VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Database Management Systems (23CS3PCDBM)

Submitted by

SUMIT KUMAR MAHATO(1BF24CS305)

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-2025 to Dec-2026

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(AffiliatedToVisvesvarayaTechnologicalUniversity, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Database Management Systems (23CS3PCDBM)" carried out by **Sumit Kumar Mahato(1BF24CS305),** who is Bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2025. 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.

Kanchana Dixit Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

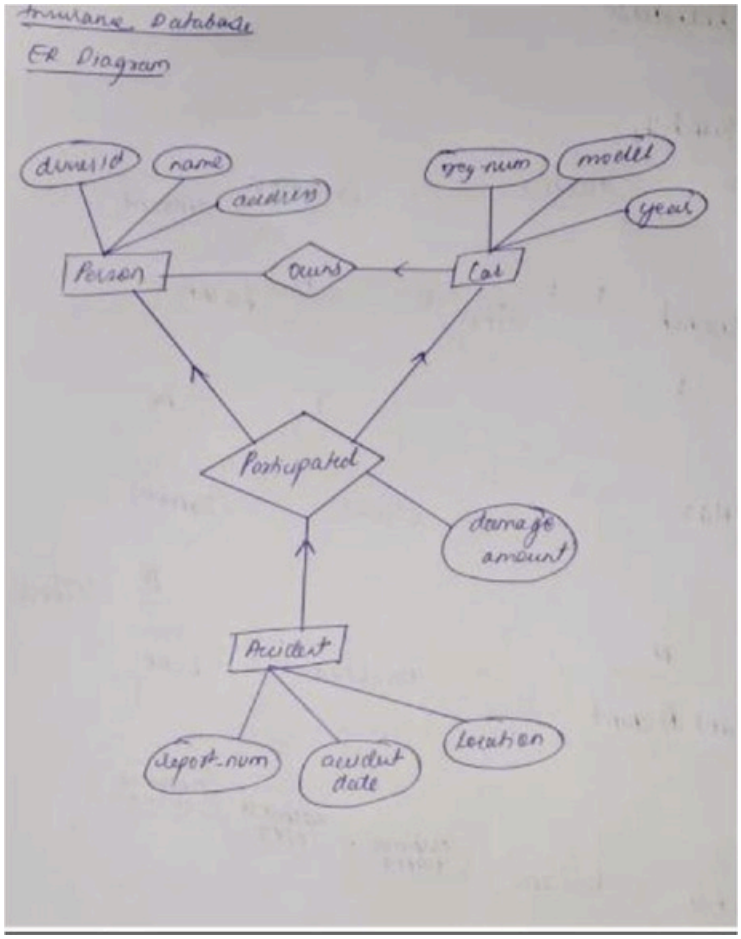
Insurance Database

Experiment 1: Insurance Database

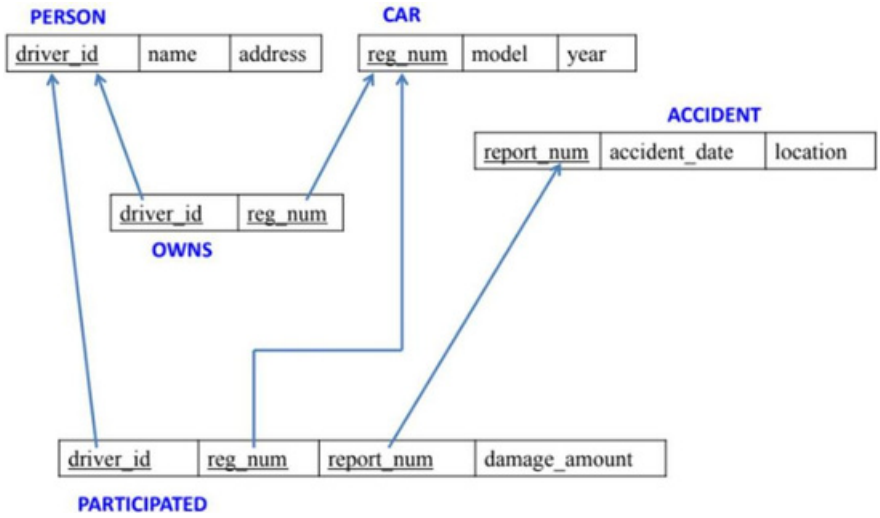
Specification of Insurance Database Application

The insurance database must maintain information about drivers, the cars they own, the accidents reported, and the participation of each driver and car in those accidents. Each driver in the system is uniquely identified by a driver ID, along with their name and address, and each car is
uniquely identified by its registration number together with details such as model and manufacturing year. The system must allow storing ownership information that links a driver to one or more cars, while also allowing a car to be linked to one or more drivers if shared ownership occurs; duplicate ownership
records for the same driver and car must not exist. Accident information must be stored using a un ique report number assigned to each accident, along with the date on which the accident occurred and the
location where it happened. Every accident reported in the system must have at least one participa ting
driver and car, and this participation is recorded by linking the driver, the involved car, and the ac cident
report together with the corresponding damage amount for that particular involvement. A particip ation record must reference an existing driver, an existing car, and an existing accident, and no two
participation entries may repeat the same combination of driver, car, and accident report. The data base must ensure that damage amounts are non-negative, accident dates are valid calendar dates, and car manufacturing years fall within reasonable limits. It must also preserve referential integrity so that ownership or participation entries cannot exist without valid driver, car, and accident information already present in the system. Deletion policies must prevent
removal of drivers or cars that appear in past accident participation records unless historical consistency is preserved through controlled deletion rules or archival mechanisms. The system should maintain accurate links between drivers, cars, and accidents at all times, ensuring reliable retrieval of ownership histories, accident histories, and damage information for administrative, legal, and insurance-related purposes.

Entity Relationship Diagram



Schema Diagram



Create database

- CREATEDATABASE IF NOT EXISTS INSURANCE; SHOW DATABASES;
- USEINSURANCE;

Create table

- CREATE TABLE IF NOT EXISTS PERSON (driver_id VARCHAR
- (10)PRIMARYKEY,nameVARCHAR(50)NOTNULL,addressVARCHAR(100));
- CREATE TABLE IF NOT EXISTS CAR (reg_num VARCHAR(10) PRIMARY KEY, m

  odel VARCHAR(20),year INT);
- CREATE TABLE

  IFNOT EXISTS ACCIDENT(report_numINTPRIMARYKEY,accident_date
- DATE, l ocation VARCHAR(50));

```
CREATE TABLE IF NOT EXISTS OWNS (driver_id VARCHAR(10),reg_num VARCH
AR(10), PRIMARY KEY (driver_id, reg_num),FOREIGN KEY
(driver_id) REFERENCES PERSON(driver_id), FOREIGN KEY
(reg_num) REFERENCES CAR(reg_num));
CREATE TABLE IF NOT EXISTS PARTICIPATED (
driver_id
VARCHAR(10),
reg_num
VARCHAR(10),
report_num INT,
damage_amount INT,
PRIMARY KEY (driver_id, reg_num, report_num),
FOREIGN KEY (driver_id) REFERENCES
PERSON(driver_id), FOREIGN KEY (reg_num) REFERENCES
CAR(reg_num),
FOREIGN KEY (report_num) REFERENCES ACCIDENT(report_num) );
```

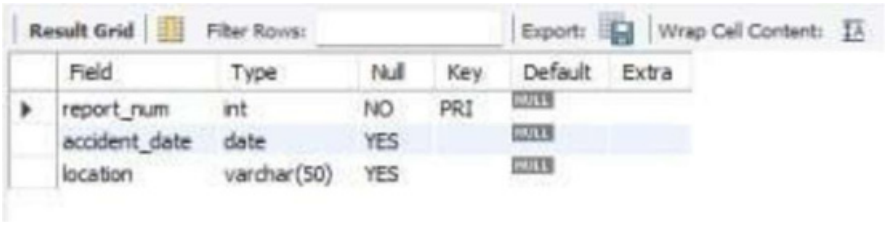
Structure of the table

Desc person;

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
driver_id	varchar(20)	NO	PRI	NULL	
reg_num	varchar(10)	NO	PRI	NULL	
report_num	int	NO	PRI	NULL	
damage_amount	int	YES		NULL	

desc accident;



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
report_num	int	NO	PRI	NULL	
accident_date	date	YES		NULL	
location	varchar(50)	YES		NULL	

desc participated;

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

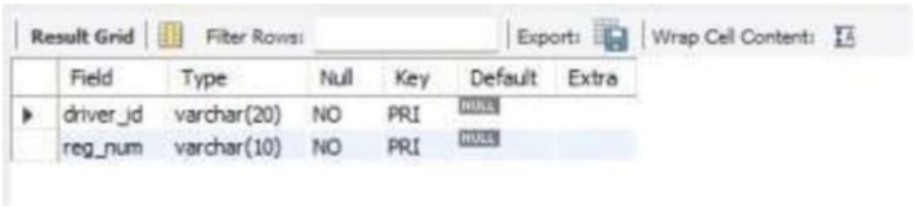
Field	Type	Null	Key	Default	Extra
driver_id	varchar(20)	NO	PRI	NULL	
reg_num	varchar(10)	NO	PRI	NULL	
report_num	int	NO	PRI	NULL	
damage_amount	int	YES		NULL	

desc car;

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
reg_num	varchar(15)	NO	PRI	NULL	
model	varchar(10)	YES		NULL	
year	int	YES		NULL	

desc owns;

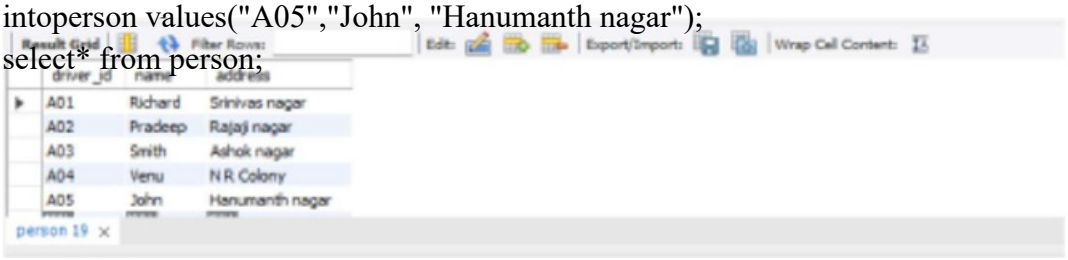


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
driver_id	varchar(20)	NO	PRI	NULL	
reg_num	varchar(10)	NO	PRI	NULL	

Inserting Values to the table

insert intoperson values("A01","Richard", "Srinivas nagar"); insert into person values("A02","Pradeep", "Rajaji nagar"); insert into person values("A03","Smith", "Ashok nagar"); insert into person values("A04","Venu", "N R Colony"); insert intoperson values("A05","John", "Hanumanth nagar"); select* from person;



driver_id	name	address
A01	Richard	Srinivas nagar
A02	Pradeep	Rajaji nagar
A03	Smith	Ashok nagar
A04	Venu	N R Colony
A05	John	Hanumanth nagar

person 19

insert into car values("KA052250","Indica", "1990"); insert into carvalues("KA031181","Lancer", "1957"); insert into car values("KA095477","Toyota", "1998"); insert into car values("KA053408","Honda", "2008"); insert into car values("KA041702","Audi", "2005"); Select * from car;

reg_num	model	year
KA031181	Lancer	1957
KA041702	Audi	2005
KA052250	Indica	1990
KA053408	Honda	2008
KA095477	Toyota	1998

car 20

insert into owns values("A01","KA052250"); insert into owns values("A02","KA031181"); insert into owns values("A03","KA095477"); insert into owns values("A04","KA053408"); insert into owns values("A05","KA041702"); select * from owns;

insert into accident values(11,'2003-01-01',"Mysore Road");
insert into accident values(12,'2004-02-02',"South end
Circle"); insert into accident values(13,'2003-01-21',"Bull
temple Road"); insert into accident values(14,'2008-02-
17',"Mysore Road"); insert into accident values(15,'2004-
03-05',"Kanakpura Road"); select * from accident;

Queries :-

Display Accident date and location

```
66
67
68 •    select accident_date,location from accident;
69
70
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

accident_date	location
2003-01-01	Mysore road
2004-02-02	South end Circle
2003-01-21	Bull temple Road
2008-02-17	Mysore road
2005-03-04	Kanakpura Road

Update the damage amount to 25000 for the car with a specific reg_num (example 'KA053408') for which the accident report number was 12.

```
59
60 •    update participated set damage_amount=25000 where reg_num="KA053408" and report_num=12;
61
62 •    select * from participated;
63
```

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

driver_id	reg_num	report_num	damage_amount
A01	KA052250	11	10000
A02	KA053408	12	25000
A03	KA095477	13	25000
A04	KA031181	14	3000
A05	KA041702	15	5000
NULL	NULL	NULL	NULL

Display Accident date and location

```
45 •    select * from accident;
46
47 •    create table participated(driver_id varchar(10), reg_num varchar(10),
48      report_num int, damage_amount int, primary key(driver_id, reg_num, report_num),
49      foreign key(driver_id) references person(driver_id),
50      foreign key(reg_num) references car(reg_num), foreign key(report_num) references accident(report_num));
51
52 •    insert into participated values("A01","KA052250",11,10000);
53 •    insert into participated values("A02","KA053408",12,50000);
54 •    insert into participated values("A03","KA095477",13,25000);
55 •    insert into participated values("A04","KA031181",14,3000);
```

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

report_num	accident_date	location
11	2003-01-01	Mysore road
12	2004-02-02	South end Circle
13	2003-01-21	Bull temple Road
14	2008-02-17	Mysore road
15	2005-03-04	Kanakpura Road
NULL	NULL	NULL

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

```
69
70
71 •  select driver_id from participated p, accident a where p.report_num=a.report_num and damage_amount>=25000;
72
73
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

driver_id
A02
A03

More Queries on Insurance Database

Queries :-

Display the entire CAR relation in the ascending order of manufacturing year.

```
78
79
80 •  select * from car order by year asc;
81
82
```

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

reg_num	model	year
KA031181	Lancer	1957
KA052250	Indica	1990
KA095477	Toyota	1998
KA041702	Audi	2005
KA053408	Honda	2008
NULL	NULL	NULL

Find the number of accidents in which cars belonging to a specific model (example 'Lancer') were involved

```
82
83 •  select count(model) CNT from car c, participated p, accident a where c.model="Lancer" and c.reg_num=p.reg_num
84    and p.report_num=a.report_num;
85
86
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

CNT
1

Find the Average Damage Amount

```
75
76
77 •  select avg(damage_amount) from participated;
78
79
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

avg(damage_amount)
13600.0000

Delete the tuple whose Damage Amount is below the Average Damage Amount

```
86
87 • delete from participated where damage_amount < (select avg_damage from (select avg(damage_amount) as
88    avg_damage from participated)as t);
89
90 •  select * from participated;
```

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

driver_id	reg_num	report_num	damage_amount
A02	KA053408	12	25000
A03	KA095477	13	25000
NULL	NULL	NULL	NULL

Listthe name of drivers whose Damage is Greater than the Average Damage Amount

```
91
92
93 •  select name from person a, participated b where a.driver_id = b.driver_id and damage_amount >
94    (select avg(damage_amount) from participated);
95
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name
Pradeep
Smith

Find Maximum Damage Amount.

```
92
93 •  select name from person a, participated b where a.driver_id = b.driver_id and damage_amount >
94    (select avg(damage_amount) from participated);
95
96
97 •  select max(damage_amount) from participated;
```

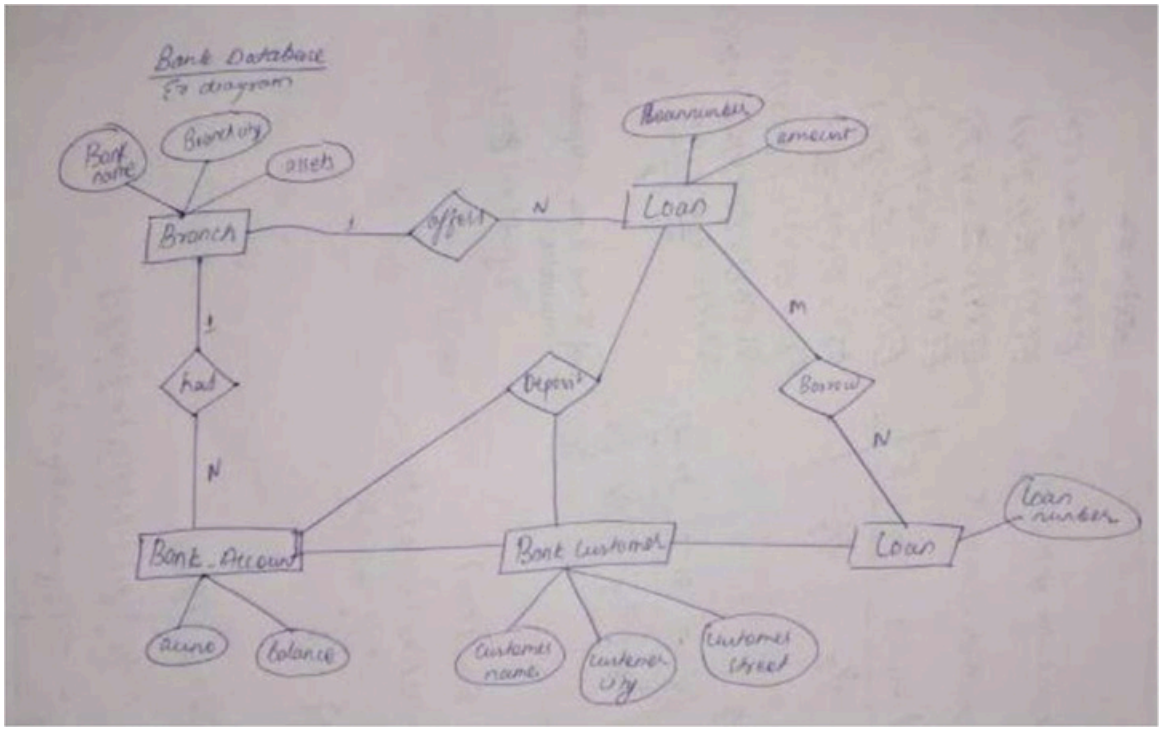
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

max(damage_amount)
25000

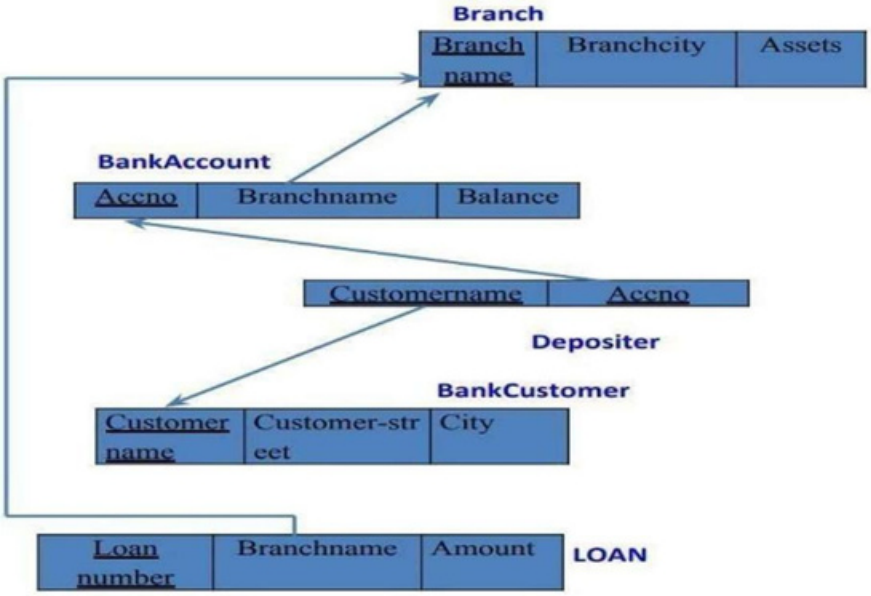
Experiment 2: BANKING DATABASE

The banking system must store information about branches, bank accounts, customers, deposit ,relationships, and loans so that branch details (identified by branch name together with city and total assets) are linked to accounts and loans, each account (identified by an account number) records the branch it belongs to and the current balance, customers are recorded with their name, street , city and a depositor relationship associates a customer with an account; loans are recorded by a unique loan number together with the branch name that issued the loan and the loan amount. Account numbers and loan numbers must be unique identifiers, branch names are used to associate accounts and loans to a branch, and customer names (as model) are used to identify customers referenced by depositor entries; every depositor entry must reference an existing customer and an existing account so that ownership and access relationships are always valid, and duplicate depositor records linking the same customer and account are disallowed. The system must maintain referential integrity so accounts cannot reference a non-existent branch, depositor rows cannot reference missing customers or accounts, and loans must reference an existing branch; deletion of a branch, account, or customer that is referenced by dependent records should be controlled (either disallowed or handled by archival/controlled reassignment) to preserve historical transaction and loan consistency. Numeric and temporal constraints must be enforced: account balances should be constrained to valid values (for example non-negative where overdraft is not allowed), branch assets and loan amounts must be non- negative and within specified business limits, and updates to balance or loan amounts should be auditable. Cardinality rules implied by the schema are enforced: a branch may host many accounts and issue many loans, an account belongs to exactly one branch, a customer may be linked to many accounts through depositor relationships, and an account may have many depositors if joint accounts are permitted by policy. Implementation must prevent orphaned records, ensure uniqueness where required, and rely on application logic or database-level triggers to enforce complex rules such as cascading effects on deletion, business rules about allowed balance operations or overdrafts, and any required validation when transferring accounts between branches or when converting a customer's identifying details; the database should thus reliably support queries for branch-wise account lists, customer account ownership, account balances, and loan portfolios while preserving historical and referential integrity for auditing and regulatory reporting.

Entity Relationship Diagram



Schema Diagram



Create Database

create database

show bank_database;

usebank_database;

Create Table

createtable branch(branch_name varchar(50), branch_city varchar(50), assests real, Primary key(branch_name)

);

create table bankAccount(

accno int,

branch_name varchar(50),

balance real, primary key

(accno), Foreign key(branch_name) references branch(branch_name)

);

create table bankCustomer(customer_name varchar(50), customer_street varchar(50), customer_city varchar(50) , primary key (customer_name)

);

create table depositor(customer_name varchar(50), accno int, primary key (customer_name,accno), foreign key(customer_name) references bankCustomer(customer_name), foreign key(accno) references bankAccount(accno)

);

desc depositor; create table loan(

loan_number int,

branch_name varchar(50),

amount real,

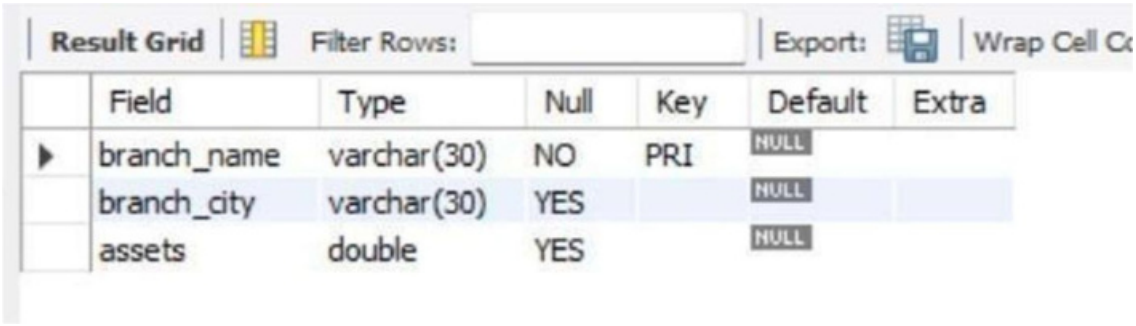
primary key(loan_number),

foreign key(branch_name) references branch(branch_name)

);

Structure of the table

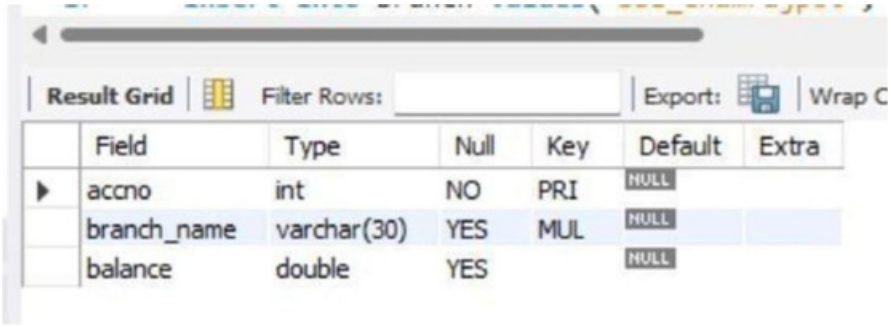
Desc branch;



Result Grid | Filter Rows: | Export: | Wrap Cell Co

	Field	Type	Null	Key	Default	Extra
▶	branch_name	varchar(30)	NO	PRI	NULL	
	branch_city	varchar(30)	YES		NULL	
	assets	double	YES		NULL	

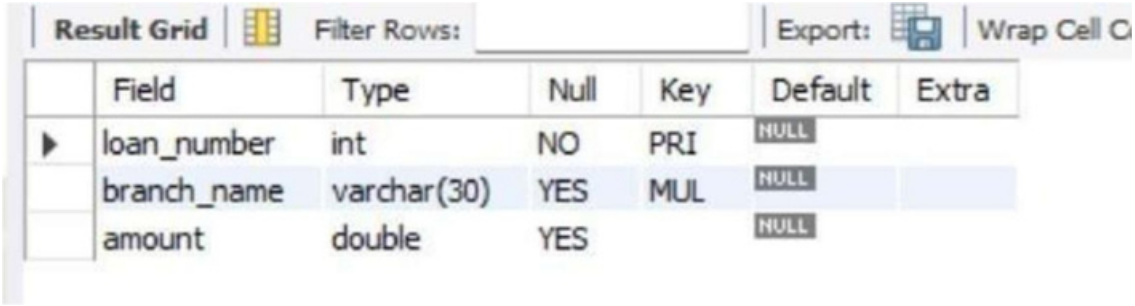
Desc bankaccount;



Result Grid | Filter Rows: | Export: | Wrap C

	Field	Type	Null	Key	Default	Extra
▶	accno	int	NO	PRI	NULL	
	branch_name	varchar(30)	YES	MUL	NULL	
	balance	double	YES		NULL	

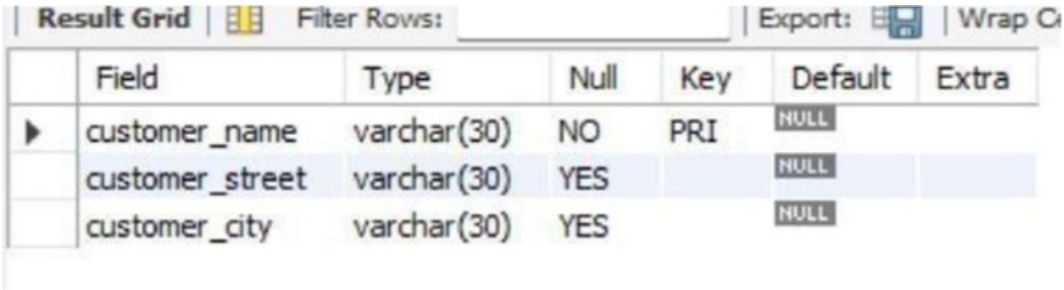
Desc loan;



Result Grid | Filter Rows: | Export: | Wrap Cell C

	Field	Type	Null	Key	Default	Extra
▶	loan_number	int	NO	PRI	NULL	
	branch_name	varchar(30)	YES	MUL	NULL	
	amount	double	YES		NULL	

Desc bankcustomer;



Result Grid | Filter Rows: | Export: | Wrap C

	Field	Type	Null	Key	Default	Extra
▶	customer_name	varchar(30)	NO	PRI	NULL	
	customer_street	varchar(30)	YES		NULL	
	customer_city	varchar(30)	YES		NULL	

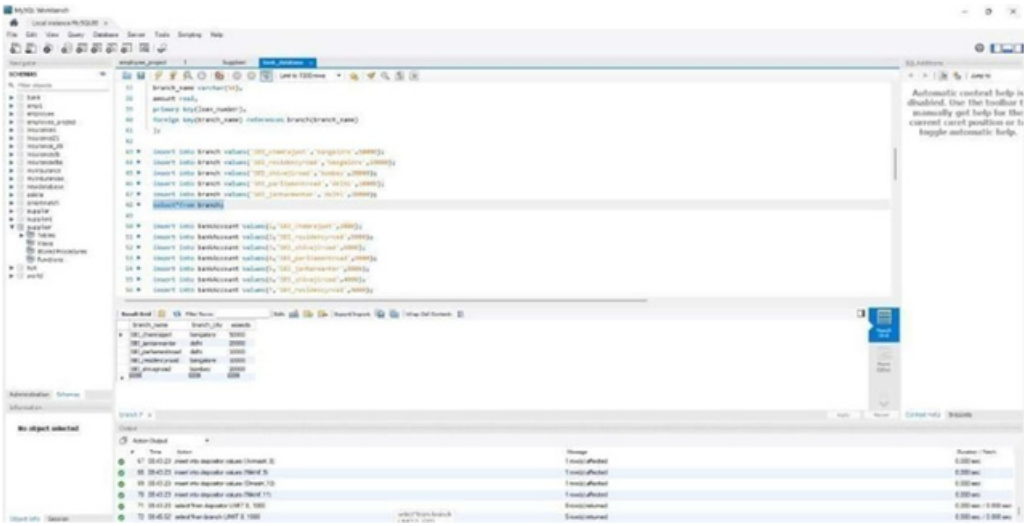
Desc depositer;

Result Grid | Filter Rows: | Export: | Wrap Cell Cor

Field	Type	Null	Key	Default	Extra
customer_name	varchar(30)	NO	PRI	NULL	
accno	int	NO	PRI	NULL	

Insertion of values into tables

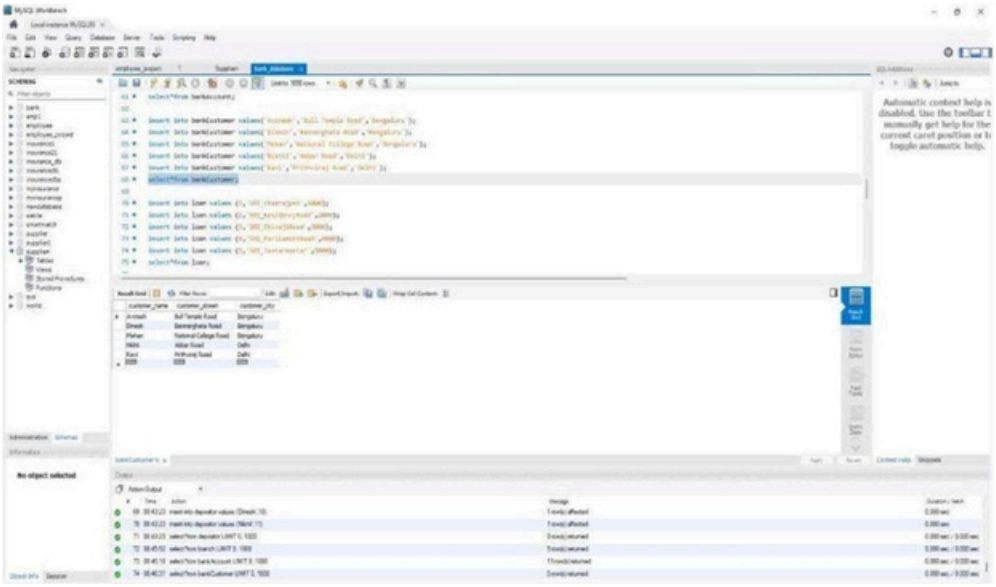
insert intobranch
values('SBI_chamrajpet','bangalore',50000); insert into
branch values('SBI_residencyroad','bangalore',10000);
insert into branch
values('SBI_shivajiroad','bombay',20000); insert into
branch values('SBI_parliamentroad','delhi',10000); insert
into branch values('SBI_jantarmantar','delhi',20000);
select*from branch;



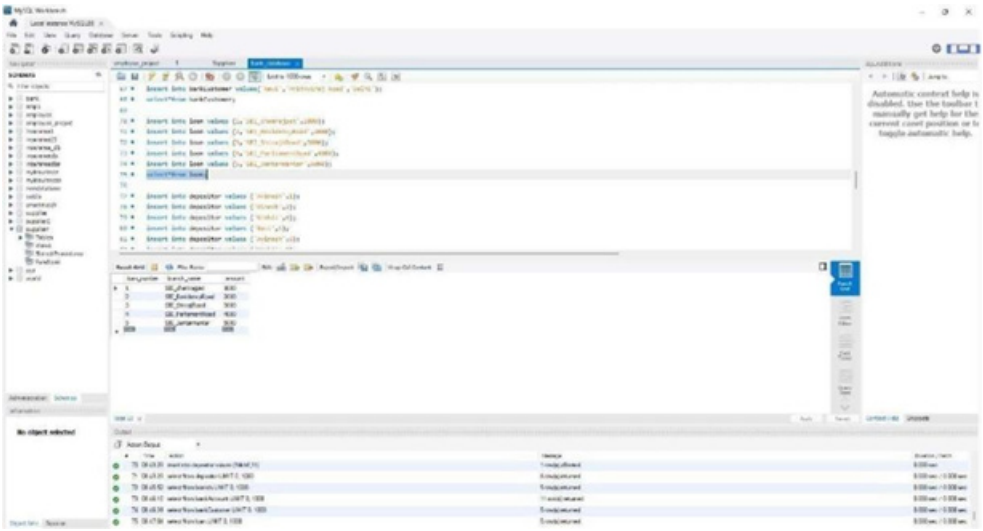
insert into bankAccount values(1,'SBI_chamrajpet',2000);
insert into bankAccount
values(2,'SBI_residencyroad',5000); insert into
bankAccount values(3,'SBI_shivajiroad',6000); insert into
bankAccount values(4,'SBI_parliamentroad',9000); insert
into bankAccount values(5,'SBI_jantarmantar',8000);
insert into bankAccount values(6,'SBI_shivajiroad',4000);
insert into bankAccount
values(7,'SBI_residencyroad',4000); insert into
bankAccount values(8,'SBI_parliamentroad',3000);

insert into bankAccount
values(9,'SBI_residencyroad',5000); insert into
bankAccount values(10,'SBI_jantarmantar',2000); insert
into bankAccount values(11,'SBI_jantarmantar',2000);

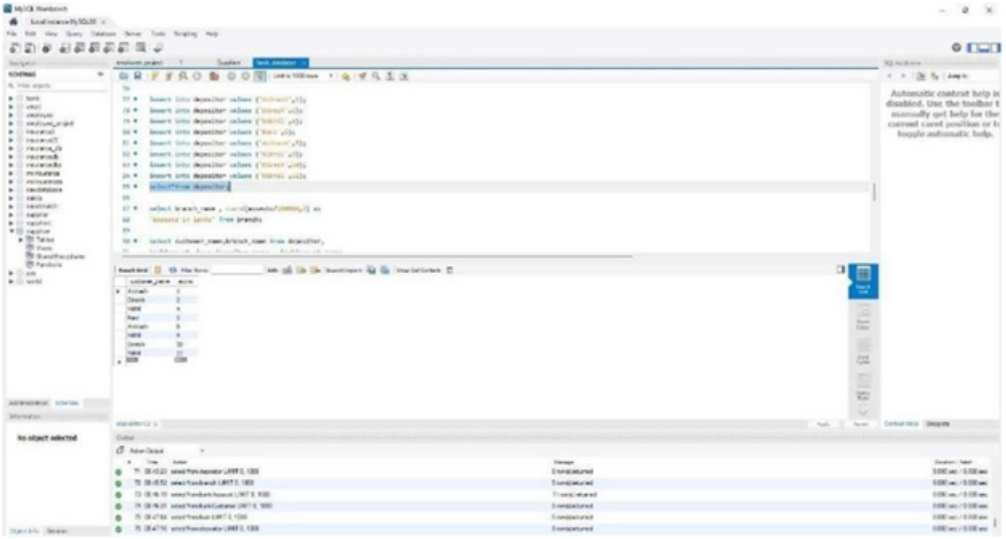
insert into bankCustomer values('Avinash','Bull Temple
Road','Bengaluru'); insert into bankCustomer
values('Dinesh','Bannerghata Road','Bengaluru'); insert into
bankCustomer values('Mohan','National College Road','Bengaluru');
insert into bankCustomer values('Nikhil','Akbar Road','Delhi');
insert into bankCustomer values('Ravi','Prithviraj Road','Delhi'); select*from bankCustomer;



insert into loan values (1,'SBI_chamrajpet',1000);
insert into loan values
(2,'SBI_ResidencyRoad',2000); insert into loan
values (3,'SBI_ShivajiRoad',3000); insert into
loan values (4,'SBI_ParliamentRoad',4000);
insert into loan values (5,'SBI_Jantarmantar',5000); select*from loan;

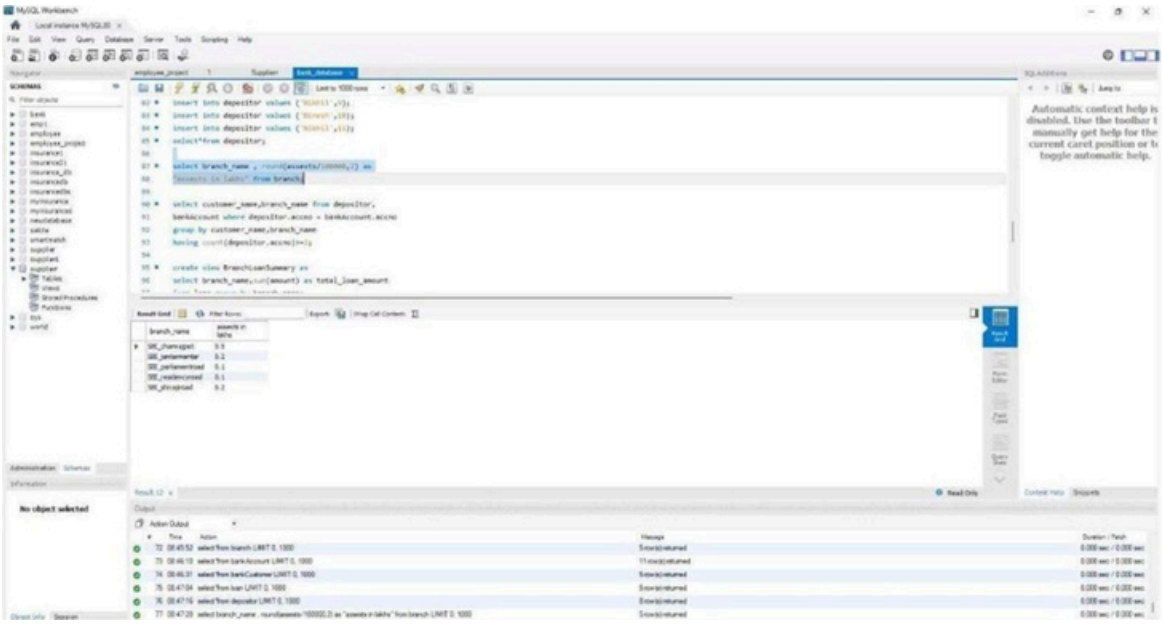


insert into depositor values
('Avinash',1); insert into depositor
values ('Dinesh',2); insert into depositor
values ('Nikhil',4); insert into depositor
values ('Ravi',5);
insert into depositor values ('Avinash',8);
insert into depositor values ('Nikhil',9);
insert into depositor values ('Dinesh',10);
insert into depositor values ('Nikhil',11); Select *
from depositor

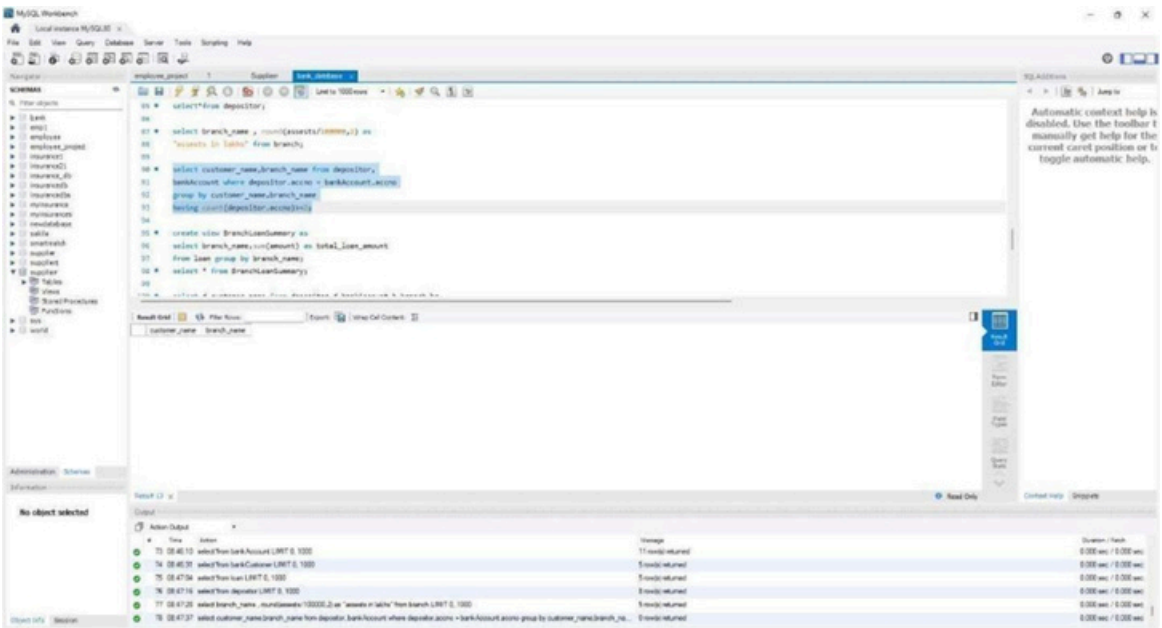


Queries :-

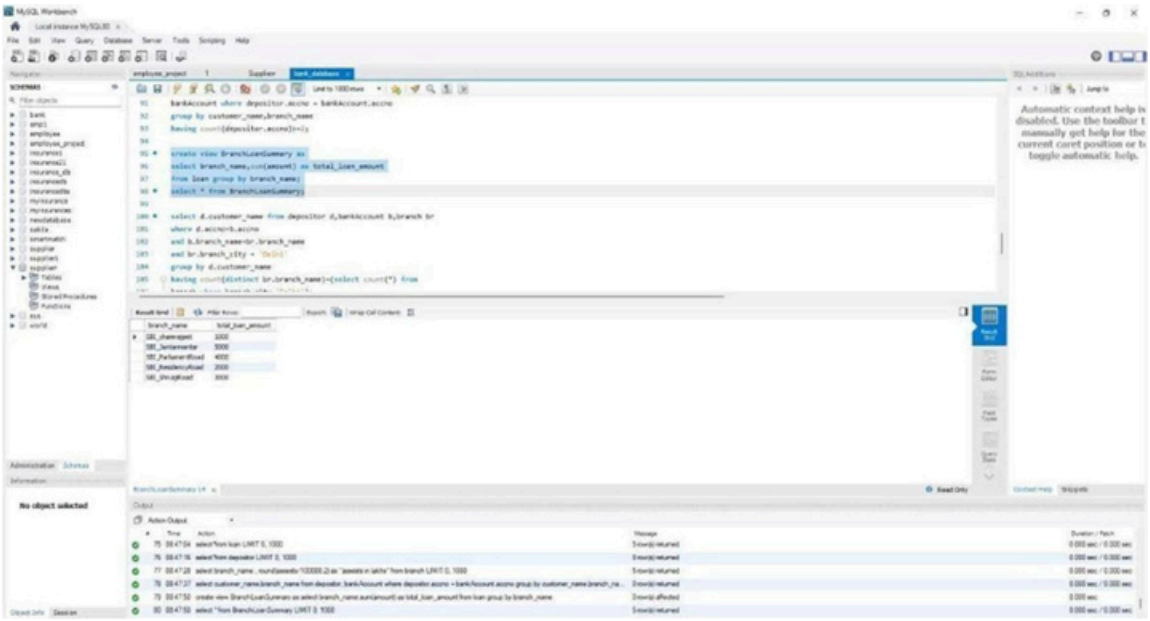
Display the branch name and assets from all branches in lakhs of rupees and rename the assets column to 'assets in lakhs'.



Find all the customers who have at least two accounts at the same branch (ex. SBI_ResidencyRoad).

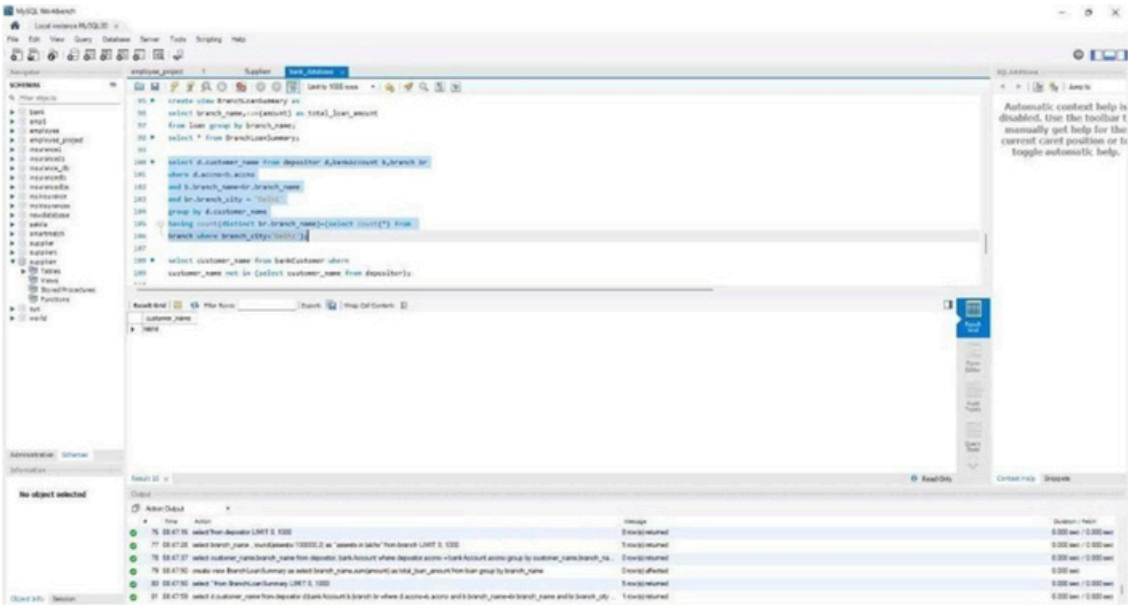


Create a view which gives each branch the sum of the amount of all the loans at the branch.

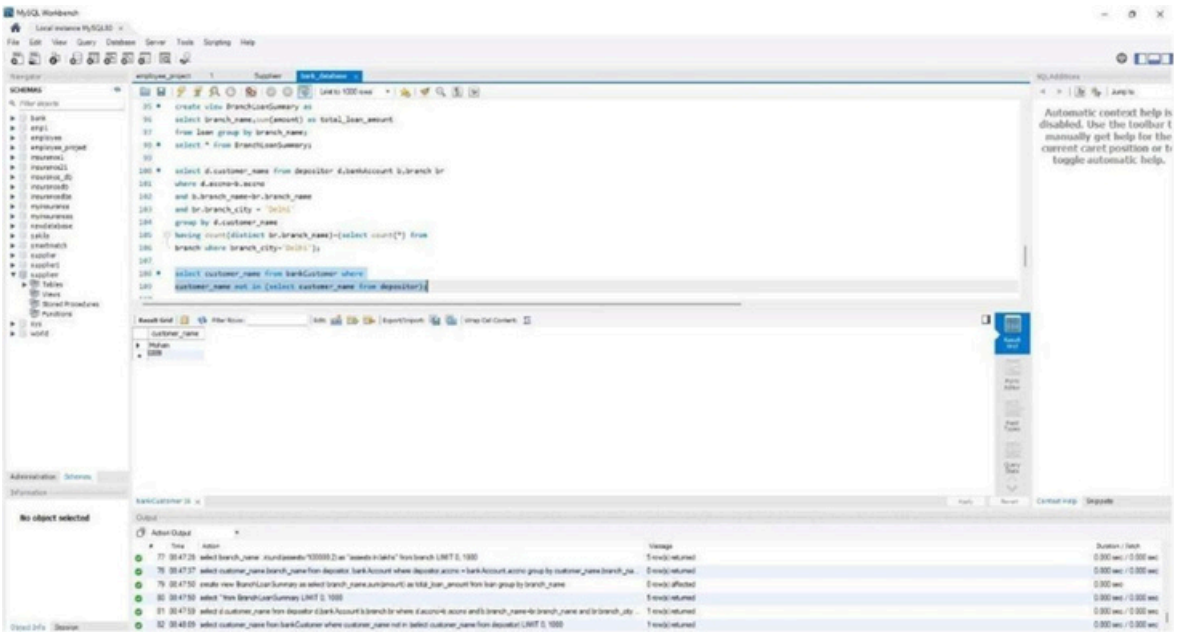


Experiment 3: More Queries on Bank Database

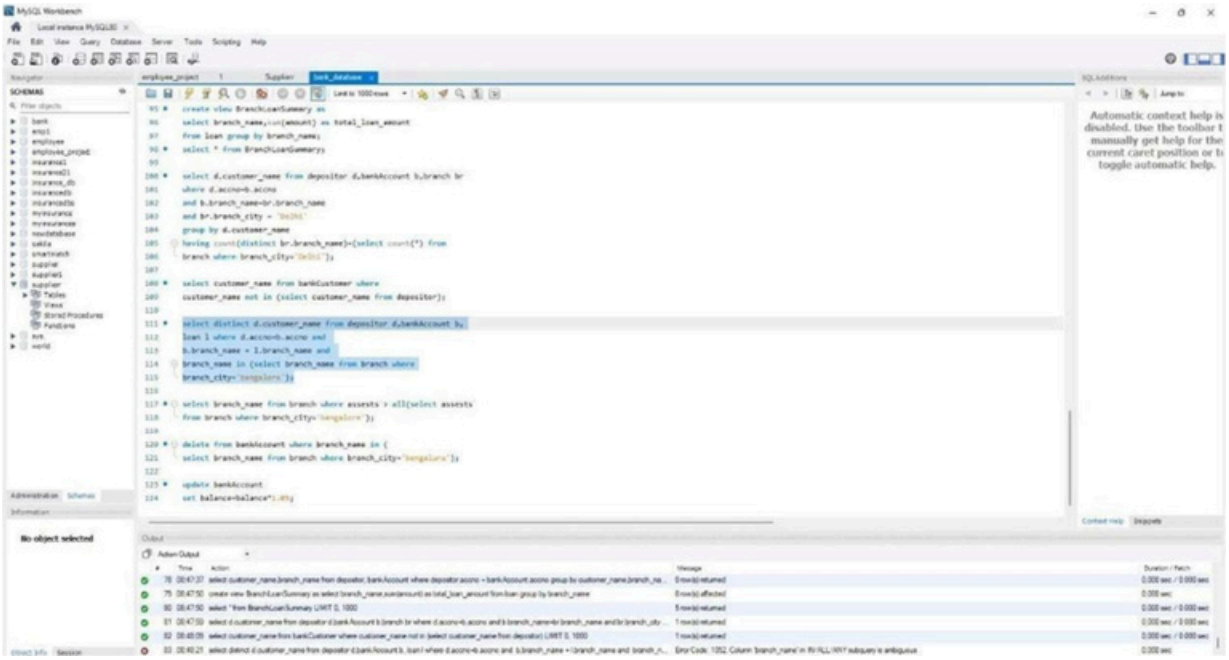
Find all the customers who have an account at all the branches located in a specific city (Ex. Delhi).



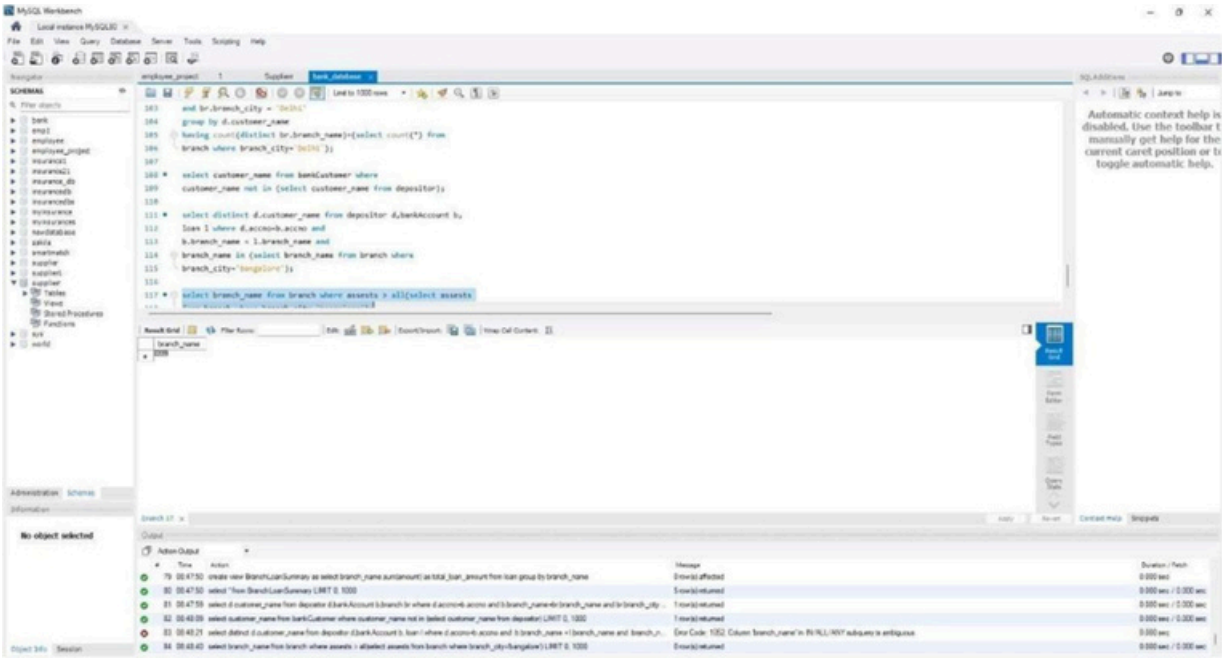
Find all customers who have a loan at the bank but do not have an account.



Find all customers who have both an account and a loan at the Bangalore branch.



Demonstrate how you delete all account tuples at every branch located in a specific city (Ex. Bombay).

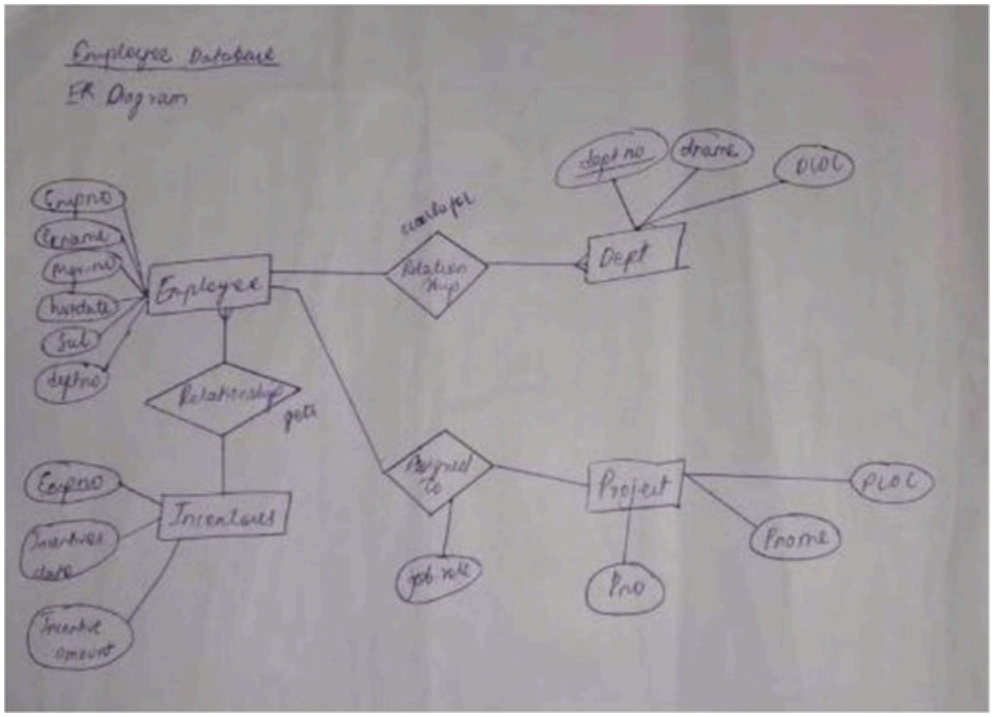


Experiment 5: Employee Database

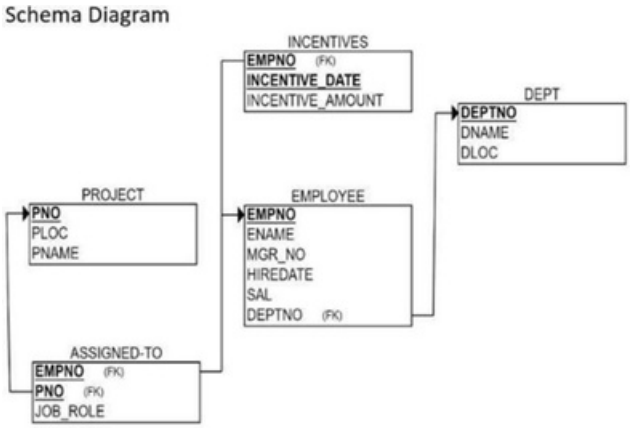
The employee database must record each employee's identifying number, name, manager reference, hire date, salary, and department affiliation while also tracking departmental details, project
assignments (including the role an employee plays on a project), and any incentive payments given to employees. Every employee is represented by a unique employee number and has a hire date and
salary that must be valid; the manager field is a self-referencing link that must, if present, point to an existing employee and must never create a circular management chain or reference the employee
themself. Departments are identified by a unique department number and include a department name and location; every department referenced by an employee or by other structures must exist in the
department table, and departments may contain zero or many employees. Projects are recorded with a unique project number, project name and project location; employees may be assigned to multiple projects and each project may have many employees, with each assignment carrying the employee's job role for that project — duplicate assignments of the same employee to the same project are
disallowed. Incentive payments are recorded with the employee reference, the incentive date and the incentive amount; an incentive entry must reference an existing employee and incentive amounts must be non-negative and dated on or after the employee's hire date. Referential integrity must be enforced so that employee records cannot reference non-existent departments, projects, or managers, and
assignment and incentive records cannot exist without corresponding employee, project, or department records as appropriate. Salary, incentive amounts, and any monetary fields must be constrained to valid numeric ranges and hire/ incentive dates must be valid calendar dates (and typically not future-dated unless business rules permit). Deletion and update policies must preserve historical consistency: deleting an employee who appears as a manager, as a project assignee, or in
incentive records should be prevented or should be handled via controlled archival, reassignment, or soft-delete flags rather than hard deletion to preserve audit trails; similarly, changing a department or project identifier must either be disallowed if it would orphan historical records or handled by
introducing immutable surrogate keys. Business rules include preventing circular manager chains, ensuring an employee's manager (if specified) cannot be the employee themself, disallowing duplicate project-assignments, requiring that incentive dates fall within the employee's employment window, and optionally requiring at least one project assignment or at least one incentive record depending on policy for reporting. Implementation should use primary-key and foreign-key constraints for identity and linkage, unique constraints to

prevent duplicate assignments, check constraints for monetary and date ranges, and application logic or triggers for complex temporal or graph constraints (like cycle detection in management relationships and enforcing non-overlap or other schedule-related rules if assignments gain temporal attributes later). The system must therefore reliably support queries such as employee reporting lines, department staffing lists, project rosters with job roles, incentive payment histories, salary analyses, and audit reports while maintaining data integrity, preventing inconsistent deletions, and preserving a complete historical record for HR and compliance needs.

Entity Relationship Diagram



Schema Diagram



Create Database

```
create database Employee_project;
show databases;
useemployee_project;
```

Create table

```
CREATE TABLE DEPT (
DEPTNO INT PRIMARY KEY,
DNAME VARCHAR(30) NOT NULL,
DLOC VARCHAR(30) NOT NULL
);

descdept;

CREATE TABLE EMPLOYEE (
EMPNO INT PRIMARY KEY,
ENAME VARCHAR(30) NOT NULL,
MGR_NO INT,
HIREDATE DATE,
SAL DECIMAL(10,2),
DEPTNO INT,
FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)
);
descemployee;

CREATE TABLE PROJECT (
PNO INT PRIMARY KEY,
PLOC VARCHAR(30) NOT NULL,
PNAME VARCHAR(30) NOT NULL
);

descproject;

CREATE TABLE ASSIGNED_TO (
EMPNO INT,
PNO INT,
JOB_ROLE VARCHAR(30),
PRIMARY KEY (EMPNO, PNO),
FOREIGN KEY (EMPNO) REFERENCES EMPLOYEE(EMPNO),
FOREIGN KEY (PNO) REFERENCES PROJECT(PNO)
);
desc assigned_to;
```

```
CREATE TABLE INCENTIVES (
EMPNO INT,
INCENTIVE_DATE DATE,
INCENTIVE_AMOUNT DECIMAL(10,2),
PRIMARY KEY (EMPNO, INCENTIVE_DATE),
FOREIGN KEY (EMPNO) REFERENCES EMPLOYEE(EMPNO) );
```

Structure of the table desc dept;

desc employeee;

Field	Type	Null	Key	Default	Extra
empno	int	NO	PRI	NULL	
ename	varchar(30)	YES		NULL	
mgr_no	int	YES		NULL	
hiredate	date	YES		NULL	
sal	decimal(10,2)	YES		NULL	
deptno	int	YES	MUL	NULL	

desc project;

Field	Type	Null	Key	Default	Extra
pno	int	NO	PRI	NULL	
pname	varchar(30)	YES		NULL	
dloc	varchar(30)	YES		NULL	

desc assigned;

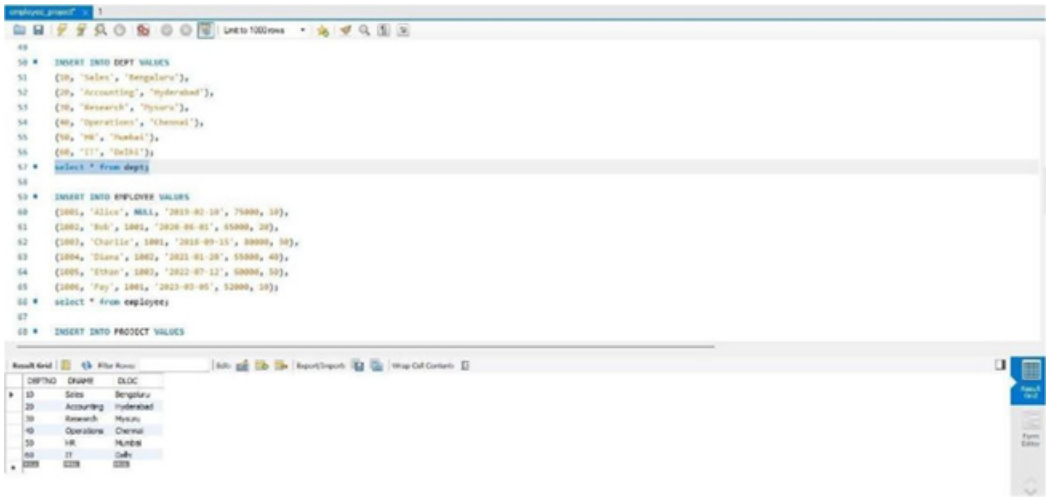
Field	Type	Null	Key	Default	Extra
empno	int	NO	PRI	NULL	
pno	int	NO	PRI	NULL	
job_role	varchar(30)	YES		NULL	

desc incentives;

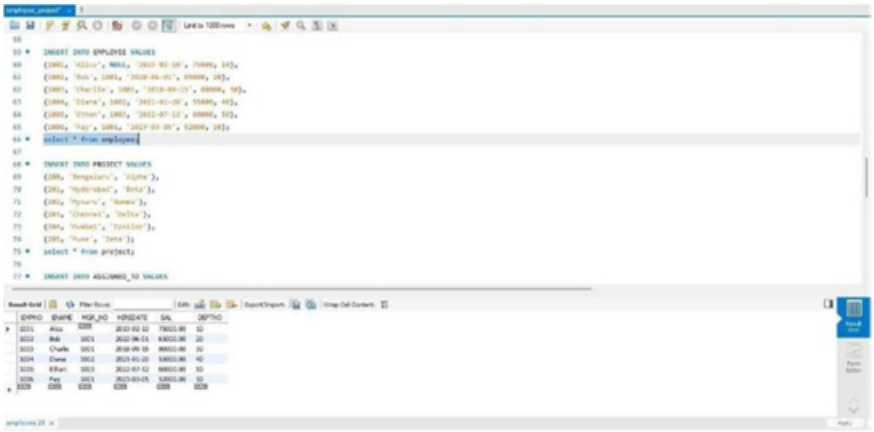
Field	Type	Null	Key	Default	Extra
empno	int	YES	MUL	NULL	
incentives_date	date	YES		NULL	
incentives_amount	decimal(10,2)	YES		NULL	

Insertion of values into Table

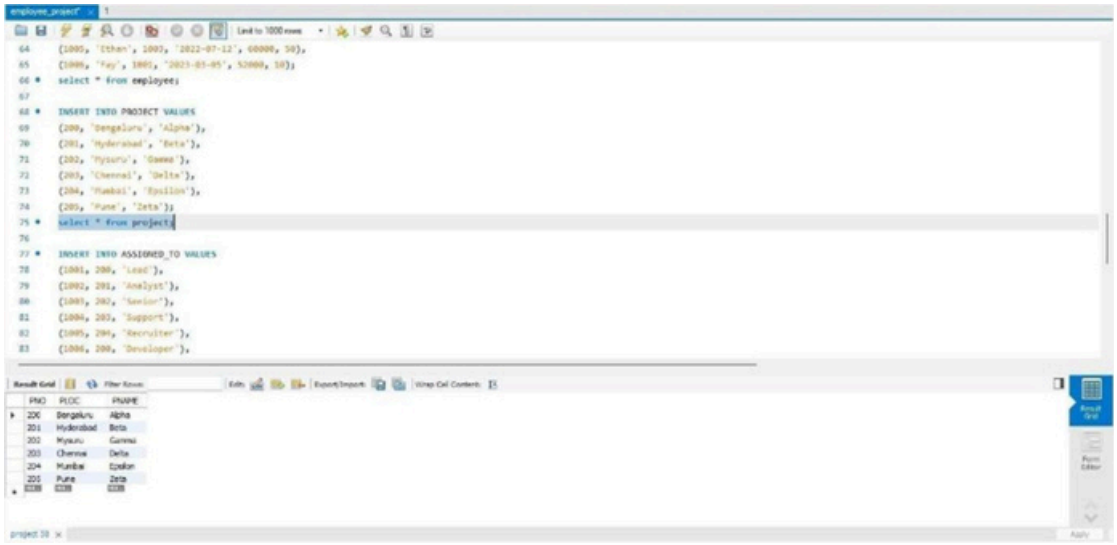
INSERTINTODEPTVALUES
(10, 'Sales','Bengaluru'),
(20, 'Accounting', 'Hyderabad'),
(30, 'Research', 'Mysuru'),
(40, 'Operations', 'Chennai'),
(50, 'HR', 'Mumbai'),
(60, 'IT', 'Delhi');
select * from dept;



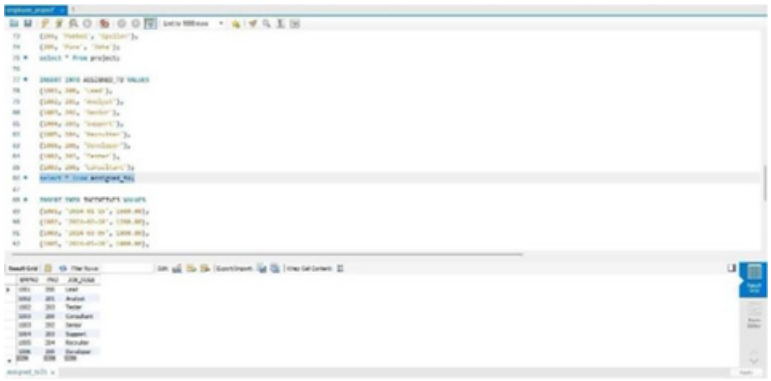
INSERT INTO EMPLOYEE VALUES
(1001, 'Alice', NULL, '2019-02-10', 75000, 10),
(1002, 'Bob', 1001, '2020-06-01', 65000, 20),
(1003, 'Charlie', 1001, '2018-09-15', 80000, 30),
(1004, 'Diana', 1002, '2021-01-20', 55000, 40),
(1005, 'Ethan', 1003, '2022-07-12', 60000, 50),
(1006, 'Fay', 1001, '2023-03-05', 52000, 10);
Select * from employee;



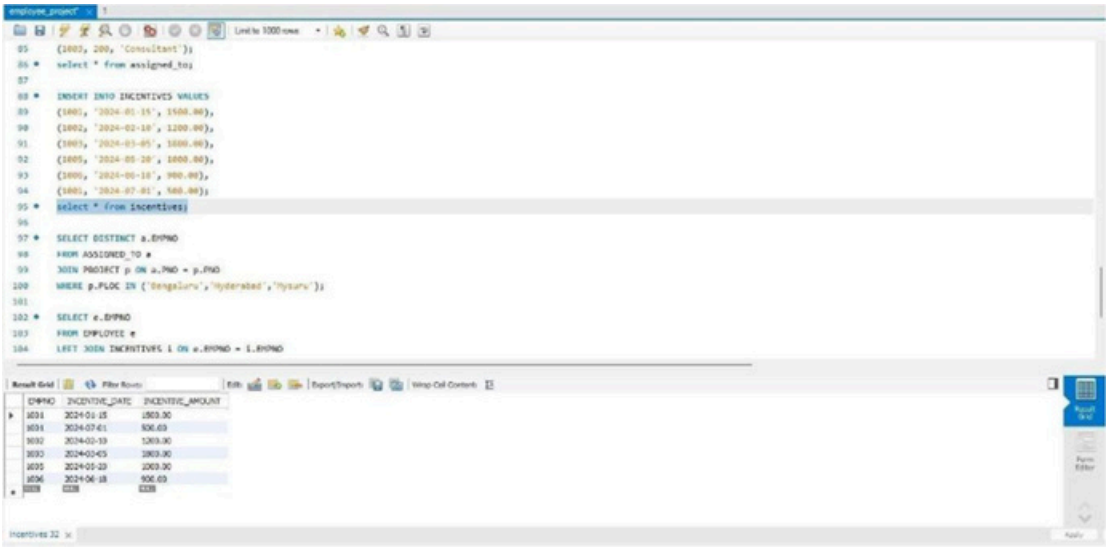
```
INSERT INTO PROJECT VALUES
(200, 'Bengaluru', 'Alpha'), (201,
'Hyderabad', 'Beta'), (202,
'Mysuru', 'Gamma'), (203,
'Chennai', 'Delta'), (204,
'Mumbai','Epsilon'), (205,
'Pune','Zeta'); select * from project;
```



```
INSERT INTO ASSIGNED_TO
VALUES (1001, 200, 'Lead'),
(1002, 201, 'Analyst'),
(1003, 202, 'Senior'),
(1004, 203, 'Support'),
(1005, 204, 'Recruiter'),
(1006, 200, 'Developer'),
(1002, 203, 'Tester'),
(1003, 200,'Consultant');
select * from
assigned_to;
```

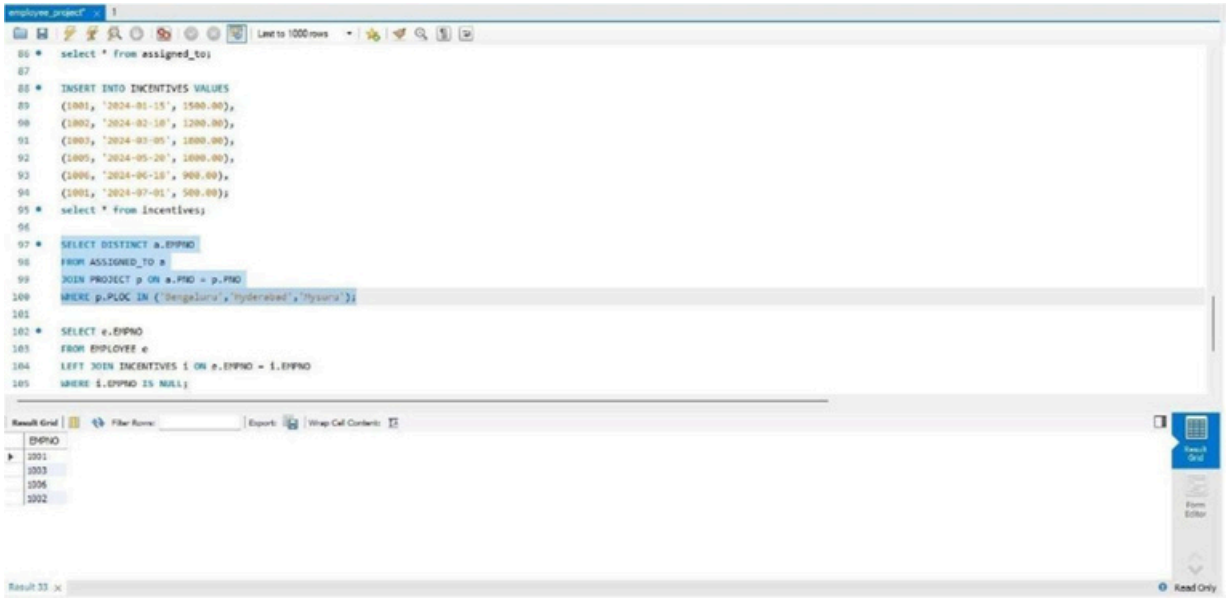


INSERT INTO INCENTIVES VALUES
(1001, '2024-01-15', 1500.00),
(1002, '2024-02-10', 1200.00), (1003,
'2024-03-05', 1800.00),
(1005, '2024-05-20', 1000.00),
(1006, '2024-06-18', 900.00),
(1001,'2024-07-01', 500.00);
select * from incentives;

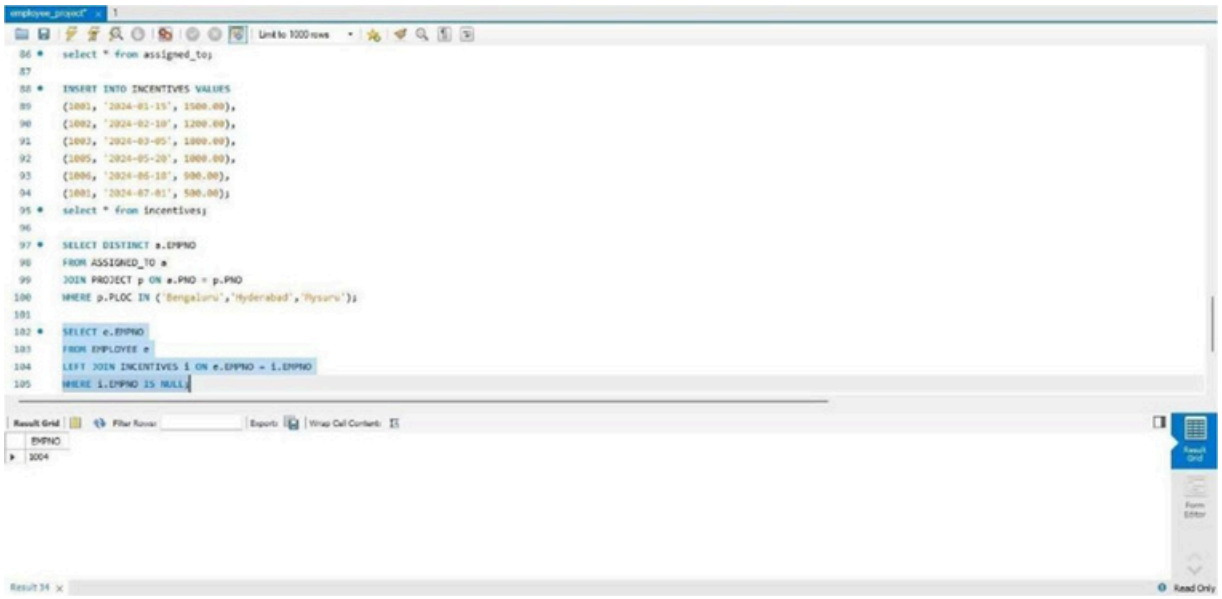


Queries

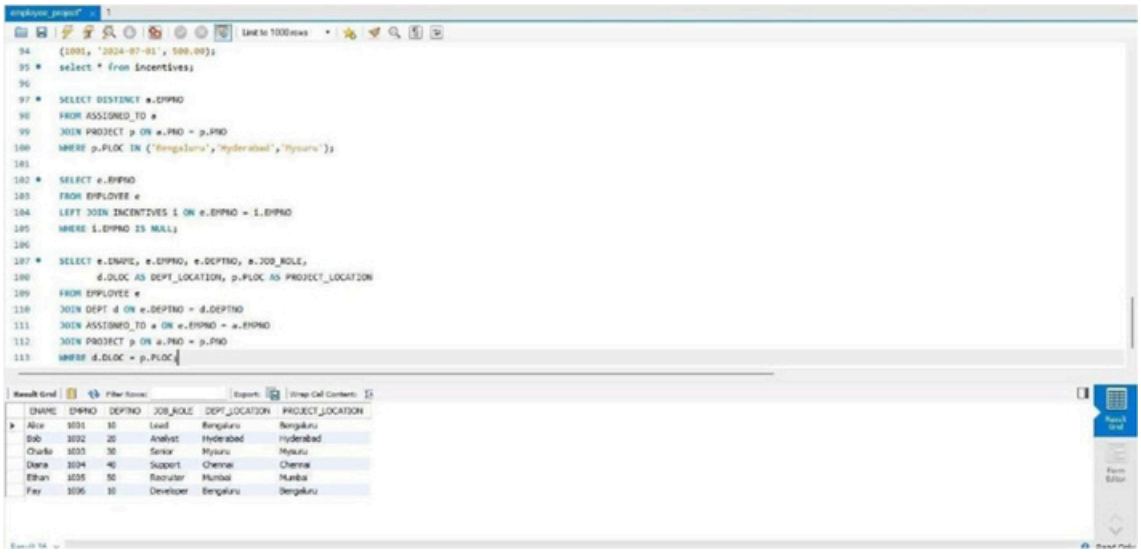
Retrieve the employee numbers of all employees who work on project located in Bengaluru, Hyderabad, or Mysuru.



Get Employee ID's of those employees who didn't receive incentives

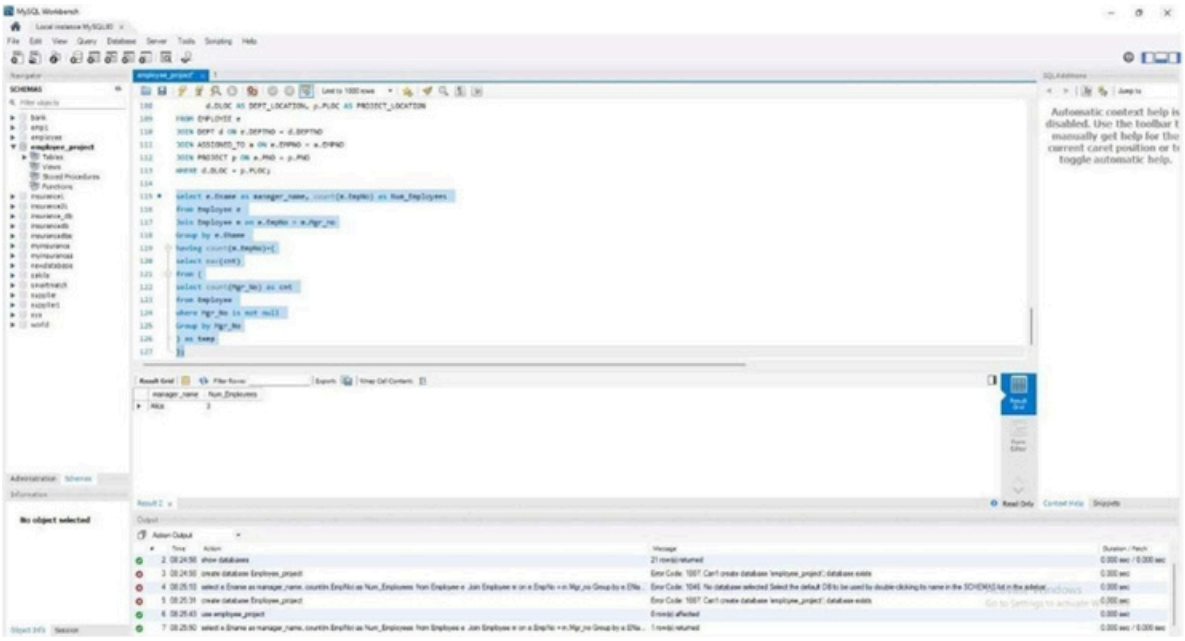


Write a SQL query to find the employees name, number, dept, job_role, department location and project location who are working for a project location same as his/her department location.

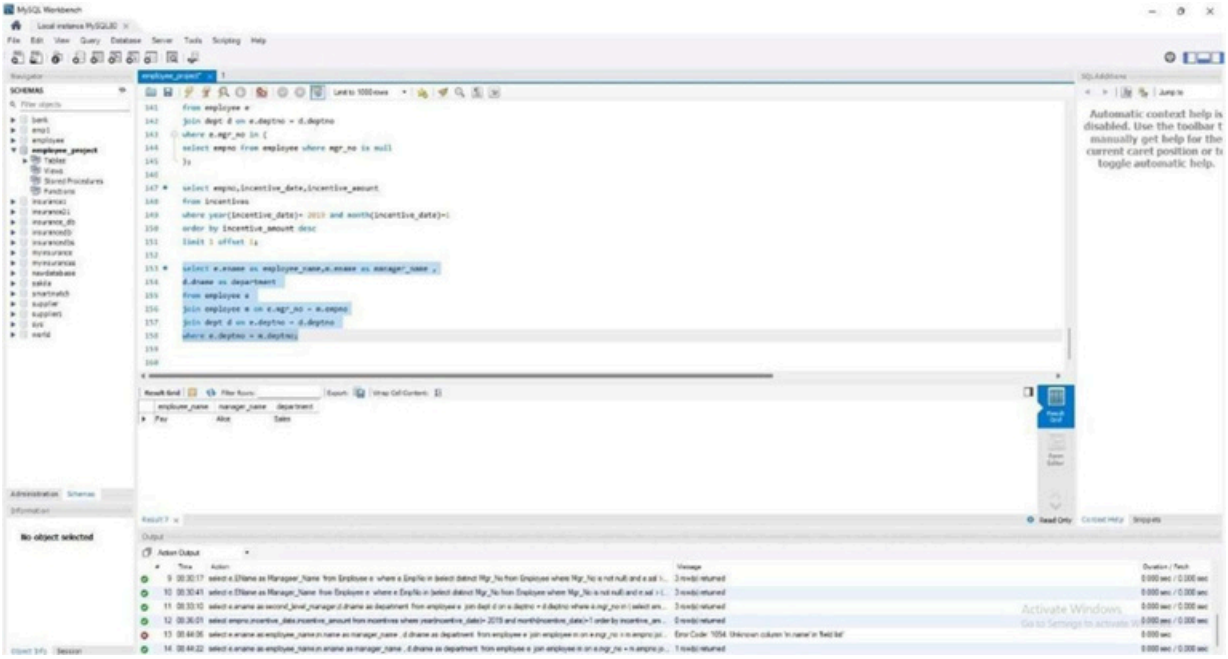


Experiment 6: More Queries on Employee Database

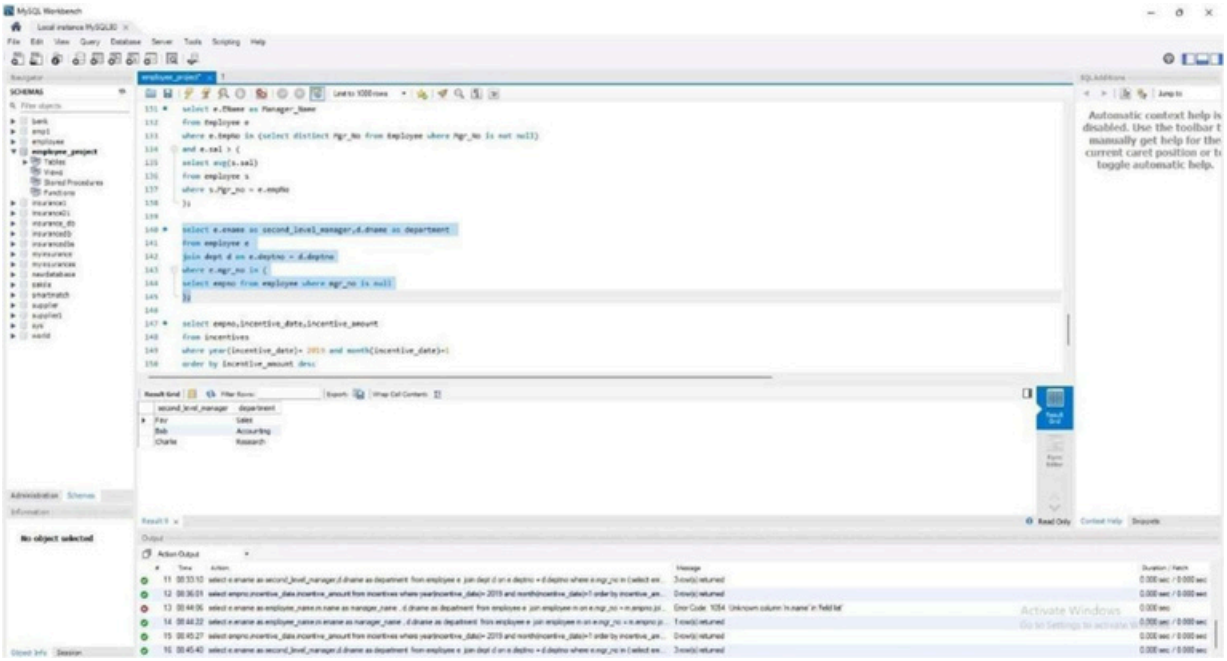
List the name of the managers with the maximum employees.



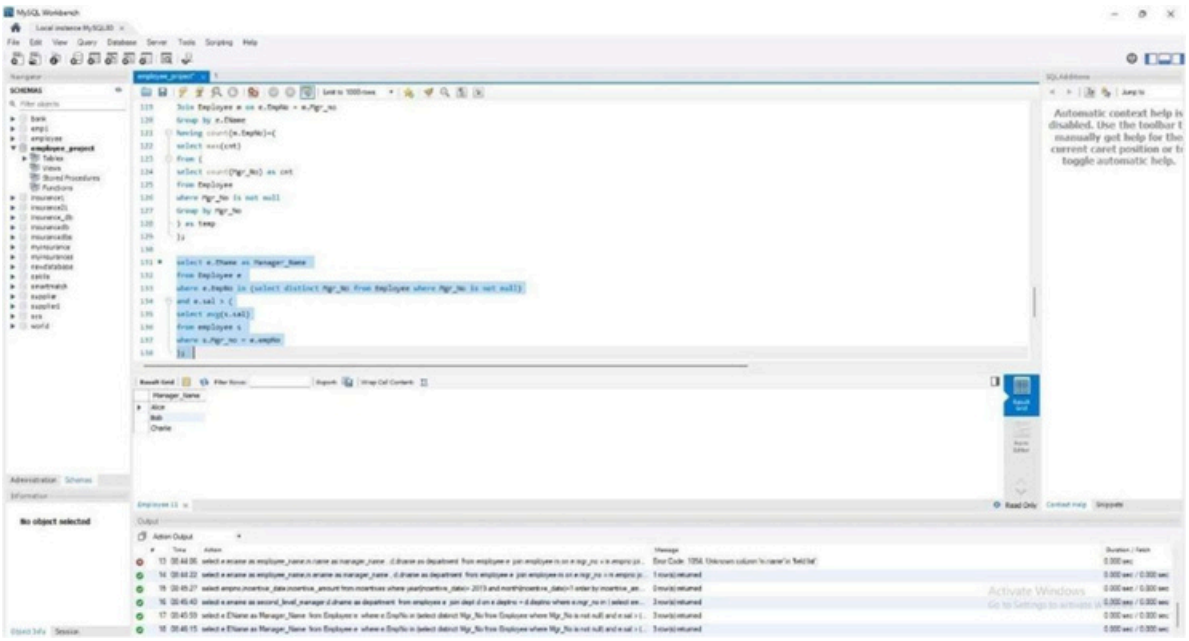
Display those managers name whose salary is more than average salary of his employee.



Find the employee details who got second maximum incentive in January 2019.



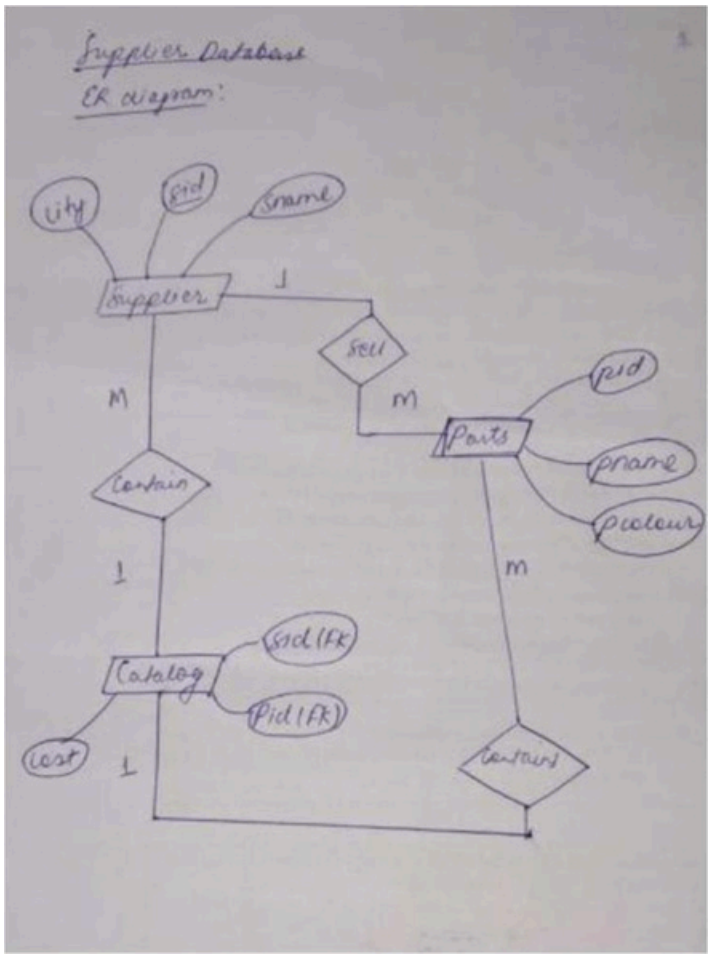
Display those employees who are working in the same department where his manager is working



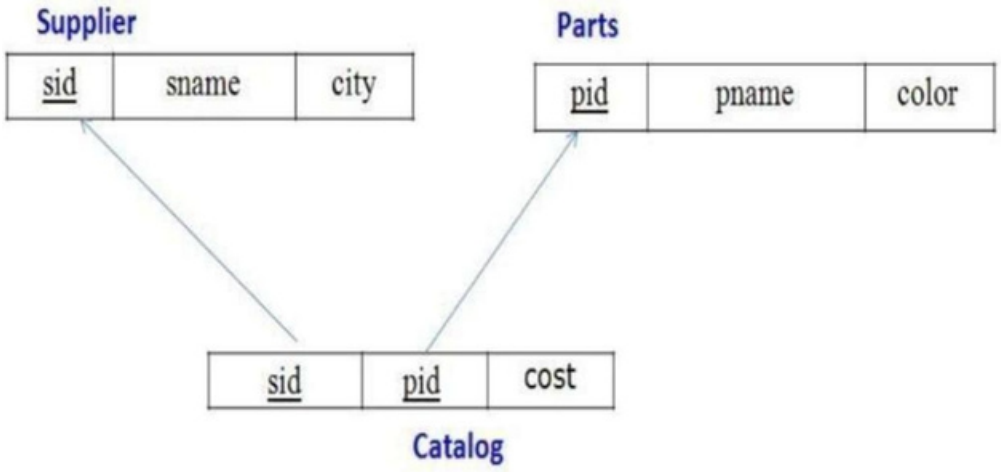
Experiment 7: Supplier Database

The supplier database must store information about suppliers, the parts they provide, and the prices at which each part is offered so that purchasing, analysis, and reporting can be done accurately. Each supplier is uniquely identified by a supplier ID and is recorded with a name and the city in which the supplier is located; each part is uniquely identified by a part ID and includes a part name and a color. The system must maintain a catalog that links suppliers to the parts they supply and records the cost at which a given supplier sells a given part. Every catalog entry must reference an existing supplier and an existing part, and there must be no duplicate entries for the same combination of supplier and part, so that at most one current price record exists per supplier–part pair. Costs must be valid numeric values and strictly non-negative, and business rules may specify upper limits or currency formats that must be enforced consistently. The data model must support the possibility that a supplier can provide many different parts, that a part can be supplied by many different suppliers, and that some suppliers or parts may temporarily have no catalog entries if they are inactive or not currently traded. Referential integrity must be enforced so that a supplier or part cannot be deleted while still referenced in the catalog unless such deletion is handled by controlled archival or cascade rules that preserve historical price information; in general, historical catalog data should not be lost, as it may be required for audits or trend analysis. The system should allow queries such as “find all suppliers for a given part,” “list all parts provided by a given supplier,” “retrieve the cheapest supplier for each part,” and “analyze supplier coverage by city,” and must therefore guarantee that identifiers are unique, relationships between suppliers, parts, and catalog entries are consistent, and price information is accurate and reliably maintained over time.

Entity Relationship Diagram



Schema Diagram



Create database

createdatabaseif not exists Supplierr ; use supplierr;

Create Table

```
CREATE TABLE Supplier
( sid INT PRIMARY KEY,
sname VARCHAR(50),
city VARCHAR(50)
);
CREATE TABLE Parts
( pid INT PRIMARY
KEY, pname
VARCHAR(50),
color VARCHAR(20)
);
CREATE TABLE Catalog (
sid INT,
pid INT,
cost INT,
PRIMARY KEY (sid, pid),
FOREIGN KEY (sid) REFERENCES Supplier(sid),
FOREIGN KEY (pid) REFERENCES Parts(pid)
);
```

Structure of the table

desc supplier;

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
sid	int	NO	PRI	NULL	
sname	varchar(40)	YES		NULL	
city	varchar(40)	YES		NULL	

desc parts;

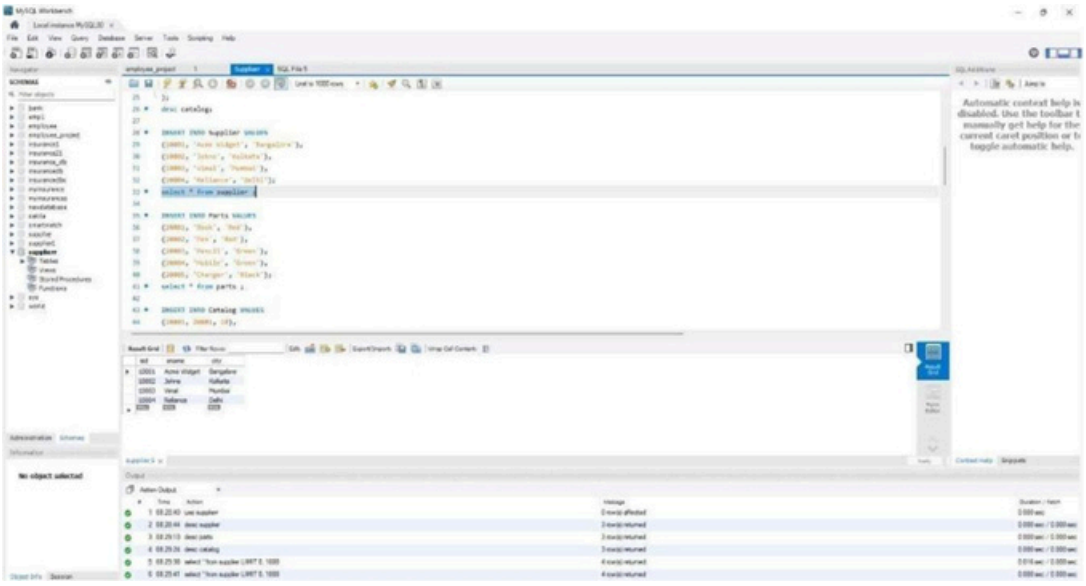
Field	Type	Null	Key	Default	Extra
pid	int	NO	PRI	NULL	
pname	varchar(40)	YES		NULL	
colour	varchar(20)	YES		NULL	

desc catalog;

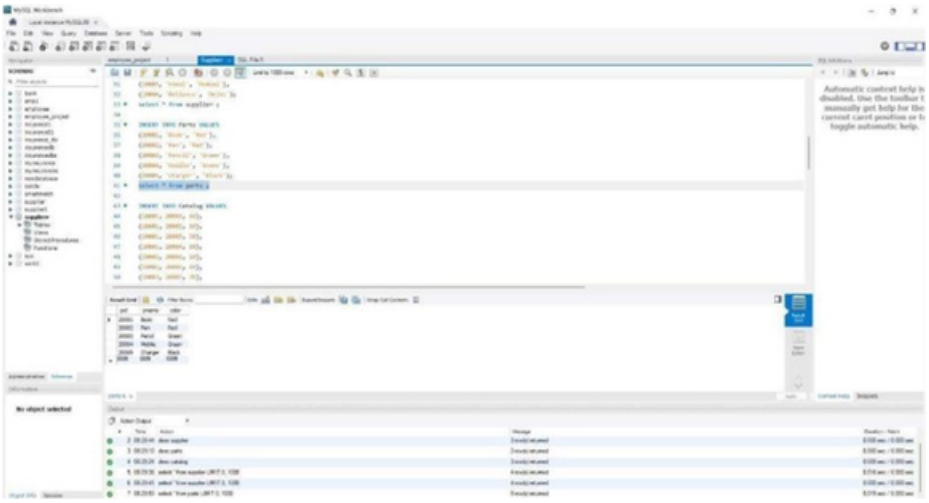
	Field	Type	Null	Key	Default	Extra
▶	sid	int	YES	MUL	NULL	
	pid	int	YES	MUL	NULL	
	cost	int	NO	PRI	NULL	

Insertion of values into the table

INSERT INTO Supplier VALUES
(10001, 'Acme Widget', 'Bangalore'),
(10002, 'Johns', 'Kolkata'),
(10003, 'Vimal', 'Mumbai'),
(10004,'Reliance', 'Delhi');

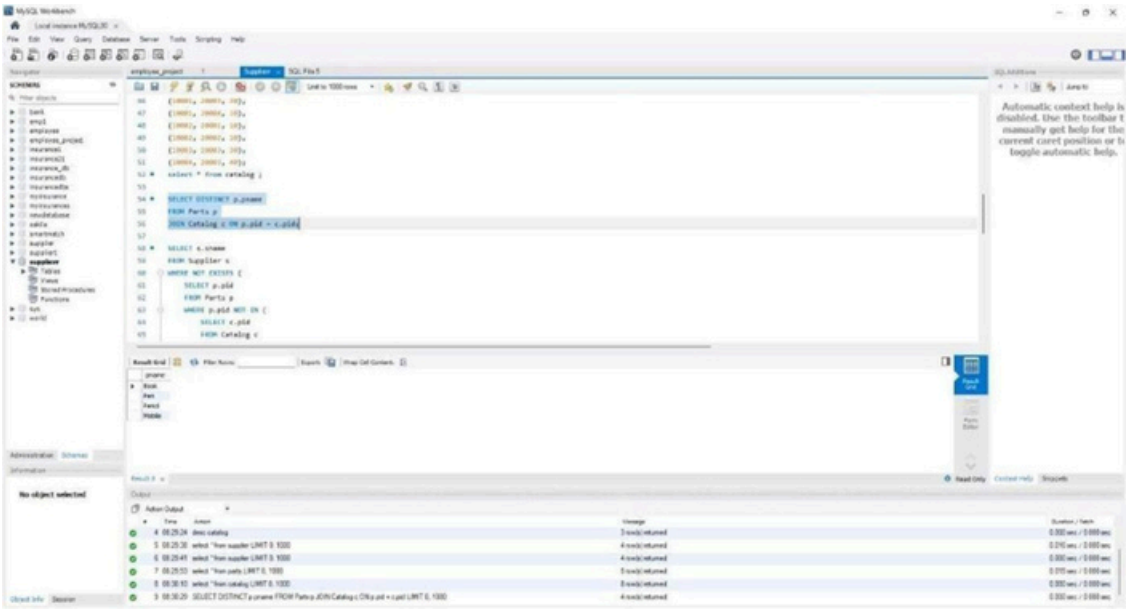


```
INSERT INTO Parts VALUES
(20001, 'Book', 'Red'),
(20002, 'Pen', 'Red'),
(20003, 'Pencil', 'Green'),
(20004, 'Mobile', 'Green'),
(20005, 'Charger', 'Black');
```

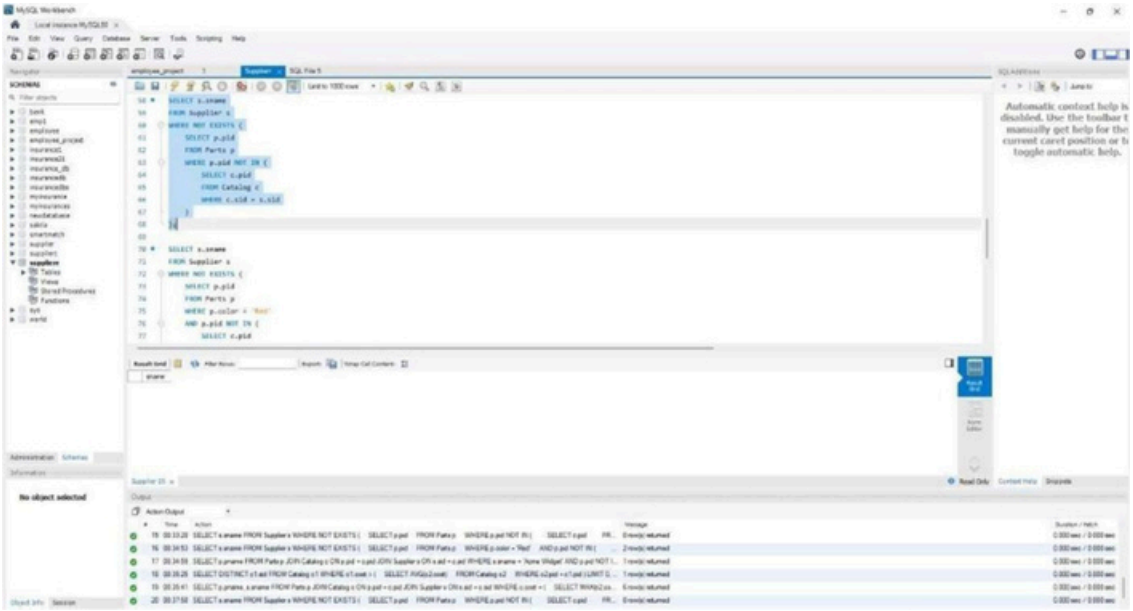


Queries

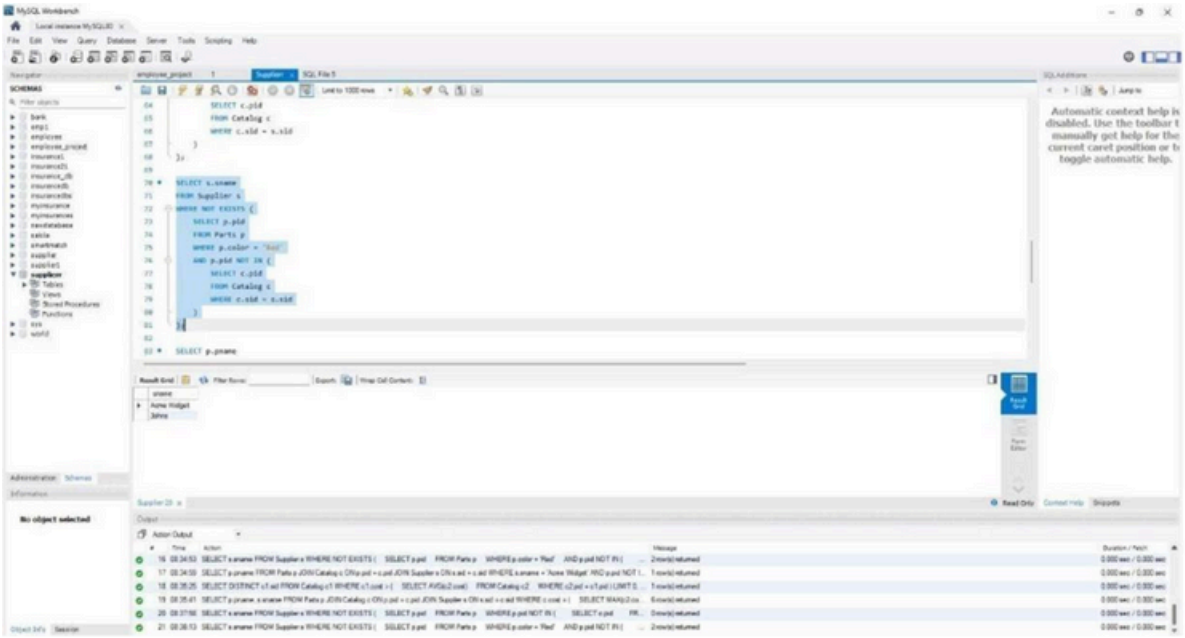
Find the pnames of parts for which there is some supplier



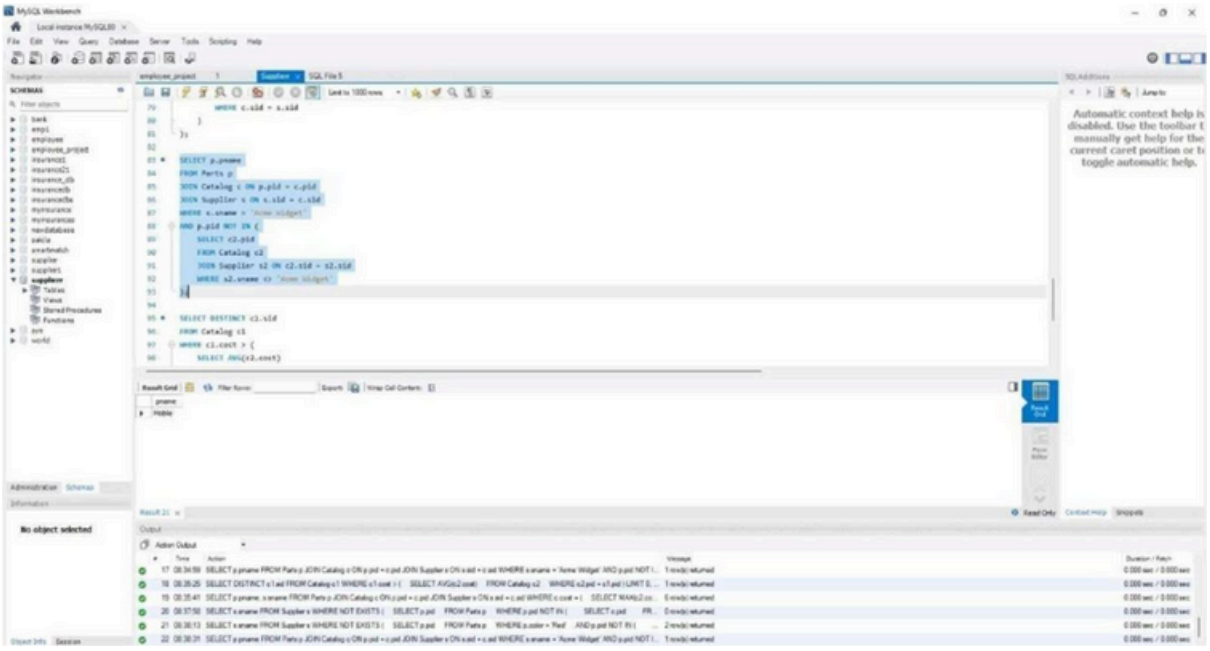
Find the snames of suppliers who supply every part



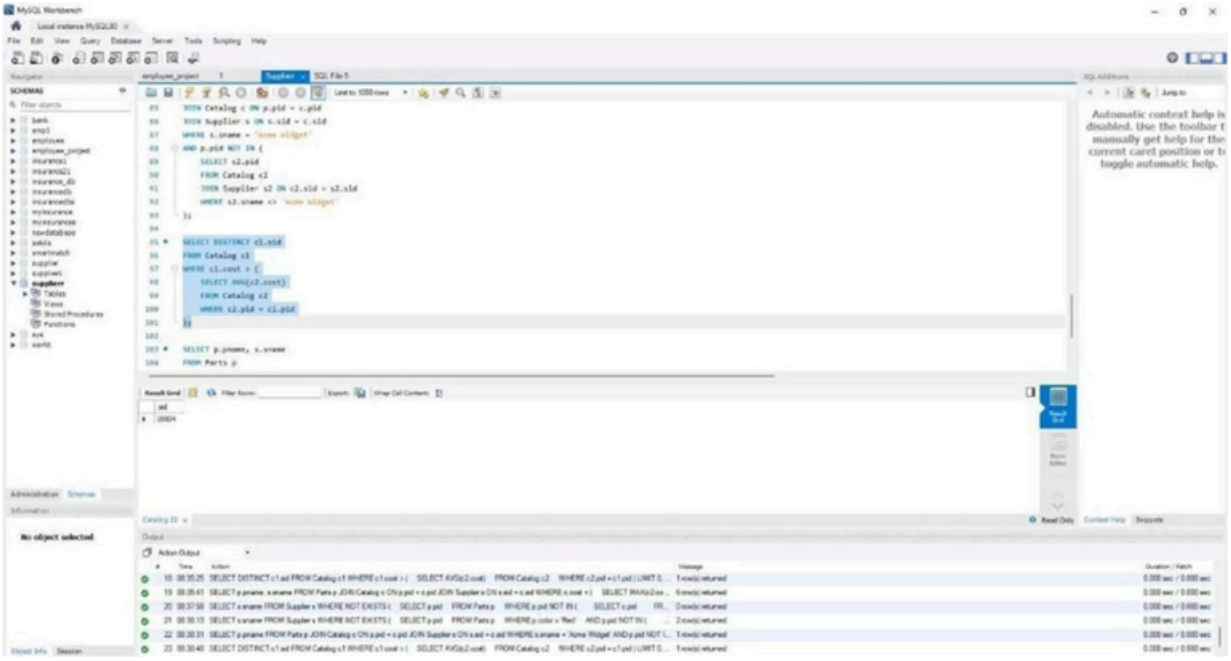
Find the snames of suppliers who supply every red part.



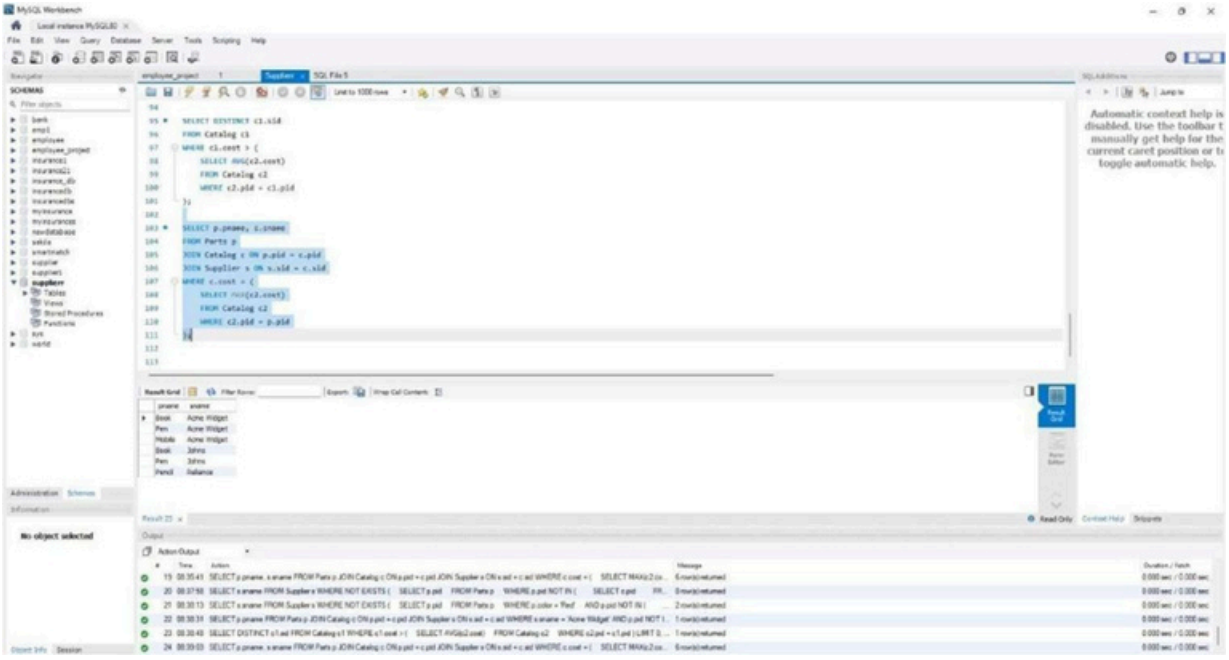
Find the pnames of parts supplied by Acme Widget Suppliers and by no one else.



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).



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



Lab8-MORE QUERIES ON SUPPLIER DATABASE

- **Find the most expensive part overall and the supplier who supplies it.**
  Select p.pname,s.sname,c.cost from catalog_202 c join parts_202 p on c.pid=p.pid join supplier_202 s on c.sid=s.sid where c.cost=(select max(cost) from catalog);

	Pname	Sname	Cost
▶	Pencil	Reliance	40

- **Find suppliers who do NOT supply any red parts.**
  Select s.sname from supplier_202 s where s.sid not in (select c.sid from catalog_202 c join parts_202 p on c.pid=p.pid where p.color='Red');

	Sname
▶	Vimal
	Reliance

- **Show each supplier and total value of all parts they supply.**
  select s.sname,sum(c.cost) as total_value from supplier_202 s join catalog_202 c on s.sid=c.sid group by s.sname

	Sname	Total_Value
▶	Acme Widget	70
	Johns	30
	Vimal	30
	Reliance	40

- **Find suppliers who supply at least 2 parts cheaper than ₹20.**
  select s.sname from supplier s join catalog c on s.sid=c.sid where c.cost<20 group by s.sname having count(c.pid)>=2;

	Sname
▶	Acme Widget

- **List suppliers who offer the cheapest cost for each part.**
  select p.pname, s.sname, c.cost from parts p join catalog c on p.pid = c.pid join supplier s on c.sid = s.sid where c.cost = (select min(c1.cost) from catalog c1 where c1.pid = p.pid);

	Pname	Sname	Cost
▶	Book	Acme Widget	10
	Pen	Acme Widget	10
	Pencil	Acme Widget	30
	Mobile	Acme Widget	10
	Charger	Acme Widget	10
	Book	Johns	10
	Pencil	Vimal	30

- **Create a view showing suppliers and the total number of parts they supply.**
  create view supplierpartcount_view as select s.sname, count(c.pid) as part_count from supplier s join catalog c on s.sid = c.sid group by s.sname;
  select * from SupplierPartCount_View;

	Sname	Part_Count
▶	Acme Widget	5
	Johns	2
	Vimal	1
	Reliance	1

- **Create a view of the most expensive supplier for each part.**
  create view mostexpensivesupplier_view as select p.pname, s.sname, c.cost from parts p join catalog c on p.pid = c.pid join supplier s on c.sid = s.sid where c.cost = (select max(c1.cost) from catalog c1 where c1.pid = p.pid); select * from MostExpensiveSupplier_View;

	Pname	Sname	Cost
▶	Book	Acme ...	10
	Mobile	Acme ...	10
	Charger	Acme ...	10
	Book	Johns	10
	Pen	Johns	20
	Pencil	Reliance	40

- **Create a Trigger to prevent inserting a Catalog cost below 1.**

```
delimiter //
create trigger min_cost_check_new
before insert on catalog
for each row
begin
        if new.cost < 1 then
signal sqlstate '45000' set message_text = 'catalog cost cannot be less than 1.';
```

```
end if;
end//
delimiter ;
insert into catalog values (101, 201, 0);
```

```
Error Code: 1644. Catalog cost cannot be less than 1.
```

- **Create a trigger to set to default cost if not provided.**

```
delimiter //
create trigger set_default_cost_new
before insert on catalog
for each row
begin
        if new.cost is null then
                set new.cost = 10;
        end if;
end//
delimiter ;
insert into catalog values (10002, 20005, null);
select * from catalog where sid=10002 and pid= 20005;
```

	sid	pid	cost
▶	10002	20005	10
*	NULL	NULL	NULL

Experiment 9: Nosql lab1(Student)

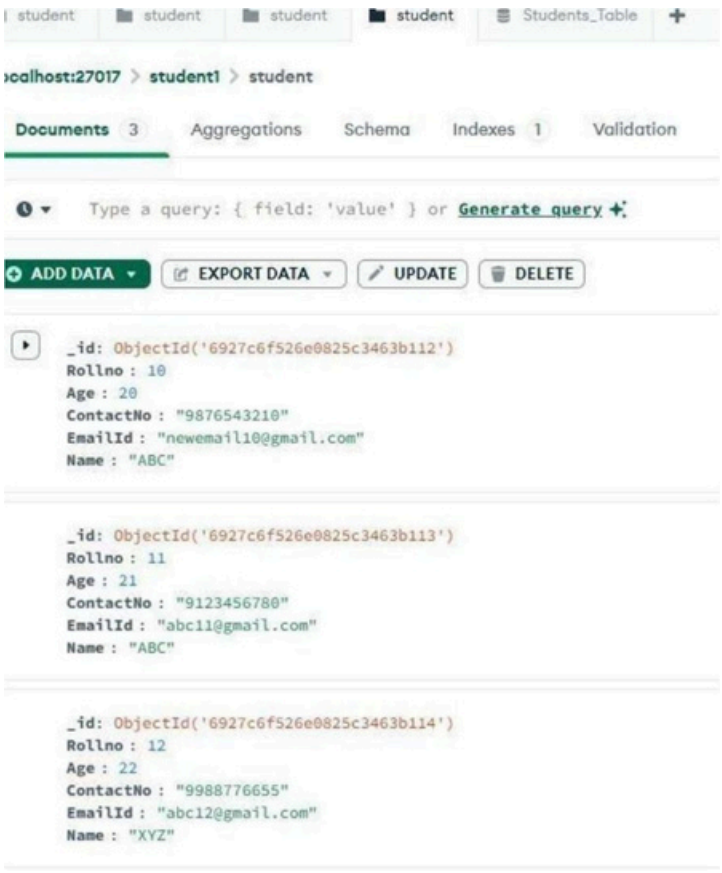
Create a database "Student" with the following attributesRollno, Age, ContactNo, EmailId.

```
>_MONGOSH

> use Student
< switched to db Student
Student>
```

Insert appropriate values.

```
switched to db Students_Table
Students_Table> db.student.insertMany([
...    { Rollno: 10, Age: 20, ContactNo: "9876543210", EmailId: "rahul10@gmail.com", Name: "RAHUL" },
...    { Rollno: 11, Age: 21, ContactNo: "9123456780", EmailId: "raj11@gmail.com", Name: "RAJ" },
...    { Rollno: 12, Age: 22, ContactNo: "9988776655", EmailId: "rajvardhan12@gmail.com", Name: "RAJVARDHAN" }
... ])
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6927c9b61771b37f9963b112'),
    '1': ObjectId('6927c9b61771b37f9963b113'),
    '2': ObjectId('6927c9b61771b37f9963b114')
  }
}
```



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

```
Students_Table> db.student.updateOne(
...    { Rollno: 10 },
...    { $set: { EmailId:raman10@gmail.com" } }
... )
```

Replace the student name from "ABC" to "FEM" of rollno 11.

```
Students_Table> db.student.updateOne(
...    { Rollno: 10 },
...    { $set: { EmailId:"raman10@gmail.com" } }
... )
...
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Students_Table> db.student.updateOne(
...    { Rollno: 11, Name: "RAJ" },
...    { $set: { Name: "RAJU" } }
... )
...
{
  acknowledged: true,
  insertedId: null,
```

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

```
mongoexport --db=Student --collection=students --out=students.json
```

Export the created table into local file system.

```
2025-02-20T16:22:13.543+0530    connected to: localhost
2025-02-20T16:22:13.678+0530    exported 3 records
```

Experiment 10 : Nosql lab2(Customer)

Create a collection by name Customers with the following attributes. Cust_id, Acc_Bal, Acc_Type

```
>_MONGOSH

> use CustomerDB

< switched to db CustomerDB
> db.Customers.insertMany([

    { Cust_id: 101, Acc_Bal: 1500, Acc_Type: "Z" },

    { Cust_id: 102, Acc_Bal: 900,  Acc_Type: "A" },

    { Cust_id: 101, Acc_Bal: 1800, Acc_Type: "Z" },

    { Cust_id: 103, Acc_Bal: 1300, Acc_Type: "Z" },

    { Cust_id: 104, Acc_Bal: 700,  Acc_Type: "B" }

  ])

< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('693a3695a2d039c4696aa0a8'),
      '1': ObjectId('693a3695a2d039c4696aa0a9'),
      '2': ObjectId('693a3695a2d039c4696aa0aa'),
      '3': ObjectId('693a3695a2d039c4696aa0ab'),
      '4': ObjectId('693a3695a2d039c4696aa0ac')
    }
  }
```

Write a query to display those records whose total account balance is greater than 1200 of account type ‘Z’ for each customer_id.

```
>_MONGOSH

    }
  }
> db.Customers.find(
    { Acc_Type: "Z", Acc_Bal: { $gt: 1200 } }
  )
< {
    _id: ObjectId('693a3695a2d039c4696aa0a8'),
    Cust_id: 101,
    Acc_Bal: 1500,
    Acc_Type: 'Z'
  }
  {
    _id: ObjectId('693a3695a2d039c4696aa0aa'),
    Cust_id: 101,
    Acc_Bal: 1800,
    Acc_Type: 'Z'
  }
  {
    _id: ObjectId('693a3695a2d039c4696aa0ab'),
    Cust_id: 103,
    Acc_Bal: 1300,
    Acc_Type: 'Z'
  }
```

Determine Minimum and Maximum account balance for each customer_id.

```
>_MONGOSH
    Acc_Type: 'Z'
  }
  {
    _id: ObjectId('693a3695a2d039c4696aa0ab'),
    Cust_id: 103,
    Acc_Bal: 1300,
    Acc_Type: 'Z'
  }
> db.Customers.aggregate([
    {
      $group: {
        _id: "$Cust_id",
        Min_Balance: { $min: "$Acc_Bal" },
        Max_Balance: { $max: "$Acc_Bal" }
      }
    }
  ])
< {
    _id: 104,
    Min_Balance: 700,
    Max_Balance: 700
  }
  {
    _id: 102,
    Min_Balance: 900,
    Max_Balance: 900
  }
  {
    _id: 101,
    Min_Balance: 1500,
    Max_Balance: 1800
  }
  {
    _id: 103,
    Min_Balance: 1300,
    Max_Balance: 1300
  }
```

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

```
mongoexport --db=CustomerDB --collection=Customers --out=customers.json
```

Export the created collection into local file system.

```
2025-02-20T16:55:12.543+0530    connected to: localhost
2025-02-20T16:55:12.684+0530    exported 5 records
```