# DEPARTMENT OF COMPUTER APPLICATION TKM COLLEGE OF ENGINEERING

**KOLLAM – 691005**

**Logo

Description automatically generated**

**20MCA134 ADVANCED DATABASE MANAGEMENT SYSTEMS LAB**

PRACTICAL RECORD BOOK

Second Semester MCA 2021-2022

**Submitted by:**

**NAME: STEFI T**

**ROLL NO: TKM21MCA-2038**

# DEPARTMENT OF COMPUTER APPLICATION TKM COLLEGE OF ENGINEERING

**KOLLAM – 691005**

**Logo

Description automatically generated**

**Certificate**

This is a bonafide record of the work done by STEFI T (TKM21MCA-2038) in the Second Semester in ADVANCED DATABASE MANAGEMENT SYSTEMS LAB, Course(20MCA134) towards the partial fulfillment of the degree of Master of Computer Applications during the academic year 2021-2022.

Staff Member in-charge Examiner

……………………….. ………………………..

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SL.NO.** | **EXPERIMENTS** | **PAGE NO.** |
| 1 | INTRODUCTION TO SQL | 1 |
| 2 | DML(Department database) | 2 |
| 3 | DDL(Movie database) | 4 |
| 4 | Aggregate Functions(Employees db) | 7 |
| 5 | DCL and TCL | 10 |
| 6 | Join | 12 |
| 7 | Aggregate Functions(Emp db) | 16 |
| 8 | PL SQL(Insertion) | 20 |
| 9 | PL SQL(Insentive Calculation) | 21 |
| 10 | PL SQL(Display book information) | 22 |
| 11 | PL SQL(Company fund amount details) | 24 |
| 12 | PL SQL(Update levels of employees) | 26 |
| 13 | Trigger(Billing Details) | 29 |
| 14 | Cursor | 31 |
| 15 | Installation of MongoDB | 33 |
| 16 | MongoDB Design | 39 |
| 17 | CRUD operations | 40 |
| 18 | MongoDB(update& Delete) | 45 |
| 19 | Mongodb (Aggregate Functions) | 50 |
| 20 | Regular Expressions | 52 |
| 21 | Backup and Monitoring | 54 |
| 22 | Users and roles | 56 |
| 23 | Replication | 58 |
| 24 | Indexing | 60 |

**INTRODUCTION TO SQL**

Pronounced as SEQUEL: Structured English QUERY Language

* Pure non-procedural query language
* Designed and developed by IBM, Implemented by Oracle
* 1978 System/R IBM- 1st Relational DBMS
* 1979 Oracle and Ingres
* 1982 SQL/DS and DB2 IBM
* Accepted by both ANSI + ISO as Standard Query Language for any RDBMS
* SQL86 (SQL1) : first by ANSI and ratified by ISO (SQL-87), minor revision on 89 (SQL- 89)
* SQL92 (SQL2) : major revision
* SQL99 (SQL3) : add recursive query, trigger, some OO features, and non-scholar type
* SQL2003 : XML, Window functions, and sequences (Not free) Supports all the three sublanguages of DBMS: DDL, DML, DCL
* Supports Aggregate functions, String Manipulation functions, Set theory operations, Date Manipulation functions, rich set of operators ( IN, BETWEEN, LIKE, IS NULL,EXISTS)
* Supports REPORT writing features and Forms for designing GUI based applications

# Experiment 1

### AIM

### 

### CODE:

CREATE DATABASE Departments;

USE Departments;

CREATE TABLE Dept(

DEPTNO int NOT NULL,

DNAME varchar(30) NOT NULL,

LOC varchar(30) NOT NULL

);

SELECT \* FROM dept;

SELECT \* FROM department;

**Query 1:**

RENAME TABLE dept TO department;

**Query 2:**

ALTER TABLE department ADD COLUMN PINCODE int(6) NOT NULL;

**Query 3:**

ALTER TABLE department DROP COLUMN PINCODE;

**Query 4:**

ALTER TABLE department CHANGE DNAME DEPT\_NAME varchar(30);

**Query 5:**

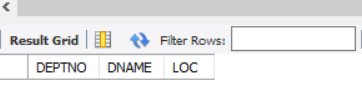
ALTER TABLE department MODIFY COLUMN LOC char(10);

**Query 6:**

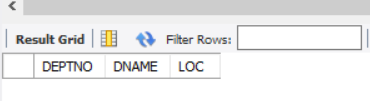
DROP TABLE department;

### OUTPUT

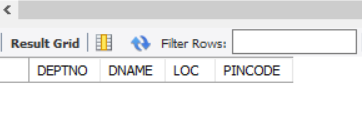
**Query 1:**



**Query 2:**



**Query 3:**



**Query 4:**



**Query 5:**



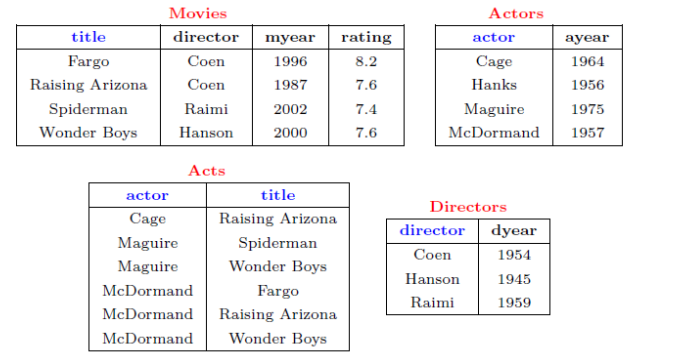
### RESULT

Query processed successfully and output obtained.

# Experiment 2

### AIM

Consider the MOVIE DATABASE



Write following relational algebra queries for a given set of relations.

1. Find movies made after 1997
2. Find movies made by Hanson after 1997
3. Find all movies and their ratings
4. Find all actors and directors
5. Find Coen’s movies with McDormand

### CODE:

CREATE DATABASE MOVIE;

USE MOVIE;

CREATE TABLE directors ( director varchar(20) NOT NULL, dyear int NOT NULL,

PRIMARY KEY(director) );

CREATE TABLE movies ( title varchar(30) NOT NULL, director varchar(20) NOT NULL,

myear int NOT NULL, rating float NOT NULL, PRIMARY KEY(title),

FOREIGN KEY (director) REFERENCES directors (director) );

CREATE TABLE actors ( actor varchar(20) NOT NULL, ayear int NOT NULL,

PRIMARY KEY(actor) );

CREATE TABLE acts ( actor varchar(20) NOT NULL, title varchar(30) NOT NULL,

FOREIGN KEY (actor) REFERENCES actors (actor),

FOREIGN KEY (title) REFERENCES movies (title) );

INSERT INTO directors(director,dyear) VALUES("Coen",1954);

INSERT INTO directors(director,dyear) VALUES("Hanson",1945);

INSERT INTO directors(director,dyear) VALUES("Raimi",1959);

INSERT INTO movies(title,director,myear,rating) VALUES("Fargo","Coen",1996,8.2);

INSERT INTO movies(title,director,myear,rating) VALUES("Raising Arizona","Coen",1987,7.6);

INSERT INTO movies(title,director,myear,rating) VALUES("Spiderman","Raimi",2002,7.4);

INSERT INTO movies(title,director,myear,rating) VALUES("Wonder Boys","Hanson",2000,7.6);

INSERT INTO actors(actor,ayear) VALUES("Cage",1964);

INSERT INTO actors(actor,ayear) VALUES("Hanks",1956);

INSERT INTO actors(actor,ayear) VALUES("Maguire",1975);

INSERT INTO actors(actor,ayear) VALUES("McDormand",1957);

INSERT INTO acts(actor,title) VALUES("Cage","Raising Arizona");

INSERT INTO acts(actor,title) VALUES("Maguire","Spiderman");

INSERT INTO acts(actor,title) VALUES("Maguire","Wonder Boys");

INSERT INTO acts(actor,title) VALUES("McDormand","Fargo");

INSERT INTO acts(actor,title) VALUES("McDormand","Raising Arizona");

INSERT INTO acts(actor,title) VALUES("McDormand","Wonder Boys");

SELECT \* FROM movies;

SELECT \* FROM directors;

SELECT \* FROM actors;

SELECT \* FROM acts;

**Query 1:**

SELECT title FROM movies WHERE myear>1997;

**Query 2:**

SELECT title FROM movies WHERE director="Hanson" AND myear>1997;

**Query 3:**

SELECT title,rating FROM movies;

**Query 4:**

CREATE VIEW actdir AS SELECT actors.actor,directors.director FROM actors,directors;

SELECT \* FROM actdir;

**Query 5:**

ALTER TABLE department MODIFY COLUMN LOC char(10);

**Query 6:**

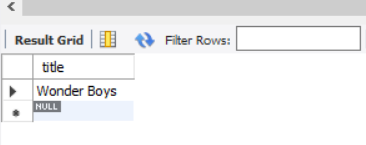
SELECT movies.title FROM movies,acts WHERE director="Coen" AND actor="McDormand" AND movies.title=acts.title;

### OUTPUT

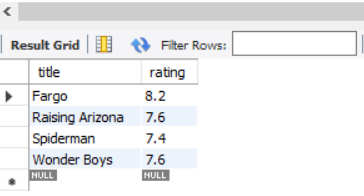
**Query 1:**



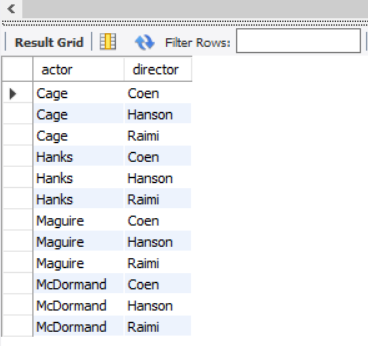
**Query 2:**



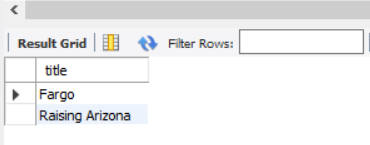
**Query 3:**



**Query 4:**



**Query 5:**



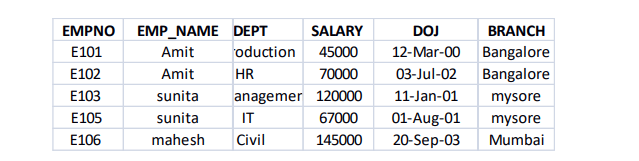
### RESULT

Query processed successfully and output obtained.

# Experiment 3

### AIM

Consider Employee table



Perform the following

1. Display all the fields of employee table

2. Retrieve employee number and their salary

3. Retrieve average salary of all employee

4. Retrieve number of employee

5. Retrieve distinct number of employee

6. Retrieve total salary of employee group by employee name and count similar names

7. Retrieve total salary of employee which is greater than >120000

8. Display name of employee in descending order

9. Display details of employee whose name is AMIT and salary greater than 50000

### CODE:

CREATE DATABASE EMPLOYEES;

USE EMPLOYEES;

CREATE TABLE EMPLOYEE( EMPNO char(4) not null, EMPNAME varchar(30) not null,

DEPT varchar(30) not null, SALARY int(8) not null, DOJ date not null,

BRANCH varchar(20) not null, PRIMARY KEY(EMPNO) );

INSERT INTO EMPLOYEE VALUES("E101","Amit","Production",45000,"2000-03-12","Banglore");

INSERT INTO EMPLOYEE VALUES("E102","Amit","HR",70000,"2002-07-03","Banglore");

INSERT INTO EMPLOYEE VALUES("E103","Sunitha","Management",120000,"2001-01-11","Mysore");

INSERT INTO EMPLOYEE VALUES("E105","Sunitha","IT",67000,"2001-08-01","Mysore");

INSERT INTO EMPLOYEE VALUES("E106","Mahesh","Civil",145000,"2003-09-20","Mumbai");

**Query 1:**

SELECT \* FROM EMPLOYEE;

**Query 2:**

SELECT EMPNO,SALARY FROM EMPLOYEE;

**Query 3:**

SELECT AVG(SALARY) FROM EMPLOYEE;

**Query 4:**

SELECT COUNT(\*) FROM EMPLOYEE;

**Query 5:**

SELECT DISTINCT EMPNO FROM EMPLOYEE;

**Query 6:**

SELECT SUM(SALARY),EMPNAME,COUNT(EMPNAME) AS OCCURENCE FROM EMPLOYEE GROUP BY EMPNAME;

**Query 7:**

SELECT SUM(SALARY) FROM EMPLOYEE WHERE SALARY>120000;

**Query 8:**

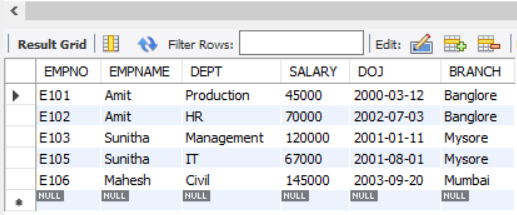
SELECT EMPNAME FROM EMPLOYEE ORDER BY EMPNAME DESC;

**Query 9:**

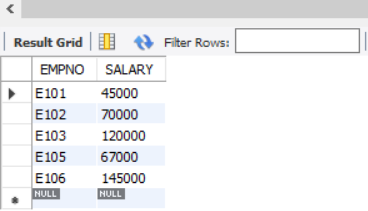
SELECT \* FROM EMPLOYEE WHERE EMPNAME="Amit" AND SALARY>50000;

### OUTPUT

**Query 1:**



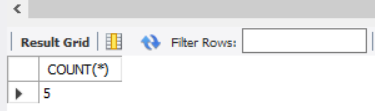
**Query 2:**



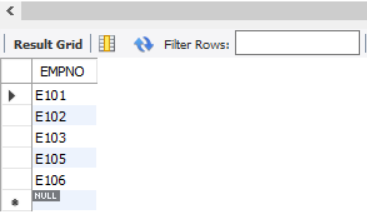
**Query 3:**



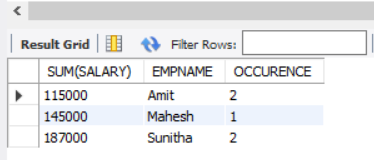
**Query 4:**



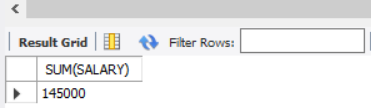
**Query 5:**



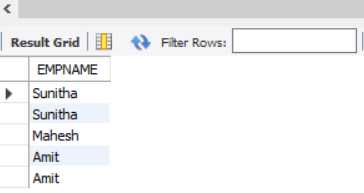
**Query 6:**



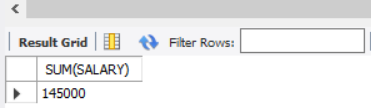
**Query 7:**



**Query 8:**



**Query 9:**



### RESULT

Query processed successfully and output obtained.

**Experiment 4**

**AIM**

Apply DCL and TCL commands to impose restrictions on database.

### CODE:

### DCL:

CREATE TABLE employee(empno VARCHAR(20) NOT NULL PRIMARY KEY,

emp\_name VARCHAR(20) NOT NULL,dept VARCHAR(20) NOT NULL,

salary INT NOT NULL,dob DATE NOT NULL,branch VARCHAR(20) NOT NULL);

DESCRIBE employee;

INSERT INTO employee VALUES ('E101','Amit','Production',45000,'2000-03-12','Bangalore');

INSERT INTO employee VALUES ('E102','Amit','HR',70000,'2002-07-03','Bangalore');

INSERT INTO employee VALUES ('E103','sunita','Manager',120000,'2001-01-11','Mysore');

INSERT INTO employee VALUES ('E104','sunita','IT',67000,'2001-08-01','Mysore');

INSERT INTO employee VALUES ('E105','mahesh','Civil',145000,'2003-09-20','Mumbai');

SELECT \* FROM employee;

delete from employee where empno="E101";

use employee1;

GRANT DELETE ON employee TO 'heylo'@'localhost';

REVOKE DELETE ON employee FROM 'heylo'@'localhost';

REVOKE DELETE ON \*.\* FROM 'heylo'@'localhost';

SHOW GRANTS FOR 'heylo'@'localhost';

### TCL:

CREATE DATABASE tcl;

USE tcl;

CREATE TABLE dept(deptno varchar(20) not null,dname varchar(20) not null,loc varchar(20) not null,primary key(deptno));

INSERT INTO dept VALUES ("d001","finance","kollam");

INSERT INTO dept VALUES ("d002","it","ernakulam");

INSERT INTO dept VALUES ("d003","management","thrissur");

set autocommit=0;

INSERT INTO dept VALUES ("d004","it","kozhikode");

savepoint b;

rollback;

select \*from dept;

INSERT INTO dept VALUES ("d005","finance","kozhikode");

savepoint c;

INSERT INTO dept VALUES ("d006","finance","malappuram");

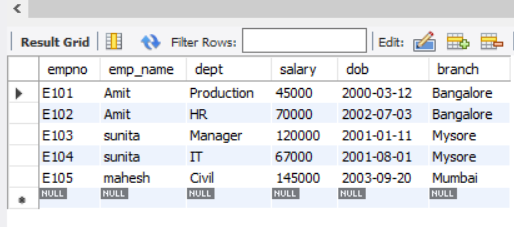
savepoint d;

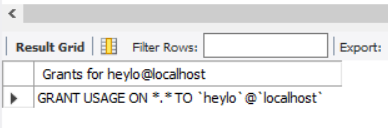
rollback to c;

commit;

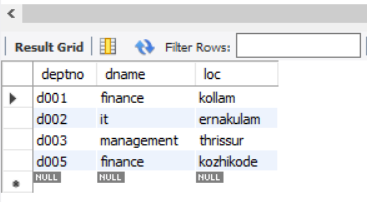
### OUTPUT

**DCL:**





**TCL:**



### RESULT

Query processed successfully and output obtained.

**Experiment 5**

**AIM**

.Consider the schema for MovieDatabase:

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

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

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

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

RATING **(Mov\_id,** Rev\_Stars)

Write SQL queries to

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

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

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

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

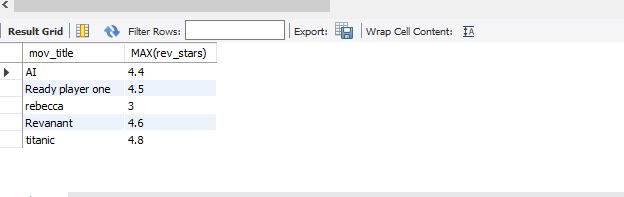
5. Update rating of all movies directed by ‘Steven Spielberg’ to5.

### CODE:

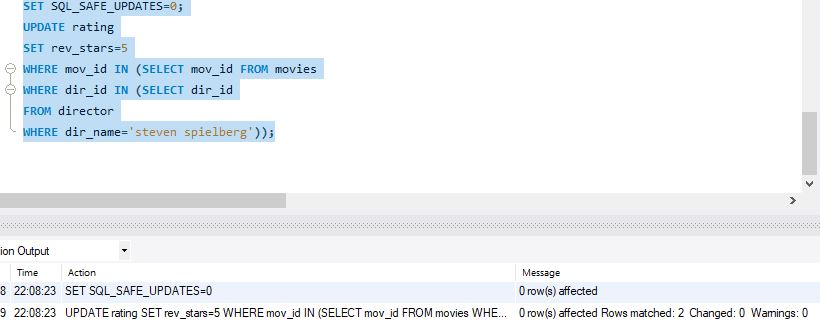
CREATE DATABASE clock;

|  |  |
| --- | --- |
|  |  |
|  | USE clock; |
|  | CREATE TABLE actor(Act\_id int, Act\_Name varchar(50), Act\_Gender varchar(10),PRIMARY KEY(Act\_id)); |
|  | CREATE TABLE director (Dir\_id int, Dir\_Name varchar(50), Dir\_Phone int,PRIMARY KEY(Dir\_id)); |
|  | CREATE TABLE movies (Mov\_id int, Mov\_Title varchar(50), Mov\_Year int, Mov\_Lang varchar(20), Dir\_id int,PRIMARY KEY(Mov\_id),FOREIGN KEY(Dir\_id) REFERENCES director(Dir\_id) ON DELETE CASCADE) ; |
|  | CREATE TABLE movies\_cast (Act\_id int, Mov\_id int,Role varchar(20),FOREIGN KEY(Act\_id) REFERENCES actor(Act\_id) ON DELETE CASCADE,FOREIGN KEY(Mov\_id) REFERENCES movies(Mov\_id) ON DELETE CASCADE); |
|  | CREATE TABLE rating (Mov\_id int, Rev\_Stars varchar(20),FOREIGN KEY(Mov\_id) REFERENCES movies(Mov\_id) ON DELETE CASCADE) ; |
|  |  |
|  | INSERT INTO actor(Act\_id,Act\_Name,Act\_Gender) VALUES (101,'kate','female'); |
|  | INSERT INTO actor(Act\_id,Act\_Name,Act\_Gender) VALUES (102,'leo','male'); |
|  | INSERT INTO actor(Act\_id,Act\_Name,Act\_Gender) VALUES (103,'joan','female'); |
|  | INSERT INTO actor(Act\_id,Act\_Name,Act\_Gender) VALUES (104,'frances','female'); |
|  | INSERT INTO actor(Act\_id,Act\_Name,Act\_Gender) VALUES (105,'tyre','male'); |
|  |  |
|  | select \* from actor; |
|  |  |
|  | INSERT INTO director(Dir\_id,Dir\_Name,Dir\_phone) VALUES (301,'james cameron','1982654329'); |
|  | INSERT INTO director(Dir\_id,Dir\_Name,Dir\_phone) VALUES (302,'Hitchcock','8907654312'); |
|  | INSERT INTO director(Dir\_id,Dir\_Name,Dir\_phone) VALUES (303,'Steven Spielberg','8907654534'); |
|  | INSERT INTO director(Dir\_id,Dir\_Name,Dir\_phone) VALUES (304,'alejan','8456654534'); |
|  |  |
|  | select \* from director; |
|  |  |
|  |  |
|  | INSERT INTO movies(Mov\_id,Mov\_Title,Mov\_Year,Mov\_Lang,Dir\_id) VALUES (201,'titanic',1997,'english',301); |
|  | INSERT INTO movies(Mov\_id,Mov\_Title,Mov\_Year,Mov\_Lang,Dir\_id) VALUES (202,'rebecca',1940,'english',302); |
|  | INSERT INTO movies(Mov\_id,Mov\_Title,Mov\_Year,Mov\_Lang,Dir\_id) VALUES (203,'AI',2001,'english',303); |
|  | INSERT INTO movies(Mov\_id,Mov\_Title,Mov\_Year,Mov\_Lang,Dir\_id) VALUES (204,'Ready player one',2018,'english',303); |
|  | INSERT INTO movies(Mov\_id,Mov\_Title,Mov\_Year,Mov\_Lang,Dir\_id) VALUES (205,'Revanant',2016,'english',304); |
|  | select \* from movies; |
|  |  |
|  |  |
|  | INSERT INTO movies\_cast(Act\_id,Mov\_id,Role) VALUES (101,201,'Rose'); |
|  | INSERT INTO movies\_cast(Act\_id,Mov\_id,Role) VALUES (102,201,'jack'); |
|  | INSERT INTO movies\_cast(Act\_id,Mov\_id,Role) VALUES (102,205,'hugh glass'); |
|  | INSERT INTO movies\_cast(Act\_id,Mov\_id,Role) VALUES (103,202,'mrs.de winter'); |
|  | INSERT INTO movies\_cast(Act\_id,Mov\_id,Role) VALUES (104,203,'monica'); |
|  | INSERT INTO movies\_cast(Act\_id,Mov\_id,Role) VALUES (105,204,'wade watts'); |
|  |  |
|  |  |
|  | select \* from movies\_cast; |
|  |  |
|  | INSERT INTO rating(Mov\_id,Rev\_Stars) VALUES (201,4.8); |
|  | INSERT INTO rating(Mov\_id,Rev\_Stars) VALUES (202,3); |
|  | INSERT INTO rating(Mov\_id,Rev\_Stars) VALUES (203,4.4); |
|  | INSERT INTO rating(Mov\_id,Rev\_Stars) VALUES (204,4.5); |
|  | INSERT INTO rating(Mov\_id,Rev\_Stars) VALUES (205,4.6); |
|  |  |
|  |  |
|  | select \* from rating; |
|  |  |
|  |  |
|  | **Query 1:** |
|  | SELECT movies.Mov\_Title,director.Dir\_name from movies inner join director on movies.dir\_id=director.dir\_id where director.dir\_id=302 ; |
|  |  |
|  |  |
|  | **Query 2:** |
|  | SELECT mov\_title FROM movies WHERE mov\_id IN ( |
|  | SELECT mov\_id FROM movies\_cast WHERE act\_id IN ( |
|  | SELECT act\_id FROM actor WHERE act\_id IN ( |
|  | SELECT act\_id FROM movies\_cast GROUP BY act\_id HAVING COUNT(act\_id)>1))); |
|  | **Query 3:** |
|  | SELECT actor.Act\_Name FROM actor inner join movies\_cast on actor.act\_id=movies\_cast.act\_id WHERE mov\_id in(SELECT Mov\_id FROM movies where Mov\_year<2000 or Mov\_year>2015);  **Query 4:**  SELECT mov\_title,MAX(rev\_stars) FROM movies INNER JOIN rating USING (mov\_id) GROUP BY mov\_title HAVING MAX(rev\_stars)>0 ORDER BY mov\_title;  **Query 5:**  SET SQL\_SAFE\_UPDATES=0; UPDATE rating SET rev\_stars=5 WHERE mov\_id IN (SELECT mov\_id FROM movies WHERE dir\_id IN (SELECT dir\_id FROM director WHERE dir\_name='steven spielberg'));  **OUTPUT:**  **Query 1:**    **Query 2:** |
|  |  |
|  | **Query 3:** |

**Query 4:**



**Query 5:**



### RESULT

Query processed successfully and output obtained.

**Experiment 6**

**AIM**

Queries using aggregate functions(COUNT,AVG,MIN,MAX,SUM),Group by,Order by,Having.

|  |  |  |  |
| --- | --- | --- | --- |
| E\_ID | E\_NAME | AGE | SALARY |
| 101 | ANU | 22 | 9000 |
| 102 | Shane | 29 | 8000 |
| 103 | Rohan | 34 | 6000 |
| 104 | Scott | 44 | 10000 |
| 105 | Tiger | 35 | 8000 |
| 106 | Alex | 27 | 7000 |
| 107 | Abhi | 29 | 8000 |

1. Create Employee table containing all Records.
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. Display the Sum of age employee table.
6. Display the Average of age from Employee table
7. Create a View for age in employee table
8. Display views
9. Find grouped salaries of employees.
10. Find salaries of employee in Ascending Order
11. Find salaries of employee in Descending Order

### CODE:

CREATE DATABASE EMP;

USE EMP;

CREATE TABLE EMPP(E\_ID int not null, E\_NAME varchar(20) not null, AGE int not null,

SALARY int not null, PRIMARY KEY(E\_ID) );

INSERT INTO EMPP(E\_ID,E\_NAME,AGE,SALARY) VALUES(101,"Anu",22,9000);

INSERT INTO EMPP(E\_ID,E\_NAME,AGE,SALARY) VALUES(102,"Shane",29,8000);

INSERT INTO EMPP(E\_ID,E\_NAME,AGE,SALARY) VALUES(103,"Rohan",34,6000);

INSERT INTO EMPP(E\_ID,E\_NAME,AGE,SALARY) VALUES(104,"Scott",44,10000);

INSERT INTO EMPP(E\_ID,E\_NAME,AGE,SALARY) VALUES(105,"Tiger",35,8000);

INSERT INTO EMPP(E\_ID,E\_NAME,AGE,SALARY) VALUES(106,"Alex",27,7000);

INSERT INTO EMPP(E\_ID,E\_NAME,AGE,SALARY) VALUES(107,"Abhi",29,8000);

**Query 1:**

SELECT \* FROM EMPP;

**Query 2:**

SELECT COUNT(\*) FROM EMPP;

**Query 3:**

SELECT MAX(AGE) FROM EMPP;

**Query 4:**

SELECT MIN(AGE) FROM EMPP;

**Query 5:**

SELECT SUM(AGE) FROM EMPP;

**Query 6:**

SELECT AVG(AGE) FROM EMPP;

**Query 7:**

CREATE VIEW AGES AS SELECT AGE FROM EMPP;

**Query 8:**

SELECT COUNT(\*) FROM EMPP;

**Query 9:**

SELECT SALARY FROM EMPP GROUP BY SALARY;

**Query 10:**

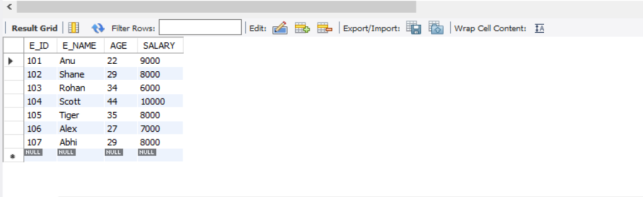
SELECT SALARY FROM EMPP ORDER BY SALARY ASC;

**Query 11:**

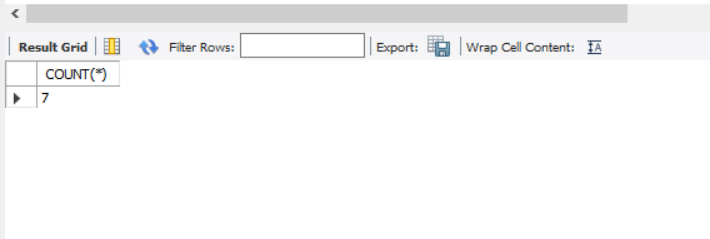
SELECT SALARY FROM EMPP ORDER BY SALARY DESC;

### OUTPUT

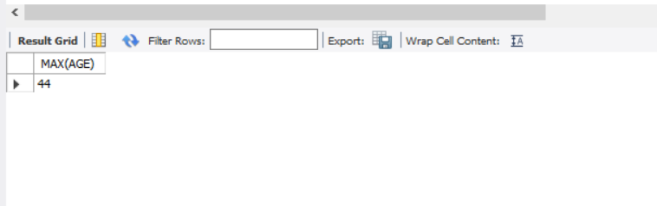
**Query 1:**



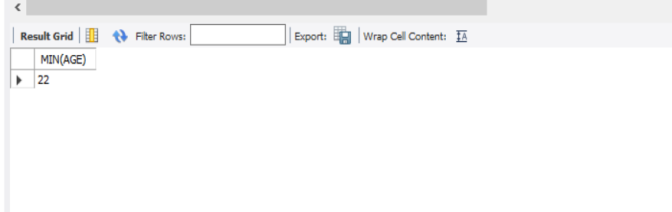
**Query 2:**



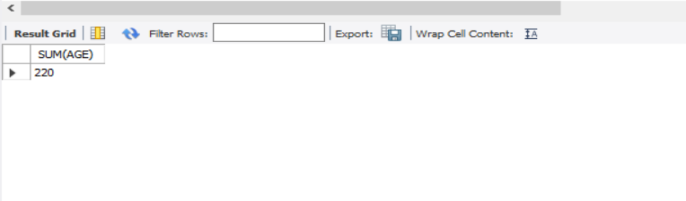
**Query 3:**



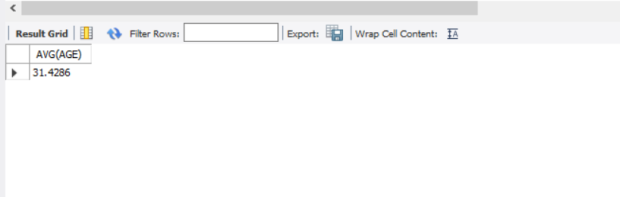
**Query 4:**



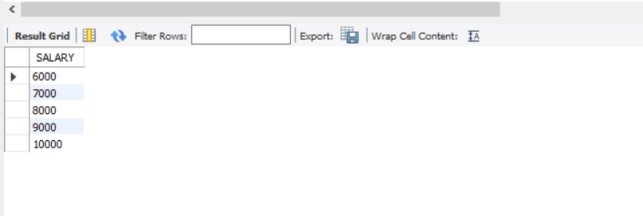
**Query 5:**



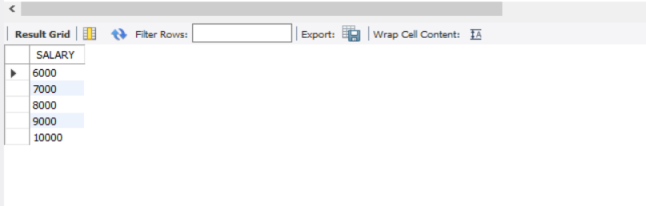
**Query 6:**



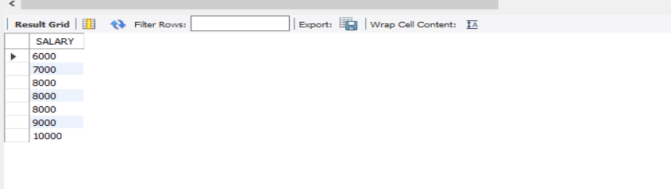
**Query 8:**



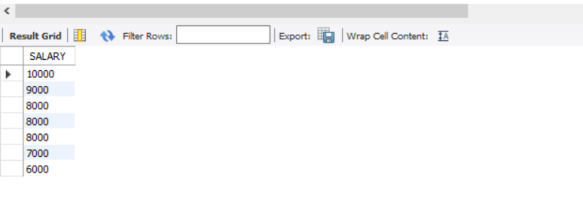
**Query 9:**



**Query 10:**



**Query 11:**



### RESULT

Query processed successfully and output obtained.

**Experiment 7**

**AIM**

Given an integer i, write a PL/SQL procedure to insert the tuple (i, 'xxx') into a given relation

**CODE**

CREATE DATABSE studentdb;

USE studentdb;

CREATE TABLE T2(rollno int,name varchar(10),primary key(rollno));

call stud('101','Stefi');

call stud('102','Sreelaya');

select \* from T2;

///STORED PROCEDURE///

CREATE DEFINER=`root`@`localhost` PROCEDURE `stud`(rollno int,name varchar(10))

BEGIN

insert into T2 values(rollno,name);

END

### OUTPUT



### RESULT

Query processed successfully and output obtained.

**Experiment 8**

**AIM**

To write a PL/SQL block to calculate the incentive of an employee whose ID is 110

**CODE**

CREATE DATABASE employeedb2;

USE employeedb2;

CREATE TABLE E1(empid int,empname varchar(10),salary int,primary key(empid));

INSERT INTO E1(empid,empname,salary)VALUES('110','Stefi',2000);

INSERT INTO E1(empid,empname,salary)VALUES('111','Sreelaya',50000);

SELECT \* from E1;

SELECT empid,empname,insentive(empid) from E1;

///FUNCTION///

CREATE DEFINER=`root`@`localhost` FUNCTION `insentive`(empid int) RETURNS varchar(20) CHARSET latin1

BEGIN

DECLARE i VARCHAR(20);

IF (empid=110)

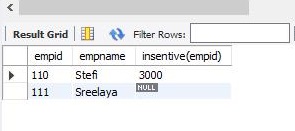
THEN SET i=3000;

END IF;

RETURN i ;

END

### OUTPUT



### RESULT

Query processed successfully and output obtained.

**Experiment 9**

**AIM**

To create the Book database and do the following: (Consider the attributes based on the question given)

book(book\_name, author\_name, price,quantity)

1. Write a query to update the quantity by double in the table book.
2. List all the book\_name whose price is greater than those of book named "Database for Dummies"
3. Retrieve the list of author\_name whose first letter is ’a’ along with the book\_name and price (Explore more about *Like* keyword)
4. Write a PL/SQL Procedure to find the total number of books of same author

**CODE:**

CREATE DATABASE books;

USE books;

CREATE TABLE book\_info(book\_name varchar (20),author varchar(20),price int,quantity int);

INSERT into book\_info VALUES('randamoozham','MT',300,5);

INSERT into book\_info VALUES ('ikigai','hector',500,7);

INSERT into book\_info VALUES ('databse of dummies','xyz',250,7);

INSERT into book\_info VALUES ('wings of flare','APJ',500,7);

INSERT into book\_info VALUES ('oopol','MT',270,3);

SELECT \* from book\_info;

**a)** set sql\_safe\_updates=0;

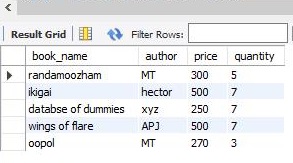
UPDATE book\_info set quantity=quantity\*2;

**b)** SELECT book\_name from book\_info where price>(select price from book\_info where book\_name='databse of dummies');

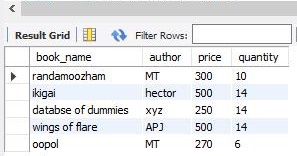
**c)** SELECT author,book\_name,price from book\_info where author like 'a%';

**OUTPUT**

**Query a:**



**Query b:**



**Query c:**



### RESULT

Query processed successfully and output obtained.

**Experiment 10**

**AIM**

Create the Company database with the following tables and do the following:

Administration (employee\_salary, development \_cost, fund\_ amount, turn\_over,bonus)

Emp\_details (emp\_no, emp\_name, DOB, address, doj, mobile\_no, dept\_no, salary).

1. Calculate the total and average salary amount of the employees of each department.
2. Display total salary spent for employees.
3. Develop a PL/SQL function to display total fund\_amount spent by the administration department

**CODE:**

CREATE DATABASE company;

USE company;

CREATE TABLE Admins(

emp\_sal double,

dvlp\_cost double,

fund\_amount double,

turn\_over double,

bonus double);

CREATE TABLE Emp\_details(

emp\_no int,

emp\_name varchar(20),

DOB date,

address varchar(20),

doj date,

mobile\_no int8,

dept\_no int,

salary double);

INSERT INTO Admins VALUES

(12000,25000,560000,65000,5000),

(70000,55000,860000,15000,1000),

(18000,45000,160000,75000,7000),

(10000,27000,520000,60000,5000),

(18000,27000,360000,35000,3000);

INSERT INTO Emp\_details VALUES

(1,"hamna","1999-10-10","Street - 2 xyz","2020-10-10",9865986598,10,12000),

(2,"ansi","1997-10-10","Street - 2 abc","2020-10-10",9865986598,10,12200),

(3,"sree","1996-10-10","Street ","2020-10-10",9865986598,11,12500),

(4,"stef","1957-10-10","Street in","2020-10-10",9865986598,11,17200),

(5,"anu","1948-10-10","gared","2020-10-10",9865986598,12,12090),

(6,"shiva","1988-10-10","Sas","2020-10-10-",9865986598,12,12050);

1. SELECT dept\_no,avg(salary) 'Average salary',sum(salary) 'Total Salary' FROM Emp\_details GROUP BY dept\_no;

**b)** SELECT sum(salary) 'SUM OF SALARY'FROM Emp\_details;

**c)** //FUCTION//

CREATE DEFINER=`root`@`localhost` FUNCTION `fund\_total`() RETURNS double

BEGIN

DECLARE f DOUBLE;

DECLARE i DOUBLE;

SELECT SUM(fund\_amount)

FROM Admins;

RETURN f;

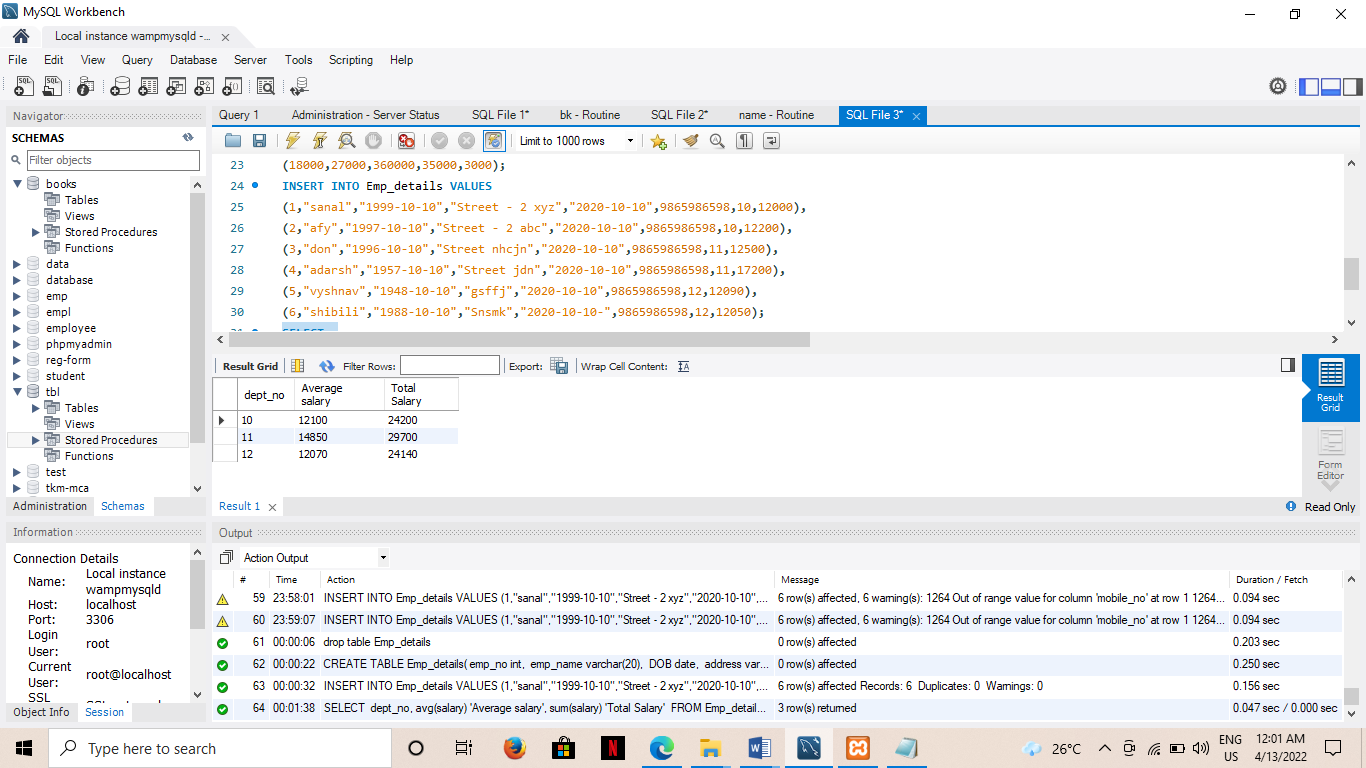
END

//FUNCTION CALL//

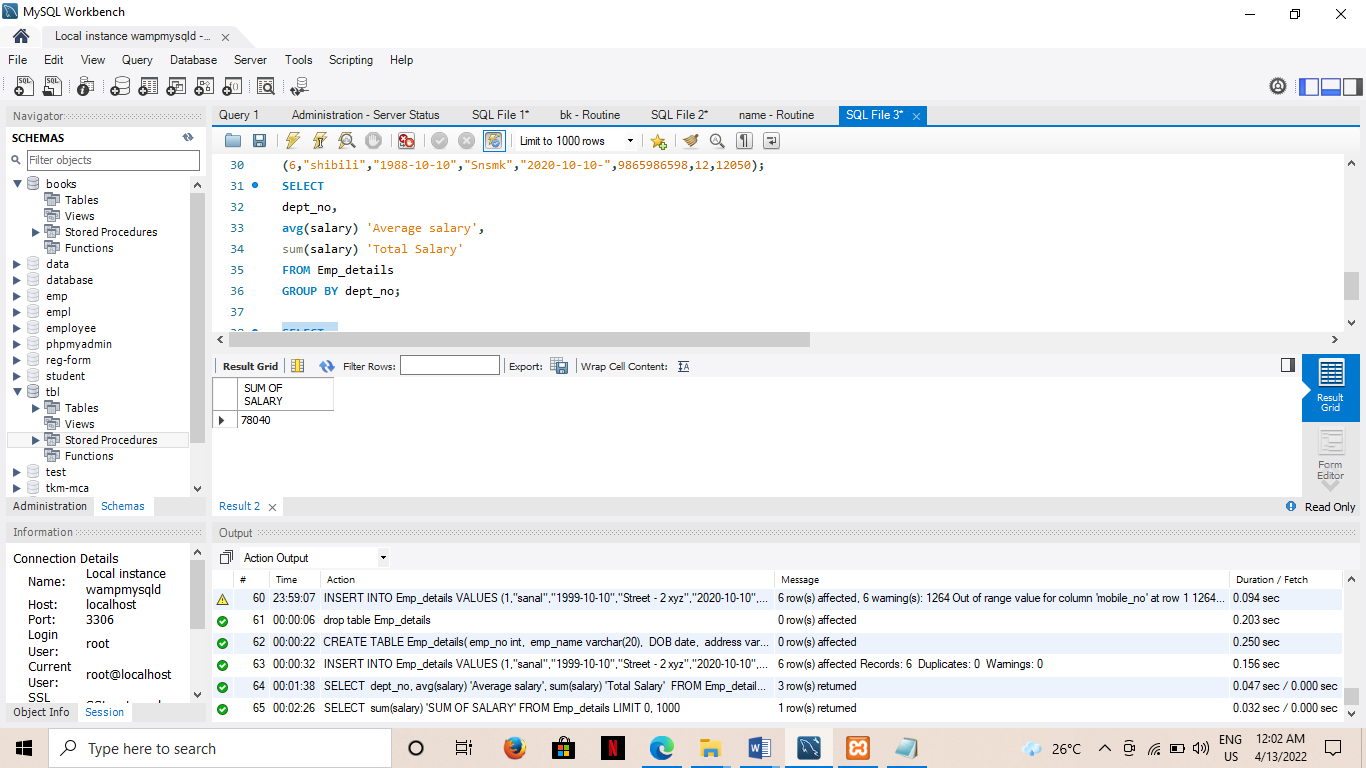
SELECT fund\_total() from Admins LIMIT 1;

**OUTPUT**

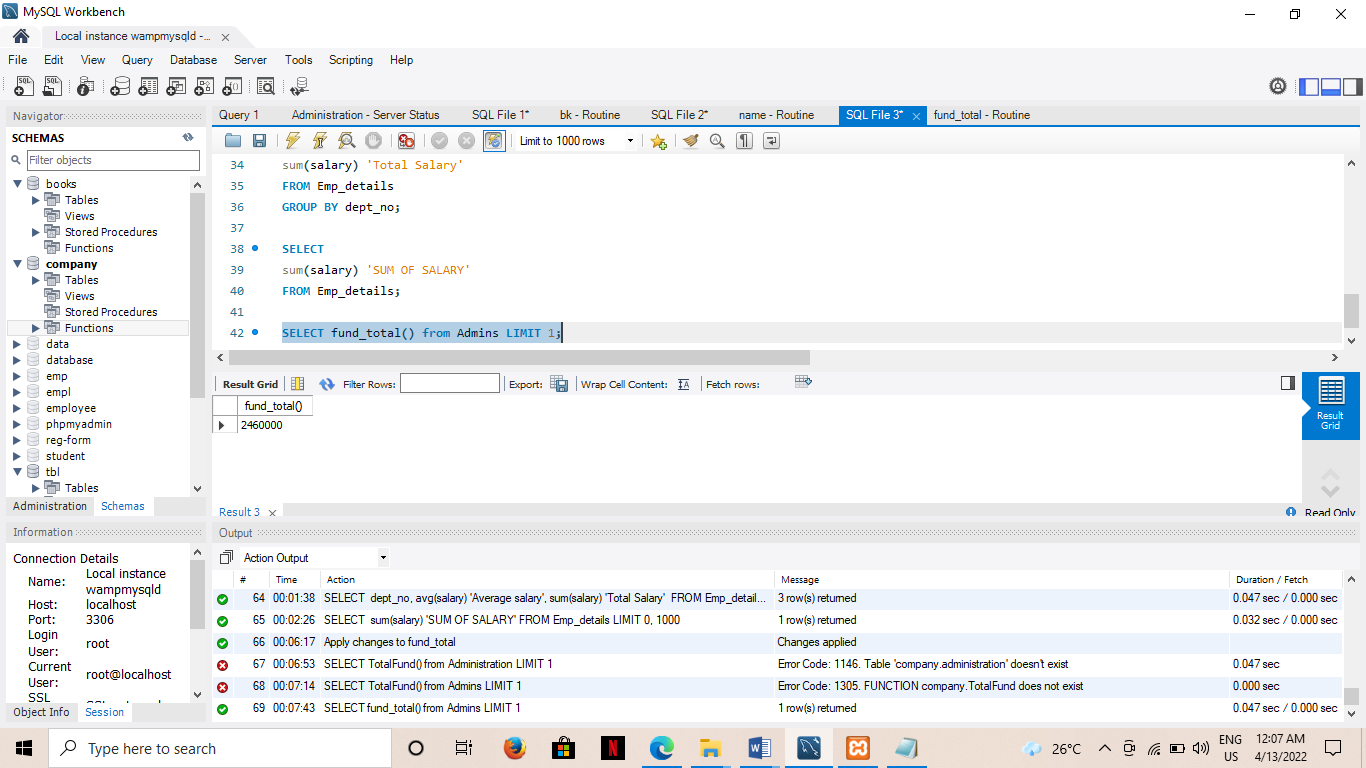
**Query a:**



**Query b:**



**Query c:**



### RESULT

Query processed successfully and output obtained.

# Experiment 11

### AIM

To create a database containing table employee with employee details.Write PLSQL to update the experience level of employee as beginner,intermediate and advanced.

**CODE:**

CREATE DATABASE company;

USE company;

CREATE TABLE emp(emp\_id int primary key,emp\_name varchar(20),salary varchar(20));

CREATE TABLE dept(dept\_id int primary key,emp\_id int,designation varchar(20),experience int(10) ,foreign key(emp\_id) references emp(emp\_id));

INSERT INTO emp(emp\_id,emp\_name,salary)values(101,'Shibu',25000);

INSERT INTO emp(emp\_id,emp\_name,salary)values(102,'Raju',35000);

INSERT INTO emp(emp\_id,emp\_name,salary)values(103,'Shanku',50000);

SELECT \* from emp;

INSERT INTO dept(dept\_id,emp\_id,designation,experience)values(201,101,'Peon',2);

INSERT INTO dept(dept\_id,emp\_id,designation,experience)values(202,102,'Clerk',6);

INSERT INTO dept(dept\_id,emp\_id,designation,experience)values(203,103,'Manager',12);

SELECT \* from dept;

CREATE TABLE level(emp\_id int,dept\_id int,experience\_level varchar(20),foreign key(emp\_id) references emp(emp\_id),foreign key(dept\_id) references dept(dept\_id));

call exp(2,101,201);

call exp(6,102,201);

call exp(12,103,203);

SELECT\* from level;

SELECT emp.emp\_name,emp.salary,new\_salary(level.experience\_level,emp.salary) from emp,level where emp.emp\_id=level.emp\_id;

////STORED PROCEDURE

CREATE DEFINER=`root`@`localhost` PROCEDURE `exp`(experience int,emp\_id int,dept\_id int)

BEGIN

DECLARE

levels varchar(45);

if (experience > 0 && experience<5)

then set levels = 'beginner';

insert into employe(emp\_id,experience,salary,levels) values(emp\_id,experience,salary,levels);

end if;

if( exp>=6 && exp <10)

then set levels = 'intermediate';

insert into employe(emp\_id,experience,salary,levels) values(emp\_id,experience,salary,levels);

end if;

if (exp >= 10)

then set levels = 'Experienced';

insert into employe(emp\_id,experience,salary,levels) values(emp\_id,experience,salary,levels);

end if;

END

////FUNCTION////

CREATE DEFINER=`root`@`localhost` FUNCTION `new\_salary`(experience\_level varchar(20),sal varchar(10)) RETURNS int(11)

BEGIN

if(experience\_level = 'Experienced')

then

return(sal+1000);

else

return(sal);

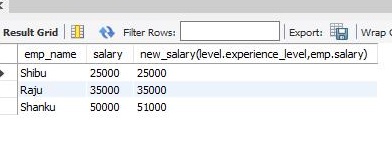
end if;

RETURN 1;

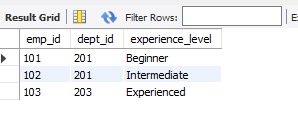
END

**OUTPUT**

**PROCEDURE**



**FUNCTION**



### RESULT

Query processed successfully and output obtained.

# Experiment 12

### AIM

Create a table bookings with attributes bk\_id,name,source,dist on insert and update ,update

billing details in bill table(id,bill)

**CODE:**

CREATE DATABSE train;

USE train;

CREATE TABLE booking(bk\_id int(50),name varchar(50),source varchar(50),dist int(50));

CREATE TABLE bill(id int(50),bill int(50));

INSERT INTO booking values(10,&quot;Farhana&quot;,&quot;kyj&quot;,500);

SELECT \* from bill;

SELECT \* from booking;

INSERT INTO booking values(11,&quot;Arun&quot;,&quot;kollam&quot;,100);

UPDATE booking set dist=150 where bk\_id=10;

set SQL\_SAFE\_UPDATES=0;

DELETE from bill;

DELETE from booking;

**TRIGGER INSERT:**

CREATE DEFINER=`root`@`localhost` TRIGGER `train`.`booking\_AFTER\_INSERT`

AFTER INSERT ON `booking` FOR EACH ROW

BEGIN

if(new.dist&gt;100 and new.dist&lt;=200)

then

insert into bill set id=new.bk\_id,bill=800;

end if;

if(new.dist&lt;=100)

then

insert into bill set id=new.bk\_id,bill=500;

end if;

if(new.dist&gt;200)

then

insert into bill set id=new.bk\_id,bill=0;

end if;

END

**TRIGGER UPDATE:**

CREATE DEFINER=`root`@`localhost` TRIGGER `train`.`booking\_AFTER\_UPDATE`

AFTER UPDATE ON `booking` FOR EACH ROW

BEGIN

if(new.dist&gt;100 and new.dist&lt;=200)

then

update bill set bill=800 where id=new.bk\_id ;

end if;

if(new.dist&lt;=100)

then

update bill set bill=500 where id=new.bk\_id;

end if;

if(new.dist&gt;200)

then

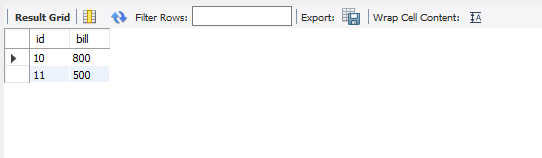
update bill set bill=0 where id=new.bk\_id;

end if;

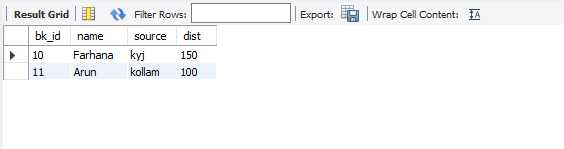
END

**OUTPUT:**

**TRIGGER INSERT**



**TRIGGER UPDATE**



### RESULT

Query processed successfully and output obtained.

# Experiment 13

### AIM

### To write a program to implement cursor.

### CODE

CREATE DATABASE college;

USE college;

CREATE TABLE library(shelf\_no int,category varchar(10),book\_name varchar(20));

INSERT INTO library values(101,'Topology','Real Analysis');

INSERT INTO library values(102,'Algebra','Linear Algebra');

INSERT INTO library values(103,'Analysis','Complex Analysis');

INSERT INTO library values(104,'OR','Operations Research');

INSERT INTO library values(106,'NumberSys','AbstractAlg');

CREATE TABLE book\_by\_order(book\_shelf int,book\_category varchar(20),bookname varchar(20));

SELECT\* from library;

call book\_details();

/\*

CREATE DEFINER=`root`@`localhost` PROCEDURE `book\_details`()

BEGIN

declare book\_shelf int;

declare bookname varchar(20);

declare book\_category varchar(10);

declare C\_finished integer default 0;

declare C1 cursor for select shelf\_no,category,book\_name from library;

declare continue handler for not found set C\_finished = 1;

open C1;

book\_details:loop

if C\_finished=1 then

leave book\_details;

end if;

if C\_finished = 0 then

Fetch from C1 into book\_shelf,book\_category,bookname;

if book\_category = 'OR' then

insert into book\_by\_order values(book\_shelf,bookname,book\_category);

end if;

end if;

end loop;

close C1;

END

\*/

**OUTPUT**

### Description: Image

### RESULT

Query processed successfully and output obtained.

# Experiment 14

### AIM

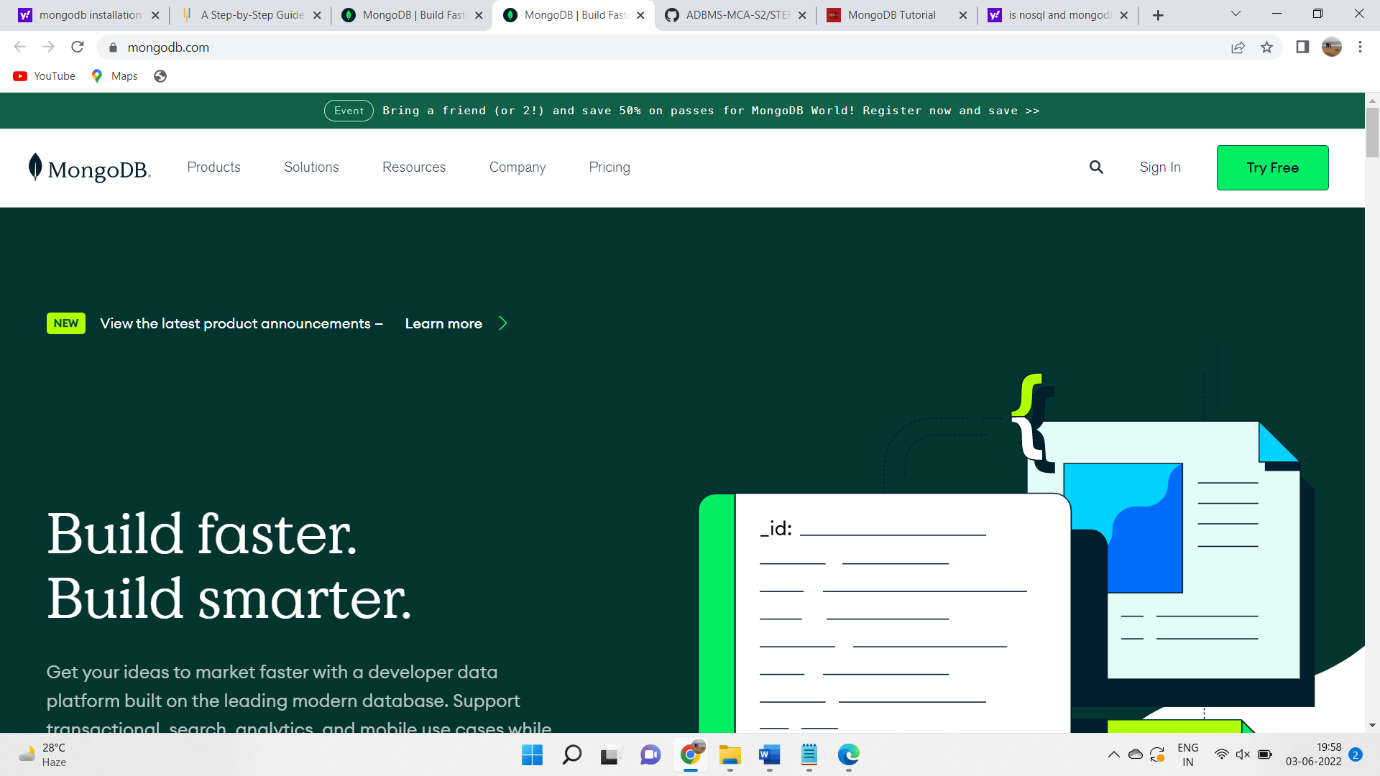
To understand the installation and configuration of NOSQL databases.

**CODE**

MongoDB is a cross-platform, document oriented NoSql database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

**STEP 1:**

Navigate to the official MongoDB website.



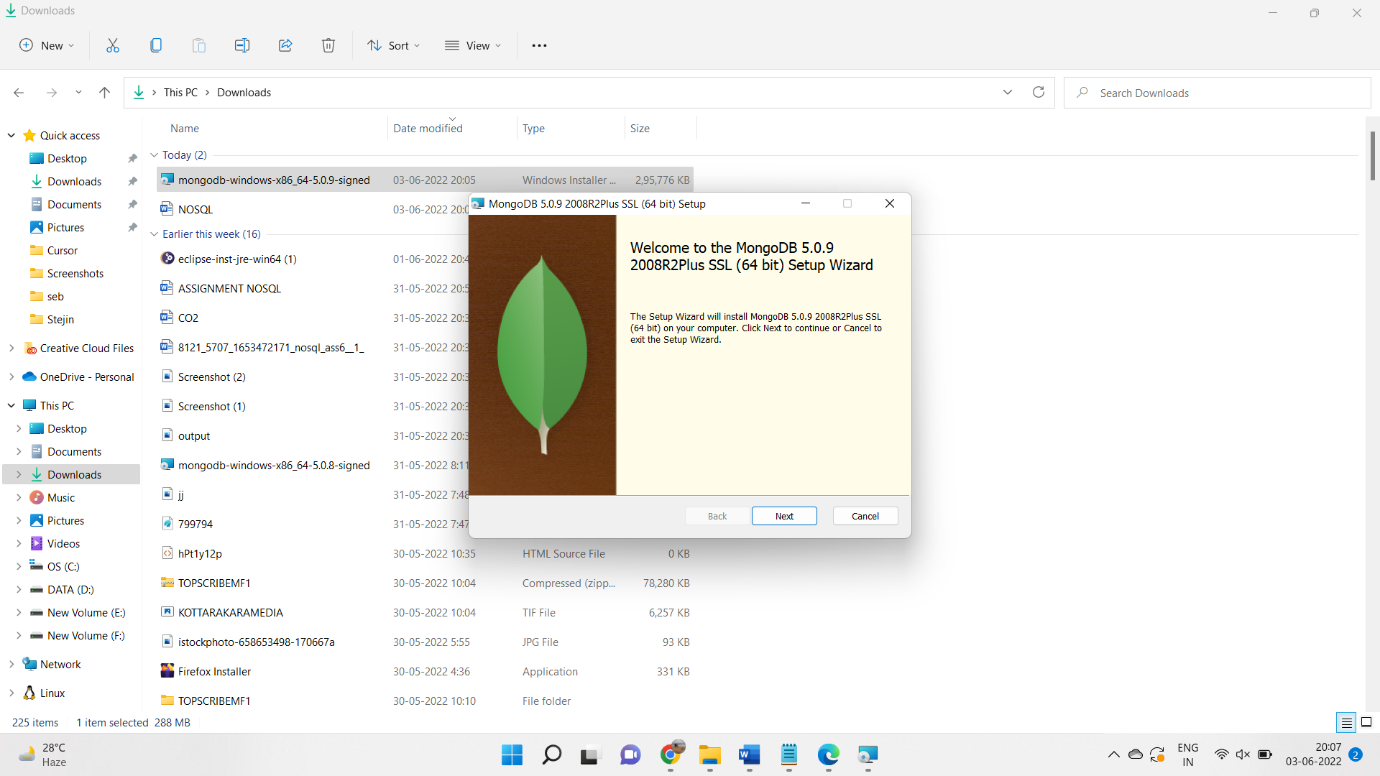
**STEP 2:**

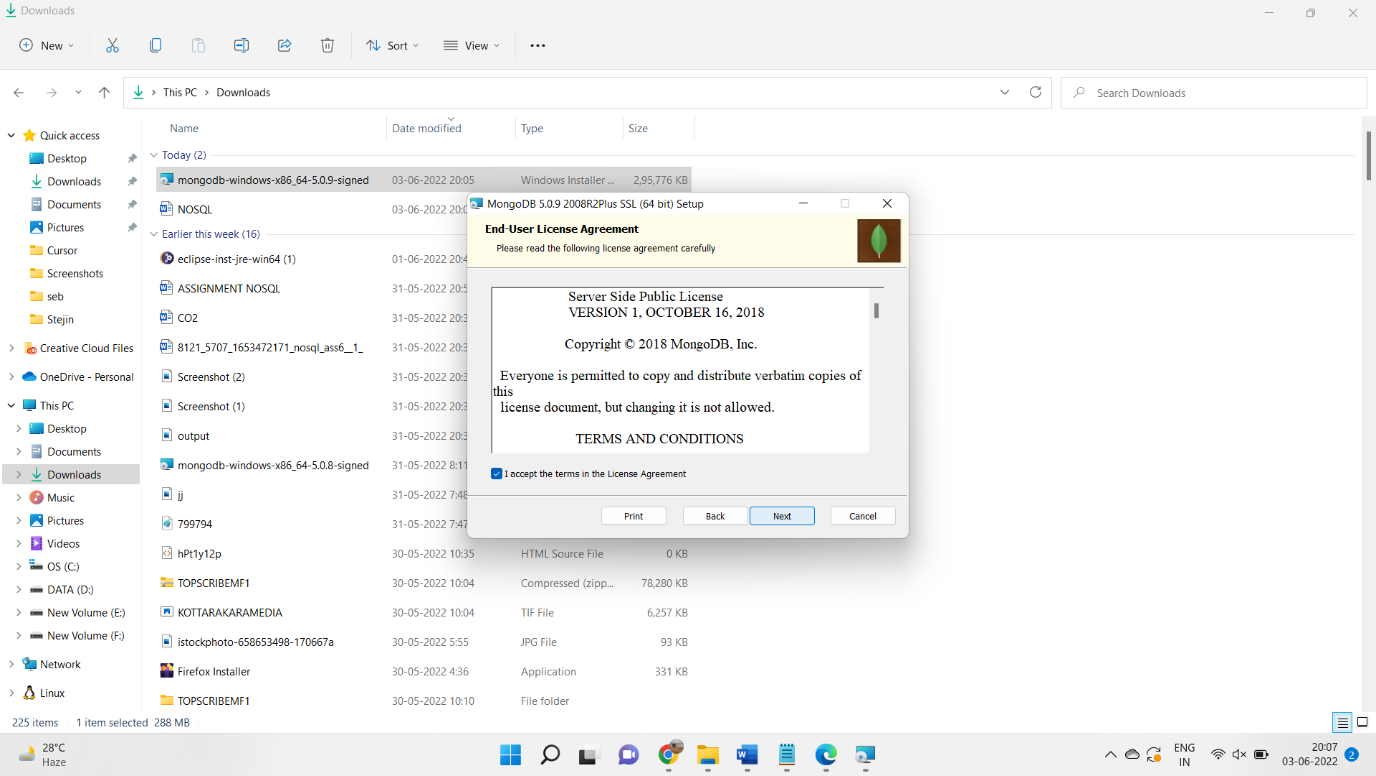
Under the products section, click on the Community server version. Make sure that the specifications to the right of the screen are correct. At the time of writing, the latest version is 4.4.5. Ensure that the platform is Windows, and the package is MSI. Go ahead and click on download.

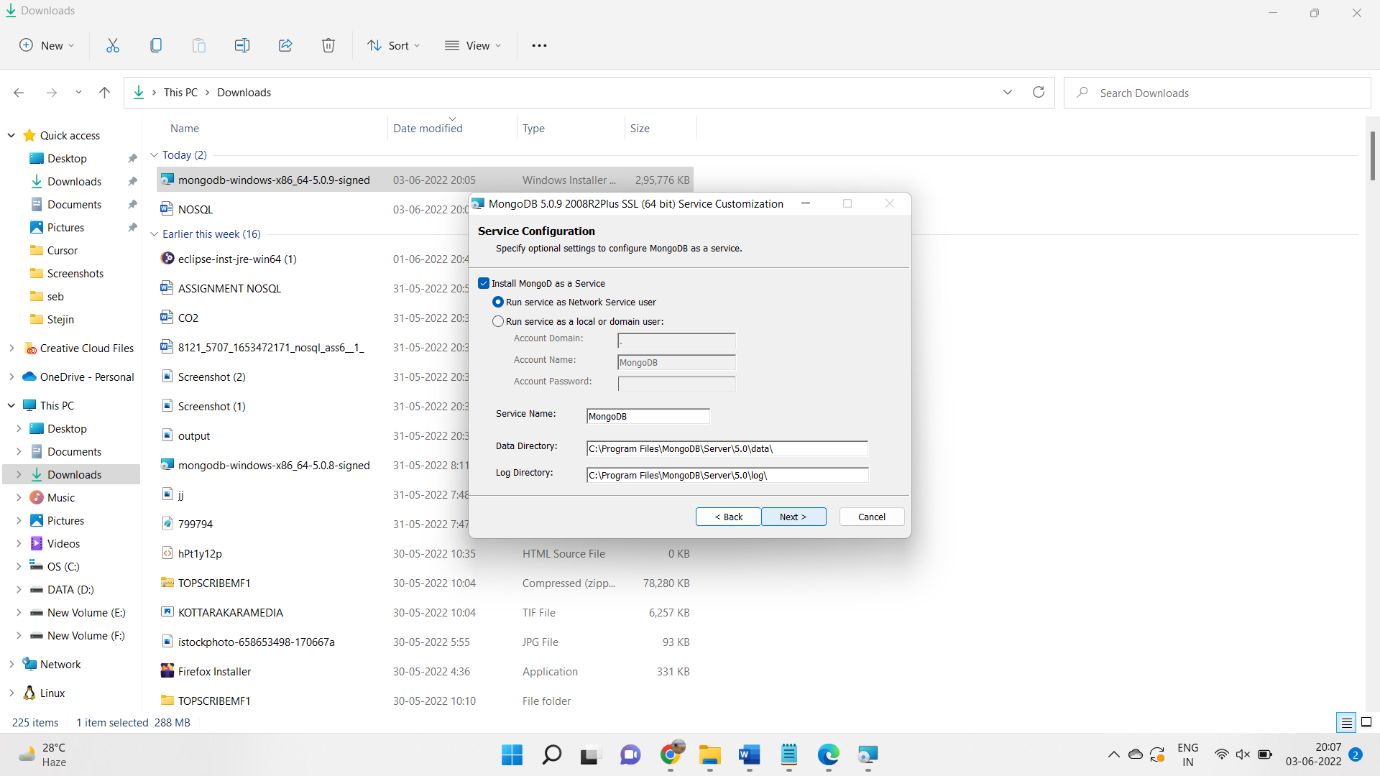


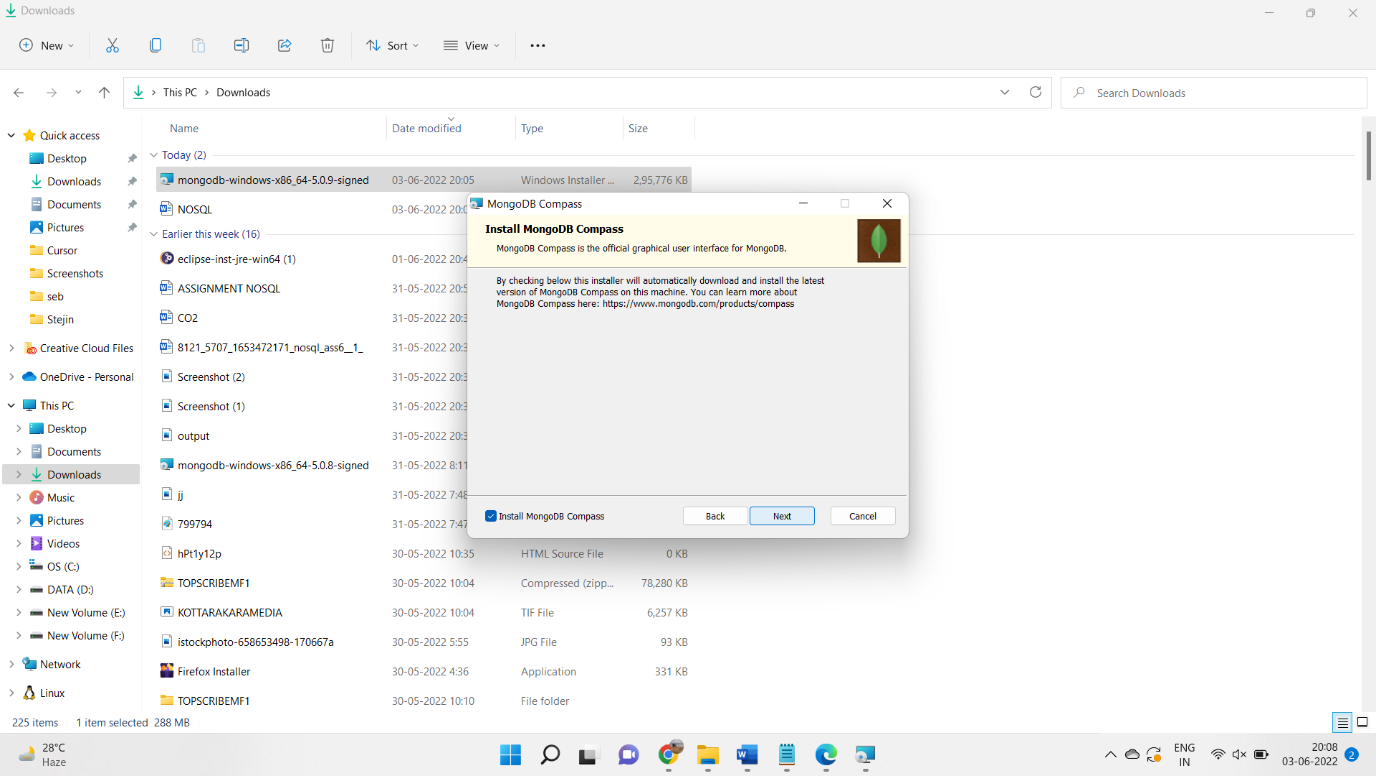
**STEP 3**:

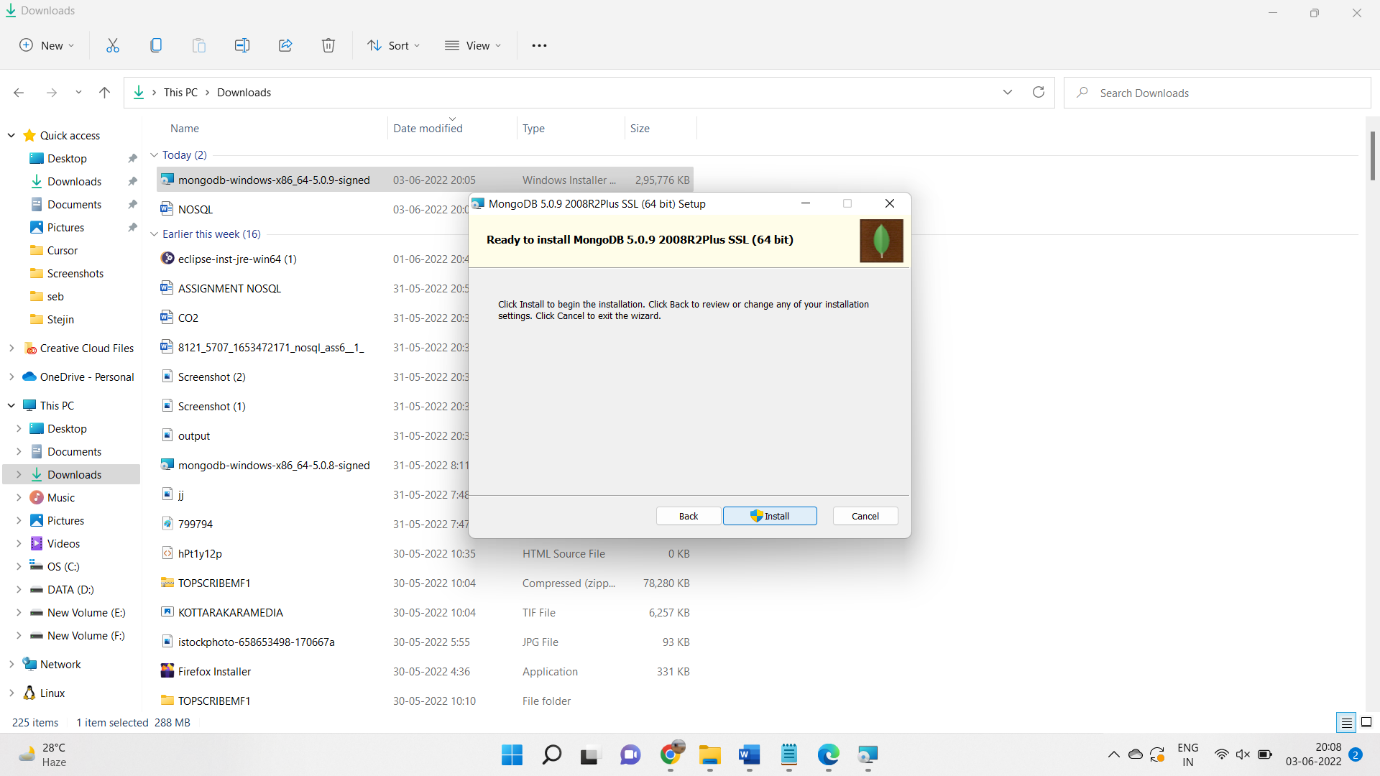
You can find the downloaded file in the downloads directory. Install the software step by step.

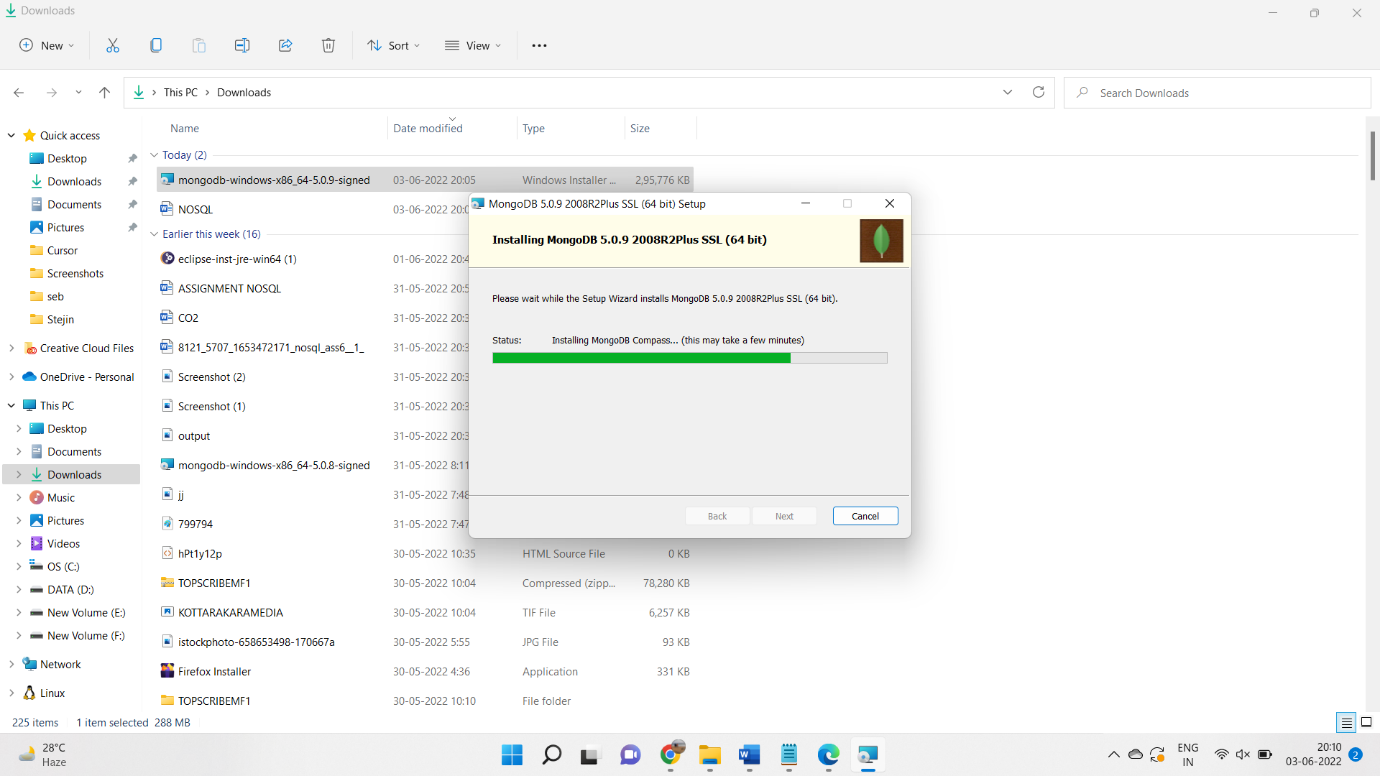






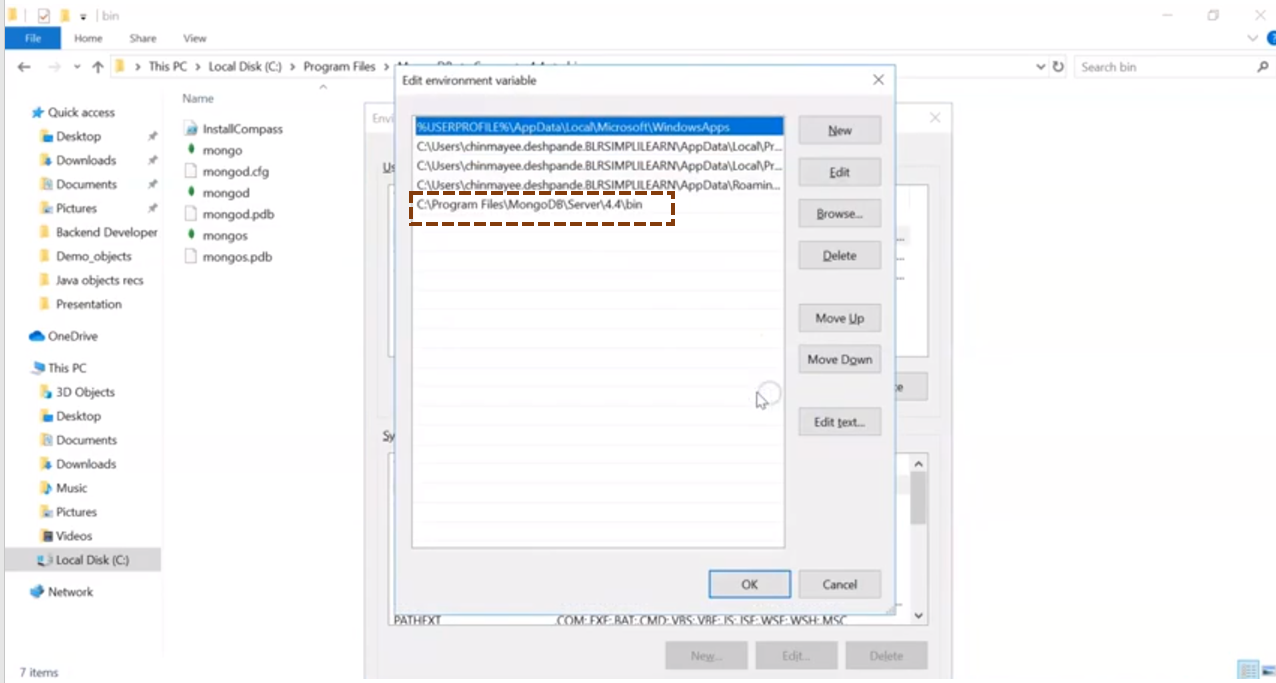






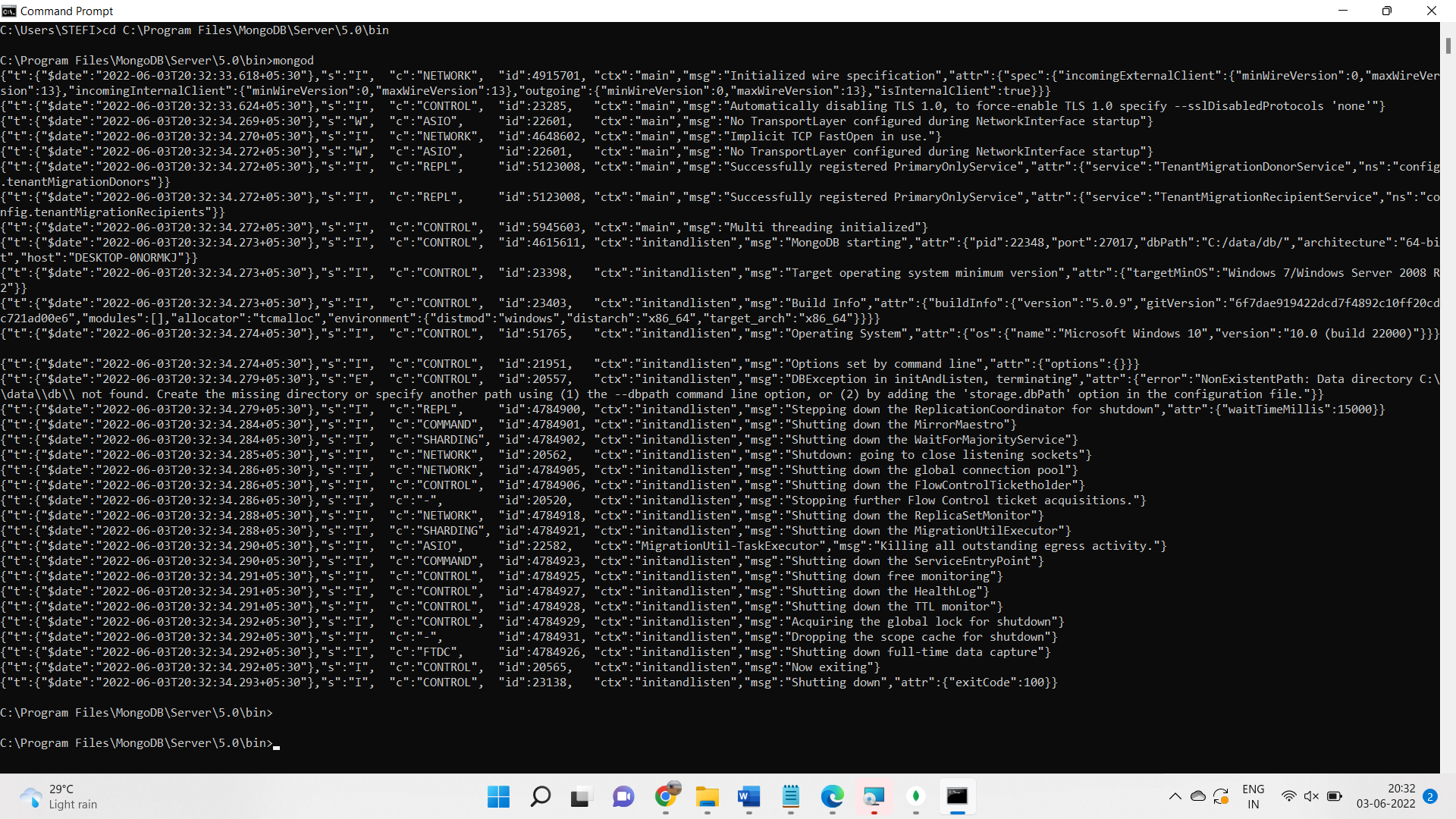
**STEP:4**

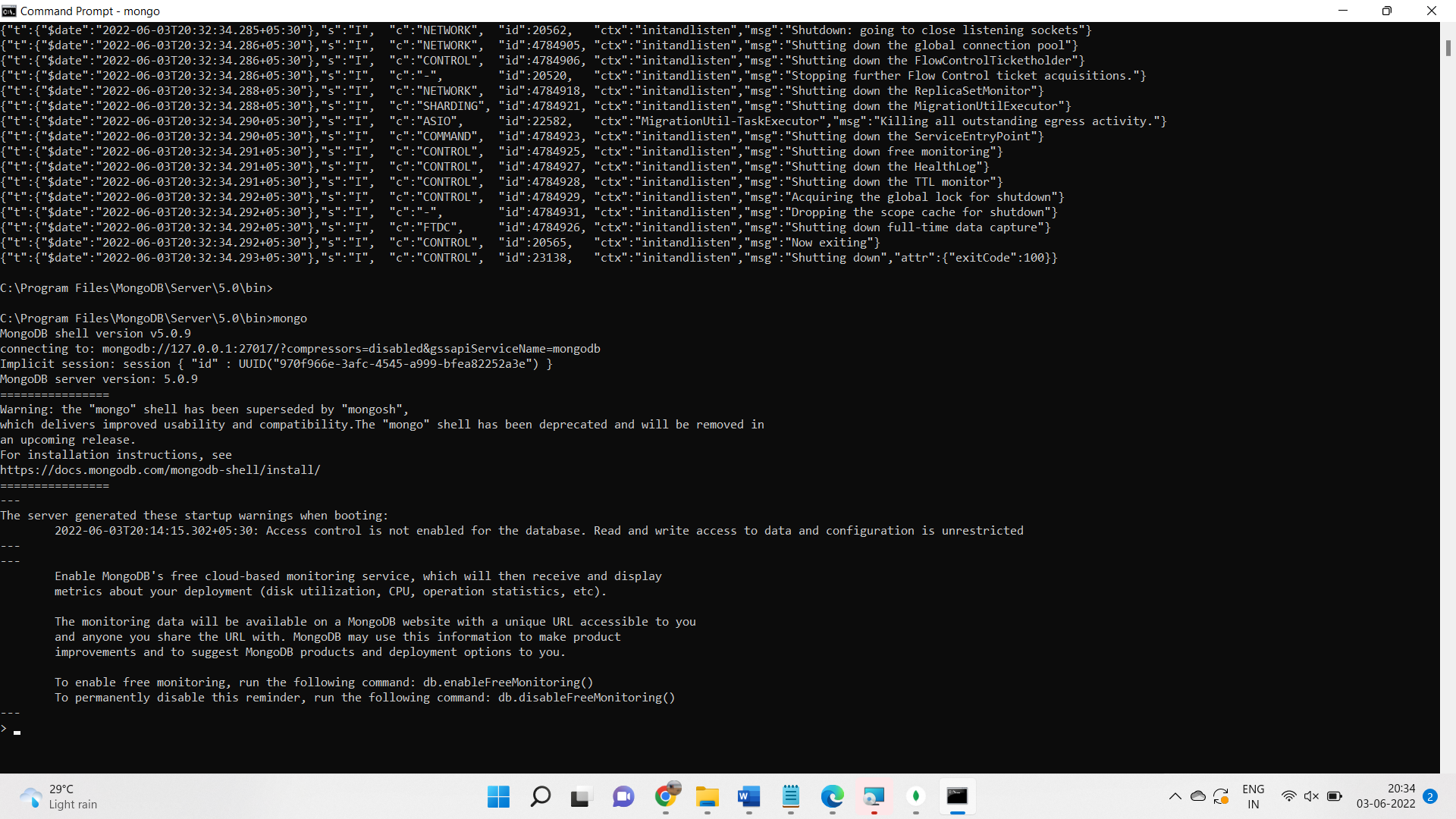
create an environment variable for the executable file so that we don’t have to change the directory structure every time we want to execute the file.



**STEP:5**

After creating an environment path, we can open the command prompt and type mongod and then mongo.





# Experiment 15

### AIM

### To build sample collection/documents to perform query operations.

### CODE:

### 

### OUTPUT

### RESULT

Query processed successfully and output obtained.

# Experiment 16

### AIM

### To perform CRUD operations on the student database.

### 1. Display all the documents.

### 2. Display all the students in Bca.

### 3. Display all the students in ascending order.

### 4. Display all the top three students.

### 5. Display the students 1,2,3.

### 6. Display the degree of the student Rahul.

### 7.Display the student details of 3,4,5 in descending order of cgpa

### 8. Display the number of sttudents in Bca

### 9. Display all the degree without the \_id.

### 10. Display the distinct degree.

### CODE:

### Query 1:

student> db.stud.find().pretty()

**Query 2:**

student> db.stud.find({"degree":"Bca"}).pretty()

**Query 3:**

student> db.stud.find({},{sname:1,\_id:0}).sort({sname:1})

**Query 4:**

student> db.stud.find({},{sname:1,\_id:0}).limit(3).sort({cgpa:-1})

**Query 5:**

student> db.stud.find().skip(2).limit(3)

**Query 6:**

db.stud.find({sname:'Rahul'},{degree:1,\_id:0}).pretty()

### Query 7:

### db.stud.find().skip(2).limit(3).sort({cgpa:-1})

### Query 8:

### db.stud.countDocuments({degree:'Bca'})

### Query 9:

### db.stud.find({},{degree:1,\_id:0}).pretty()

### Query 10:

### db.stud.distinct("degree")

### Query 11:

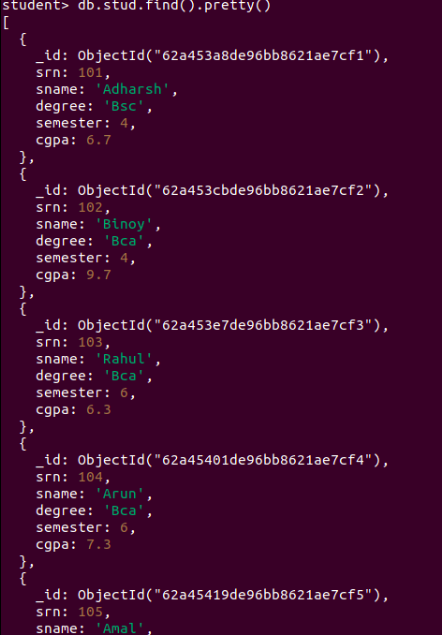
### db.stud.find({$and:[{cgpa:{$gt:6,$lt:7}},{degree:'Bca'}]})

### Query 12:

### db.stud.find({$and:[{degree:'Bca'},{semester:6}]},{sname:1,\_id:0,semester:1})

### OUTPUT

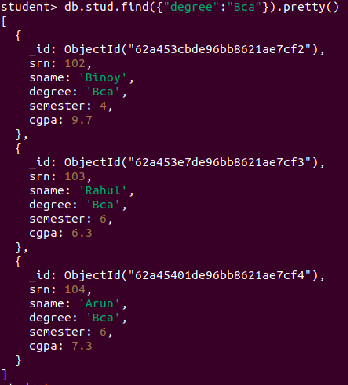
**Query 1:**

****

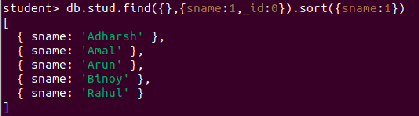


### 

**Query 2:**

****

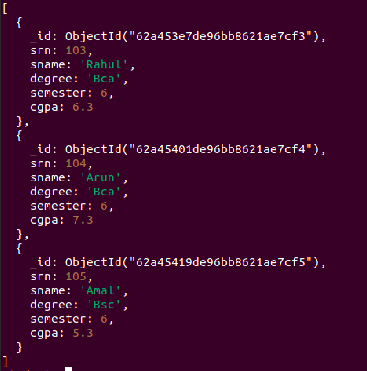
**Query 3:**

****

**Query 4:**

[ { sname: 'Binoy' }, { sname: 'Arun' }, { sname: 'Adharsh' } ]

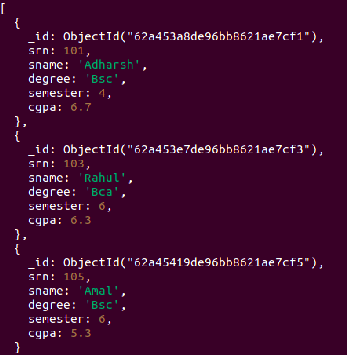
**Query 5:**

****

**Query 6:**

****

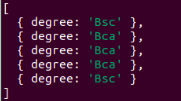
**Query 7:**

****

**Query 8:**

****

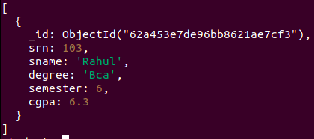
**Query 9:**

****

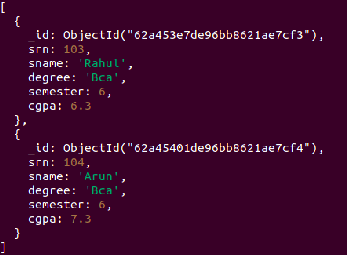
**Query 10:**

****

**Query 11:**

****

**Query 12:**

****

### 

### RESULT

Query processed successfully and output obtained.

**Experiment 17**

**AIM:**

Create an employee database with the field, eid, ename, dept, desig, salary, yoj, address: {dno, street, locality, city}

1.Display the all the epmployee with salary in the range(5000,7500)

2.Display all the employee with design developments

3.Display the salary of Rahul

4.Display the city of employee

5.Update the salary of developers by 5000

6.Add field age to Rahul

7.Remove yoj from Rahul

8.Add an array field project to Rahul

9.Add p2,p3 project to Rahul

10.Remove p3 from Rahul

11.Add a new embedded object ‘contact’ with email and phone as array object to Rahul

**CODE:**

**Query 1:**

Use emp;

Db.emp.find({salary:{$gt50000,$lt:75000}},{\_id:0,ename:1})

**Query 2:**

db.emp.find({desg:'dev'},{\_id:0,ename:1})

**Query 3:**

db.emp.find({ename:'Rahul'},{ename:1,salary:1,\_id:0})

**Query 4:**

db.emp.find({},{ename:1,address:{city:1},\_id:0})

**Query 5:**

db.emp.updateMany({desg:'dev'},{$inc:{salary:5000}})

**Query 6:**

db.emp.updateMany({},{$set:{age:25}})

**Query 7:**

db.emp.updateOne({ename:'Rahul'},{$unset:{yoj:""}})

**Query 8:**

db.emp.updateOne({ename:'Rahul'},{$push:{project:'p1'}})

**Query 9:**

db.emp.updateOne({ename:'Rahul'},{$addToSet:{project:{$each:['p2','p3']}}})

**Query 10:**

db.emp.updateOne({ename:'Rahul'},{$pull:{project:'p3'}})

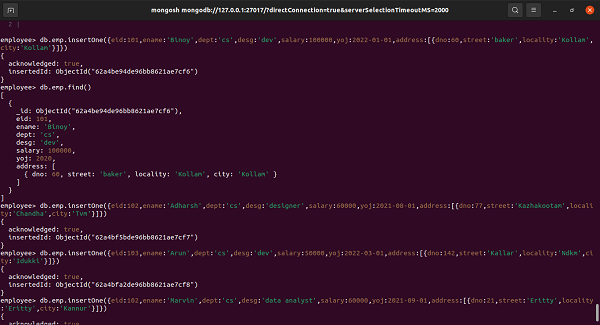
**Query 11:**

db.emp.updateOne({ename:'Rahul'},{$push:{contacts:{phone:626762232,email:'rahul@gmail.com'}}})

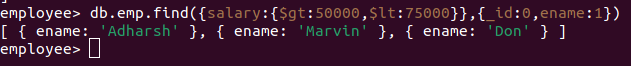
**OUTPUT:**

**Creation:**

****

****

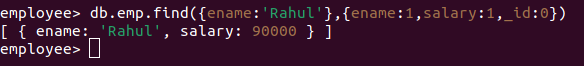
**Query 1:**

****

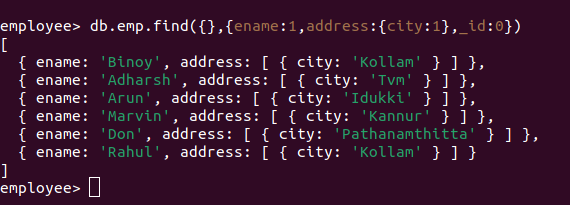
**Query 2:**

****

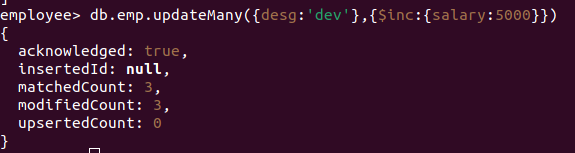
**Query 3:**

****

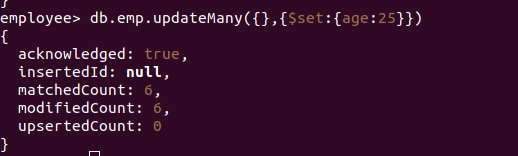
**Query 4:**

****

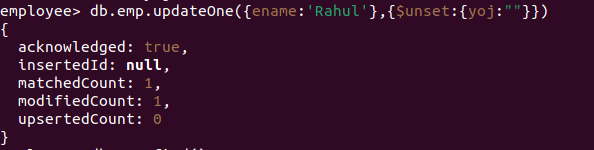
**Query 5:**



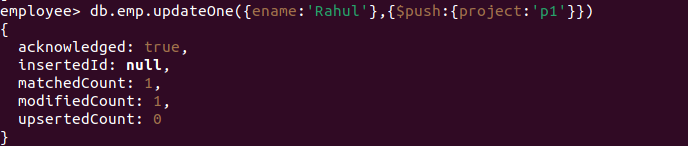
**Query 6:**

****

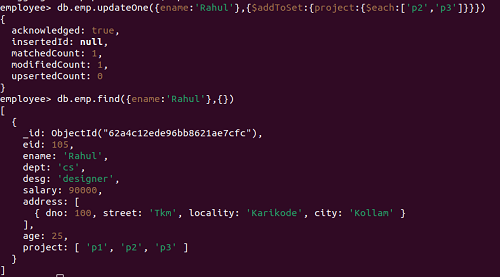
**Query 7:**

****

**Query 8:**

****

**Query 9:**

****

**Query 10:**

****

**Query 11:**

****

### RESULT

Query processed successfully and output obtained.

**Experiment 18**

**AIM:**

Create database candidate and collection details.

1.Query customer who are either male or younger than 25 using aggregate method.

2.Calculate total purchase amount for male and female using aggregate method.

3.Select customer who are older than 25 and calculate the average purchase amount for males and females.

4.Sort the data based on average amount.

**CODE:**

**Query 1:**

use candidate;

db.details.find({$or:[{gender:'M'},{age:{$lt:25}}]},{name:1,\_id:0})

**Query 2:**

db.details.aggregate([{$group:{\_id:"$gender","amount":{$sum:"$amount"}}}])

**Query 3:**

db.details.aggregate([{$match:{age:{$gt:25}}},{$group:{\_id:'$gender',amount:{$avg:'$amount'}}}])

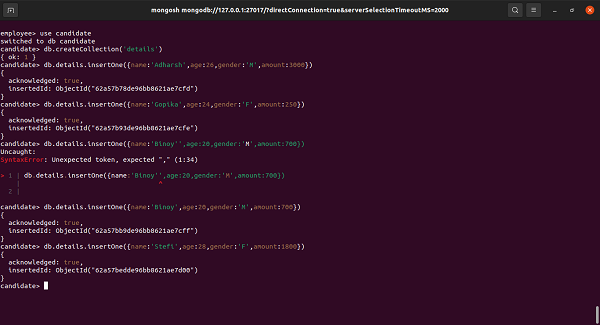
**Query 4:**

db.details.aggregate([{$group:{\_id:'$gender',amount:{$avg:'$amount'}}},{$sort:{amount:1}}])

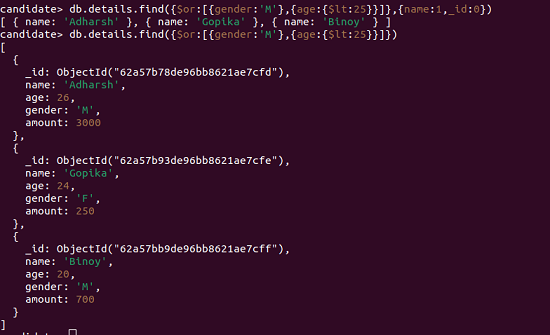
db.details.aggregate([{ $group: { \_id: '$gender', amount: { $avg: '$amount' } } }, { $sort: { amount: -1 } }])

**OUTPUT:**

**Creation:**

****

**Query 1:**

****

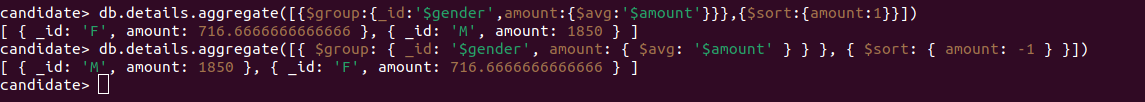
**Query 2:**

****

**Query 3:**

****

**Query 4:**

****

### RESULT

Query processed successfully and output obtained.

**Experiment 19**

**AIM:**

Create database college and collection student.

1.Display the details of the students who have their name starting with the letter c using $regex operator.

2. Display the details of the students who have their name ending with the letter r using $regex operator.

3. Display the details of the students who having cs as their department.

4. Remove the details of the students who having ec as their department.

**CODE:**

**Query 1:**

use college;

db.student.find({sname:{$regex:"^c"}});

**Query 2:**

db.student.find({sname:{$regex:"r$"}});

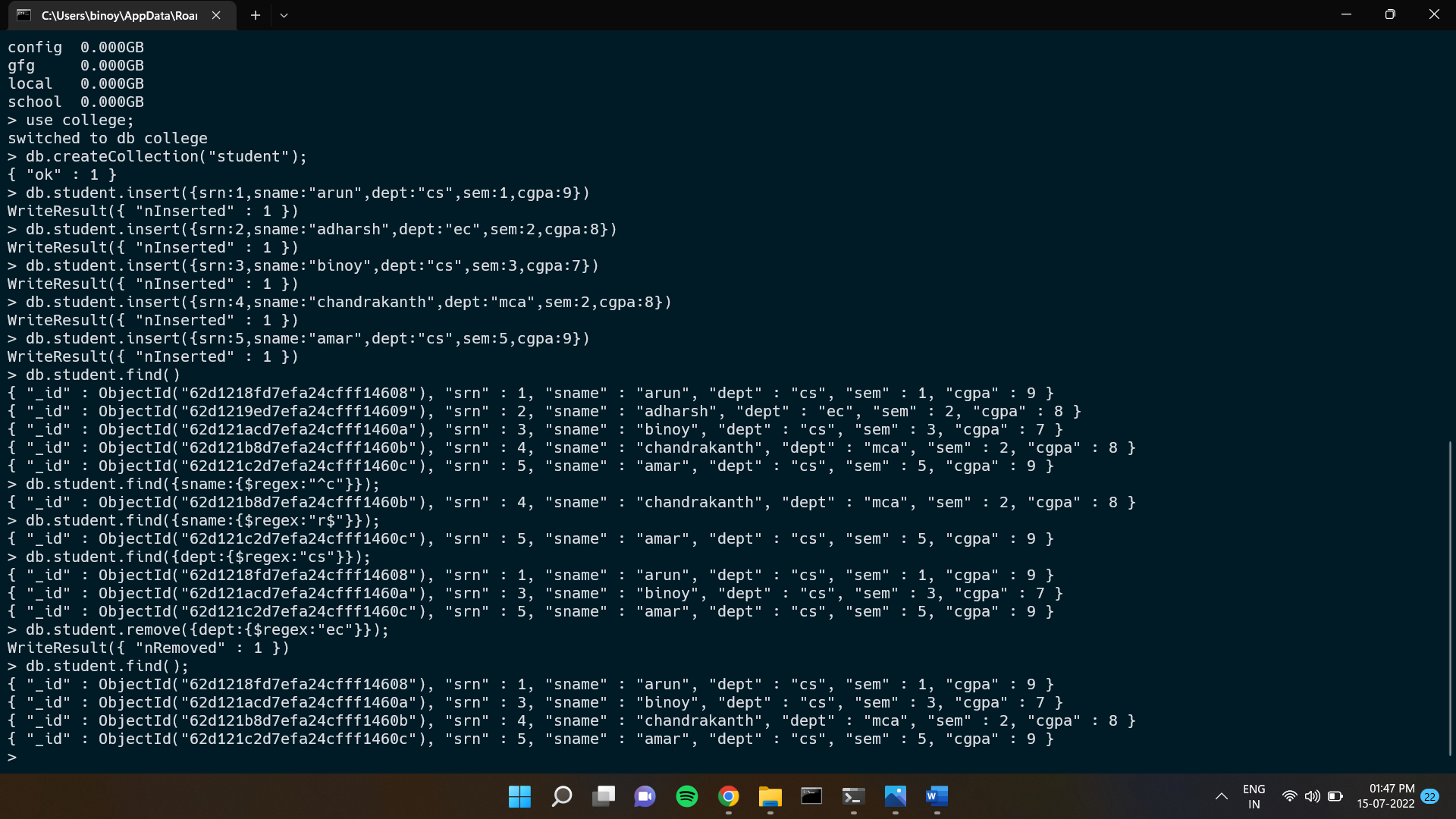
**Query 3:**

db.student.find({dept:{$regex:"cs"}});

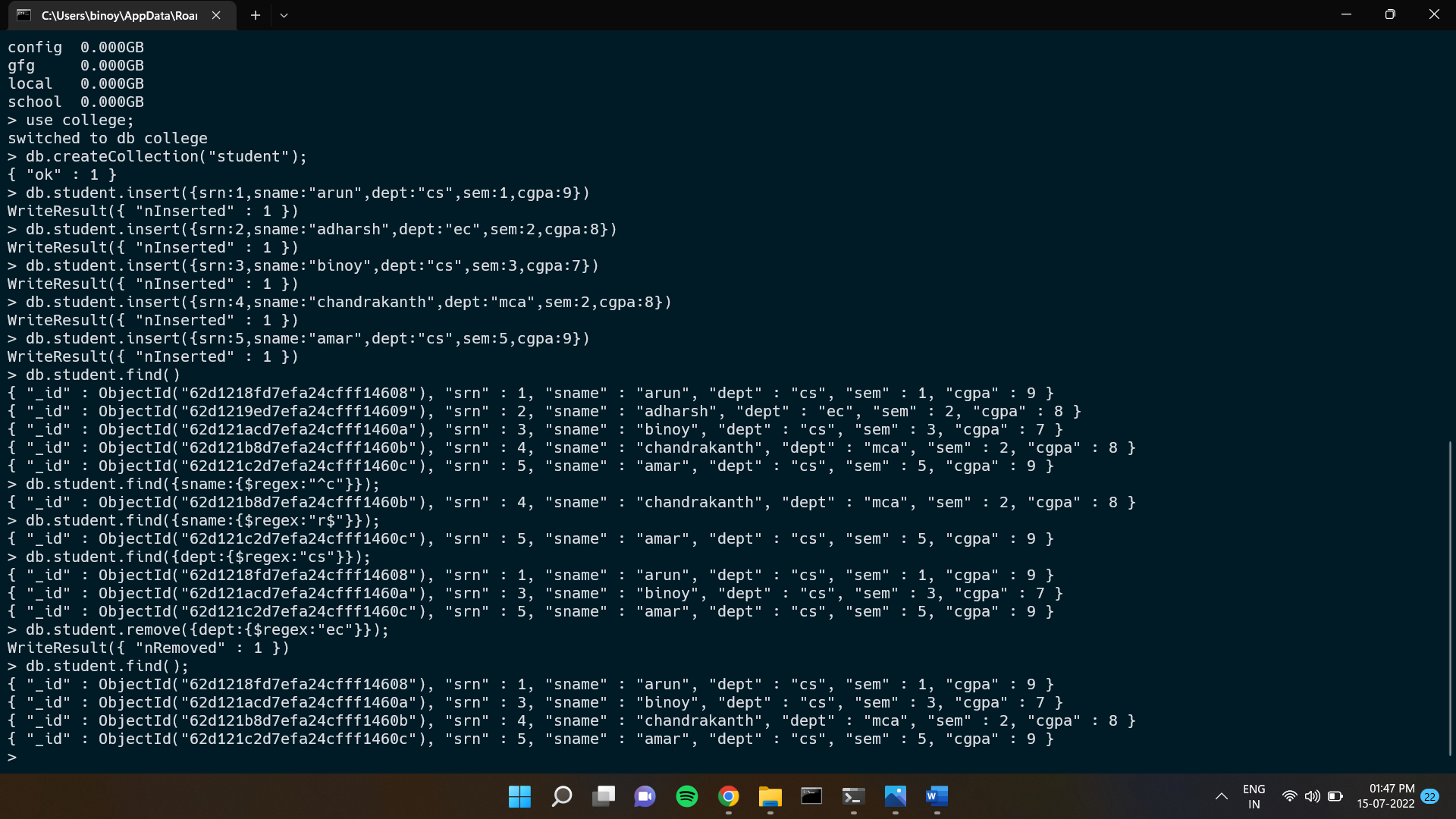
**Query 4:**

db.student.remove({dept:{$regex:"ec"}});

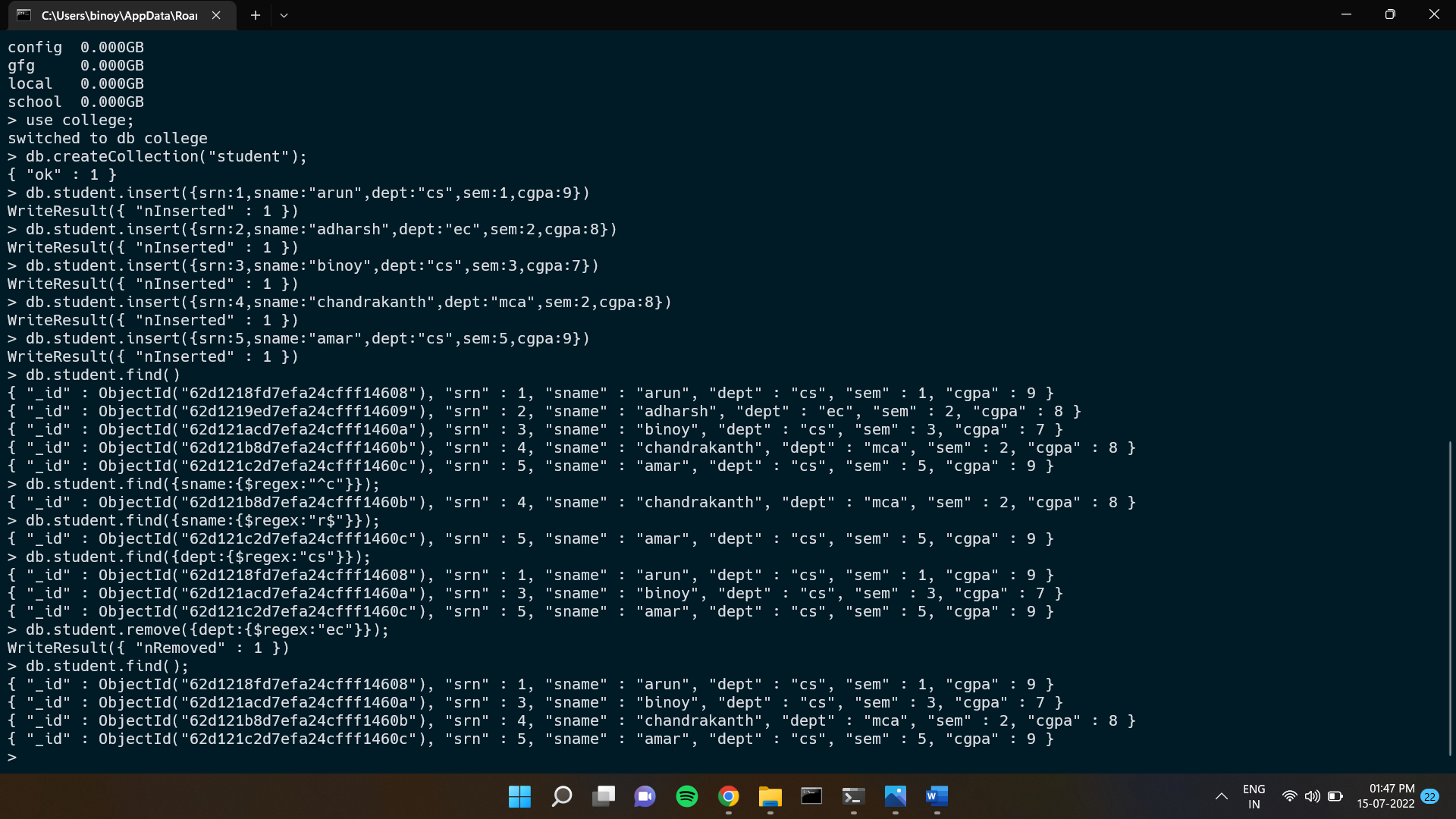
**OUTPUT:**



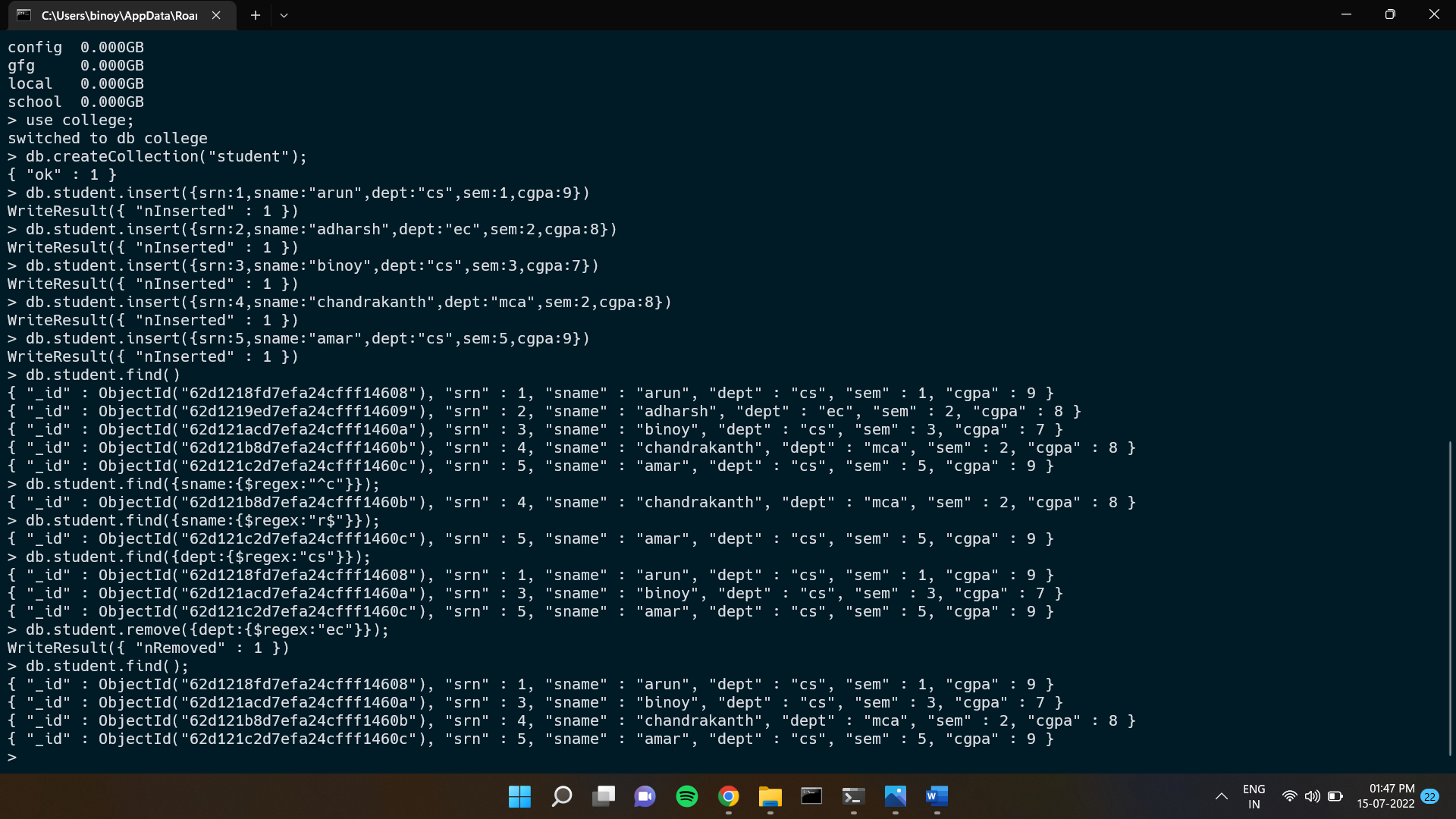
**Query 1:**

****

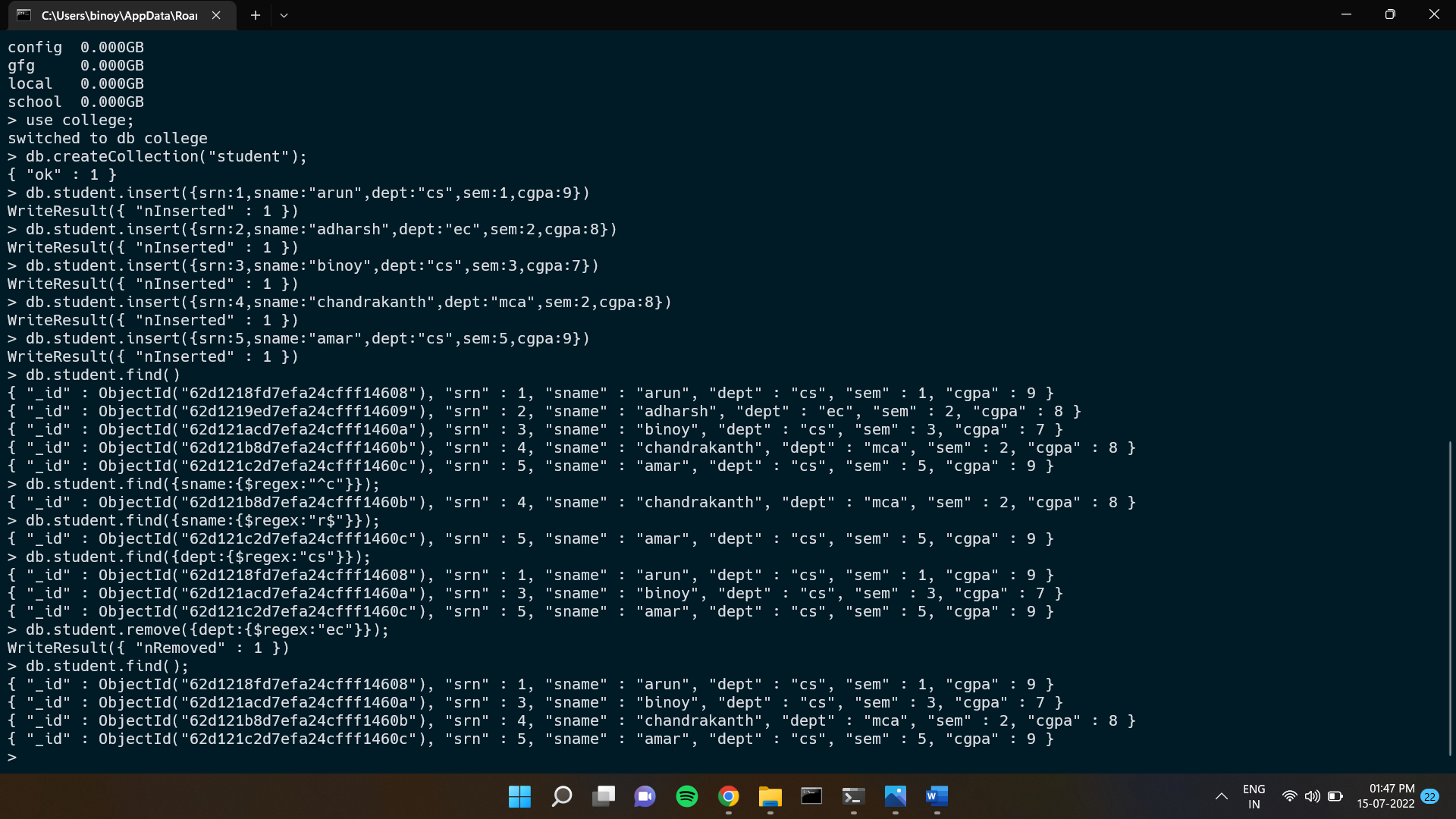
**Query 2:**

****

**Query 3:**

****

**Query 4:**



### RESULT

Query processed successfully and output obtained.

**Experiment 20**

**AIM:**

To write queries to implement backup and monitoring in mongodb.

**CODE:**

**Query 1:**

mongodump

**Query 2:**

mongorestore ./dump/

**Query 3:**

mongodump --db=employee

**Query 4:**

mongorestore --db employee dump/employee

**Query 5:**

mongodump --db student --collection Details

**Query 6:**

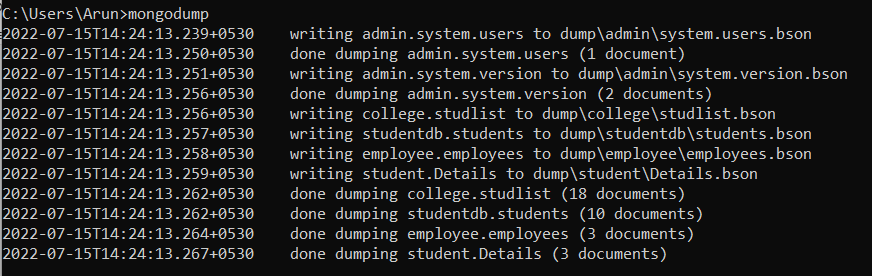
mongorestore --db student --collection Details dump/student/Details.bson

**Query 7:**

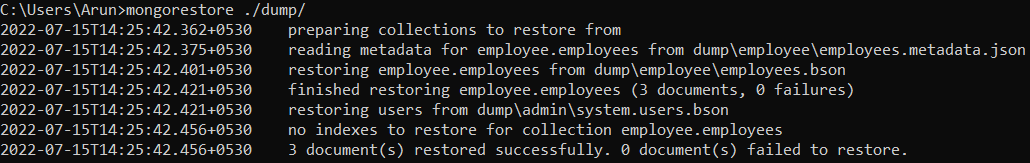
mongostat

**OUTPUT:**

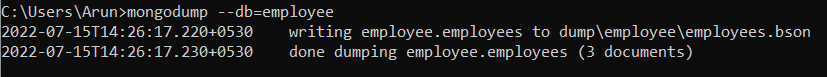
**Query 1:**



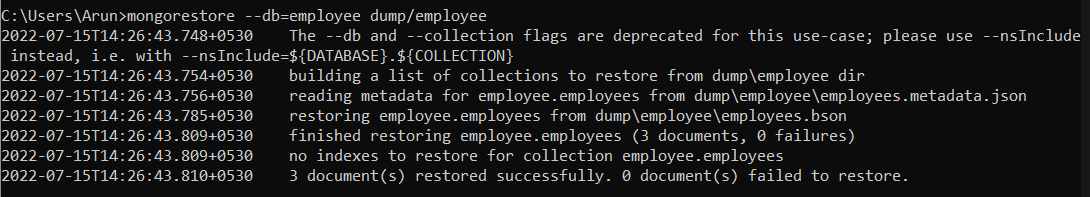
**Query 2:**



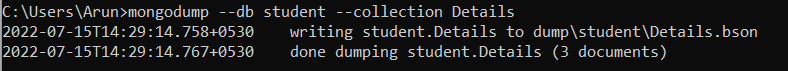
**Query 3:**



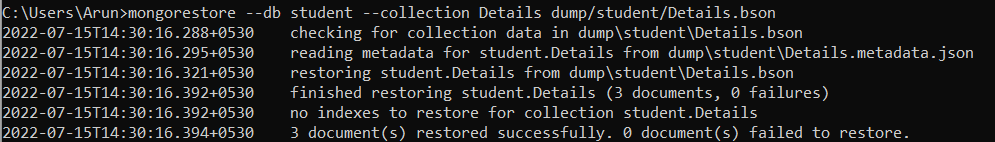
**Query 4:**



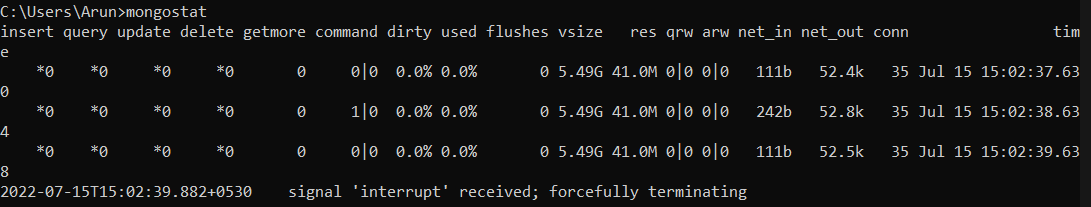
**Query 5:**



**Query 6:**



**Query 7:**



### RESULT

Query processed successfully and output obtained.

**Experiment 21**

**AIM:**

Create database college and collection student. Write Queries to implement Users and roles

**CODE:**

**Query 1:**

use student

**Query 2:**

db.createUser({user:"amal",pwd:"1234",roles:[{role:"readwrite",db:"student"}]})

**Query 3:**

show users

**Query 4:**

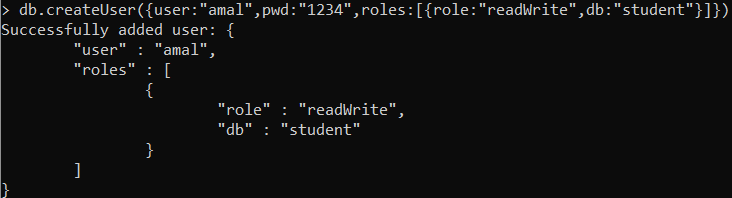
show roles

**OUTPUT:**

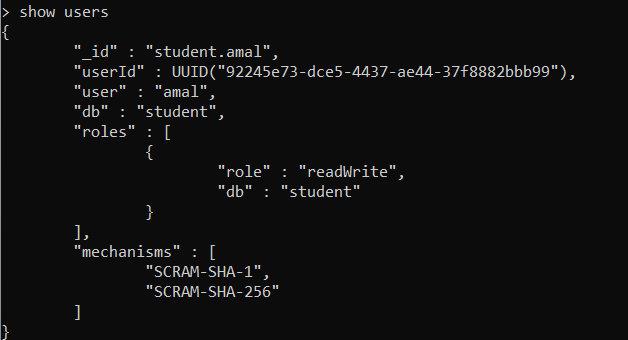
**Query 1:**



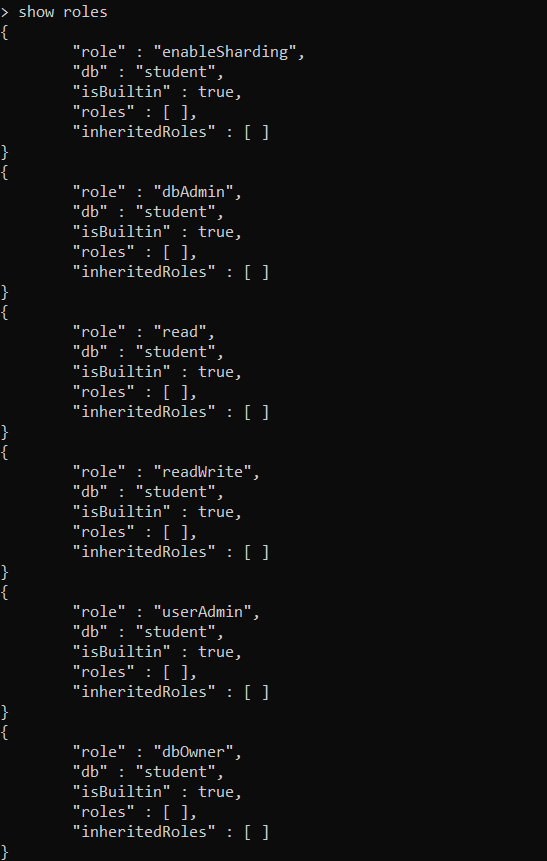
**Query 2:**



**Query 3:**



**Query 4:**



### RESULT

Query processed successfully and output obtained.

**Experiment 22**

**AIM:**

Create database college and collection student. Write Queries to implement Replication

**CODE:**

**Step 1:**

Create a folder named Data in the C Drive.

Create rs1,rs2,rs3 as sub directories to Data folder.

Open a cmd in the bin folder of the mongo.

**Query 1:**

>mongo --port 27018

>start mongod -replSet qwerty -logpath \Data\rs1\1.log -dbpath \Data\rs1 --port 27018

>start mongod -replSet qwerty -logpath \Data\rs2\2.log -dbpath \Data\rs2 --port 27019

>start mongod -replSet qwerty -logpath \Data\rs3\3.log -dbpath \Data\rs3 --port 27020

**Step 2:**

Open a mongoshell.

**Query 2:**

>config={\_id:"qwerty",members:[{\_id:0,host:"localhost:27018"},{\_id:1,host:"localhost:27019"},{\_id:2,host:"localhost:27020"}]}

> rs.initiate(config)

> show dbs

**Step 3:**

Open another mongosh

**Query 3:**

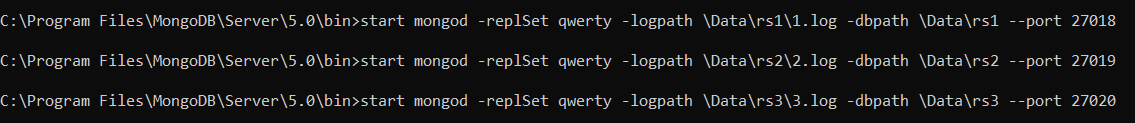
> mongo --port 27019

> rs.secondaryOk()

>show dbs

**OUTPUT:**

**Query 1:**

****

**Query 2:**

**Text

Description automatically generated**

**Query 3:**

****

**Text

Description automatically generated with medium confidence**

### RESULT

Query processed successfully and output obtained.

**Experiment 23**

**AIM:**

Create a collection college and comment and do the following basic indexing operations

1)create collection college

2)insert data

3)Get all indexes

4)Create new index and show all indexes

5)Drop index and show all indexes

6)create collection comment and insert data

8)Create a text Index and sow all indexes

9)With the help of text index search on the Collection

**CODE:**

**Query 1:**

db.createCollection("college")

**Query 2:**

db.college.insert({RegNo:2101,Name:"Raichal",Mark:[{cs:95,maths:88,phy:75,chem:85,eng:91}]})

db.college.insert({RegNo:2102,Name:"Abina",Mark:[{cs:99,maths:82,phy:76,chem:95,eng:91}]})

db.college.insert({RegNo:2103,Name:"Riya",Mark:[{cs:100,maths:85,phy:71,chem:91,eng:94}]})

**Query 3:**

db.college.getIndexes()

**Query 4:**

db.college.createIndex({RegNo:1})

db.college.getIndexes()

**Query 5:**

db.college.dropIndex({RegNo:1})

db.college.getIndexes()

**Query 6:**

db.createCollection("comment")

db.comment.insert({name:"Abina",post:"hello"})

db.comment.insert({name:"Anu",post:"hai"})

db.comment.insert({name:"Sona",post:"hola"})

db.comment.createIndex({post:"text"})

**Query 7:**

db.comment.createIndex({post:"text"})

db.comment.getIndexes()

**Query 8:**

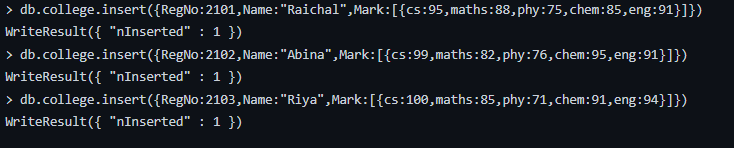
db.comment.find({$text:{$search:"\"hai\""}})

**OUTPUT:**

**Query 1:**



**Query 2:**



**Query 3:**



**Query 4:**

Text

Description automatically generated

**Query 5:**

A screenshot of a computer

Description automatically generated with low confidence

**Query 6:**

Text

Description automatically generated

**Query 7:**

Text

Description automatically generated

**Query 8:**



### RESULT

Query processed successfully and output obtained.