VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“Jnana Sangama”, Belgaum -590014, Karnataka.

LAB REPORT

on

Database Management Systems (23CS3PCDBM)

Submitted by

Nandini Yuvraj(1BM24CS182)

in partial fulfilment 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 Jan-2026

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 **Nandini Yuvraj(1BM24CS182),** who is bona-fide 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 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.

Divyashree S Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

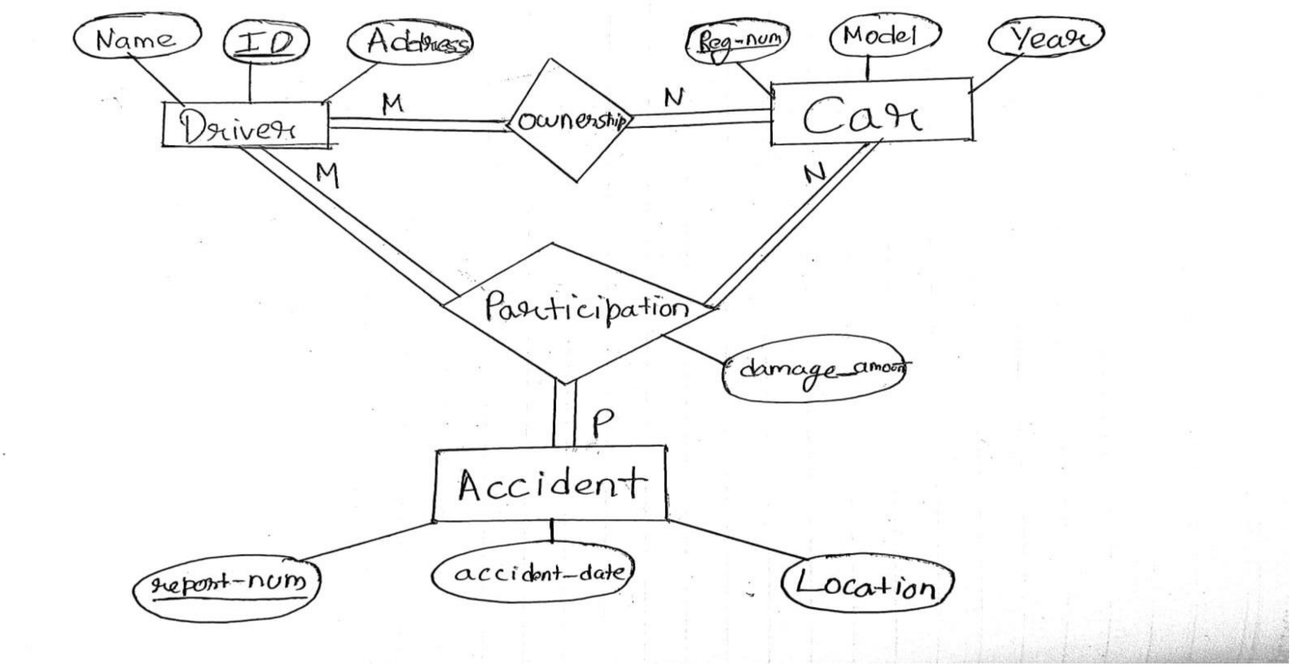
INDEX

Experiment 1: INSURANCE DATABASE

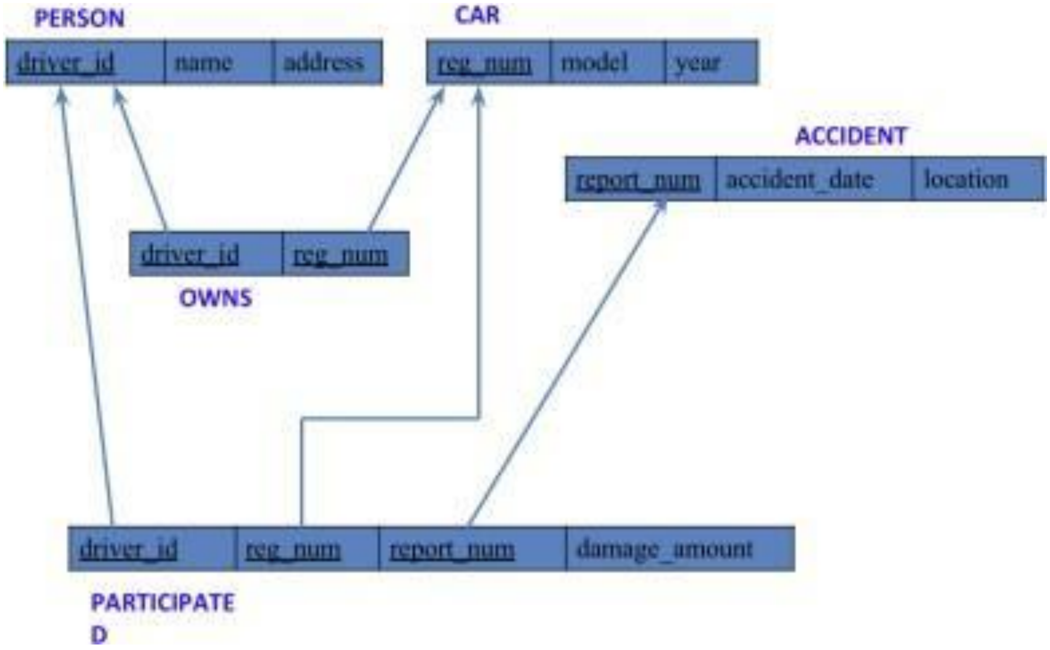
Specification of Insurance Database Application -

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 unique 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 participating driver and car, and this participation is recorded by linking the driver, the involved car, and the accident report together with the corresponding damage amount for that particular involvement. A participation 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 database 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


PERSON

driver_id	name	address

CAR

reg_num	model	year

ACCIDENT

report_num	accident_date	location

driver_id	reg_num

OWNS

driver_id	reg_num	report_num	damage_amount

PARTICIPATED

- **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 database

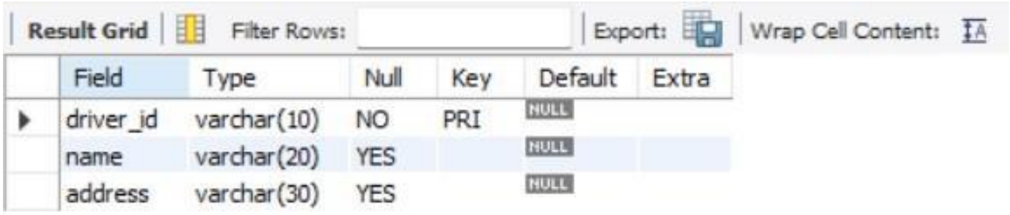
```
create database insurance;
 use insurance;
```

Create table

```
 create table person (
driver_id varchar(10), name varchar(20),
address varchar(30), primary key(driver_id)
);
create table car (
reg_num varchar(10), model varchar(10),
year int, primary key(reg_num)
);
create table accident (
report_num int, accident_date date, location varchar(20),
 primary key(report_num)
);
create table owns(
driver_id varchar(10), reg_num varchar(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 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)
);
```

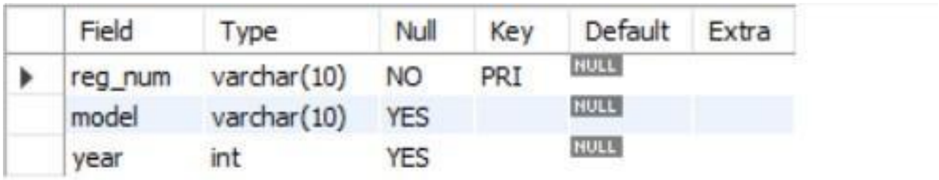
Structure of the table

- desc person;



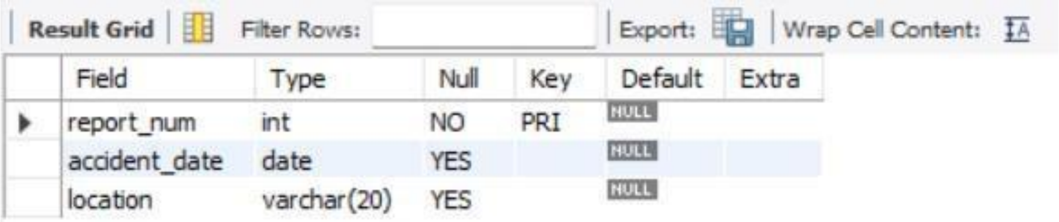
Field	Type	Null	Key	Default	Extra
driver_id	varchar(10)	NO	PRI	NULL	
name	varchar(20)	YES		NULL	
address	varchar(30)	YES		NULL	

- desc car;



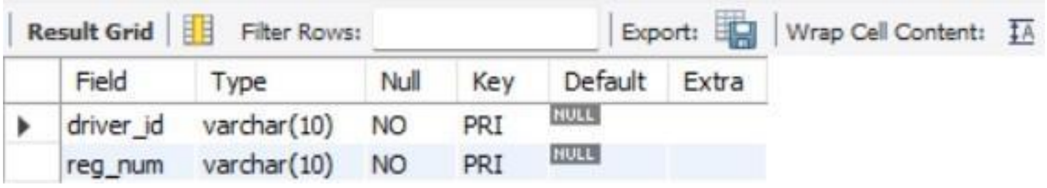
Field	Type	Null	Key	Default	Extra
reg_num	varchar(10)	NO	PRI	NULL	
model	varchar(10)	YES		NULL	
year	int	YES		NULL	

- desc accident;



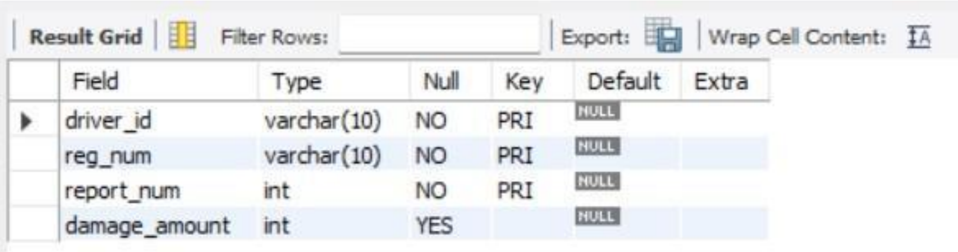
Field	Type	Null	Key	Default	Extra
report_num	int	NO	PRI	NULL	
accident_date	date	YES		NULL	
location	varchar(20)	YES		NULL	

- desc owns;



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

- desc participated;

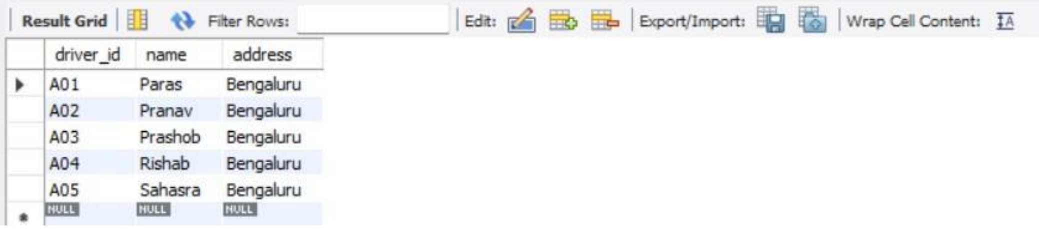


Field	Type	Null	Key	Default	Extra
driver_id	varchar(10)	NO	PRI	NULL	
reg_num	varchar(10)	NO	PRI	NULL	
report_num	int	NO	PRI	NULL	
damage_amount	int	YES		NULL	

Inserting Values to the table

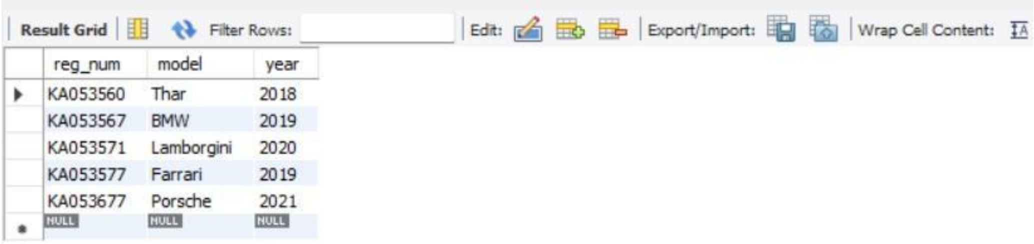
- insert into person values('A01','Paras','Bengaluru');
- insert into person values('A02','Pranav','Bengaluru');
- insert into person values('A03','Prashob','Bengaluru');
- insert into person values('A04','Rishab','Bengaluru');
- insert into person values('A05','Sahasra','Bengaluru');

- select *from person;



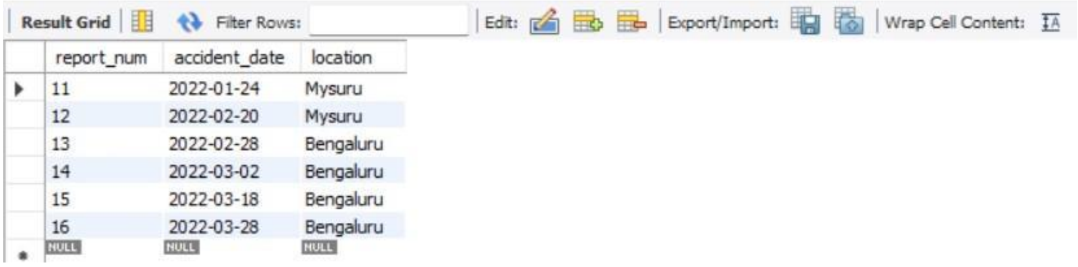
driver_id	name	address
A01	Paras	Bengaluru
A02	Pranav	Bengaluru
A03	Prashob	Bengaluru
A04	Rishab	Bengaluru
A05	Sahasra	Bengaluru
NULL	NULL	NULL

- insert into car values('KA053560','Thar',2018);
- insert into car values('KA053571','Lamborgini',2020);
- insert into car values('KA053567','BMW',2019);
- insert into car values('KA053577','Farrari',2019);
- insert into car values('KA053677','Porsche',2021);
- select *from car;



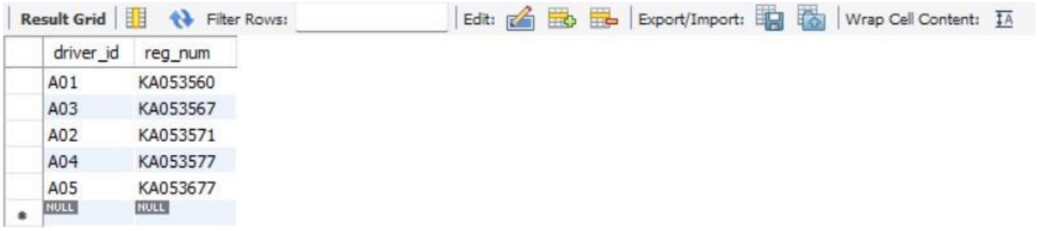
reg_num	model	year
KA053560	Thar	2018
KA053567	BMW	2019
KA053571	Lamborgini	2020
KA053577	Farrari	2019
KA053677	Porsche	2021
NULL	NULL	NULL

- insert into accident values(11,'2022-01-24','Mysuru');
- insert into accident values(12,'2022-02-20','Mysuru');
- insert into accident values(13,'2022-02-28','Bengaluru');
- insert into accident values(14,'2022-03-02','Bengaluru');
- insert into accident values(15,'2022-03-18','Bengaluru');
- select *from accident;



report_num	accident_date	location
11	2022-01-24	Mysuru
12	2022-02-20	Mysuru
13	2022-02-28	Bengaluru
14	2022-03-02	Bengaluru
15	2022-03-18	Bengaluru
16	2022-03-28	Bengaluru
NULL	NULL	NULL

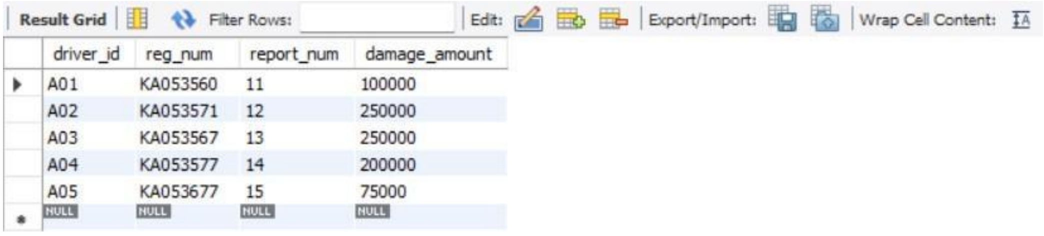
- insert into OWNS values('A01','KA053560');
- insert into OWNS values('A02','KA053571');
- insert into OWNS values('A03','KA053567');
- insert into OWNS values('A04','KA053577');
- insert into OWNS values('A05','KA053677');
- select *from owns;



driver_id	reg_num
A01	KA053560
A03	KA053567
A02	KA053571
A04	KA053577
A05	KA053677
NULL	NULL

- insert participated values('A01','KA053560',11,100000);
- insert into participated values('A02','KA053571',12,500000);
- insert into participated values('A03','KA053567',13,250000)

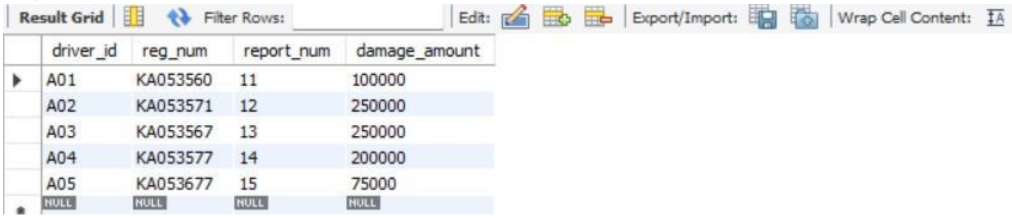
- insert into participated values('A04','KA053577',14,200000);
- insert into participated values('A05','KA053677',15,75000);
- select *from participated;



driver_id	reg_num	report_num	damage_amount
A01	KA053560	11	100000
A02	KA053571	12	250000
A03	KA053567	13	250000
A04	KA053577	14	200000
A05	KA053677	15	75000
NULL	NULL	NULL	NULL

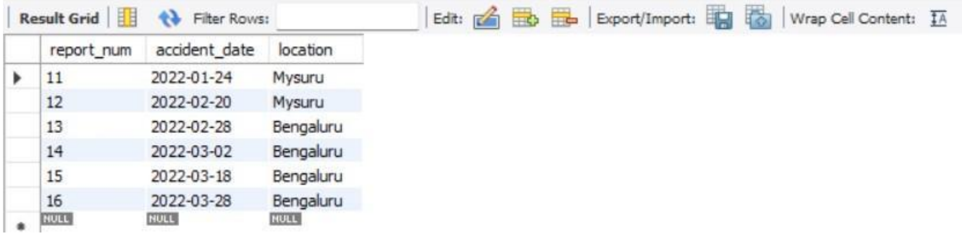
Queries

- **Update the damage amount to 25000 for the car with a specific reg_num (example 'KA053408') for which the accident report number was 12.**
  - update participated set damage_amount=250000 where reg_num='KA053571' AND report_num=12;



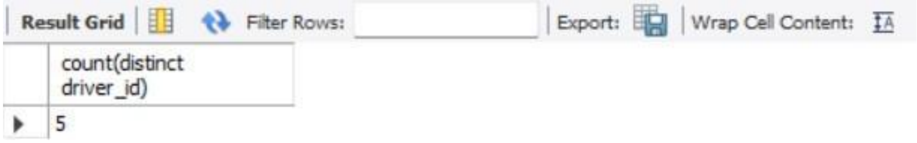
driver_id	reg_num	report_num	damage_amount
A01	KA053560	11	100000
A02	KA053571	12	250000
A03	KA053567	13	250000
A04	KA053577	14	200000
A05	KA053677	15	75000
NULL	NULL	NULL	NULL

- **Add a new accident to the database.**
  - insert into accident values(16,'2022-03-28','Bengaluru');



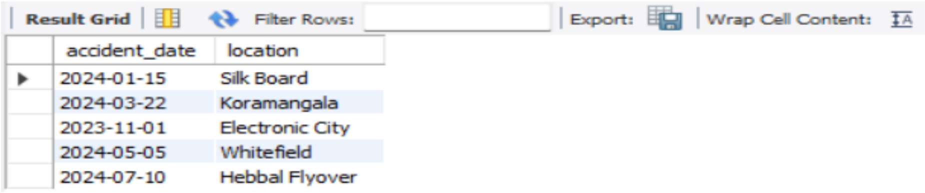
report_num	accident_date	location
11	2022-01-24	Mysuru
12	2022-02-20	Mysuru
13	2022-02-28	Bengaluru
14	2022-03-02	Bengaluru
15	2022-03-18	Bengaluru
16	2022-03-28	Bengaluru
NULL	NULL	NULL

- **Find the total number of people who owned cars that were involved in accidents in 2008.**
- SELECT count(distinct driver_id) from participated a, accident b where a.report_num=b.report_num and b.accident_date like '_22%';



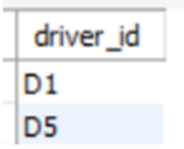
count(distinct driver_id)
5

- **Display Accident date and location**
- SELECT accident_date, location from accident;



accident_date	location
2024-01-15	Silk Board
2024-03-22	Koramangala
2023-11-01	Electronic City
2024-05-05	Whitefield
2024-07-10	Hebbal Flyover

- **Display driver id who did accident with damage amount greater than or equal to Rs.25000**
- SELECT DISTINCT driver_id
  FROM PARTICIPATED
  WHERE damage_amount >= 25000;



driver_id
D1
D5

Experiment 2:MORE QUERIES ON INSURANCE DATABASE

Question

i. List the entire participated relation in the descending order of damage amount.

ii. Find the average damage amount.

iii. Delete the tuple whose damage amount is below the average damage amount.

iv. List the name of drivers whose damage is greater than the average damage amount.

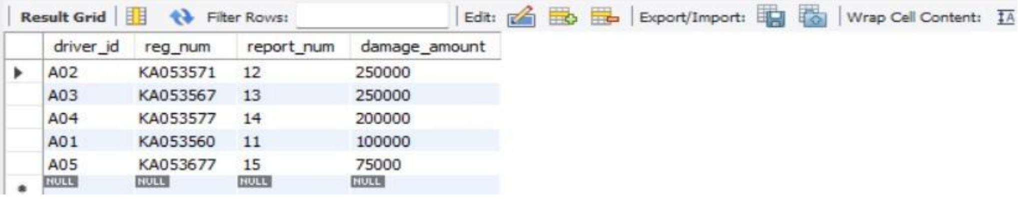
v. Find maximum damage amount.

vi. Cars that never had an accident

vii. Latest accident

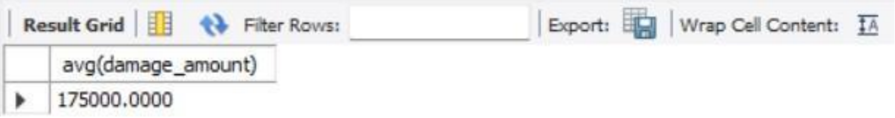
Queries

- **List the entire participated relation in the descending order of damage amount.**
  - select * from participated order by damage_amount desc;



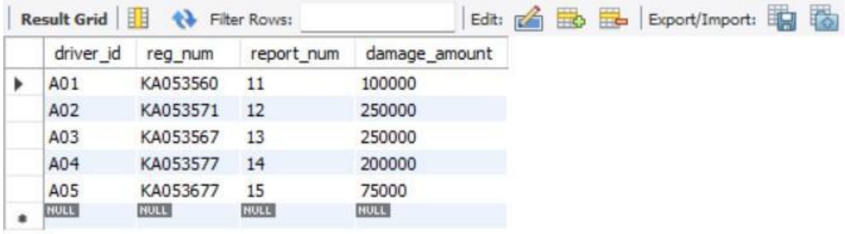
driver_id	reg_num	report_num	damage_amount
A02	KA053571	12	250000
A03	KA053567	13	250000
A04	KA053577	14	200000
A01	KA053560	11	100000
A05	KA053677	15	75000
NULL	NULL	NULL	NULL

- **Find the average damage amount.**
  - select avg(damage_amount) from participated;



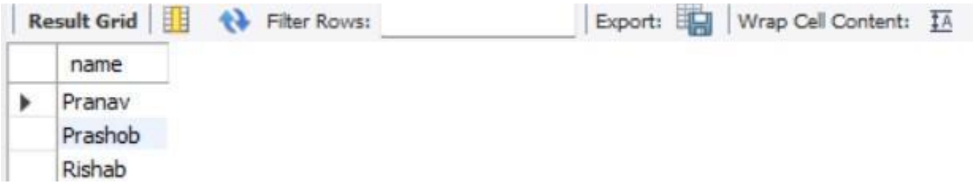
avg(damage_amount)
175000.0000

- **Delete the tuple whose damage amount is below the average damage amount.**
  - delete from participated where damage_amount < (175000.0000);



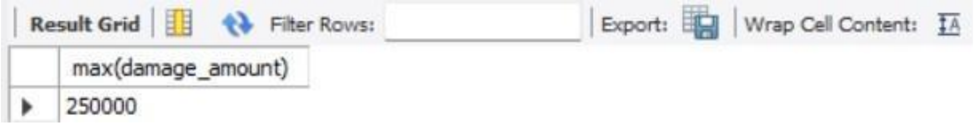
driver_id	reg_num	report_num	damage_amount
A01	KA053560	11	100000
A02	KA053571	12	250000
A03	KA053567	13	250000
A04	KA053577	14	200000
A05	KA053677	15	75000
NULL	NULL	NULL	NULL

- **List the name of drivers whose damage is greater than the average damage amount.**
    - select name from person a, participated b where a.driver_id=b.driver_id and damage_amount>(select avg(damage_amount) from participated);



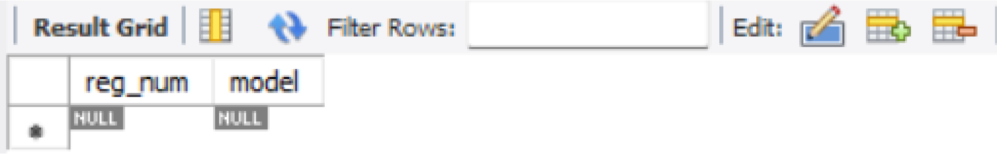
name
Pranav
Prashob
Rishab

- **Find maximum damage amount.**
    - select max(damage_amount) from participated;



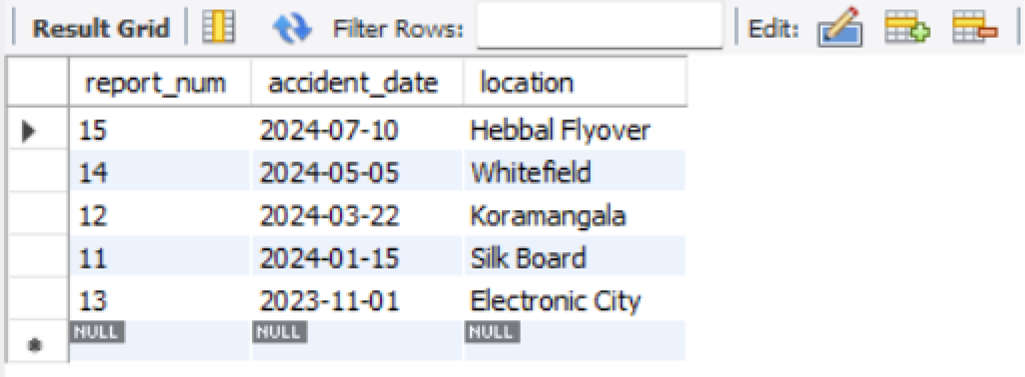
max(damage_amount)
250000

- **Cars that never had an accident**
- SELECT reg_num, model FROM CAR WHERE reg_num NOT IN (
  SELECT reg_num FROM PARTICIPATED);



reg_num	model
NULL	NULL

- **Latest accident**
- SELECT report_num, accident_date, location FROM ACCIDENT ORDER BY accident_date DESC;



report_num	accident_date	location
15	2024-07-10	Hebbal Flyover
14	2024-05-05	Whitefield
12	2024-03-22	Koramangala
11	2024-01-15	Silk Board
13	2023-11-01	Electronic City
NULL	NULL	NULL

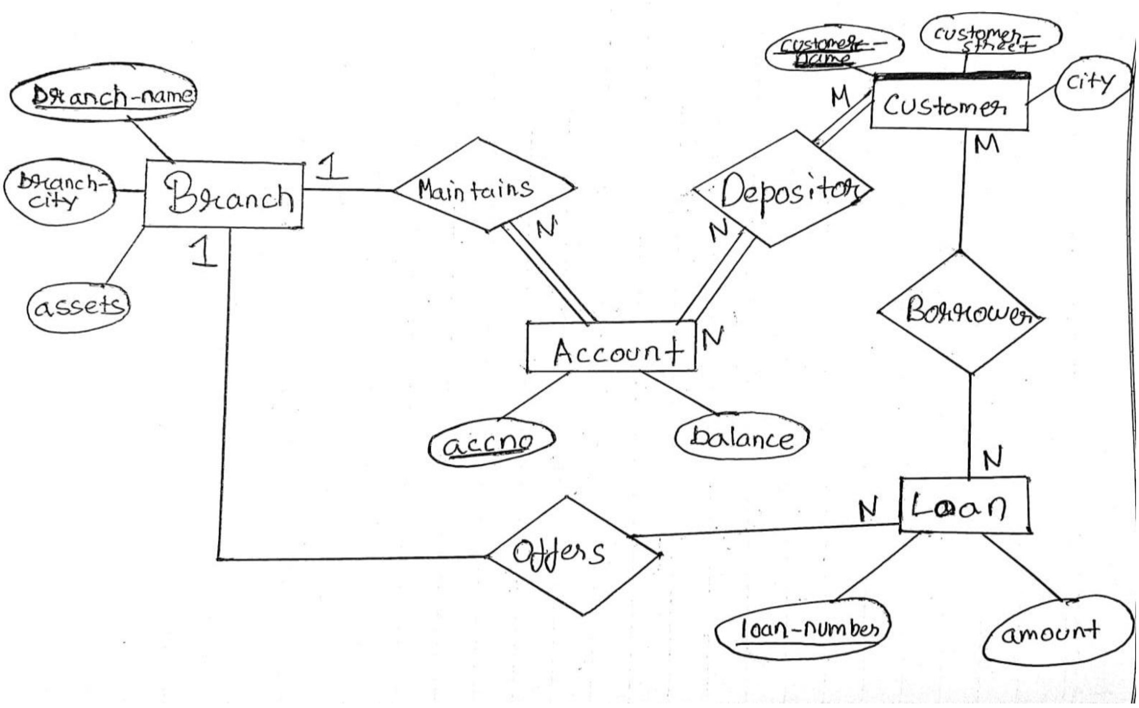
-

Experiment 3: BANK DATABASE

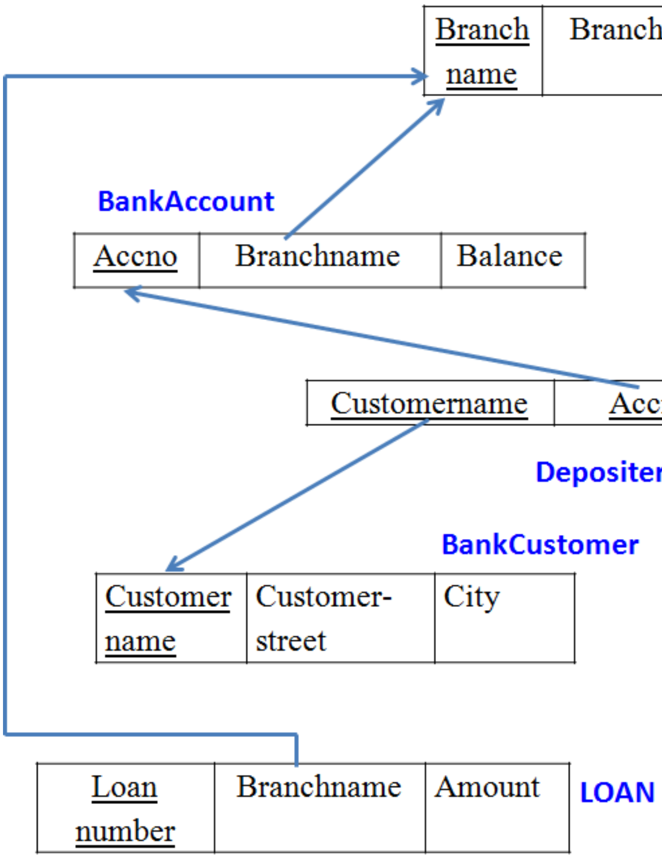
Specification of Banking Database Application –

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 and 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 modeled) 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



Branch name	Branchcity	Assets

BankAccount

Accno	Branchname	Balance

Customername	Accno

Depositer

BankCustomer

Customer name	Customer-street	City

Loan number	Branchname	Amount

LOAN

Create database

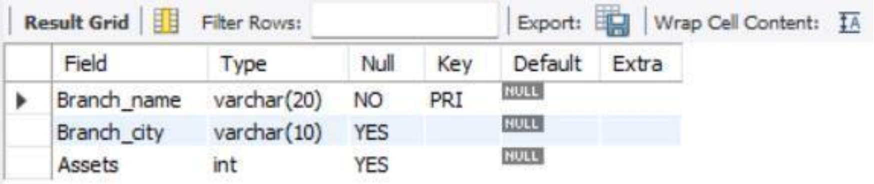
```
create database bank; use

bank;
```

Create table

```
create table branch (
Branch_name varchar(20),
Branch_city varchar(10),
Assets int,primary key (Branch_name)
);
create table bank_account(
Accno int,
Branch_name varchar(20),
balance int,
primary key (Accno),
foreign key (Branch_name)
references branch(Branch_name)
);
create  table  Bank_customer(
Customor_name  varchar(20),
Customor_street  varchar(20),
City varchar(10),
primary key(Customor_name)
);
create table depositer(
Customor_name varchar(20),
Accno int,foreign key
(Customor_name) references
Bank_customer(Customor
name), foreign key
(Accno)references
bank_account(Accno)
);
create table Loan(
lone_nm int,
Branch_name varchar(20),
Amount int,
primary key(lone_nm),
foreign key (Branch_name) references branch(Branch_name)
);
```

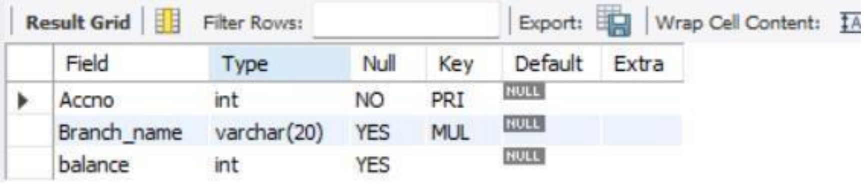
Structure of the table

- desc branch;



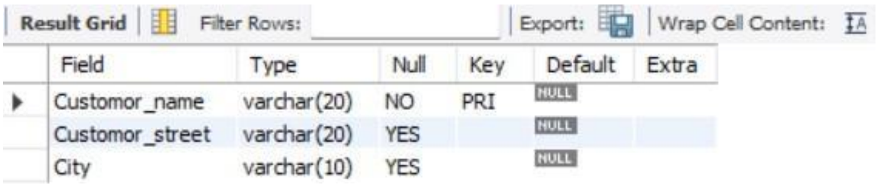
Field	Type	Null	Key	Default	Extra
Branch_name	varchar(20)	NO	PRI	NULL	
Branch_city	varchar(10)	YES		NULL	
Assets	int	YES		NULL	

- desc bank_account;



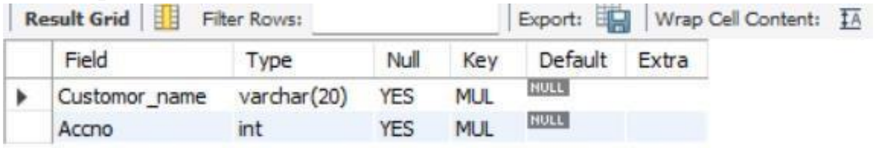
Field	Type	Null	Key	Default	Extra
Accno	int	NO	PRI	NULL	
Branch_name	varchar(20)	YES	MUL	NULL	
balance	int	YES		NULL	

- desc bank_customer;



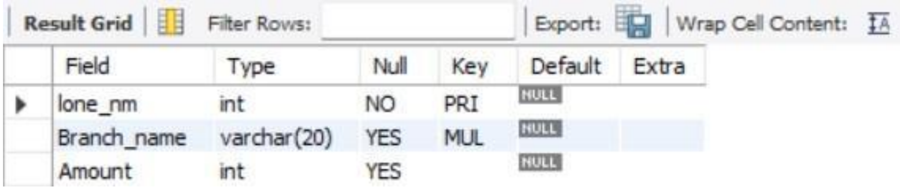
Field	Type	Null	Key	Default	Extra
Customor_name	varchar(20)	NO	PRI	NULL	
Customor_street	varchar(20)	YES		NULL	
City	varchar(10)	YES		NULL	

- desc depositor;



Field	Type	Null	Key	Default	Extra
Customor_name	varchar(20)	YES	MUL	NULL	
Accno	int	YES	MUL	NULL	

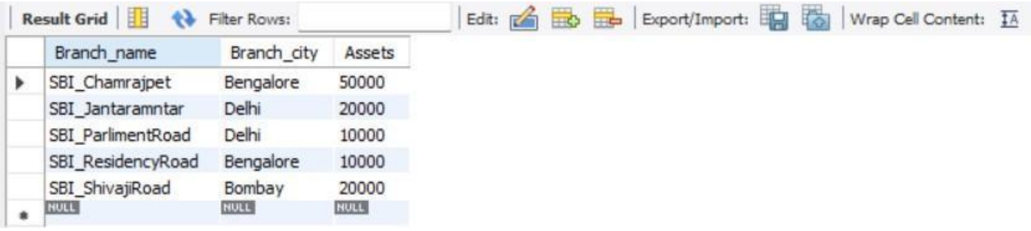
- desc lone;



Field	Type	Null	Key	Default	Extra
lone_nm	int	NO	PRI	NULL	
Branch_name	varchar(20)	YES	MUL	NULL	
Amount	int	YES		NULL	

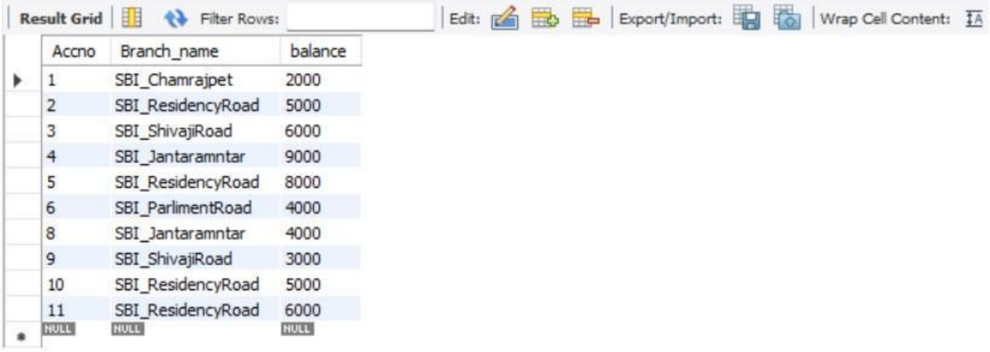
Inserting Values to the table

```
insert into branch values('SBI_Chamrajpet','Bengalore',50000);
insert into branch values('SBI_ResidencyRoad','Bengalore',10000);
insert into branch values('SBI_ShivajiRoad','Bombay',20000);
insert into branch values('SBI_ParlimentRoad','Delhi',10000);
insert into branch values('SBI_Jantaramntar','Delhi',20000);
select * from Branch;
```



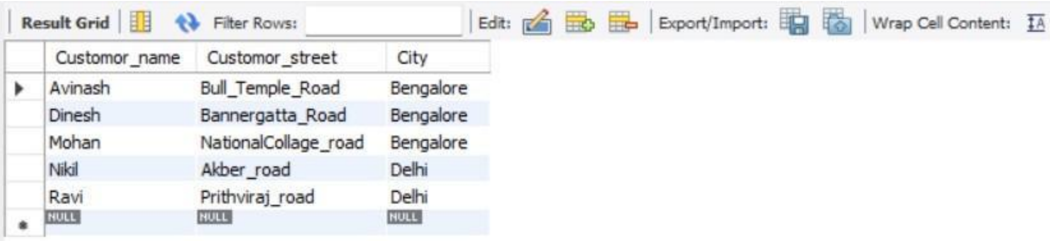
Branch_name	Branch_city	Assets
SBI_Chamrajpet	Bengalore	50000
SBI_Jantaramntar	Delhi	20000
SBI_ParlimentRoad	Delhi	10000
SBI_ResidencyRoad	Bengalore	10000
SBI_ShivajiRoad	Bombay	20000
NULL	NULL	NULL

```
insert into Bank_account values(1,'SBI_Chamrajpet',2000);
insert into Bank_account values(2,'SBI_ResidencyRoad',5000);
insert into Bank_account values(3,'SBI_ShivajiRoad',6000);
insert into Bank_account values(4,'SBI_Jantaramntar',9000);
insert into Bank_account values(5,'SBI_ResidencyRoad',8000);
insert into Bank_account values(6,'SBI_ParlimentRoad',4000);
insert into Bank_account values(8,'SBI_Jantaramntar',4000);
insert into Bank_account values(9,'SBI_ShivajiRoad',3000);
insert into Bank_account values(10,'SBI_ResidencyRoad',5000);
insert into Bank_account values(11,'SBI_ResidencyRoad',6000);
select * from Bank_account;
```



Accno	Branch_name	balance
1	SBI_Chamrajpet	2000
2	SBI_ResidencyRoad	5000
3	SBI_ShivajiRoad	6000
4	SBI_Jantaramntar	9000
5	SBI_ResidencyRoad	8000
6	SBI_ParlimentRoad	4000
8	SBI_Jantaramntar	4000
9	SBI_ShivajiRoad	3000
10	SBI_ResidencyRoad	5000
11	SBI_ResidencyRoad	6000
NULL	NULL	NULL

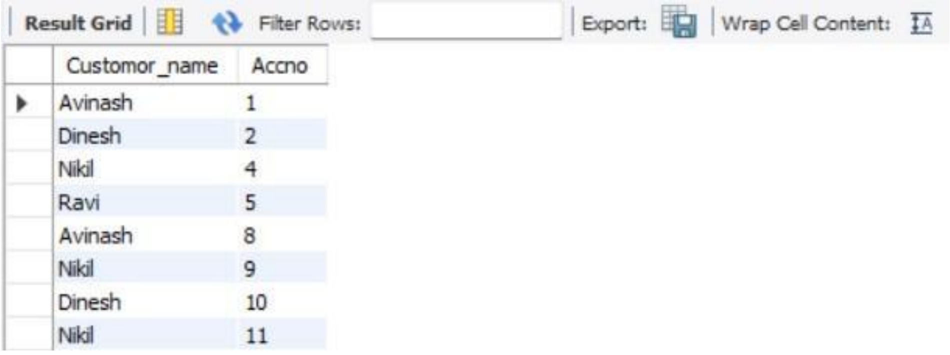
```
insert into Bank_customer values('Avinash', 'Bull_Temple_Road', 'Bengalore');
insert into Bank_customer values('Dinesh', 'Bannergatta_Road', 'Bengalore');
insert into Bank_customer values('Mohan', 'NationalCollage_road',
'Bengalore');
insert into Bank_customer values('Nikil', 'Akber_road', 'Delhi');
insert into Bank_customer values('Ravi', 'Prithviraj_road', 'Delhi');
select * from Bank_Customer;
```



Customor_name	Customor_street	City
Avinash	Bull_Temple_Road	Bengalore
Dinesh	Bannergatta_Road	Bengalore
Mohan	NationalCollage_road	Bengalore
Nikil	Akber_road	Delhi
Ravi	Prithviraj_road	Delhi
NULL	NULL	NULL

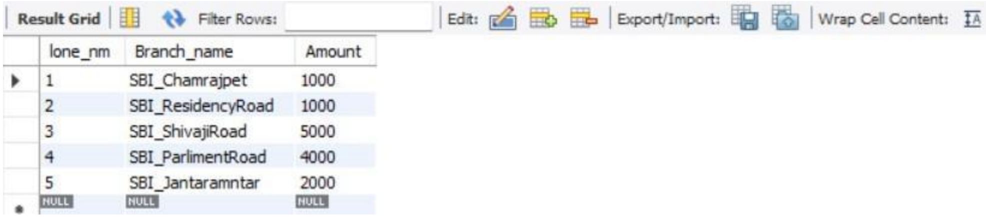
```
insert into depositer values('Avinash',1);
insert into depositer values('Dinesh',2);
insert into depositer values('Nikil',4);
insert into depositer values('Ravi',5);
insert into depositer values('Avinash',8);
insert into depositer values('Nikil',9);
insert into depositer values('Dinesh',10);
insert into depositer values('Nikil',11);
```

select * from depositer;



Customor_name	Accno
Avinash	1
Dinesh	2
Nikil	4
Ravi	5
Avinash	8
Nikil	9
Dinesh	10
Nikil	11

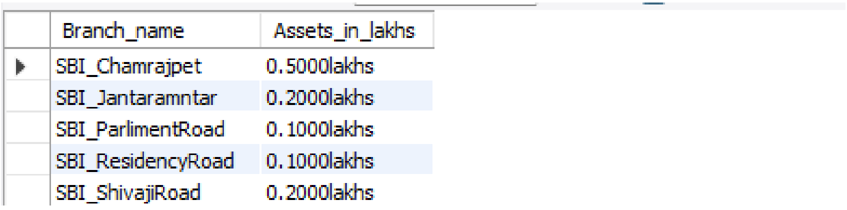
insert into loan values(1,'SBI_Chamrajpet',1000);
insert into loan values(2,'SBI_ResidencyRoad',1000);
insert into loan values(3,'SBI_ShivajiRoad',5000);
insert into loan values(4,'SBI_ParlimentRoad',4000);
insert into loan values(5,'SBI_Jantaramntar',2000);
select * from loan;



lone_nm	Branch_name	Amount
1	SBI_Chamrajpet	1000
2	SBI_ResidencyRoad	1000
3	SBI_ShivajiRoad	5000
4	SBI_ParlimentRoad	4000
5	SBI_Jantaramntar	2000
NULL	NULL	NULL

Queries

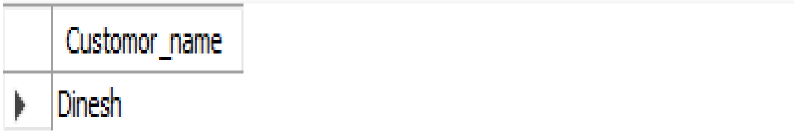
- **Display the branch name and assets from all branches in lakhs of rupees and rename the assets column to 'assets in lakhs'.**
    - select Branch_name,CONCAT(Assets/100000,'lakhs') Assets_in_lakhs from branch;



Branch_name	Assets_in_lakhs
SBI_Chamrajpet	0.5000lakhs
SBI_Jantaramntar	0.2000lakhs
SBI_ParlimentRoad	0.1000lakhs
SBI_ResidencyRoad	0.1000lakhs
SBI_ShivajiRoad	0.2000lakhs

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

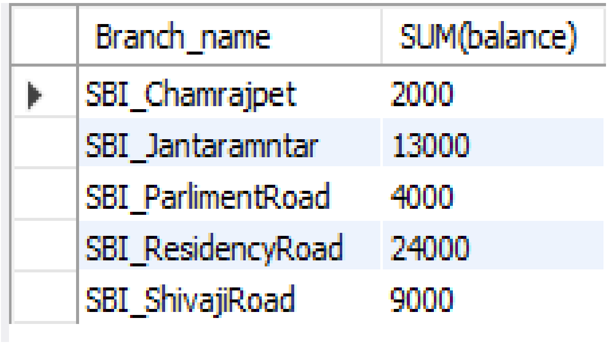
select d.Customor_name from depositer d, bank_account b
where b.Branch_name='SBI_ResidencyRoad' and d.Accno=b.Accno
group by d.Customor_name having count(d.Accno)>=2;



Customor_name
Dinesh

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

```
create view sum_of_loan
as select Branch_name, SUM(balance)
from bank_account
group by Branch_name;
    select * from sum_of_loan;
```



Branch_name	SUM(balance)
SBI_Chamrajpet	2000
SBI_Jantaramntar	13000
SBI_ParlimentRoad	4000
SBI_ResidencyRoad	24000
SBI_ShivajiRoad	9000

Experiment 4: MORE QUERIES ON BANK DATABASE

Question

i. Find the names of all branches that have greater assets than all branches located in Bangalore.

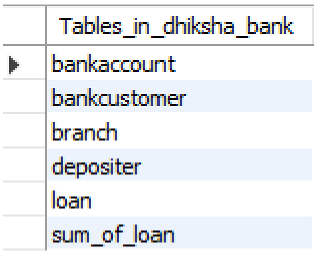
ii. Update the annual interest payments are made and all branches are to be increased by 5%.

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

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

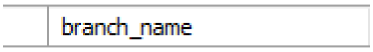
Queries

- **Find the names of all branches that have greater assets than all branches located in Bangalore.**
  - Select branch_name from branch where assets > all (select assets from branch where branch_city='Bengalore');



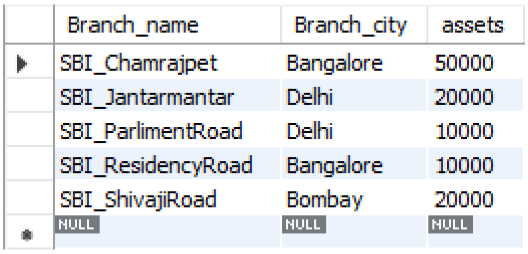
Tables_in_dhiksha_bank
bankaccount
bankcustomer
branch
depositer
loan
sum_of_loan

- **Update the annual interest payments are made and all branches are to be increased by 5%.**
  - Update bankaccount set balance = balance*1.05;



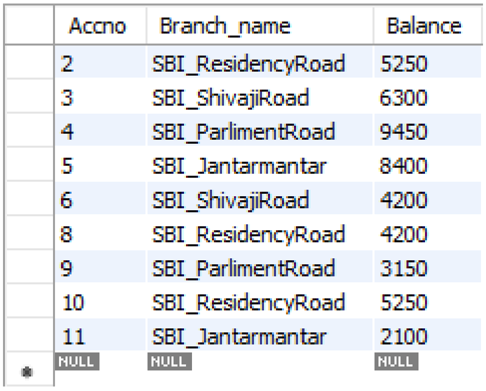
branch_name

- **Find all the customers who have an account at all the branches located in a specific city (Ex. Delhi)**
  - select distinct s.customername from depositer as s where not exists ((select branch_name from branch where branch_city = 'Delhi') except(select r.branch_name from depositer as t, bankaccount as r where t.accno = r.accno and S.customername = t.customername));



Branch_name	Branch_city	assets
SBI_Chamrajpet	Bangalore	50000
SBI_Jantarmantar	Delhi	20000
SBI_ParlimentRoad	Delhi	10000
SBI_ResidencyRoad	Bangalore	10000
SBI_ShivajiRoad	Bombay	20000
NULL	NULL	NULL

- **Demonstrate how you delete all account tuples at every branch located in a specific city (Ex.Bombay)**
    - delete from bankaccount where branch_name in (select branch_name from branch where branch_city = ‘Bombay’);



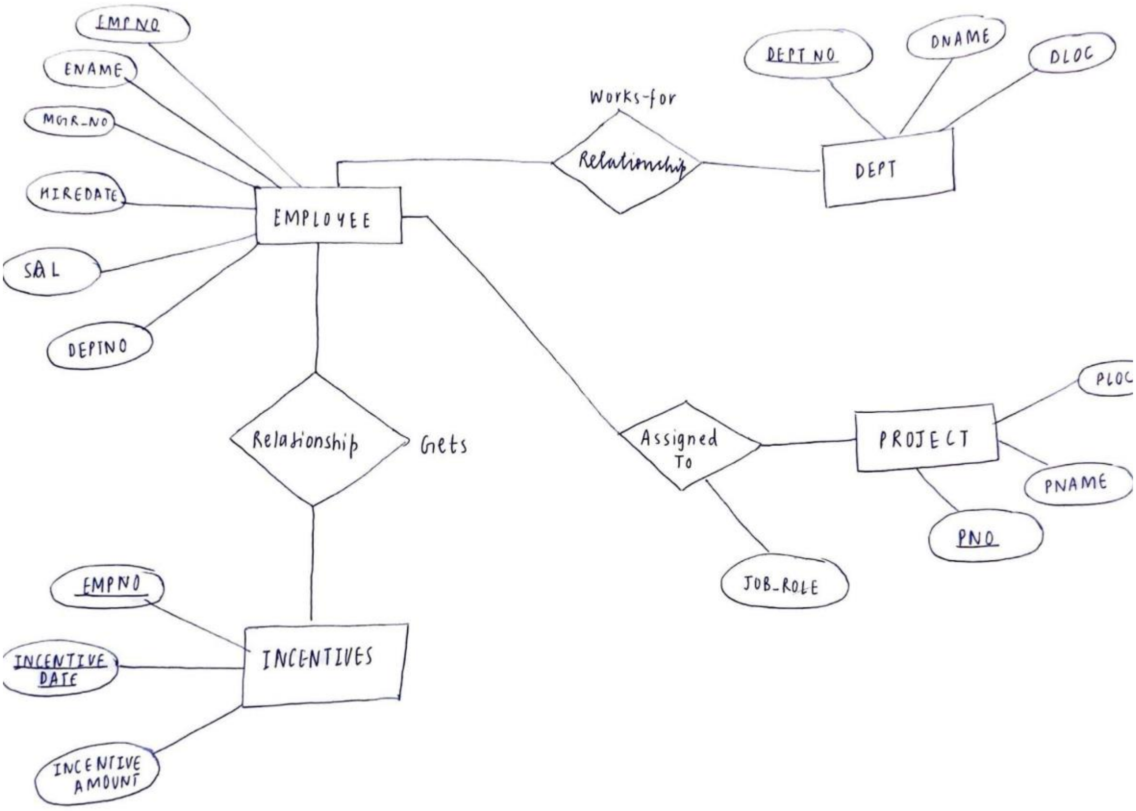
Accno	Branch_name	Balance
2	SBI_ResidencyRoad	5250
3	SBI_ShivajiRoad	6300
4	SBI_ParlimentRoad	9450
5	SBI_Jantarmantar	8400
6	SBI_ShivajiRoad	4200
8	SBI_ResidencyRoad	4200
9	SBI_ParlimentRoad	3150
10	SBI_ResidencyRoad	5250
11	SBI_Jantarmantar	2100
NULL	NULL	NULL

Experiment 5: EMPLOYEE DATABASE

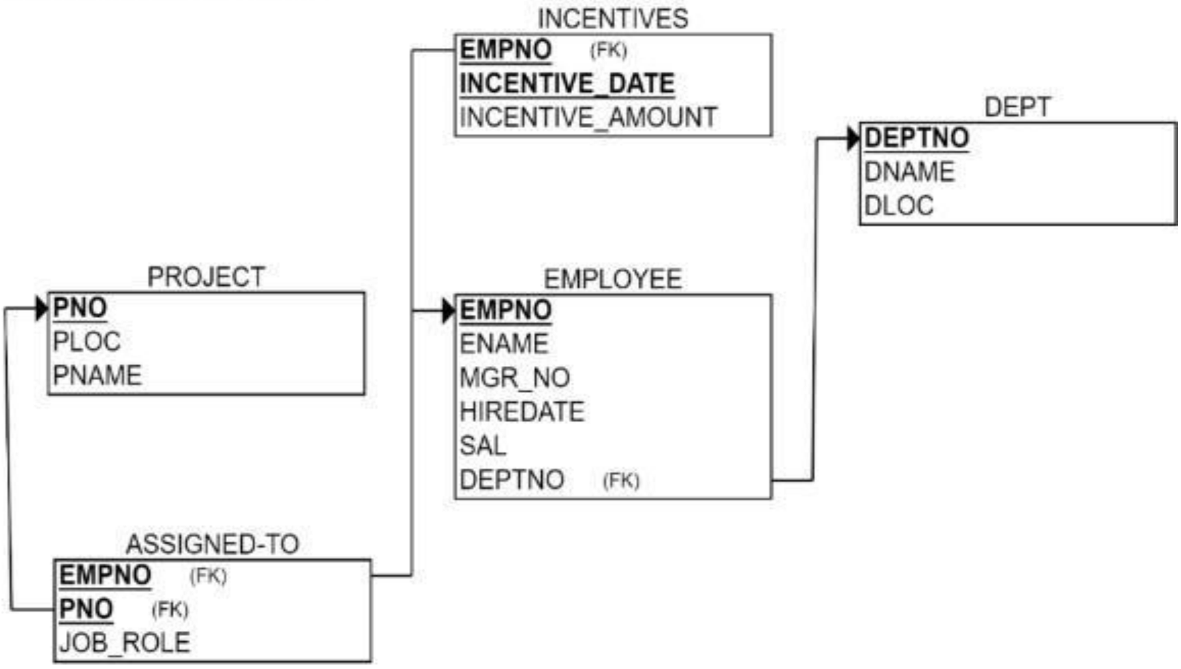
Specification of Employee Database-

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.

ER Diagram and Schema Diagram:-



Schema Diagram



INCENTIVES
EMPNO (FK)
INCENTIVE_DATE
INCENTIVE_AMOUNT

DEPT
DEPTNO
DNAME
DLOC

PROJECT
PNO
PLOC
PNAME

EMPLOYEE
EMPNO
ENAME
MGR_NO
HIREDATE
SAL
DEPTNO (FK)

ASSIGNED-TO
EMPNO (FK)
PNO (FK)
JOB_ROLE

Create database

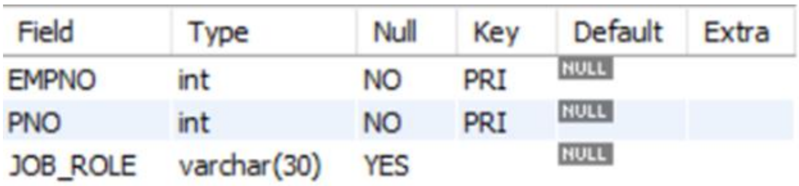
```
create database emp;
use emp;
```

Create tables

```
    create table dept(
    deptno decimal(2,0) primary key,
    dname      varchar(14)      default
    NULL,   loc   varchar(13)   default
    NULL
    );
    CREATE TABLE dept (
    deptno decimal(2,0) primary key,
    dname      varchar(14)      default
    NULL,   loc   varchar(13)   default
    NULL
    );
-   CREATE TABLE emp (
    empno decimal(4,0) primary key,
    ename varchar(10) default NULL,
    mgr_no decimal(4,0) default
    NULL, hiredate date default NULL,
    sal decimal(7,2) default NULL,
    deptno decimal(2,0) references dept(deptno) on delete cascade on update
    cascade
    );

    create table incentives (
    empno decimal(4,0) references emp(empno) on delete cascade on update
    cascade,
    incentive_date date, incentive_amount
    decimal(10,2), primary
    key(empno,incentive_date)
    );
    Create table project (
    pno int primary key,
    pname varchar(30) not null,
    ploc varchar(30)
    );
-   Create table assigned_to (
    empno decimal(4,0) references emp(empno) on delete cascade on update
cascade,pno int references project(pno) on delete cascade on update cascade,
job_role varchar(30),
primary key(empno,pno)
);
```

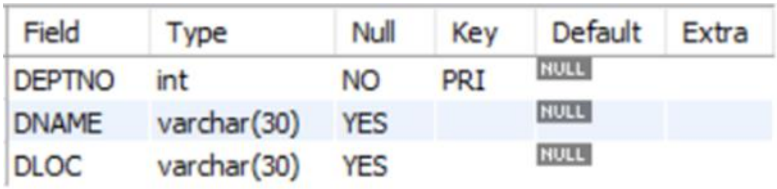
Structure of the table

desc assigned_to:



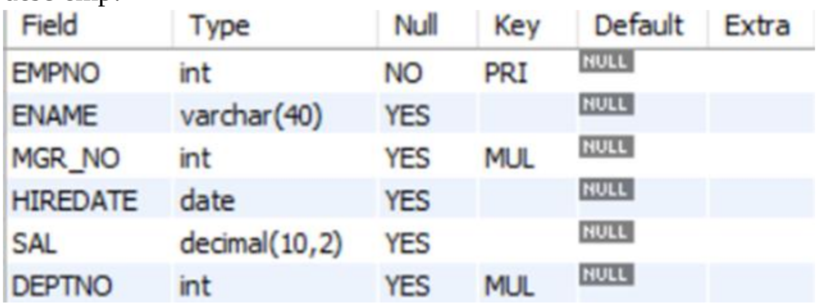
Field	Type	Null	Key	Default	Extra
EMPNO	int	NO	PRI	NULL	
PNO	int	NO	PRI	NULL	
JOB_ROLE	varchar(30)	YES		NULL	

desc dept:



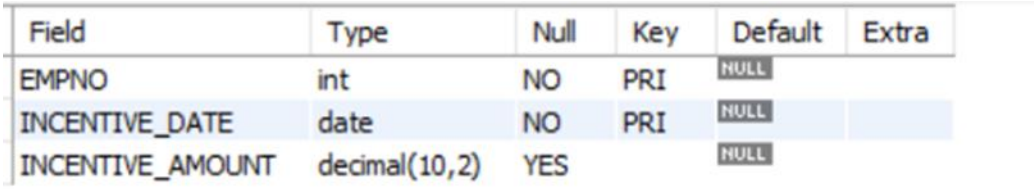
Field	Type	Null	Key	Default	Extra
DEPTNO	int	NO	PRI	NULL	
DNAME	varchar(30)	YES		NULL	
DLOC	varchar(30)	YES		NULL	

desc emp:



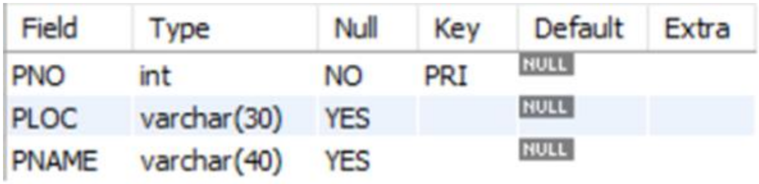
Field	Type	Null	Key	Default	Extra
EMPNO	int	NO	PRI	NULL	
ENAME	varchar(40)	YES		NULL	
MGR_NO	int	YES	MUL	NULL	
HIREDATE	date	YES		NULL	
SAL	decimal(10,2)	YES		NULL	
DEPTNO	int	YES	MUL	NULL	

desc incentives:



Field	Type	Null	Key	Default	Extra
EMPNO	int	NO	PRI	NULL	
INCENTIVE_DATE	date	NO	PRI	NULL	
INCENTIVE_AMOUNT	decimal(10,2)	YES		NULL	

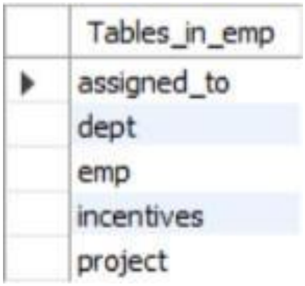
desc project:



Field	Type	Null	Key	Default	Extra
PNO	int	NO	PRI	NULL	
PLOC	varchar(30)	YES		NULL	
PNAME	varchar(40)	YES		NULL	

show tables

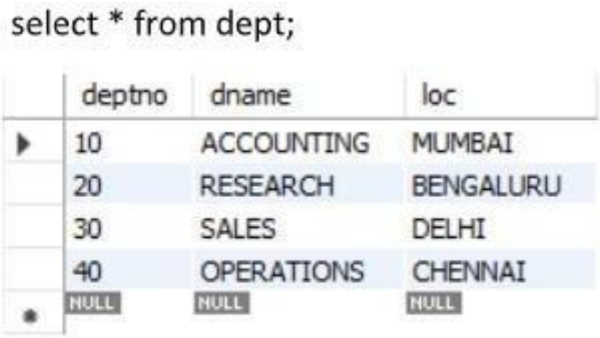
show tables;



	Tables_in_emp
▶	assigned_to
	dept
	emp
	incentives
	project

INSERT VALUES

INSERT INTO dept VALUES (10,'^Accounting','Mumbai');
INSERT INTO dept VALUES (20,'Research','Bengaluru');
INSERT INTO dept VALUES (30,'Sales','Delhi');
INSERT INTO dept VALUES (40,'Operations','Chennai');

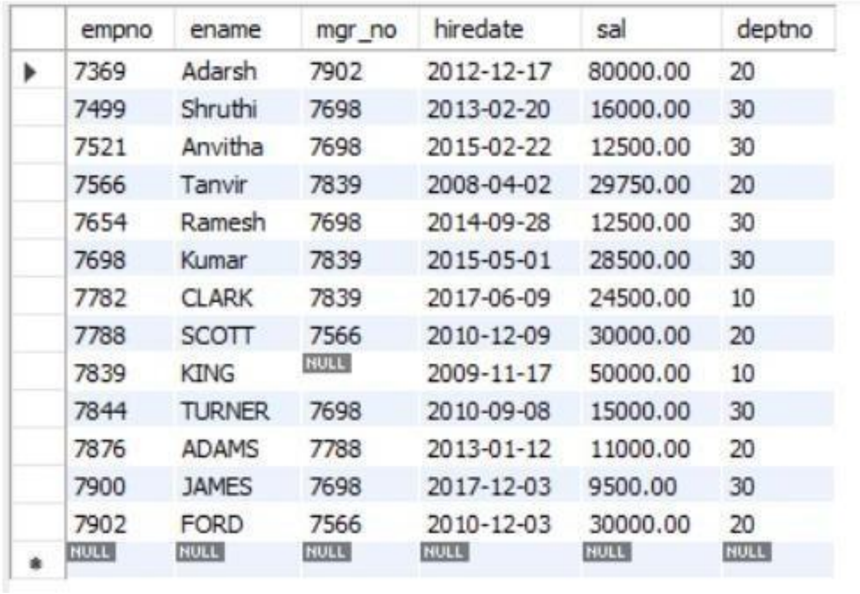


```
select * from dept;
```

	deptno	dname	loc
▶	10	ACCOUNTING	MUMBAI
	20	RESEARCH	BENGALURU
	30	SALES	DELHI
	40	OPERATIONS	CHENNAI
*	NULL	NULL	NULL

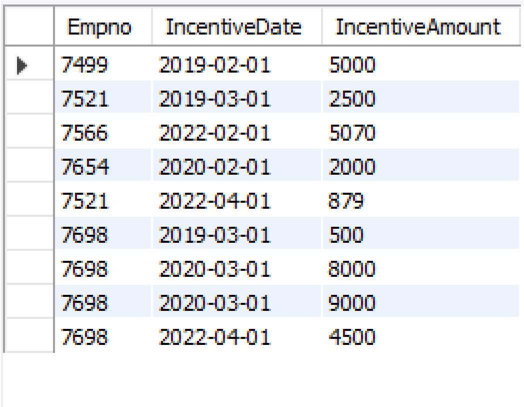
INSERT INTO employee VALUES (7369,'Adarch',7902,'2012-12-17',20,'80000.00');
INSERT INTO employee VALUES (7499,'Shruthi',7698,'2013-02- 20',30,'16000.00');
INSERT INTO employee VALUES (7521,'Anvitha',7698,'2015-02- 22',30,'12500.00');
INSERT INTO employee VALUES (7566,'Tanvir',7698,'2008-04- 02',20,'29750.00');
INSERT INTO employee VALUES (75654,'Ramesh',7698,'2014-09-28',30,'12500.00');
INSERT INTO employee VALUES (7698,'Kumar',7698,'2015-05- 01',30,'28500.00');
INSERT INTO employee VALUES (7782,'Clark',7698,'2017-06- 09',10,'24500.00');
INSERT INTO employee VALUES (7788,'Scott',7566,'2010-12- 09',20,'30000.00');
INSERT INTO employee VALUES (7844,'Turner',7698,'2010-09- 08',10,'15000.00');
INSERT INTO employee VALUES (7839,'KING',NULL,'2009-11- 17',10,'50000.00');
INSERT INTO employee VALUES (7876,'Adams',7788,'2013-01- 12',20,'11000.00');
INSERT INTO employee VALUES (7900,'James',7698,'2017-12- 03',20,'9500.00');
INSERT INTO employee VALUES (7902,'Ford',7566,'2010-12- 03',20,'30000.00');

```
select * from emp ;
```



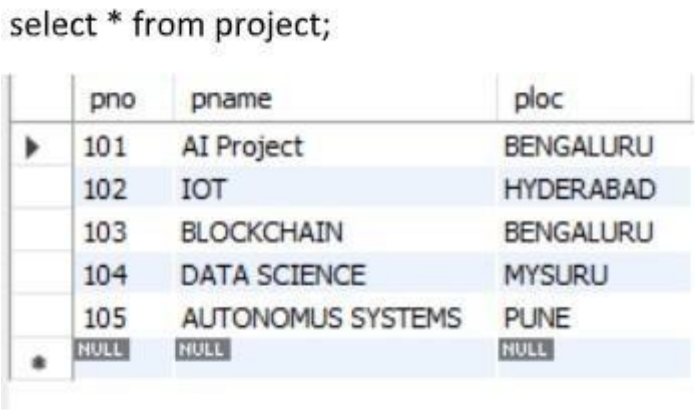
	empno	ename	mgr_no	hiredate	sal	deptno
▶	7369	Adarsh	7902	2012-12-17	80000.00	20
	7499	Shruthi	7698	2013-02-20	16000.00	30
	7521	Anvitha	7698	2015-02-22	12500.00	30
	7566	Tanvir	7839	2008-04-02	29750.00	20
	7654	Ramesh	7698	2014-09-28	12500.00	30
	7698	Kumar	7839	2015-05-01	28500.00	30
	7782	CLARK	7839	2017-06-09	24500.00	10
	7788	SCOTT	7566	2010-12-09	30000.00	20
	7839	KING	NULL	2009-11-17	50000.00	10
	7844	TURNER	7698	2010-09-08	15000.00	30
	7876	ADAMS	7788	2013-01-12	11000.00	20
	7900	JAMES	7698	2017-12-03	9500.00	30
	7902	FORD	7566	2010-12-03	30000.00	20
*	NULL	NULL	NULL	NULL	NULL	NULL

```
INSERT INTO incentives VALUES (7499,'2019-02-01',5000.00);
INSERT INTO incentives VALUES (7521,'2019-03-01',2500.00);
INSERT INTO incentives VALUES (7566,'2022-02-01',5070.00);
INSERT INTO incentives VALUES (7654,'2020-02-01',2000.00);
INSERT INTO incentives VALUES (7521,'2022-04-01',879.00);
INSERT INTO incentives VALUES (7698,'2019-03-01',500.00);
INSERT INTO incentives VALUES (7698,'2020-03-01',8000.00);
INSERT INTO incentives VALUES (7698,'2020-03-01',9000.00);
INSERT INTO incentives VALUES (7698,'2022-04-01',4500.00);
select * from incentives;
```



	Empno	IncentiveDate	IncentiveAmount
▶	7499	2019-02-01	5000
	7521	2019-03-01	2500
	7566	2022-02-01	5070
	7654	2020-02-01	2000
	7521	2022-04-01	879
	7698	2019-03-01	500
	7698	2020-03-01	8000
	7698	2020-03-01	9000
	7698	2022-04-01	4500

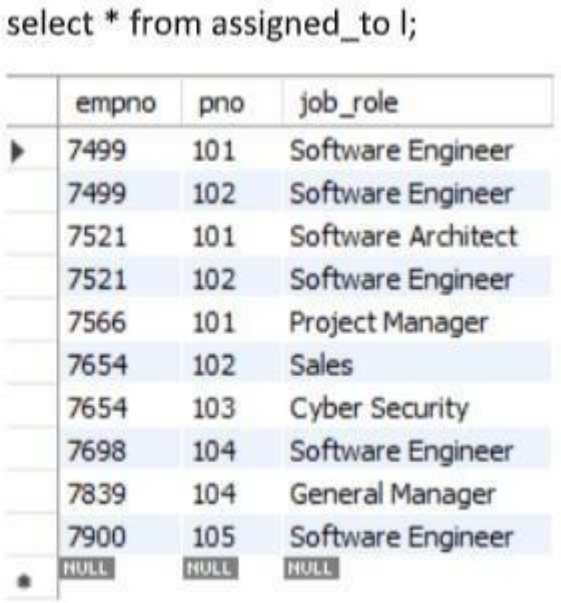
```
INSERT INTO project VALUES (101,'AI Project ','BENGALURU');
INSERT INTO project VALUES (102,'IOT','HYDERABAD');
INSERT INTO project VALUES (103,'BLOCKCHAIN','BENGALURU');
INSERT INTO project VALUES (104,'DATA SCIENCE','MYSURU');
INSERT INTO project VALUES (105,'AUTONOMOUS SYSTEMS','PUNE');
```



select * from project;

	pno	pname	ploc
▶	101	AI Project	BENGALURU
	102	IOT	HYDERABAD
	103	BLOCKCHAIN	BENGALURU
	104	DATA SCIENCE	MYSURU
	105	AUTONOMUS SYSTEMS	PUNE
*	NULL	NULL	NULL

```
INSERT INTO Assignedto VALUES (7499,101,'Software Engineer');
INSERT INTO Assignedto VALUES (7521,101,'Software Engineer');
INSERT INTO Assignedto VALUES (7566,101,'Project Manager');
INSERT INTO Assignedto VALUES (7654,102,'Sales');
INSERT INTO Assignedto VALUES (7521,102,'Software Engineer');
INSERT INTO Assignedto VALUES (7499,102,'Software Engineer');
INSERT INTO Assignedto VALUES (7654,103,'Cyber Security');
INSERT INTO Assignedto VALUES (7698,104,'Software Engineer');
INSERT INTO Assignedto VALUES (7900,105,'Software Engineer');
INSERT INTO Assignedto VALUES (7839,104,'General Manager');
```



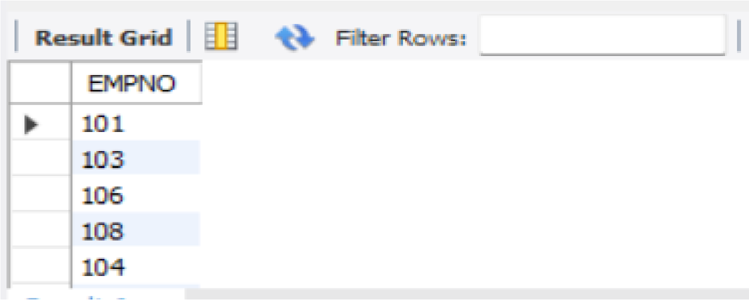
select * from assigned_to l;

	empno	pno	job_role
▶	7499	101	Software Engineer
	7499	102	Software Engineer
	7521	101	Software Architect
	7521	102	Software Engineer
	7566	101	Project Manager
	7654	102	Sales
	7654	103	Cyber Security
	7698	104	Software Engineer
	7839	104	General Manager
	7900	105	Software Engineer
*	NULL	NULL	NULL

Queries

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

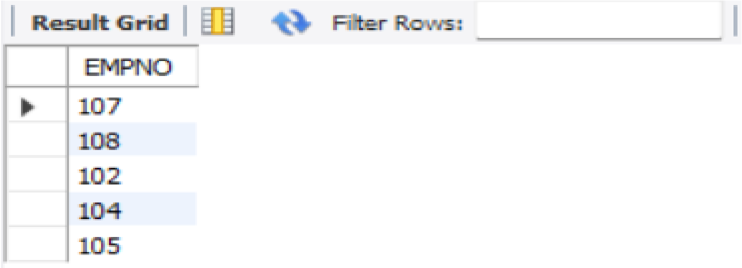
```
SELECT DISTINCT T1.EMPNO FROM ASSIGNED_TO T1 JOIN PROJECT T2
ON T1.PNO = T2.PNO WHERE T2.PLOC IN ('Bengaluru', 'Hyderabad', 'Mysuru');
```



EMPNO
101
103
106
108
104

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

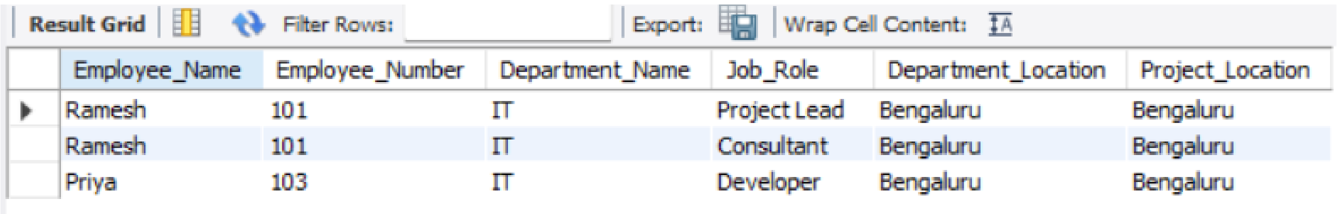
```
SELECT T1.EMPNO FROM EMPLOYEE T1
LEFT JOIN INCENTIVES T2 ON T1.EMPNO = T2.EMPNO
WHERE T2.EMPNO IS NULL;
```



EMPNO
107
108
102
104
105

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

```
SELECT E.ENAME AS Employee_Name, E.EMPNO AS Employee_Number,
 D.DNAME AS Department_Name, A.JOB_ROLE AS Job_Role,
D.DLOC AS Department_Location, P.PLOC AS Project_Location
FROM EMPLOYEE E JOIN DEPT D ON E.DEPTNO = D.DEPTNO
 JOIN ASSIGNED_TO A ON E.EMPNO = A.EMPNO
JOIN PROJECT P ON A.PNO = P.PNO
WHERE D.DLOC = P.PLOC;
```



Employee_Name	Employee_Number	Department_Name	Job_Role	Department_Location	Project_Location
Ramesh	101	IT	Project Lead	Bengaluru	Bengaluru
Ramesh	101	IT	Consultant	Bengaluru	Bengaluru
Priya	103	IT	Developer	Bengaluru	Bengaluru

Experiment 6: MORE QUERIES ON EMPLOYEE DATABASE

Questions

i. List the name of the managers with the most employees

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

iii. SQL Query to find the name of the top level manager of each department.

iv. SQL Query to find the employee details who got second maximum incentive in February 2019.

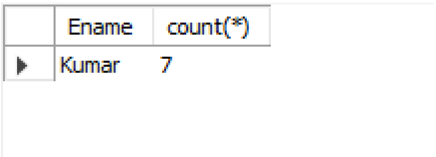
v. Display those employees who are working in the same dept where his manager is work?

vi. 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.

Queries

- **List the name of the managers with the most employees.**

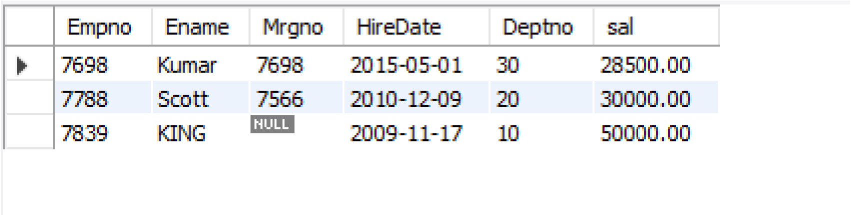
  -
    ```
    select e1.ename
    from employee e1, employee e2
    where e1.empno=e2.mgr_no group by e1.ename
    having count(e1.mgr_no)=(select count(e1.ename)
    from employee e1, employee e2 where e1.empno=e2.mgr_no group
    by e1.ename order by count(e1.ename) desc limit 1);
    ```



Ename	count(*)
Kumar	7

- **Display those managers name whose salary is more then average salary of his employee.**

```
select m.ename from employee m
where m.empno in (select mgr_no from employee) and m.sal>(select
avg(n.sal)
from employee n where n.mgr_no=m.empno);
```

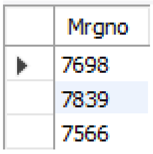


Empno	Ename	Mrgno	HireDate	Deptno	sal
7698	Kumar	7698	2015-05-01	30	28500.00
7788	Scott	7566	2010-12-09	20	30000.00
7839	KING	NULL	2009-11-17	10	50000.00

SQL query to find the name of the top level manager of each department.

```
Select distinct m.Mrgno from employee e,employee m
where e.mrgno =m.mrgno and e.deptno =m.deptno and e.empno in
```

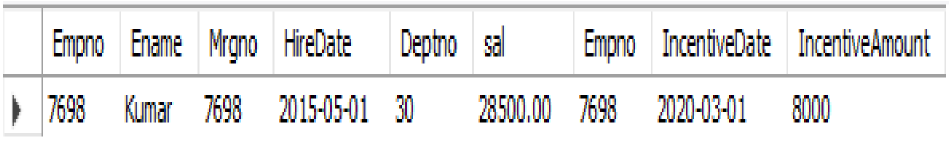
```
(select distinct m.mrgno from employee e, employee m where
e.mrgno=m.mrgnoan and e.deptno=m.dept));
```



Mrgno
7698
7839
7566

SQL query to find the employee details who got second maximum incentives in February 2019.

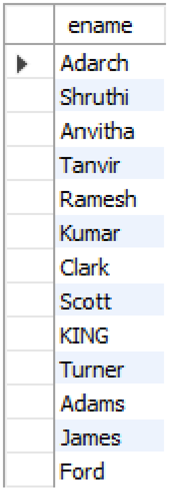
```
select * from employee where empno= (select i.empno from
incentives i where i.incentive_amount= (select
max(n.incentive_amount) from incentives n where
n.incentive_amount<(select max(inc.incentive_amount) from
incentives inc wherw inc.incentive_date between ‘2019-01-01’and
‘2019-12-31’) and incentive_date between ‘2019-01-01’and ‘2019-12-
31’));
```



Empno	Ename	Mrgno	HireDate	Deptno	sal	Empno	IncentiveDate	IncentiveAmount
7698	Kumar	7698	2015-05-01	30	28500.00	7698	2020-03-01	8000

Display those employees who are working in the same dept where this manager is work.

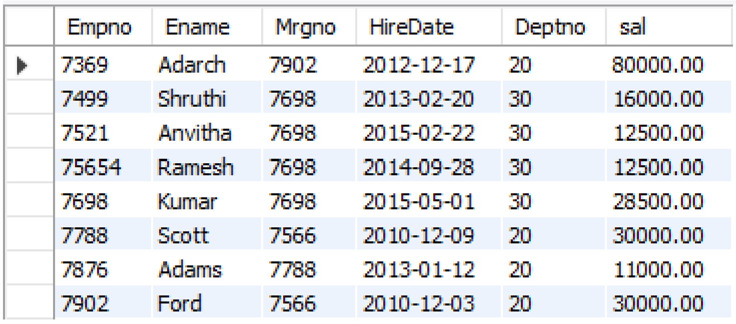
```
select e2.ename from employee e1, employee e2 where
e1.empno=e2.mgr_no and e1.deptno=e2.deptno;
```



ename
Adarch
Shruthi
Anvitha
Tanvir
Ramesh
Kumar
Clark
Scott
KING
Turner
Adams
James
Ford

- **Write a SQL Query to find those employees whose net pay are higher then or equal to the salary of any other employee in the company.**

- Select distinct e.ename from emp e,incentives I where (select max(sal+incentiveamount)from emp,incentive)>=any (select sal from emp e1 where e.deptno=e1.deptno);



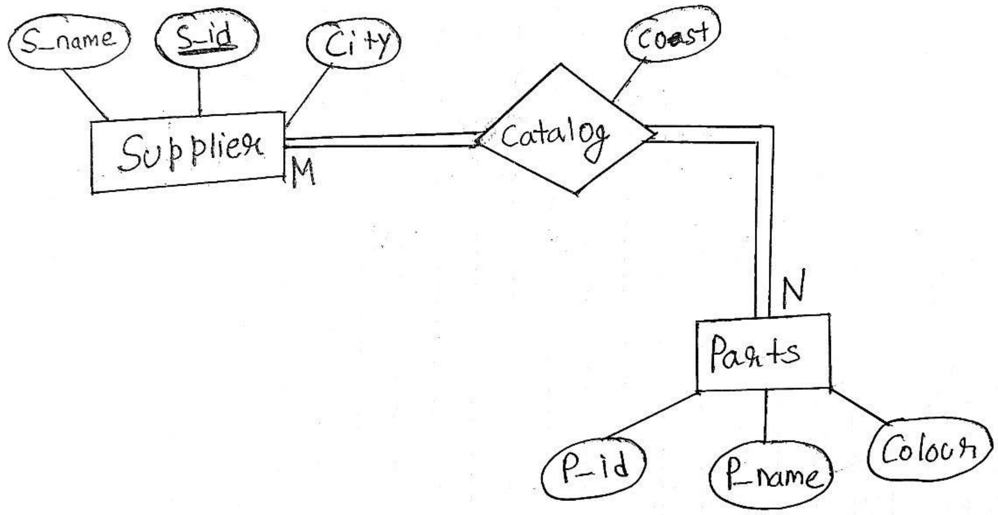
	Empno	Ename	Mrgno	HireDate	Deptno	sal
▶	7369	Adarch	7902	2012-12-17	20	80000.00
	7499	Shruthi	7698	2013-02-20	30	16000.00
	7521	Anvitha	7698	2015-02-22	30	12500.00
	75654	Ramesh	7698	2014-09-28	30	12500.00
	7698	Kumar	7698	2015-05-01	30	28500.00
	7788	Scott	7566	2010-12-09	20	30000.00
	7876	Adams	7788	2013-01-12	20	11000.00
	7902	Ford	7566	2010-12-03	20	30000.00

Experiment 7: SUPPLIER DATABASE

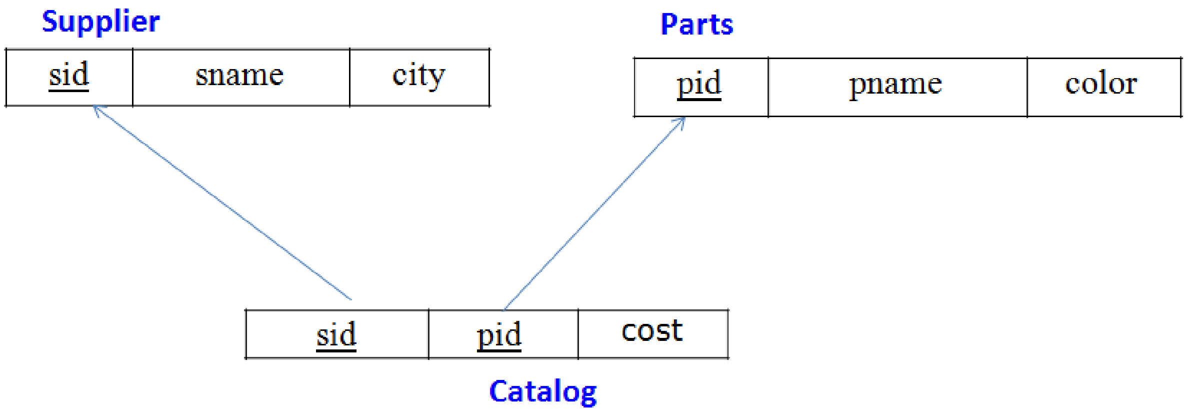
Specification of Supplier Database Application

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.

ER Diagram



Schema Diagram



Supplier (sid: int, sname: String, city: String)
Parts (pid: int, pname: String, color: String)
Catalog (sid: int, pid:
int, cost: int)

Create Database

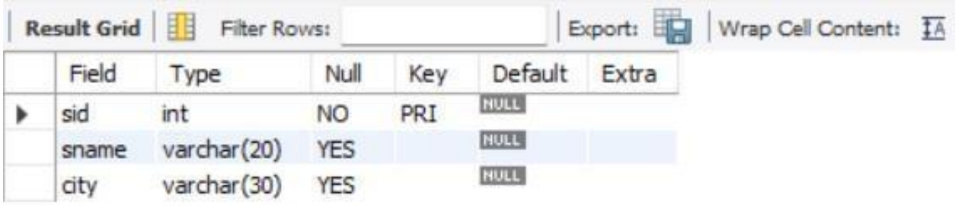
```
Create database supplierdatabase;
Use supplierdatabase;
```

Create table

```
create table supplier(
sid int primary key,
sname varchar(20),
city varchar(30));


create table parts(
pid int primary key,
pname varchar(20),
color varchar(20) );

create table catalog(
sid int,pid int, cost
int,foreign key(sid)
references
supplier(sid),
foreign key(pid) references part s(pid));
```

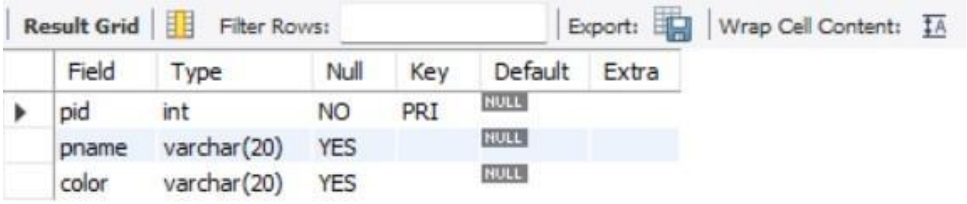
Structure of the table

- desc supplier;



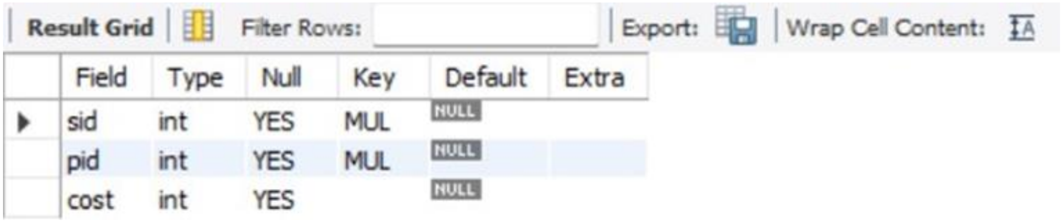
Field	Type	Null	Key	Default	Extra
sid	int	NO	PRI	NULL	
sname	varchar(20)	YES		NULL	
city	varchar(30)	YES		NULL	

- desc parts;



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

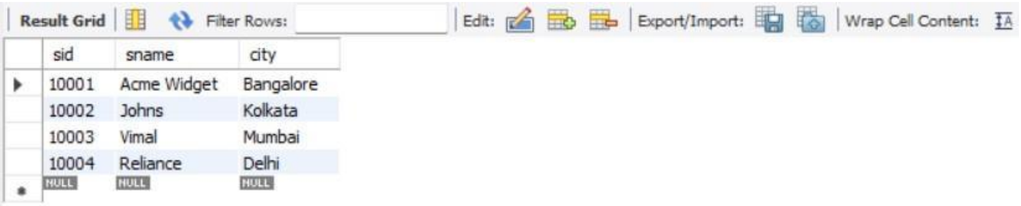
- desc catalog;



Field	Type	Null	Key	Default	Extra
sid	int	YES	MUL	NULL	
pid	int	YES	MUL	NULL	
cost	int	YES		NULL	

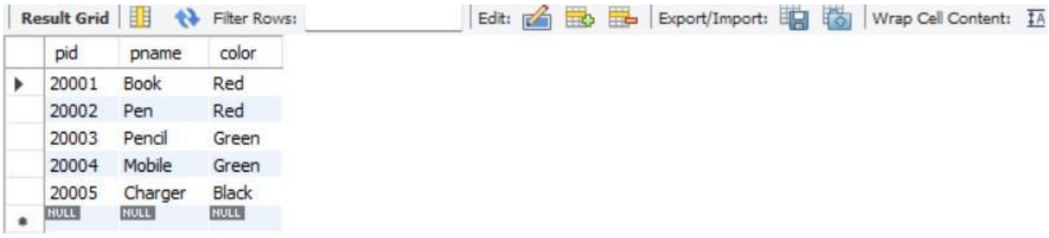
Insert values

- insert into supplier values (10001,"Acme Widget", "Bangalore"), (10002,"Johns", "Kolkata"), (10003,"Vimal", "Mumbai"),
  (10004,"Reliance","Delhi");
- select *from supplier;



sid	sname	city
10001	Acme Widget	Bangalore
10002	Johns	Kolkata
10003	Vimal	Mumbai
10004	Reliance	Delhi
NULL	NULL	NULL

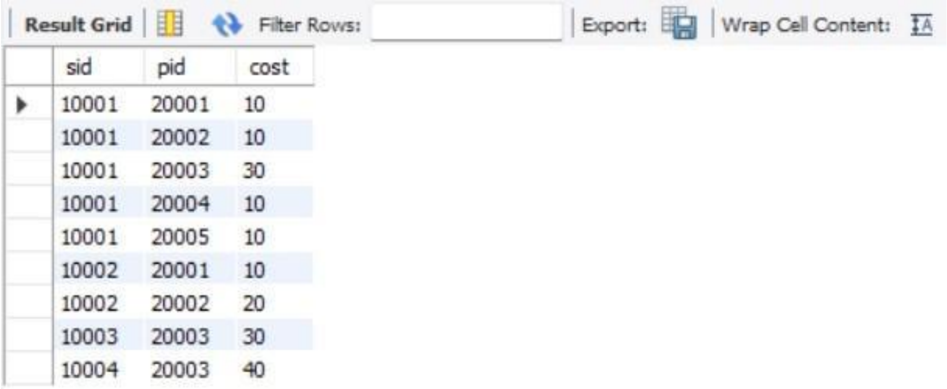
- insert into parts values (20001,"Book","Red"), (20002,"Pen","Red"), (20003,"Pencil","Green"), (20004,"Mobile","Green"), (20005,"Charger","Black");
- Select * from parts;



pid	pname	color
20001	Book	Red
20002	Pen	Red
20003	Pencil	Green
20004	Mobile	Green
20005	Charger	Black
NULL	NULL	NULL

- insert into catalog values (10001,20001,10), (10001,20002,10), (10001,20003,30), (10001,20004,10), (10001,20005,10), (10002,20001,10), (10002,20002,20), (10003,20003,30), (10004,20003,40);

- Select * from catalog;

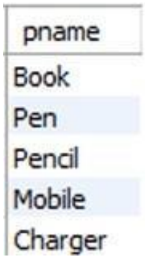


sid	pid	cost
10001	20001	10
10001	20002	10
10001	20003	30
10001	20004	10
10001	20005	10
10002	20001	10
10002	20002	20
10003	20003	30
10004	20003	40

Queries

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

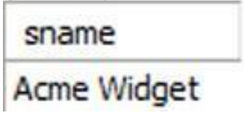
  select distinct pname from parts p,catalog c where p.pid=c.pid;



pname
Book
Pen
Pencil
Mobile
Charger

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

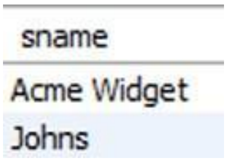
  select sname from Supplier where sid in(select sid from catalog c group by sid having count(pid)=(select count(pid) from parts));



sname
Acme Widget

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

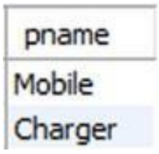
  select distinct sname from Supplier s,catalog c where s.sid=c.sid and pid in(select pid from parts where color="red");



sname
Acme Widget
Johns

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

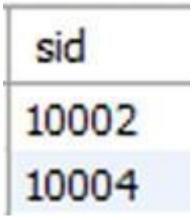
  select pname from parts p,supplier s where pid in(select pid from catalog group by pid having count(pid)=1) and s.sname="Acme Widget";



pname
Mobile
Charger

- **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).**

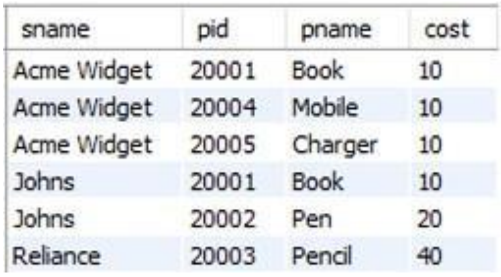
  create view c as select c.pid,p.pname,avg(cost) as co from catalog c,parts p where c.pid=p.pid group by c.pid; select ca.sid from catalog ca,c where ca.pid=c.pid and ca.cost>c.co and c.pid=ca.pid;



sid
10002
10004

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

select sname,co.pid,pname,cost from Supplier s,parts po,catalog co where co.pid=po.pid and s.sid=co.sid and co.cost =(select max(cost) from catalog where pid=po.pid) ;



sname	pid	pname	cost
Acme Widget	20001	Book	10
Acme Widget	20004	Mobile	10
Acme Widget	20005	Charger	10
Johns	20001	Book	10
Johns	20002	Pen	20
Reliance	20003	Pencil	40

Experiment 6: MORE QUERIES ON EMPLOYEE DATABASE

Questions

Find the most expensive part overall and the supplier who supplies it.

Find suppliers who do NOT supply any red parts.

Show each supplier and total value of all parts they supply.

Find suppliers who supply at least 2 parts cheaper than ₹20.

List suppliers who offer the cheapest cost for each part.

Create a view showing suppliers and the total number of parts they supply.

Create a view of the most expensive supplier for each part.

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

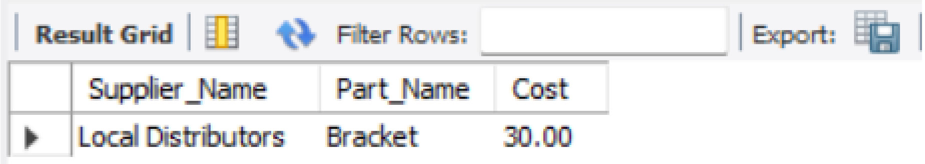
Create a trigger to set to default cost if not provided

- **Find the most expensive part overall and the supplier who supplies it**

```
SELECT T1.sname AS Supplier_Name, T3.pname AS Part_Name,
T2.cost AS Cost FROM Supplier T1 JOIN Catalog T2 ON T1.sid = T2.sid
JOIN Parts T3 ON T2.pid = T3.pid WHERE
T2.cost = (SELECT MAX(cost) FROM Catalog);
```

- **Find suppliers who do NOT supply any red parts.**

```
SELECT sname Supplier WHERE sid NOT IN (SELECT
T1.sid Catalog T1 JOIN Parts T2 ON T1.pid = T2.pid
```

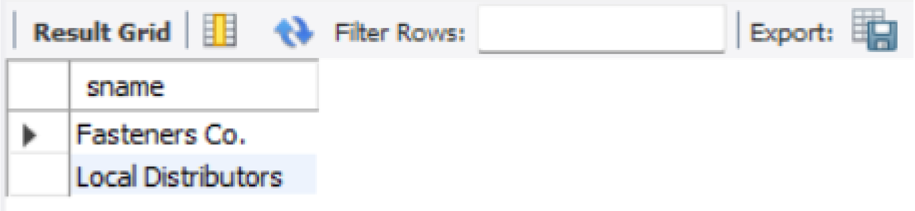


	Supplier_Name	Part_Name	Cost
▶	Local Distributors	Bracket	30.00

```
WHERE T2.color = 'Red');
```

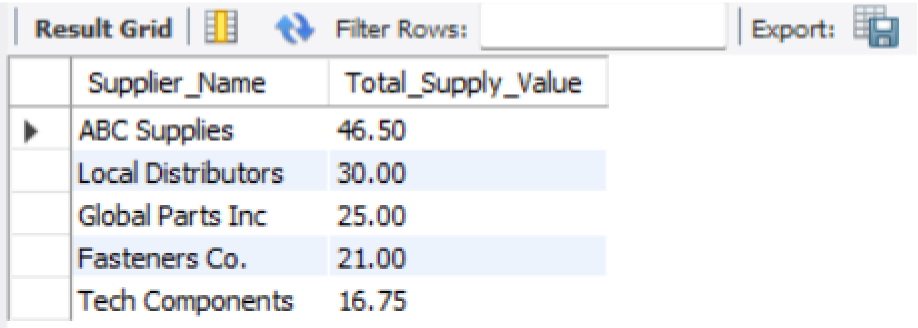
- **Show each supplier and total value of all parts they supply.**

```
SELECT T1.sname AS Supplier_Name, SUM(T2.cost) AS Total_Supply_Value
```



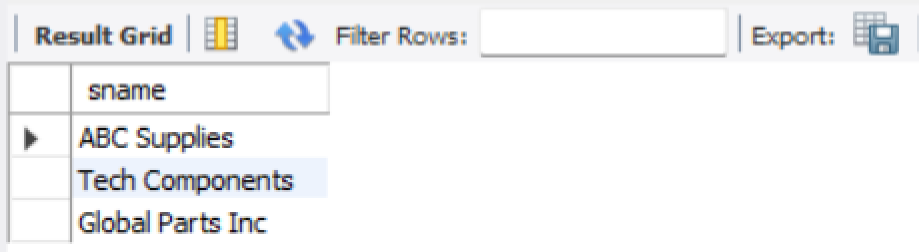
	sname
▶	Fasteners Co.
	Local Distributors

```
FROM Supplier T1 JOIN Catalog T2 ON T1.sid = T2.sid GROUP BY
T1.sname ORDER BY Total_Supply_Value DESC;
```



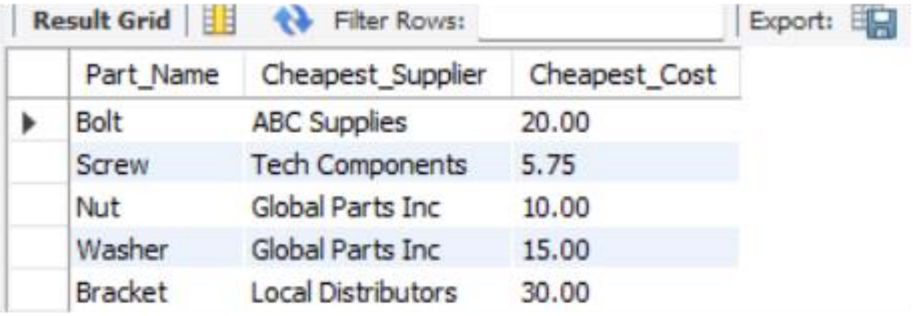
Supplier_Name	Total_Supply_Value
ABC Supplies	46.50
Local Distributors	30.00
Global Parts Inc	25.00
Fasteners Co.	21.00
Tech Components	16.75

- **Find suppliers who supply at least 2 parts cheaper than ₹20.**
  SELECT T1.sname FROM Supplier T1 JOIN (SELECT sid, pid Catalog WHERE cost < 20.00) AS Cheap_Parts ON T1.sid = Cheap_Parts.sid GROUP BY T1.sname HAVING COUNT(Cheap_Parts.pid) >= 2;



sname
ABC Supplies
Tech Components
Global Parts Inc

- **List suppliers who offer the cheapest cost for each part.**
  SELECT T1.pname AS Part_Name,T3.sname AS Cheapest_Supplier, T2.cost AS Cheapest_Cost FROM Parts T1 JOIN Catalog T2 ON T1.pid = T2.pid JOIN Supplier T3 ON T2.sid = T3.sidWHERE (T2.pid, T2.cost) IN (SELECT pid, MIN(cost) FROM Catalog GROUP BY pid);

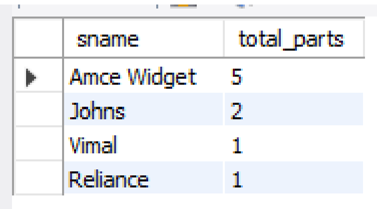


Part_Name	Cheapest_Supplier	Cheapest_Cost
Bolt	ABC Supplies	20.00
Screw	Tech Components	5.75
Nut	Global Parts Inc	10.00
Washer	Global Parts Inc	15.00
Bracket	Local Distributors	30.00

Create a view showing suppliers and the total number of parts they supply.

create view supplierpartcount as select s.sname,count(c.pid) as total_parts from supplier s join catalog c on s.sid=c.sid group by s.sname;

select * from supplierpartcount;

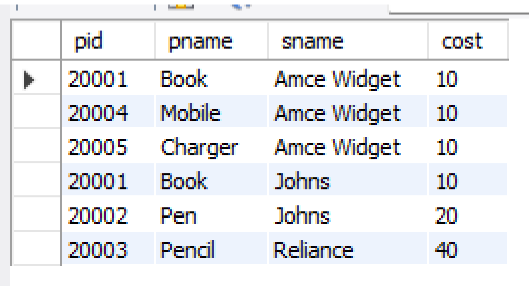


sname	total_parts
Amce Widget	5
Johns	2
Vimal	1
Reliance	1

Create a view of the most expensive supplier for each part.

create view mostExpensiveSupplier as select p.pid,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(cost) from catalog where pid=p.pid);

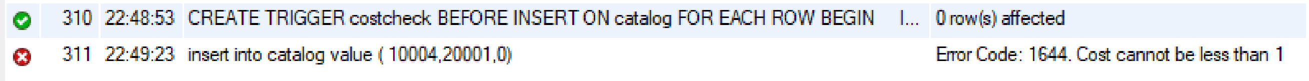
select * from mostExpensiveSupplier;



	pid	pname	sname	cost
▶	20001	Book	Amce Widget	10
	20004	Mobile	Amce Widget	10
	20005	Charger	Amce Widget	10
	20001	Book	Johns	10
	20002	Pen	Johns	20
	20003	Pencil	Reliance	40

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

```
CREATE TRIGGER costcheck
BEFORE INSERT ON catalog
FOR EACH ROW
BEGIN
   IF NEW.cost < 1 THEN
      SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'Cost cannot be less than 1';
   END IF;
END$$
DELIMITER ;
insert into catalog value ( 10004,20001,0);
```



```
✓  310  22:48:53  CREATE TRIGGER costcheck BEFORE INSERT ON catalog FOR EACH ROW BEGIN    I...  0 row(s) affected
✗  311  22:49:23  insert into catalog value ( 10004,20001,0)                                   Error Code: 1644. Cost cannot be less than 1
```

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

```
DELIMITER $$
create trigger defaultcost
before insert on catalog
for each row
begin
if new.cost is null then
set new.cost=50;
end if;
end;
```

Experiment 9: NO SQL - Customer Database

Question

Perform the following DB operations using MongoDB.

1. Create a collection by name Customers with the following attributes.

Cust_id, Acc_Bal, Acc_Type

2. Insert at least 5 values into the table

3. Write a query to display those records whose total account balance

is greater than 1200 of account type 'Checking' for each customer_id.

4. Determine Minimum and Maximum account balance for each

customer_id.

5. Export the created collection into local file system

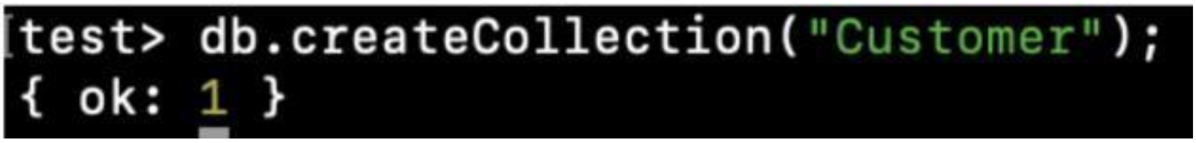
6. Drop the table

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

collection.

Create Table:

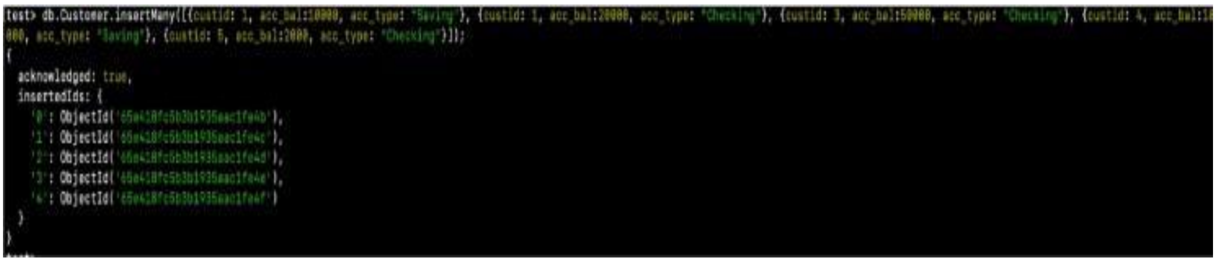
db.createCollection("Customer");



```
test> db.createCollection("Customer");
{ ok: 1 }
```

Inserting Values:

db.Customer.insertMany([{custid: 1, acc_bal:10000, acc_type: "Saving"}, {custid: 1, acc_bal:20000,acc_type: "Checking"}, {custid: 3, acc_bal:50000, acc_type: "Checking"}, {custid: 4, acc_bal:10000,acc_type: "Saving"}, {custid: 5, acc_bal:2000, acc_type: "Checking"}]

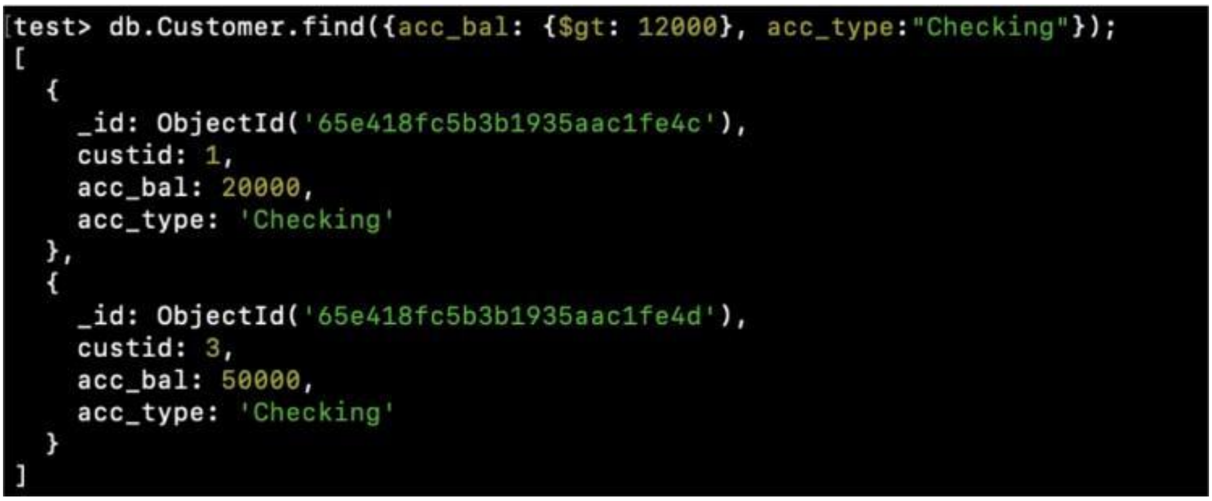


```
test> db.Customer.insertMany([{custid: 1, acc_bal:10000, acc_type: "Saving"}, {custid: 1, acc_bal:20000, acc_type: "Checking"}, {custid: 3, acc_bal:50000, acc_type: "Checking"}, {custid: 4, acc_bal:10
000, acc_type: "Saving"}, {custid: 5, acc_bal:2000, acc_type: "Checking"}]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65e418fc5b3b1935aac1fe4b'),
    '1': ObjectId('65e418fc5b3b1935aac1fe4c'),
    '2': ObjectId('65e418fc5b3b1935aac1fe4d'),
    '3': ObjectId('65e418fc5b3b1935aac1fe4e'),
    '4': ObjectId('65e418fc5b3b1935aac1fe4f')
  }
}
```

Queries:

● Finding all checking accounts with balance greater than 12000

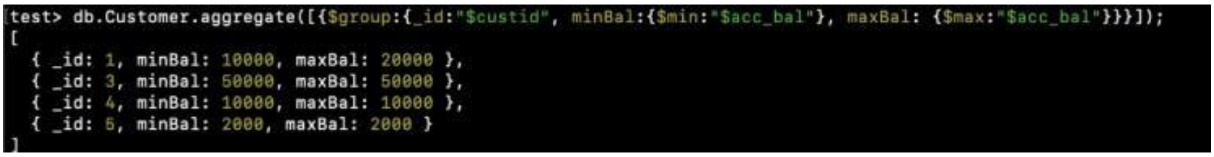
db.Customer.find({acc_bal: {$gt: 12000}, acc_type:"Checking"});



```
test> db.Customer.find({acc_bal: {$gt: 12000}, acc_type:"Checking"});
[
  {
    _id: ObjectId('65e418fc5b3b1935aac1fe4c'),
    custid: 1,
    acc_bal: 20000,
    acc_type: 'Checking'
  },
  {
    _id: ObjectId('65e418fc5b3b1935aac1fe4d'),
    custid: 3,
    acc_bal: 50000,
    acc_type: 'Checking'
  }
]
```

● Finding the maximum and minimum balance of each customer

db.Customer.aggregate([{$group:{_id:"$custid", minBal:{$min:"$acc_bal"}, maxBal: {$max:"$acc_bal"}}}]);



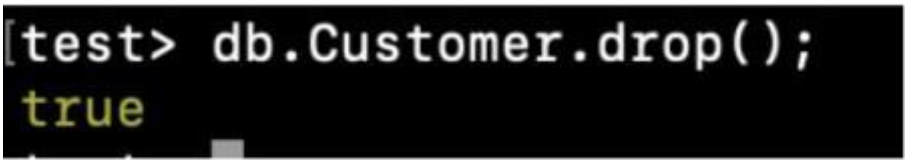
```
test> db.Customer.aggregate([{$group:{_id:"$custid", minBal:{$min:"$acc_bal"}, maxBal: {$max:"$acc_bal"}}}]);
[
  { _id: 1, minBal: 10000, maxBal: 20000 },
  { _id: 3, minBal: 50000, maxBal: 50000 },
  { _id: 4, minBal: 10000, maxBal: 10000 },
  { _id: 5, minBal: 2000, maxBal: 2000 }
]
```

● Exporting the collection to a json file

mongoexport mongodb+srv://204:<password>@cluster0.xbmgopf.mongodb.net/test --collection=Customer -- out C:\Users\nidhi\Documents\test.Customer.json

● Dropping collection “Customer”

db.Customer.drop();

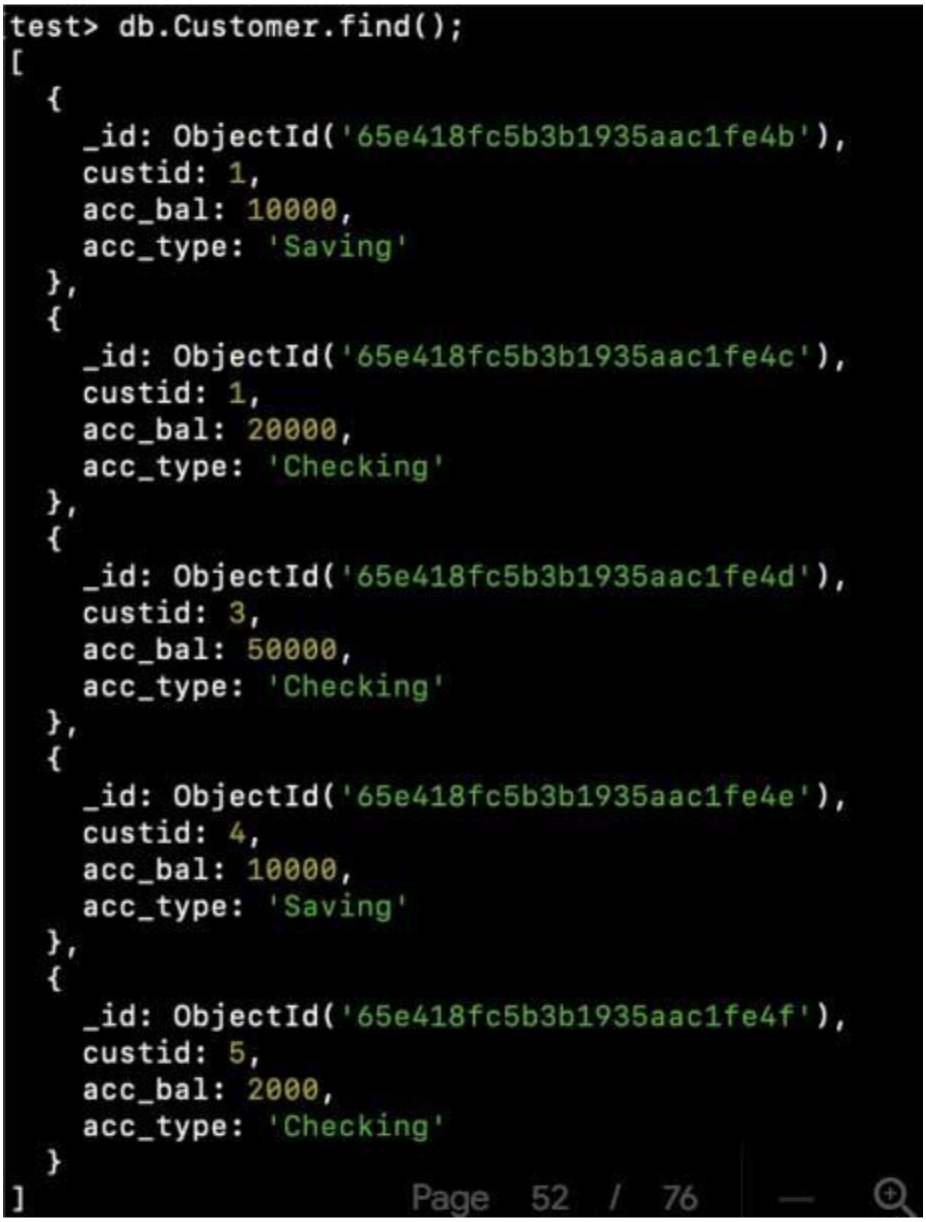


```
test> db.Customer.drop();
true
```

● **Exporting from a json file to the collection**

mongoimport mongodb+srv://204:<password>@cluster0.xbmgopf.mongodb.net/test

--collection=Customer -- type json -file C:\Users\nidhi\Documents\test.Customer.json

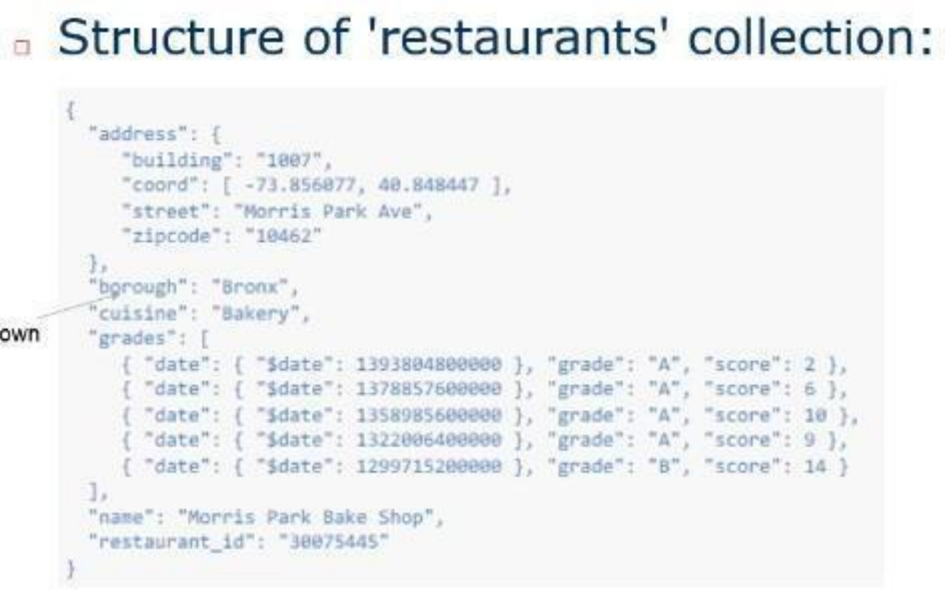


```
test> db.Customer.find();
[
  {
    _id: ObjectId('65e418fc5b3b1935aac1fe4b'),
    custid: 1,
    acc_bal: 10000,
    acc_type: 'Saving'
  },
  {
    _id: ObjectId('65e418fc5b3b1935aac1fe4c'),
    custid: 1,
    acc_bal: 20000,
    acc_type: 'Checking'
  },
  {
    _id: ObjectId('65e418fc5b3b1935aac1fe4d'),
    custid: 3,
    acc_bal: 50000,
    acc_type: 'Checking'
  },
  {
    _id: ObjectId('65e418fc5b3b1935aac1fe4e'),
    custid: 4,
    acc_bal: 10000,
    acc_type: 'Saving'
  },
  {
    _id: ObjectId('65e418fc5b3b1935aac1fe4f'),
    custid: 5,
    acc_bal: 2000,
    acc_type: 'Checking'
  }
]
```

Experiment 10: NO SQL – Restaurant Database

In MongoDB create a collection for

“Restaurant” and insert atleast 10 records



□ Structure of 'restaurants' collection:

```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

Questions

- Perform the following DB operations using MongoDB.

i. Write NoSQL Queries on “Restaurant” collection.

ii. Write a MongoDB query to display all the documents in the collection restaurants.

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

iv. 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.

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

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

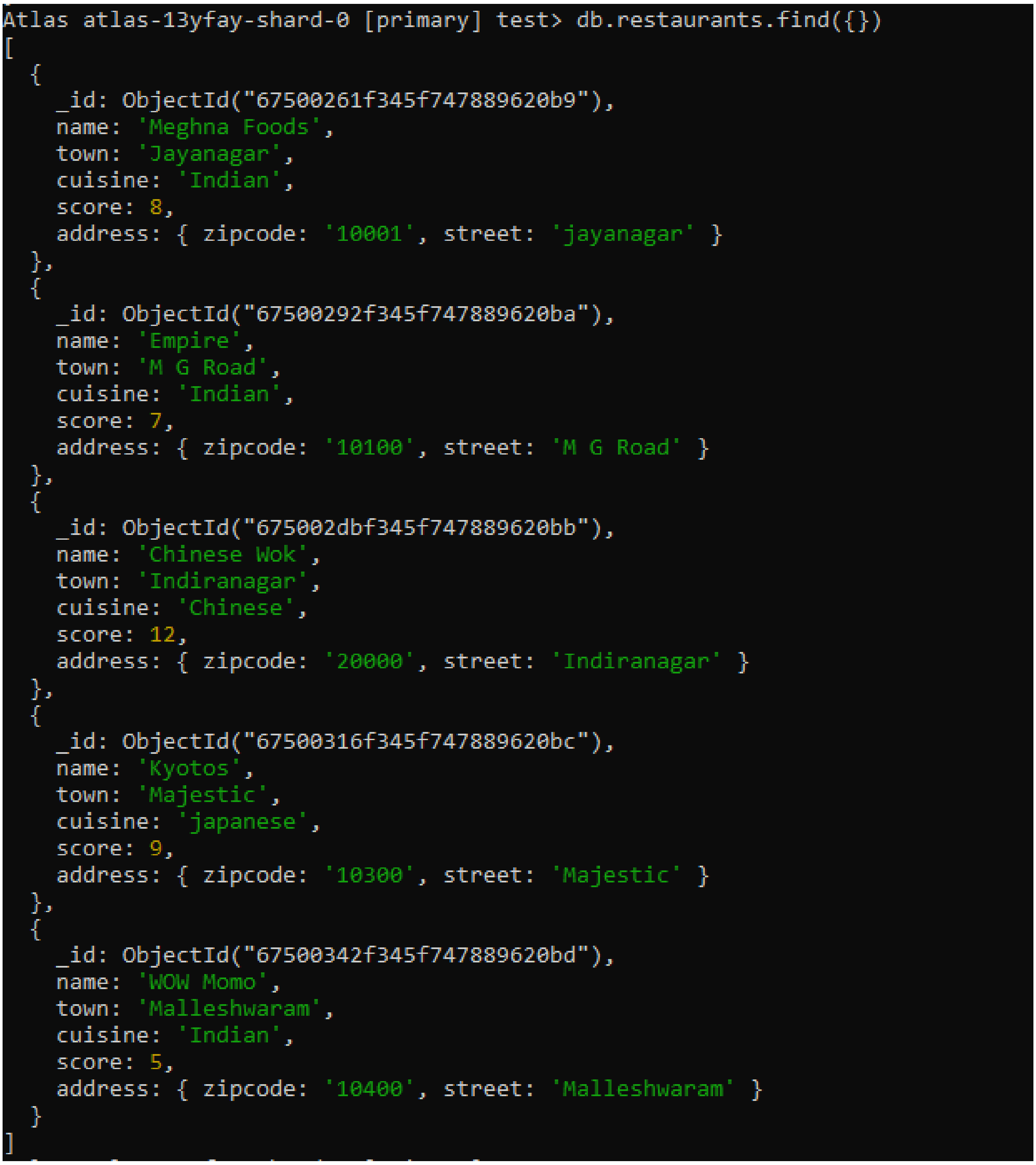
Creating Table:

```
db.createCollection("restaurants");
```

Inserting Values:

```
db.restaurants.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" }

} ]);
```

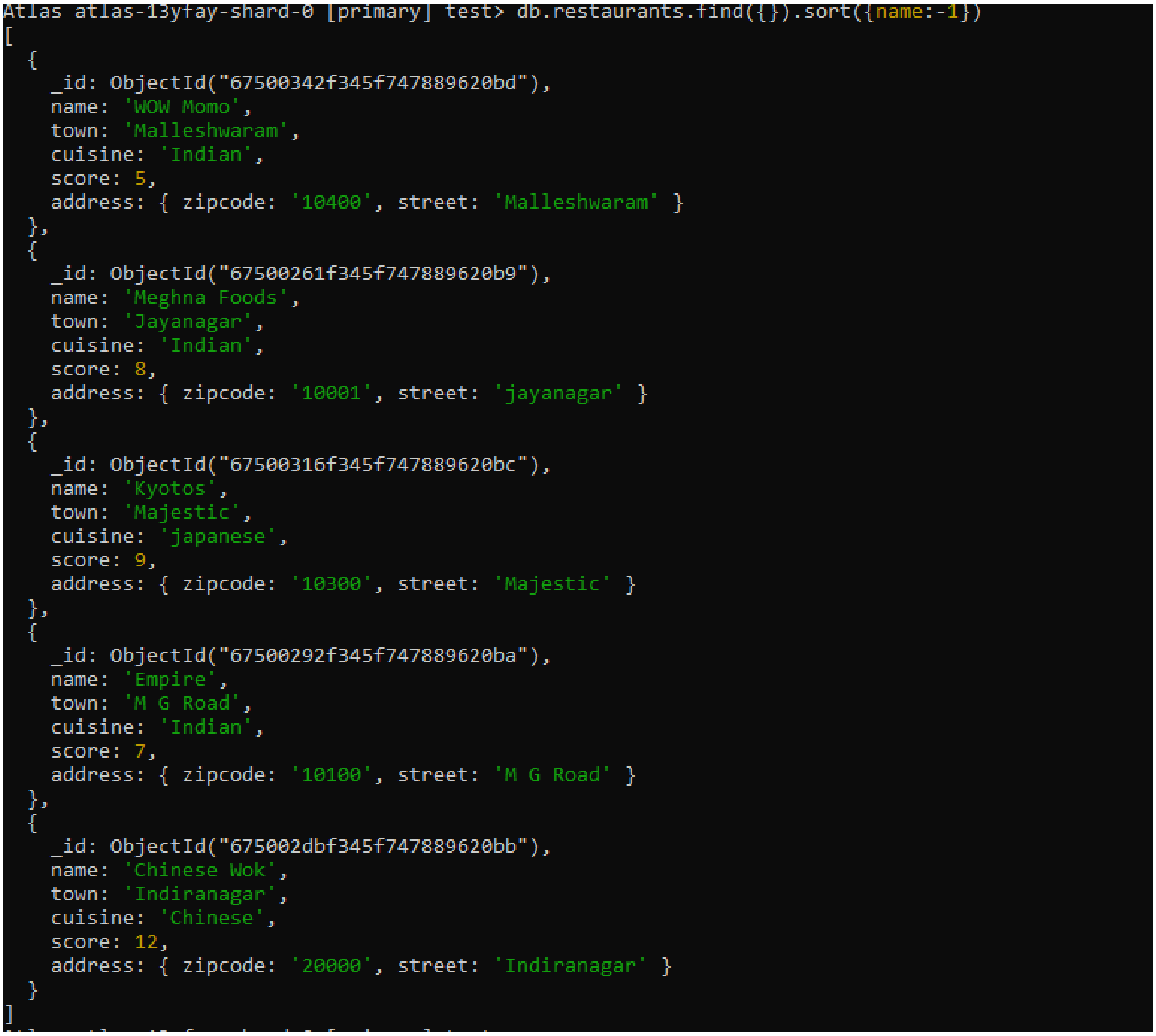
1) db.Restraunt.find()



```
Atlas atlas-13yfay-shard-0 [primary] test> db.restaurants.find({})
[
  {
    _id: ObjectId("67500261f345f747889620b9"),
    name: 'Meghna Foods',
    town: 'Jayanagar',
    cuisine: 'Indian',
    score: 8,
    address: { zipcode: '10001', street: 'jayanagar' }
  },
  {
    _id: ObjectId("67500292f345f747889620ba"),
    name: 'Empire',
    town: 'M G Road',
    cuisine: 'Indian',
    score: 7,
    address: { zipcode: '10100', street: 'M G Road' }
  },
  {
    _id: ObjectId("675002dbf345f747889620bb"),
    name: 'Chinese Wok',
    town: 'Indiranagar',
    cuisine: 'Chinese',
    score: 12,
    address: { zipcode: '20000', street: 'Indiranagar' }
  },
  {
    _id: ObjectId("67500316f345f747889620bc"),
    name: 'Kyotos',
    town: 'Majestic',
    cuisine: 'japanese',
    score: 9,
    address: { zipcode: '10300', street: 'Majestic' }
  },
  {
    _id: ObjectId("67500342f345f747889620bd"),
    name: 'WOW Momo',
    town: 'Malleshwaram',
    cuisine: 'Indian',
    score: 5,
    address: { zipcode: '10400', street: 'Malleshwaram' }
  }
]
```

2) Query to arrange the name of the restaurants in descending along with all the columns.

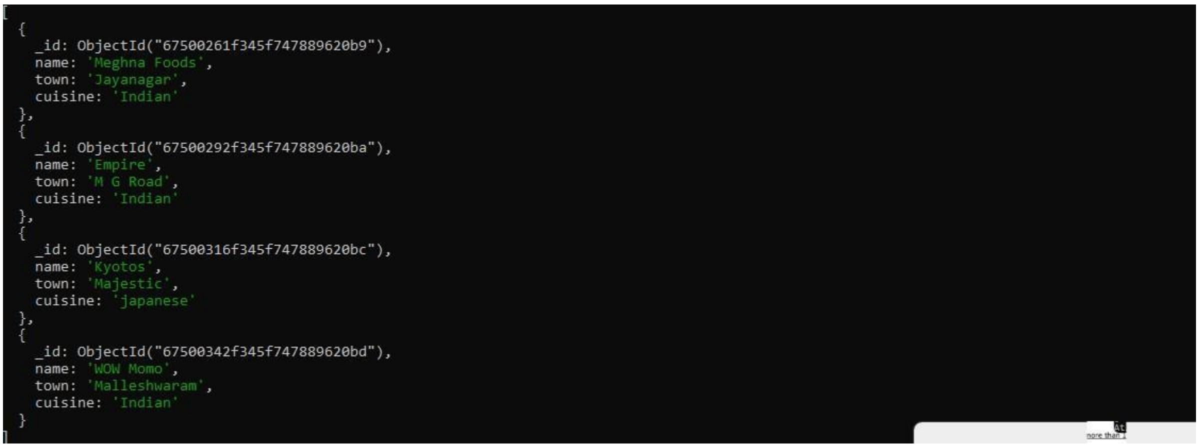
db.restaurants.find({}).sort({ name: -1 })



```
Atlas atlas-13yfay-shard-0 [primary] test> db.restaurants.find({}).sort({name:-1})
[
  {
    _id: ObjectId("67500342f345f747889620bd"),
    name: 'WOW Momo',
    town: 'Malleshwaram',
    cuisine: 'Indian',
    score: 5,
    address: { zipcode: '10400', street: 'Malleshwaram' }
  },
  {
    _id: ObjectId("67500261f345f747889620b9"),
    name: 'Meghna Foods',
    town: 'Jayanagar',
    cuisine: 'Indian',
    score: 8,
    address: { zipcode: '10001', street: 'jayanagar' }
  },
  {
    _id: ObjectId("67500316f345f747889620bc"),
    name: 'Kyotos',
    town: 'Majestic',
    cuisine: 'japanese',
    score: 9,
    address: { zipcode: '10300', street: 'Majestic' }
  },
  {
    _id: ObjectId("67500292f345f747889620ba"),
    name: 'Empire',
    town: 'M G Road',
    cuisine: 'Indian',
    score: 7,
    address: { zipcode: '10100', street: 'M G Road' }
  },
  {
    _id: ObjectId("675002dbf345f747889620bb"),
    name: 'Chinese Wok',
    town: 'Indiranagar',
    cuisine: 'Chinese',
    score: 12,
    address: { zipcode: '20000', street: 'Indiranagar' }
  }
]
```

3) Query to find the restaurant Id, name, town and cuisine for those restaurants which achieved a score which is not more than 10

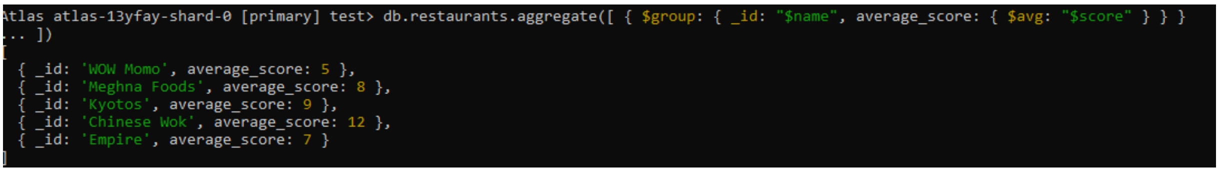
db.restaurants.find({ "score": { $lte: 10 } }, { _id: 1, name: 1, town: 1, cuisine: 1 })



```
[
  {
    _id: ObjectId("67500261f345f747889620b9"),
    name: 'Meghna Foods',
    town: 'Jayanagar',
    cuisine: 'Indian'
  },
  {
    _id: ObjectId("67500292f345f747889620ba"),
    name: 'Empire',
    town: 'M G Road',
    cuisine: 'Indian'
  },
  {
    _id: ObjectId("67500316f345f747889620bc"),
    name: 'Kyotos',
    town: 'Majestic',
    cuisine: 'japanese'
  },
  {
    _id: ObjectId("67500342f345f747889620bd"),
    name: 'WOW Momo',
    town: 'Malleshwaram',
    cuisine: 'Indian'
  }
]
```

4) Query to find the average score for each restaurant

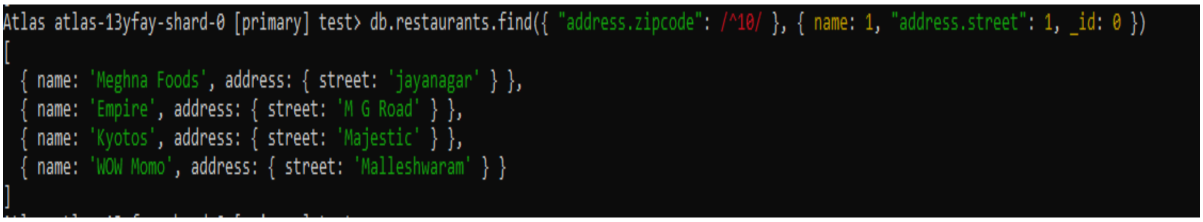
db.restaurants.aggregate([{ $group: { _id: "$name", average_score: { $avg: "$score" } } }])



```
Atlas atlas-13yfay-shard-0 [primary] test> db.restaurants.aggregate([ { $group: { _id: "$name", average_score: { $avg: "$score" } } }
... ])
[
  { _id: 'WOW Momo', average_score: 5 },
  { _id: 'Meghna Foods', average_score: 8 },
  { _id: 'Kyotos', average_score: 9 },
  { _id: 'Chinese Wok', average_score: 12 },
  { _id: 'Empire', average_score: 7 }
]
```

5) Query to find the name and address of the restaurants that have a zipcode that starts with '10'.

db.restaurants.find({ "address.zipcode": /^10/ }, { name: 1, "address.street": 1, _id: 0 })



```
Atlas atlas-13yfay-shard-0 [primary] test> db.restaurants.find({ "address.zipcode": /^10/ }, { name: 1, "address.street": 1, _id: 0 })
[
  { name: 'Meghna Foods', address: { street: 'jayanagar' } },
  { name: 'Empire', address: { street: 'M G Road' } },
  { name: 'Kyotos', address: { street: 'Majestic' } },
  { name: 'WOW Momo', address: { street: 'Malleshwaram' } }
]
```