
Empty set (0.00 sec)
```

6. Adding data to BookIssues table and checking auto trigger for books and fines table:

```
mysql> INSERT INTO BookIssues (user_id, book_id, issue_date, return_date, status)
```

```
-> VALUES
```

```
-> (4, 2, '2024-04-05', NULL, 'issued');
```

```
Query OK, 1 row affected (0.01 sec)
```

```
mysql> INSERT INTO BookIssues (user_id, book_id, issue_date, return_date, status)
```

```
-> VALUES
```

```
-> (3, 3, '2024-04-01', '2024-04-11', 'returned');
```

```
Query OK, 1 row affected (0.01 sec)
```

```
mysql> INSERT INTO BookIssues (user_id, book_id, issue_date, return_date, status)
```

```
-> VALUES
```

```
-> (3, 1, '2024-04-02', '2024-04-20', 'returned');
```

Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM BookIssues;

issue_id	user_id	book_id	issue_date	return_date	status
8	4	2	2024-04-05	NULL	issued
9	3	3	2024-04-01	2024-04-11	returned
10	3	1	2024-04-02	2024-04-20	returned

3 rows in set (0.00 sec)

mysql> SELECT book_id, title, quantity_available FROM books;

book_id	title	quantity_available
1	Book1	6
2	Book2	3
3	Book3	7

3 rows in set (0.00 sec)

mysql> SELECT * FROM fines;

fine_id	user_id	issue_id	fine_amount	fine_reason	fine_date
1	3	10	1.50	Late return	2024-04-27

1 row in set (0.00 sec)

7. Returning a Book:

mysql> DELETE FROM BookIssues

-> WHERE issue_id = 8;

Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM BookIssues;

issue_id	user_id	book_id	issue_date	return_date	status
9	3	3	2024-04-01	2024-04-11	returned
10	3	1	2024-04-02	2024-04-20	returned

2 rows in set (0.00 sec)

mysql> SELECT book_id, title, quantity_available FROM books;

book_id	title	quantity_available
1	Book1	6

```
|      2 | Book2 |      4 |  
|      3 | Book3 |      7 |  
+-----+  
3 rows in set (0.00 sec)
```

Learning outcome

