**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**



**LAB REPORT**

**on**

**Database Management Systems (23CS3PCDBM)**

***Submitted by***

**MANYU KANGANNAR MURALIDHAR (1BM24CS166)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**Sep-2024 to Jan-2025**

**B.M.S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “Database Management Systems (23CS3PCDBM)” carried out by **MANYU KANGANNAR MURALIDHAR(1BM24CS166),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of a Database Management Systems (23CS3PCDBM) work prescribed for the said degree.

|  |  |
| --- | --- |
| Rashmi H  Assistant Professor  Department of CSE, BMSCE | Dr. Joythi S Nayak  Professor & HOD  Department of CSE, BMSCE |

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 4-10-2025 | Insurance Database | 4 |
| 2 | 9-10-2025 | More Queries on Insurance Database | 12 |
| 3 | 16-10-2025 | Bank Database | 14 |
| 4 | 23-10-2025 | More Queries on Bank Database | 19 |
| 5 | 30-10-2025 | Employee Database | 21 |
| 6 | 5-11-2025 | More Queries on Employee Database | 21 |
| 7 | 12-11-2025 | Supplier Database | 31 |
| 8 | 27-11-2025 | NO SQL - Student Database | 38 |
| 9 | 4-12-2025 | NO SQL - Customer Database | 41 |
| 10 | 4-12-2025 | NO SQL – Restaurant Database | 44 |

Insurance Database

**Question:**

**Week 1**

**-** PERSON (driver\_id: String, name: String, address: String)

**-** CAR (reg\_num: String, model: String, year: int)

**-** ACCIDENT (report\_num: int, accident\_date: date, location: String)

**-** OWNS (driver\_id: String, reg\_num: String)

**-** PARTICIPATED (driver\_id: String,reg\_num: String, report\_num: int, damage\_amount: int)

**-** Create the above tables by properly specifying the primary keys and the foreign keys. **-** Enter at least five tuples for each relation

**-** Display Accident date and location

**-** Update the damage amount to 25000 for the car with a specific reg\_num (example 'K A053408' ) for which the accident report number was 12.

**-** Add a new accident to the database.

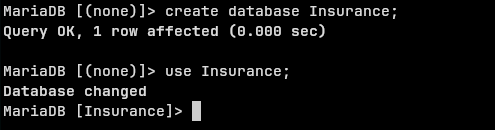
**-** To Do

**-** Display Accident date and location

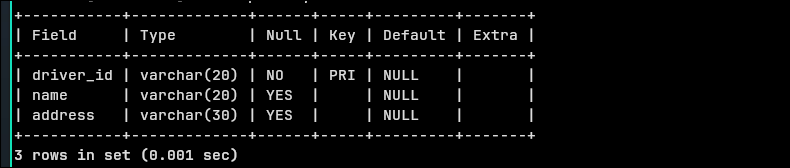
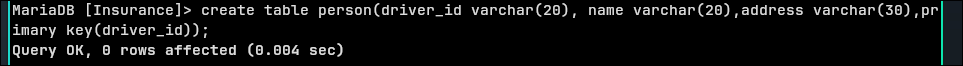
**-** Display driver id who did accident with damage amount greater than or equal to Rs.25000

**Queries**

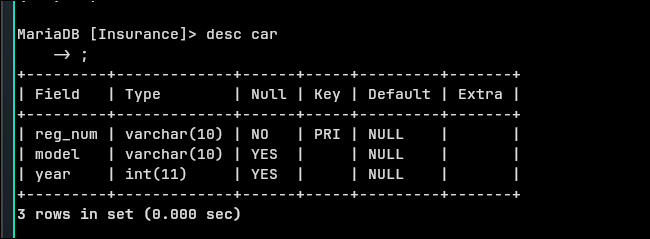
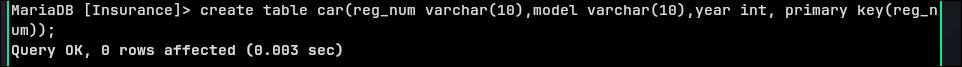
Creating database:



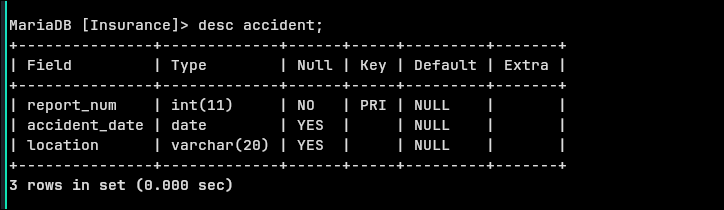
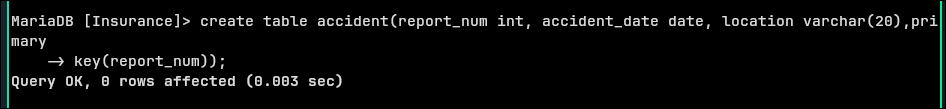
Creating person table:



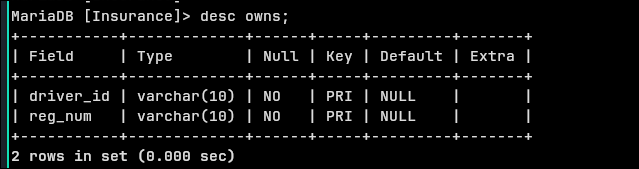
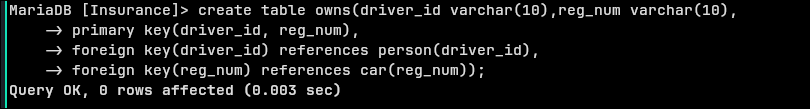
Creating car table:



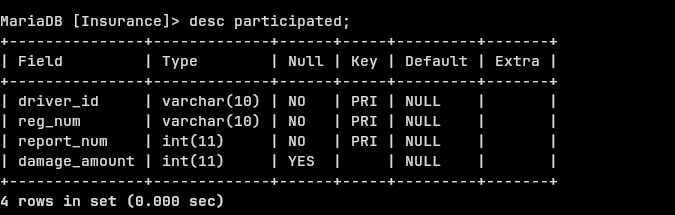
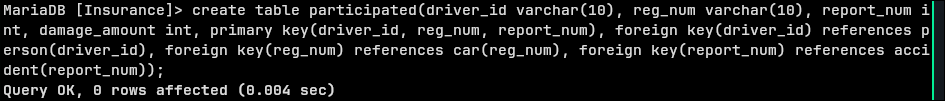
Creating accident table:



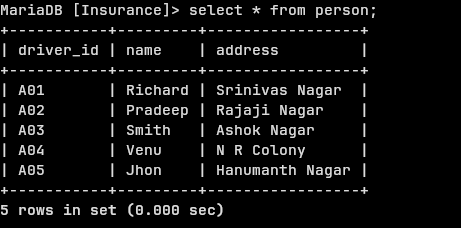
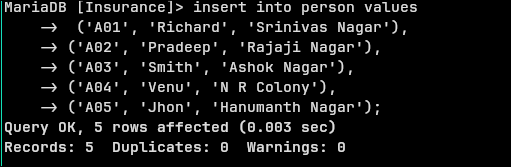
Creating owns table:



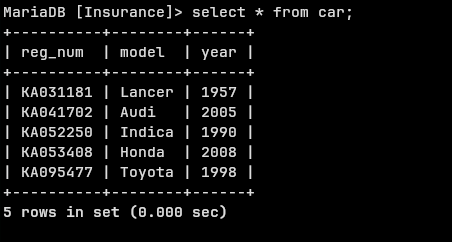
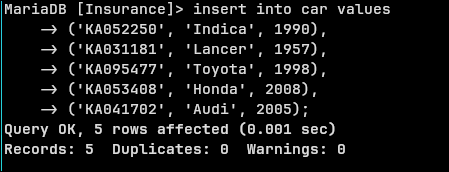
Creating participated table:



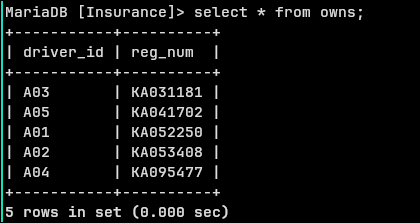
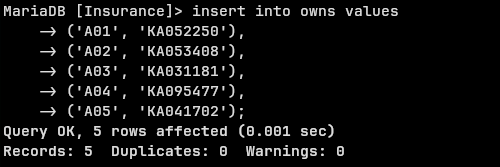
Inserting values into Person:



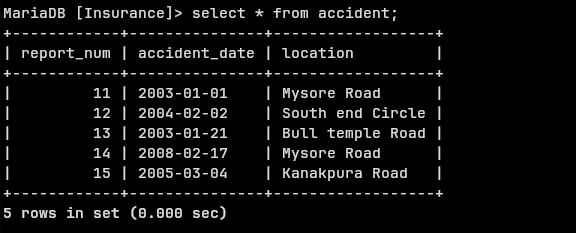
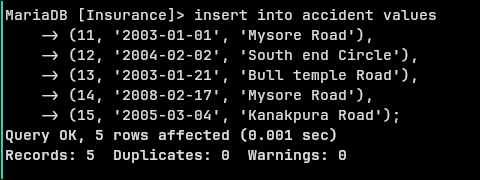
Inserting into Car:



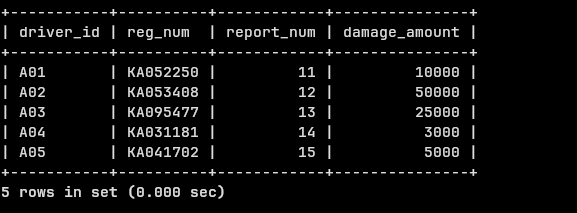
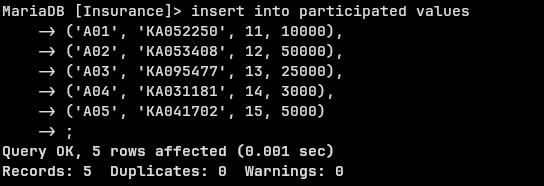
Inserting into Owns:



Inserting into Accident:

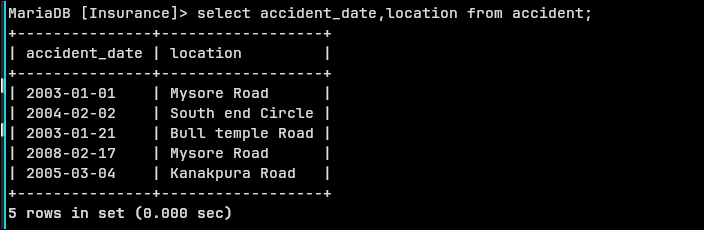


Inserting into participated:

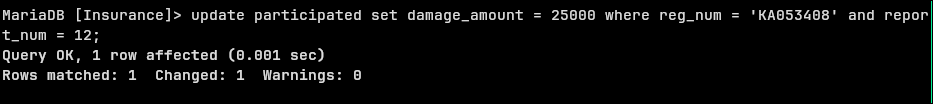


**Main Queries:**

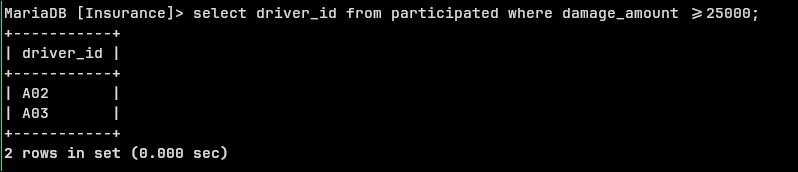
1.



2.



3.

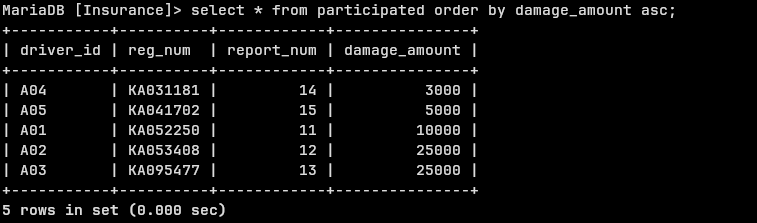


**Question:**

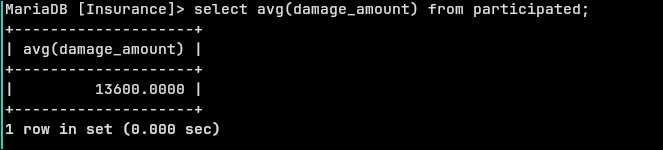
**Week 2  
More Queries on Insurance database.**

1. **LIST THE ENTIRE PARTICIPATED RELATION IN THE DESCENDING ORDER OF DAMAGE AMOUNT.**
2. **FIND THE AVERAGE DAMAGE AMOUNT**
3. **DELETE THE TUPLE WHOSE DAMAGE AMOUNT IS BELOW THE AVERAGE DAMAGE AMOUNT**
4. **LIST THE NAME OF DRIVERS WHOSE DAMAGE IS GREATER THAN THE AVERAGE DAMAGE AMOUNT.**
5. **FIND MAXIMUM DAMAGE AMOUNT.**

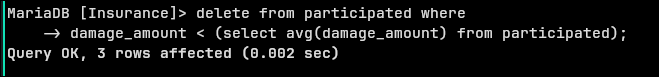
1.



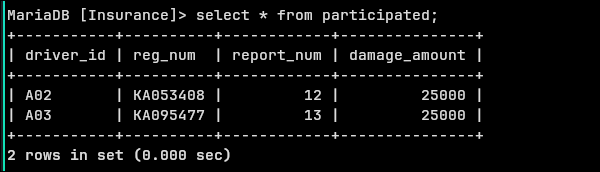
2.



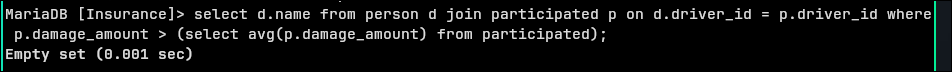
2.



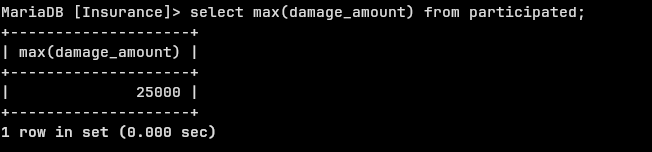
3.



4.



5.



**Week 3**

**Question:**

Consider the following database for a banking enterprise.

Branch (branch-name: String, branch-city: String, assets: real)

BankAccount(accno: int, branch-name: String, balance: real)

BankCustomer (customer-name: String, customer-street: String, customer-city: String)

Depositer(customer-name: String, accno: int)

Loan (loan-number: int, branch-name: String, amount: real)

i. Create the above tables by properly specifying the primary keys and the

foreign keys.

ii. Enter at least five tuples for each relation.

iii. Find all the customers who have at least two accounts at the Main branch (ex.

SBI\_ResidencyRoad).

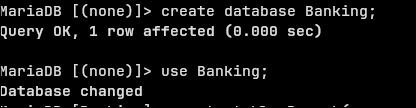
iv. Find all the customers who have an account at all the branches located in a

specific city (Ex. Delhi).

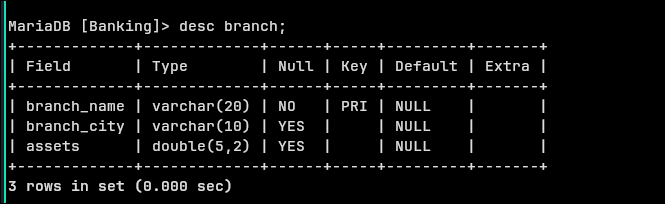
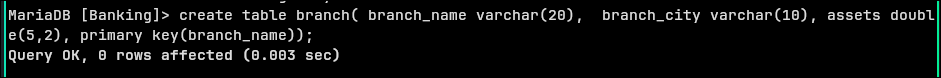
v. Demonstrate how you delete all account tuples at every branch located in

a specific city (Ex. Bombay).

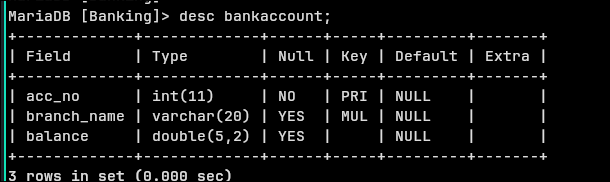
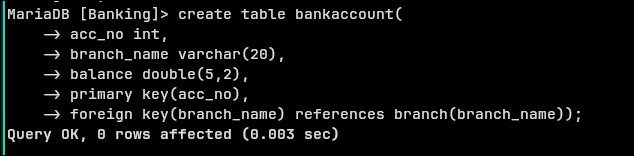
Creating database:



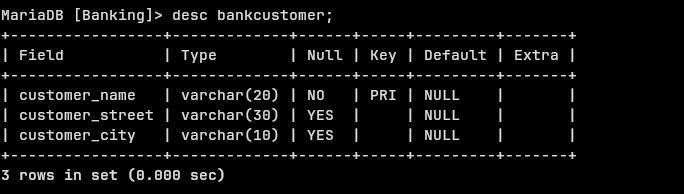
Creating branch table:



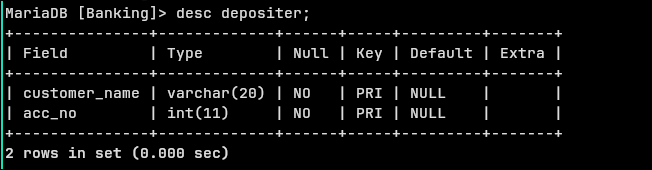
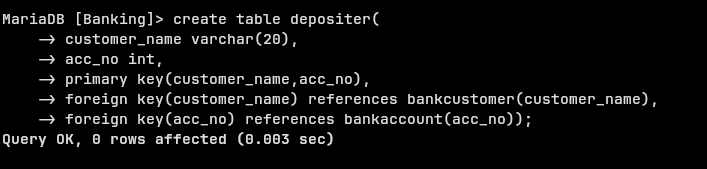
Creating BankAccount table:



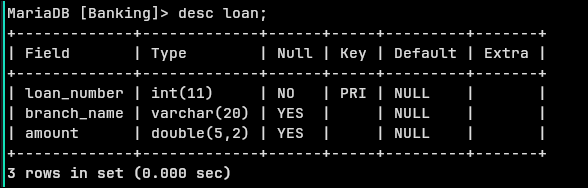
Creating BankCustomer table:



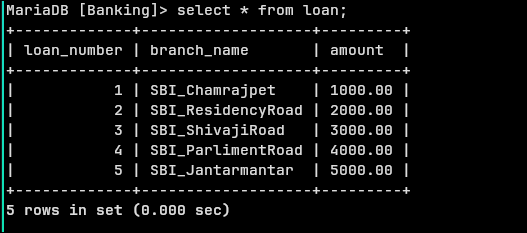
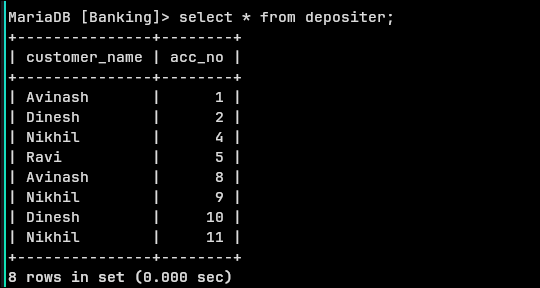
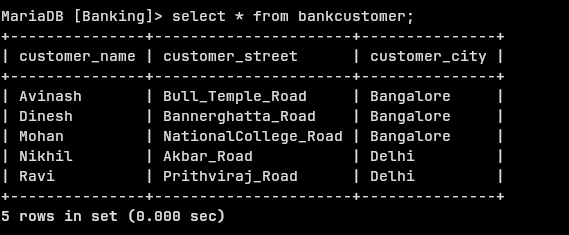
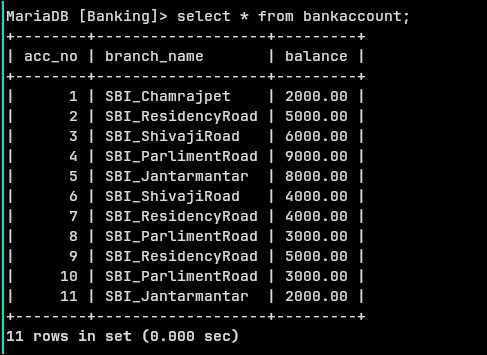
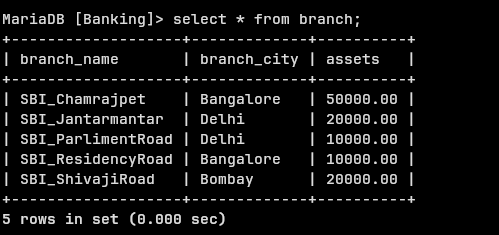
Creating depositer table:



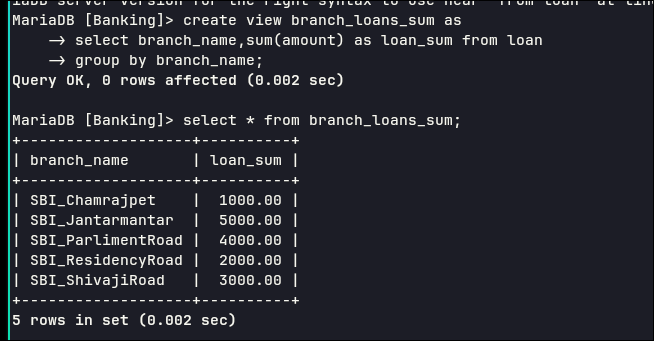
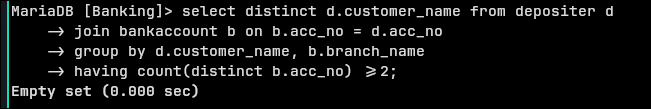
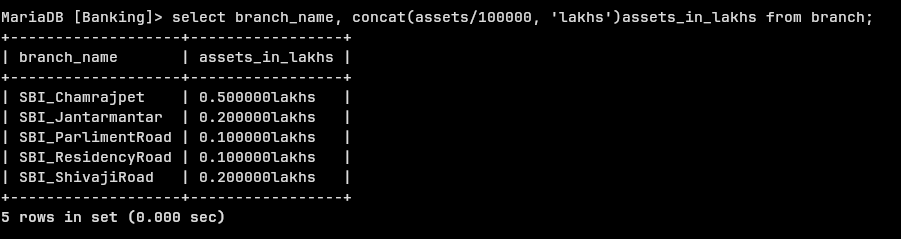
Creating loan table:



Inserting values into all tables:



**Main queries:**



**Week 4**

**More queries on Banking database:**

**Find all the customers who have an account at all the branches**

**located in a specific city (Ex. Delhi).**

**Find all customers who have a loan at the bank but do not have**

**an account.**

**Find all customers who have both an account and a loan at the**

**Bangalore branch**

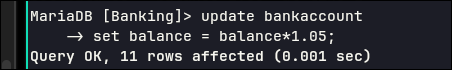
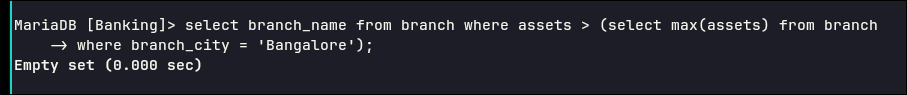
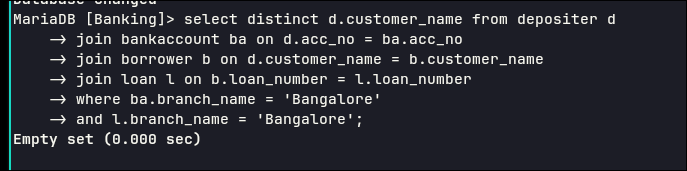
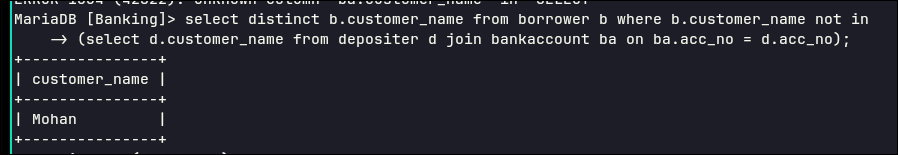
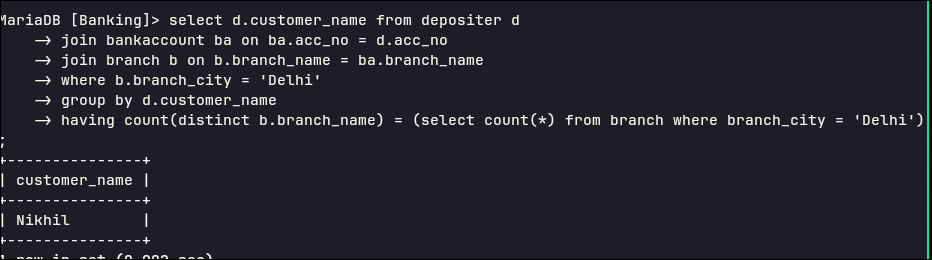
**Find the names of all branches that have greater assets than all**

**branches located in Bangalore.**

**Demonstrate how you delete all account tuples at every branch**

**located in a specific city (Ex. Bombay).**

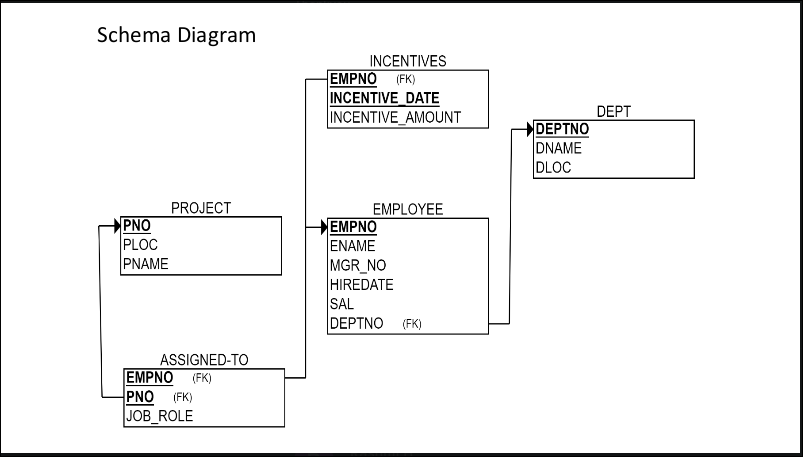
**Update the Balance of all accounts by 5%**



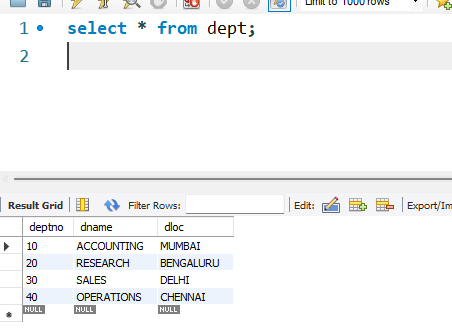
**(Week-5 & 6)**

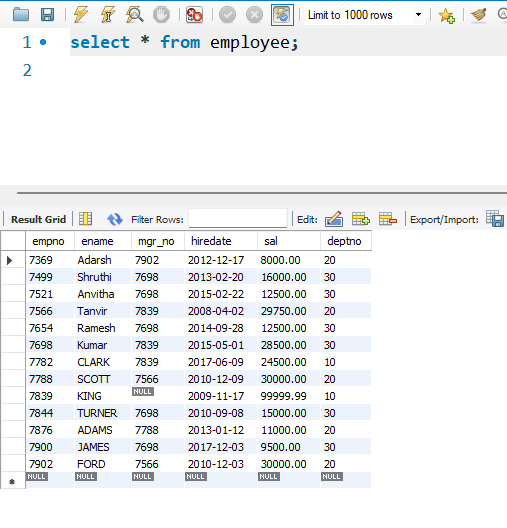
**Using Scheme diagram, Create tables by properly specifying the primary keys and the foreign keys.**

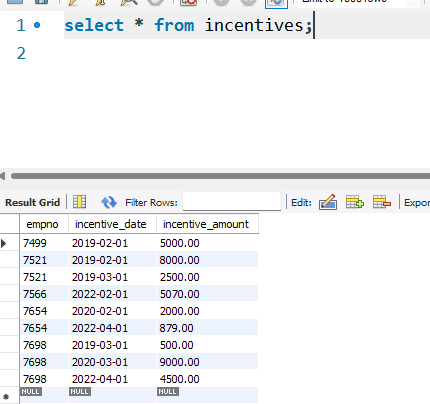
1. **Enter greater than five tuples for each table.**
2. **Retrieve the employee numbers of all employees who work on**
3. **project located in Bengaluru, Hyderabad, or Mysuru**
4. **Get Employee ID’s of those employees who didn’t receive incentives**
5. **Write a SQL query to find the employees name, number, dept,**
6. **job\_role, department location and project location who are working for**
7. **a project location same as his/her department location.**
8. **list the name of the managers with the most employees.**
9. **display the name of managers whose salary is more than the average salary of his employee?**
10. **sql query to find the name of the top level manager of each department.**
11. **sql query to find the employee details who got second maximum incentive in february 2019.**
12. **display those employees who are working in the same dept where his manager is work?**
13. **write a sql query to find those employees whose net pay are higher than or equal to the salary of any other employee in the company.**

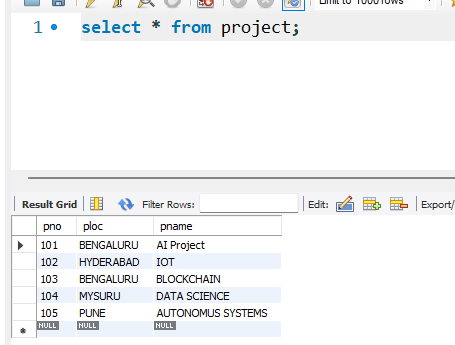


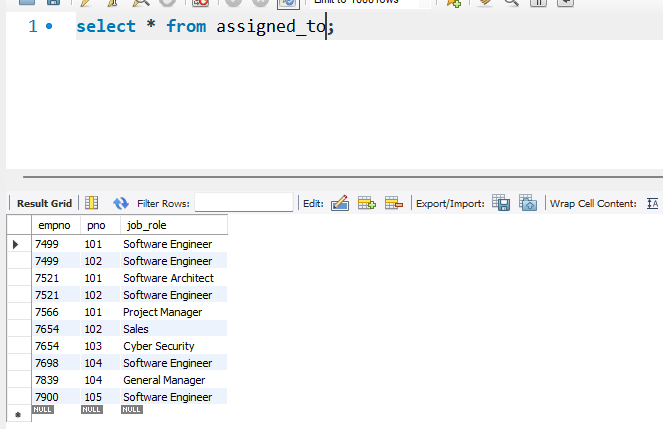
Creating tables and inserting values:



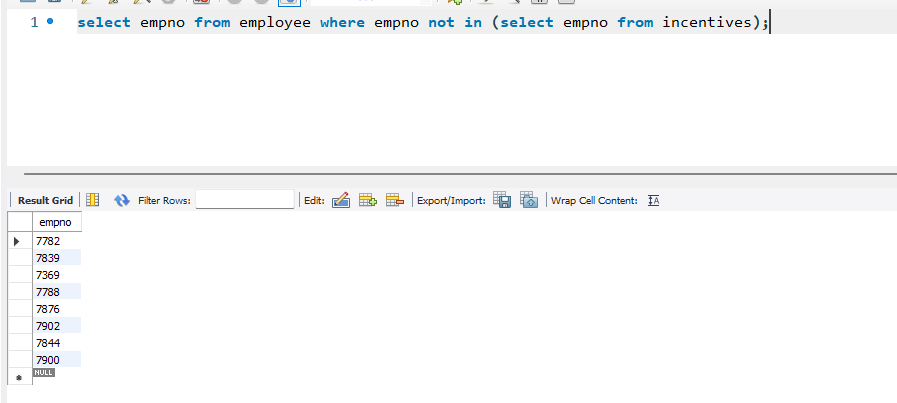


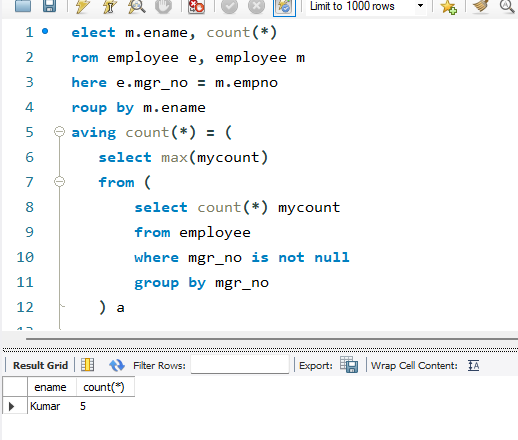


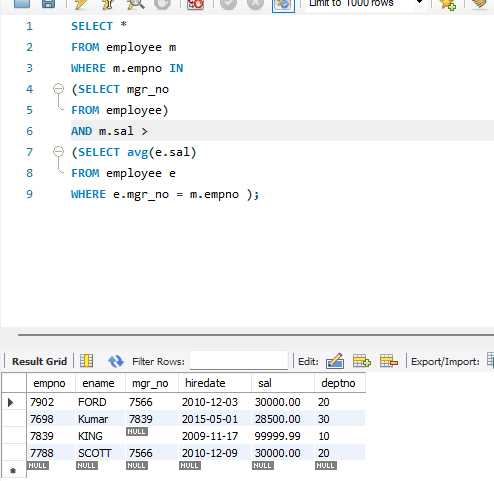


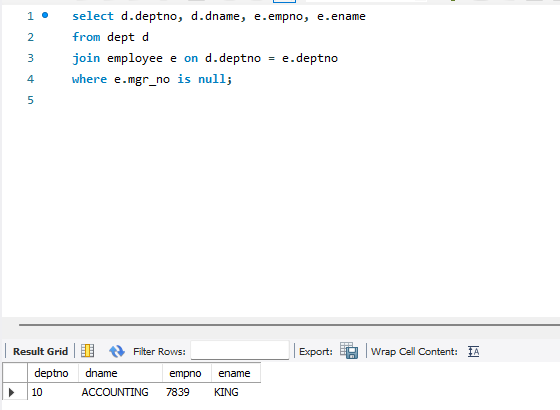


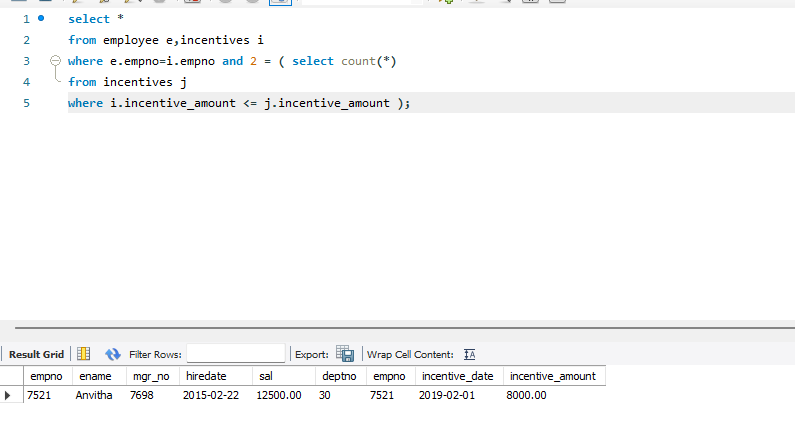
**Main Queries**

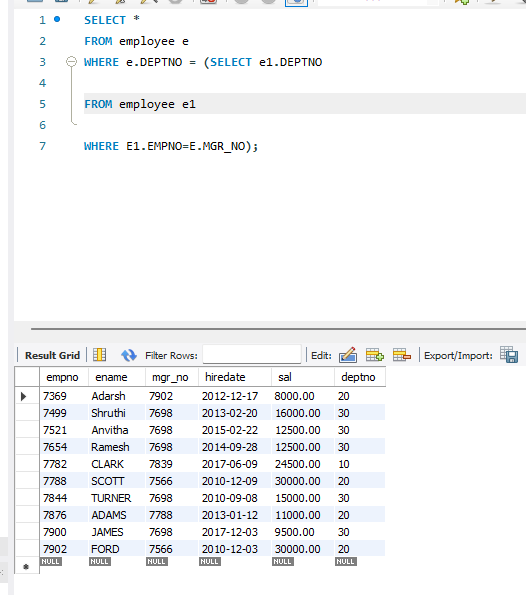


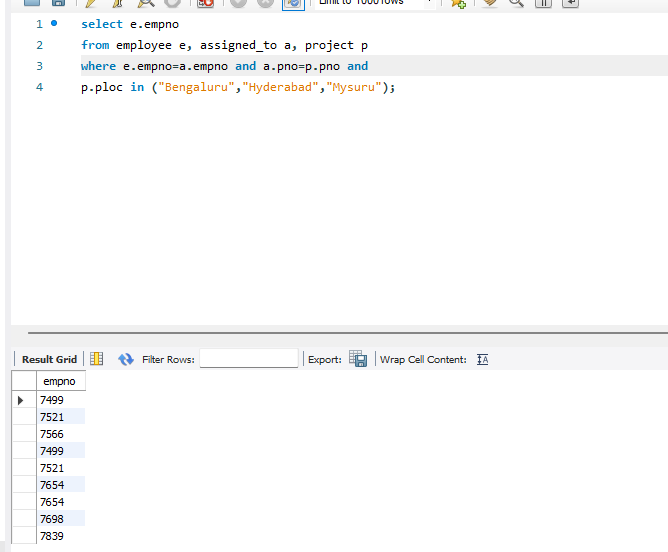
















**Week-7: To Do List**

**1. Using Scheme diagram, Create tables by properly specifying the primary**

**keys and the foreign keys.**

**2. Insert appropriate records in each table.**

**3. Find the pnames of parts for which there is some supplier.**

**4. Find the snames of suppliers who supply every part.**

**5. Find the snames of suppliers who supply every red part.**

**6. Find the pnames of parts supplied by Acme Widget Suppliers and by no**

**one else.**

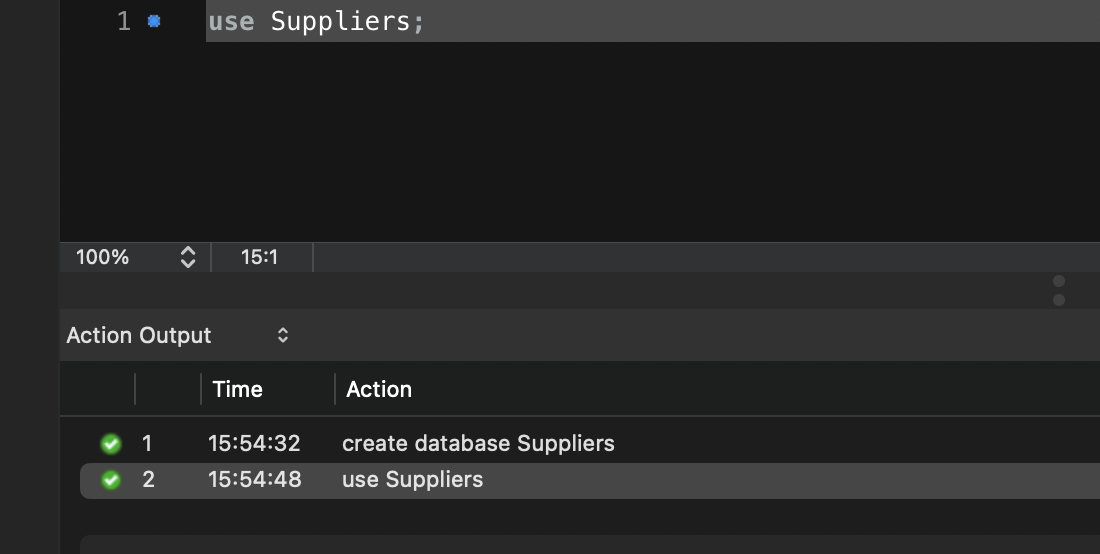
**7. Find the sids of suppliers who charge more for some part than the average**

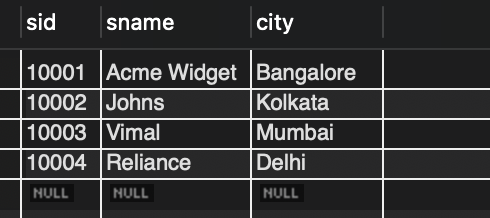
**cost of that part (averaged over all the suppliers who supply that part).**

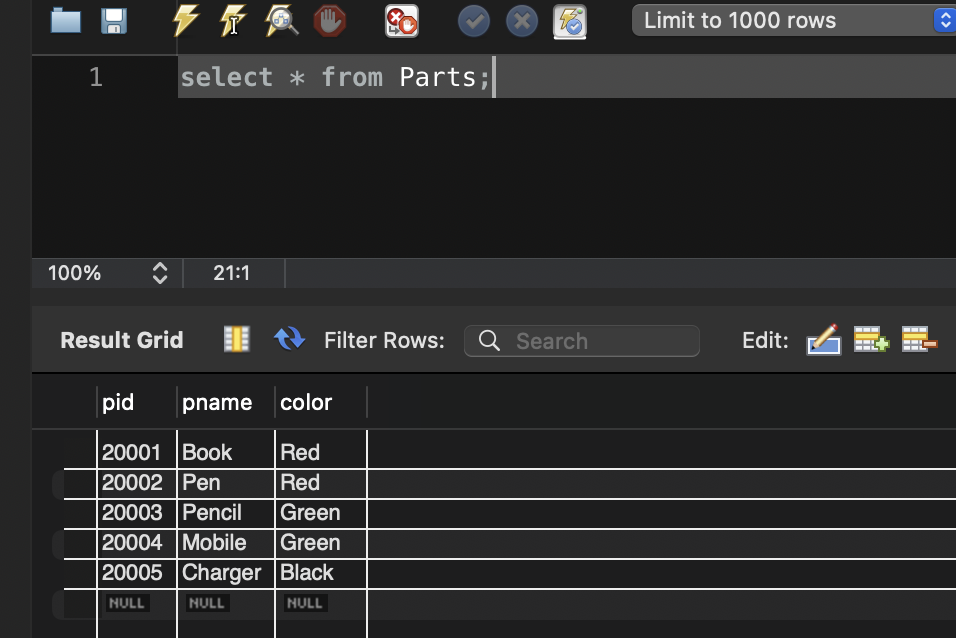
**8. For each part, find the sname of the supplier who charges the most for**

**that part.**

**Creating Database , tables and inserting values**

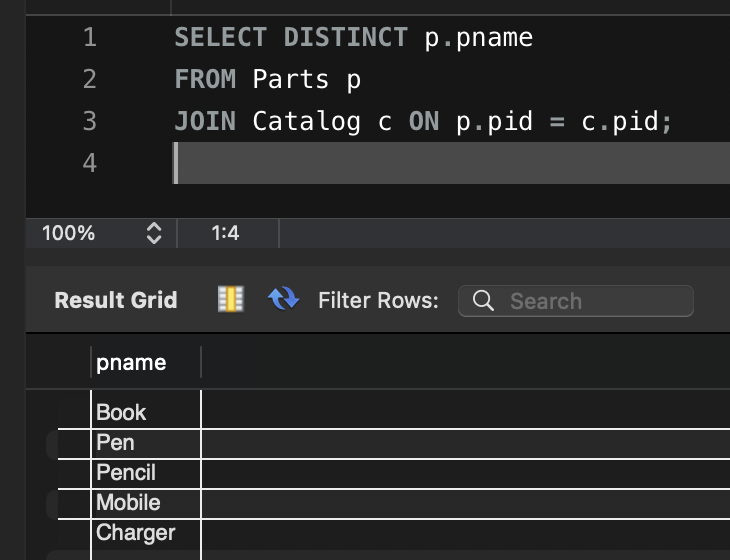
****

****

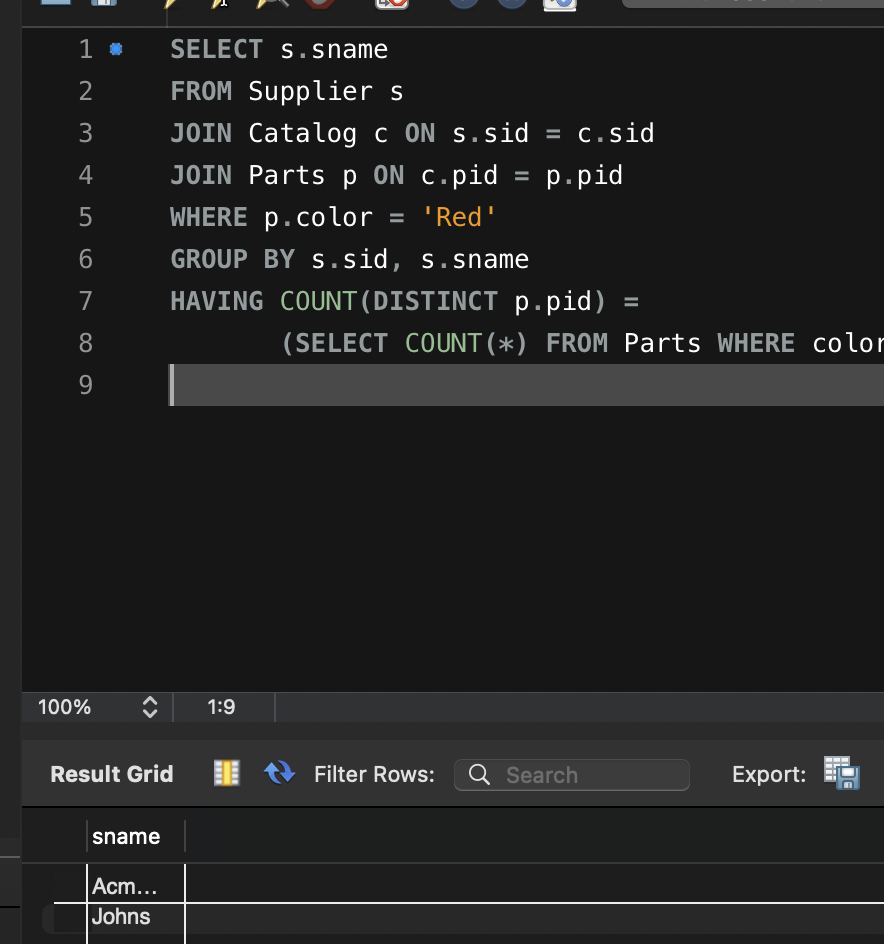
****

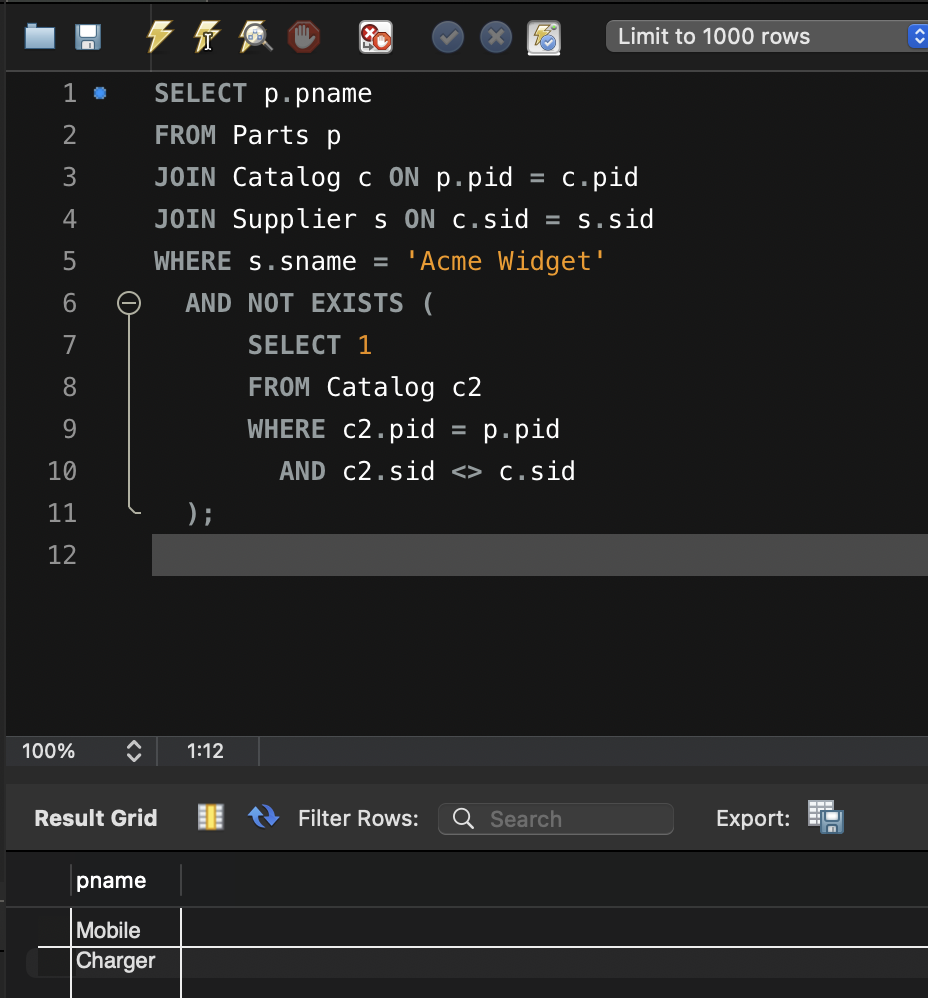
****

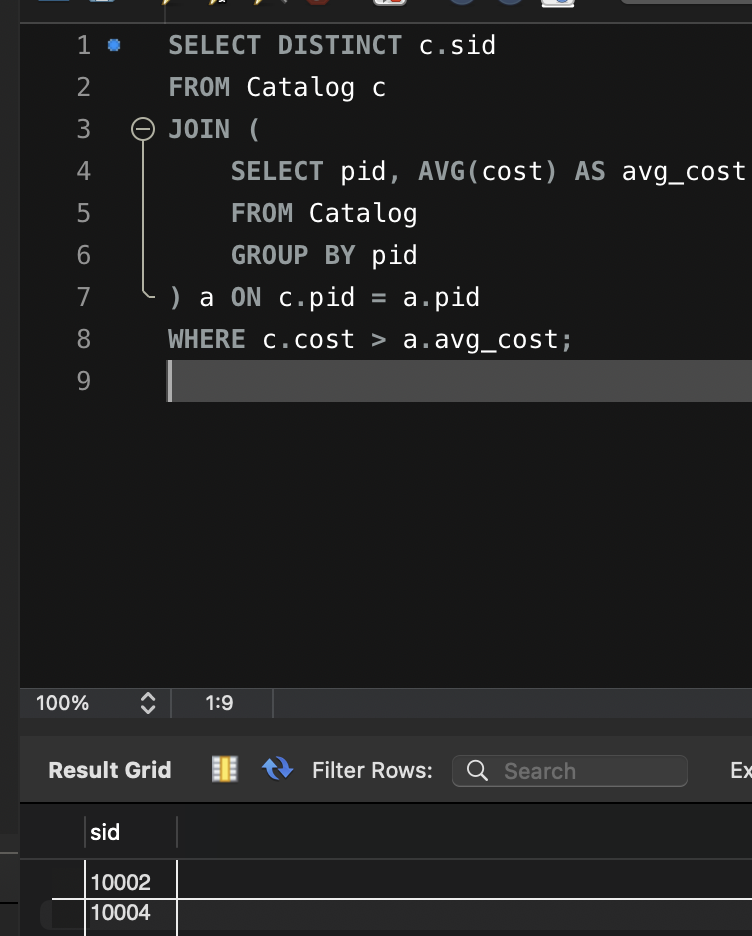
**Main Queries**

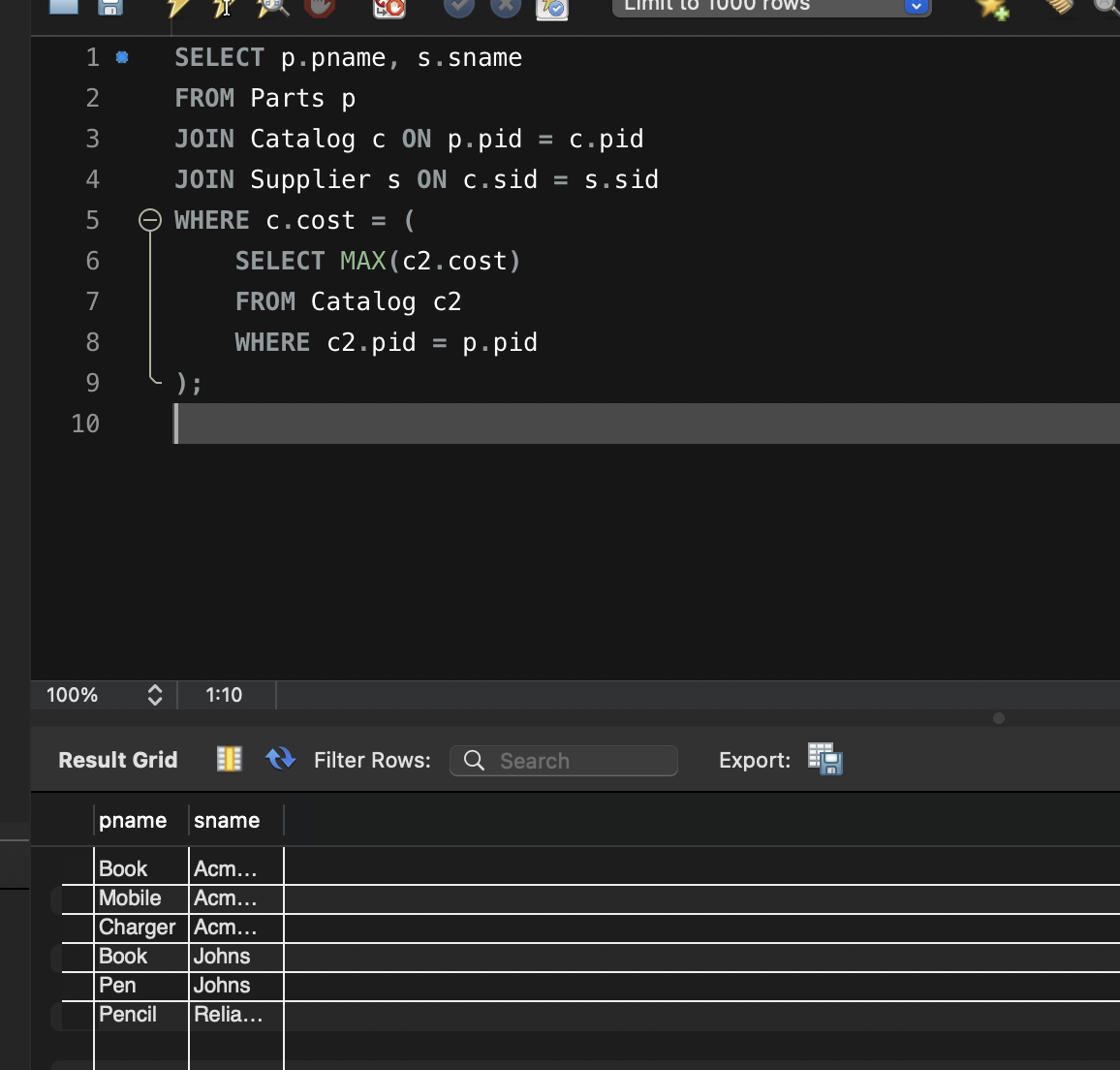
****

****

****

****

****

****

**STUDENT DATABASE-(NoSQL)**

**(WEEK 8)**

**QUERIES**

- Create a database “Student” with the following attributes Rollno, Age, ContactNo, EmailId.

use student



- Insert appropriate values

db.students.insertMany([

{ Rollno: 1, Age: 20, ContactNo: "9876543211", EmailId: "s1@gmail.com", Name: "AAA" },

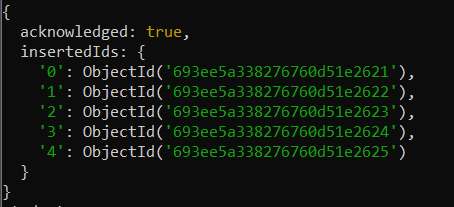
{ Rollno: 2, Age: 21, ContactNo: "9876543212", EmailId: "s2@gmail.com", Name: "BBB" },

{ Rollno: 3, Age: 22, ContactNo: "9876543213", EmailId: "s3@gmail.com", Name: "CCC" },

{ Rollno: 10, Age: 20, ContactNo: "9876543214", EmailId: "old10@gmail.com", Name: "DDD" },

{ Rollno: 11, Age: 21, ContactNo: "9876543215", EmailId: "abc@gmail.com", Name: "ABC" }

])



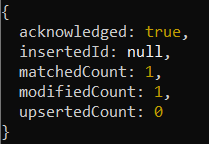
- Write query to update Email-Id of a student with rollno 10

db.students.updateOne({ Rollno: 10 }, { $set: { EmailId: "new10@gmail.com" } })



- Replace the student name from “ABC” to “FEM” of rollno 11

db.students.updateOne({ Rollno: 11 }, { $set: { Name: "FEM" } })



- Export the created table into local file system

exit

mongoexport --db student --collection students --out students.json



- Drop the table

mongosh

use student

db.students.drop()



- Import a given csv dataset from local file system into mongodb collection.

exit

mongoimport --db student --collection students --type csv --headerline --file students.csv



**NoSQL CUSTOMER DATABASE**

**WEEK(9)**

**QUERIES**

**- Create a collection by name Customers with the following attributes. Cust\_id,** Acc\_Bal, Acc\_Type

use Bank



**- Insert at least 5 values into the table.**

db.Customers.insertMany([

{ Cust\_id: 1, Acc\_Bal: 500, Acc\_Type: "Z" },

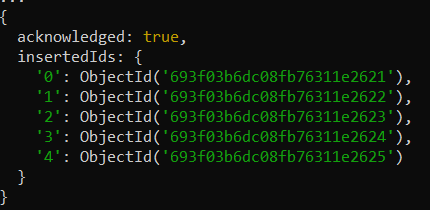
{ Cust\_id: 1, Acc\_Bal: 800, Acc\_Type: "Z" },

{ Cust\_id: 2, Acc\_Bal: 1500, Acc\_Type: "Z" },

{ Cust\_id: 3, Acc\_Bal: 400, Acc\_Type: "A" },

{ Cust\_id: 4, Acc\_Bal: 2000, Acc\_Type: "Z" }

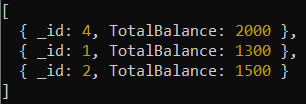
])



**- Write a query to display those records whose total account balance is greater than 1200 of account type ‘Z’ for each customer\_id**

db.Customers.aggregate([{ $match: { Acc\_Type: "Z" } }, { $group: { \_id: "$Cust\_id",

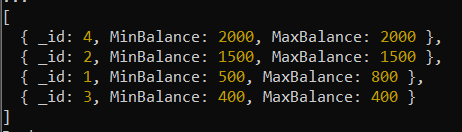
TotalBalance: { $sum: "$Acc\_Bal" }}}, { $match: { TotalBalance: { $gt: 1200 } } }])



**- Determine Minimum and Maximum account balance for each customer\_id.**

db.Customers.aggregate([{ $group: { \_id: "$Cust\_id", MinBalance: { $min: "$Acc\_Bal" },

MaxBalance: { $max: "$Acc\_Bal" }}}])



**- Export the created collection into local file system.**

exit

mongoexport --db Bank --collection Customers --out customers.json



**- Drop the table.**

use Bank

db.Customers.drop()



**- Import a given csv dataset from local file system into mongodb collection**

mongoimport --db Bank --collection Customers --type csv --headerline --file customers.csv



**RESTAURANT DATABASE – (MONGODB)**

**WEEK(10)**

**Perform the following DB operations using MongoDB.**

**1.Write NoSQL Queries on “Restaurant” collection. Write a MongoDB query to display all the documents in the collection restaurants.**

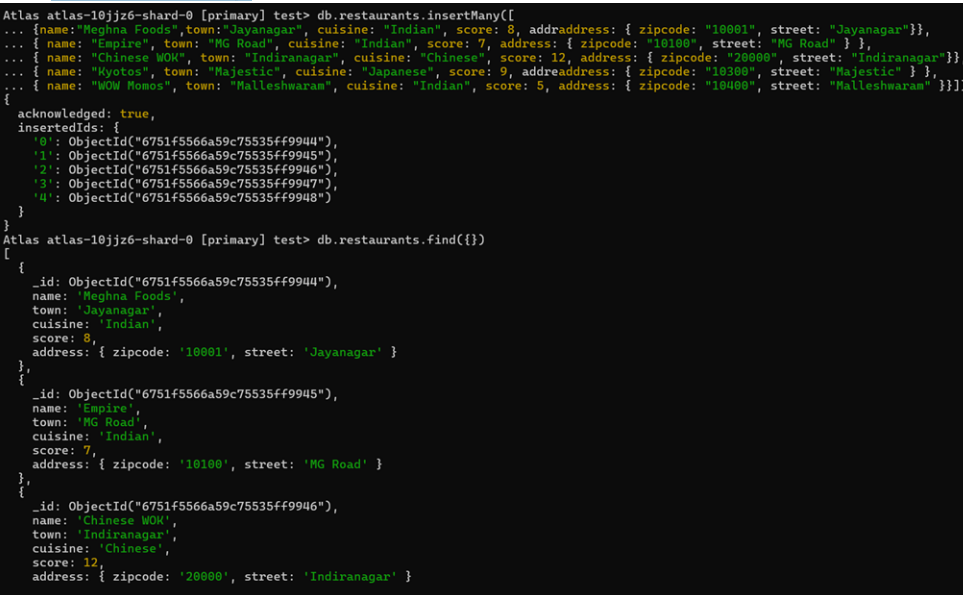
**Create database**

db.createCollection("Restaurant");

* **Inserting Values**

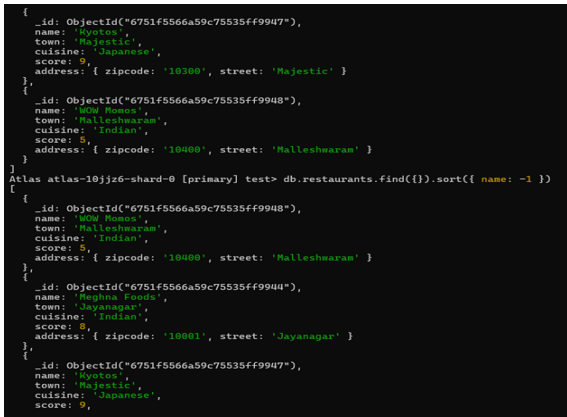
db.Restaurant.insertMany([ {name:"Meghna Foods",town:"Jayanagar", cuisine: "Indian", score: 8, address: { zipcode: "10001", street: "Jayanagar"}}, { name: "Empire", town: "MG Road", cuisine: "Indian", score: 7, address: { zipcode: "10100", street: "MG Road" } }, { name: "Chinese WOK", town: "Indiranagar", cuisine: "Chinese", score: 12, address: { zipcode: "20000", street: "Indiranagar"}}, { name: "Kyotos", town: "Majestic", cuisine: "Japanese", score: 9, address: { zipcode: "10300", street: "Majestic" } }, { name: "WOW Momos", town: "Malleshwaram", cuisine: "Indian", score: 5, address: { zipcode: "10400", street: "Malleshwaram" }}])

db.Restaurant.find();



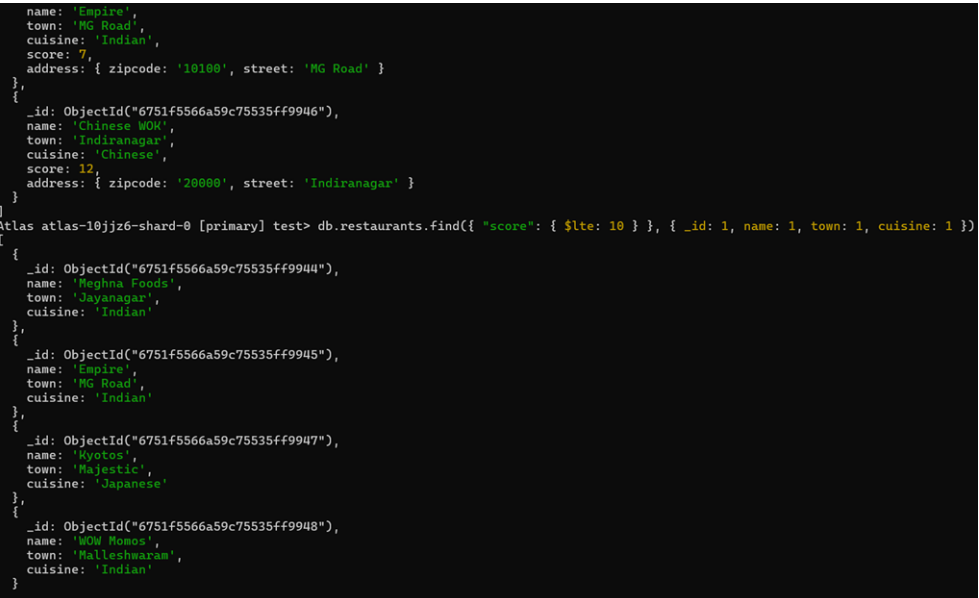
**2.Write a MongoDB query to arrange the name of the restaurants in descending order along with all the columns.**

db.Restaurant.find().sort({ "name": -1 });



**3.Write a MongoDB query to find the restaurant Id, name, town and cuisine for those restaurants which achieved a score which is not more than 10.**

db.Restaurant.find({ "grades.score": { $lte: 10 } },{ \_id: 1, name: 1, town: 1, cuisine: 1, restaurant\_id: 1 });



**4.Write a MongoDB query to find the average score for each restaurant.**

db.restaurants.aggregate([ { $group: { \_id: "$name", average\_score: { $avg: "$score" }}}])



**5.Write a MongoDB query to find the name and address of the restaurants that have a zipcode that starts with '10'.**

db.Restaurant.find( { "address.zipcode": { $regex: "^10" } },{ \_id: 0, name: 1,address: } )

