VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Database Management Systems

(23CS3PCDBM)

Submitted by

Saney Vasudha sree(1BF24CS271)

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 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 **Saney Vasudha sree(1BF24CS271)69 ,** 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 2022. 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.

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

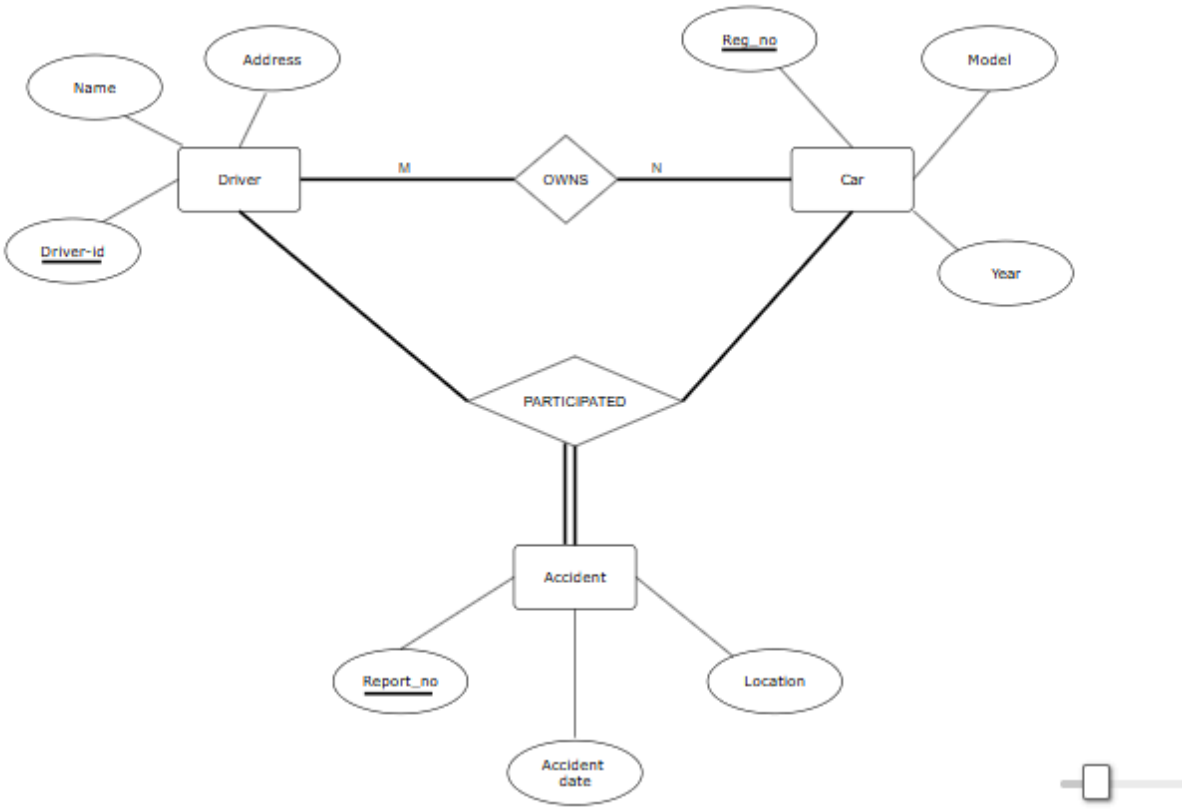
Index

Experiment 1: Insurance Database

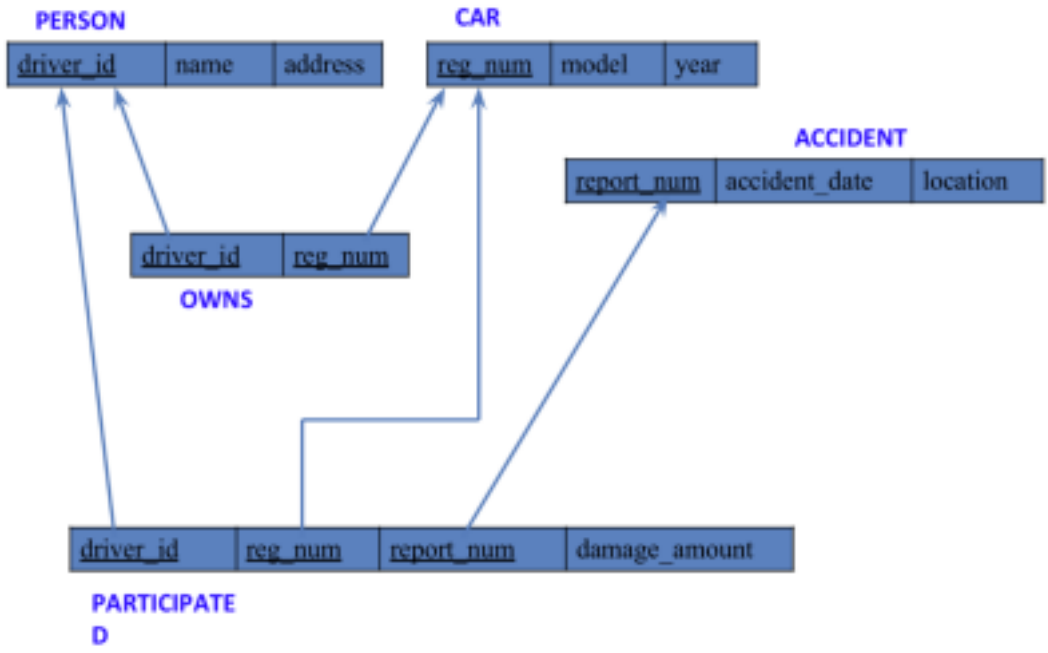
Specification of Insurance Database Application

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 (Draw it in hand)



Schema Diagram



- 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 insurancedatabase;

use insurancedatabase;
```

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;

Result Grid | Filter Rows: | Export:

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

desc accident;

Result Grid | Filter Rows: | Export: | Wra

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

desc participated;

Result Grid | Filter Rows: | Export: | Wrap

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	

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 owns;

Result Grid | Filter Rows: | Export:

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

Inserting Values to the table

insert into person 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');

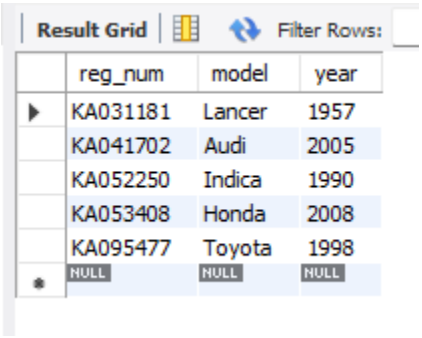
insert into person values('A05','John','Hanumanth Nagar');

select * from person;

Result Grid | Filter Rows:

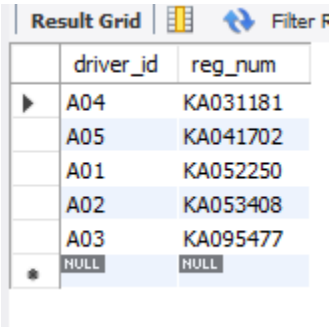
driver_id	name	address
A01	Richard	Srinivas Nagar
A02	Pradeep	Rajaji Nagar
A03	Smith	Ashok Nagar
A04	Venu	N R Colony
A05	John	Hanumanth Nagar
NULL	NULL	NULL

```
insert into car values('KA052250','Indica',1990);
insert into car values('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
NULL	NULL	NULL

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



driver_id	reg_num
A04	KA031181
A05	KA041702
A01	KA052250
A02	KA053408
A03	KA095477
NULL	NULL

```
insert into accident values(11,'01-01-03','Mysore Road');
```

```
insert into accident values(12,'02-02-04','SouthEnd Circle');
insert into accident values(13,'21-01-03','Mysore Road');
insert into accident values(14,'17-02-08','Bulltemple Road');
insert into accident values(15,'04-03-05','Kanakpura Road');
select * from accident;
```

Result Grid | Filter Rows:

report_num	accident_date	location
11	2001-01-03	Mysore Road
12	2002-02-04	SouthEnd Circle
13	2021-01-03	Mysore Road
14	2017-02-08	Bulltemple Road
15	2004-03-05	Kanakpura Road
NULL	NULL	NULL

```
insert into participated values('A01','KA052250',11,10000);
insert into participated values('A02','KA053408',12,50000);
insert into participated values('A03','KA095477',13,25000);
insert into participated values('A04','KA031181',14,3000);
insert into participated values('A05','KA041702',15,5000);
select * from participated;
```

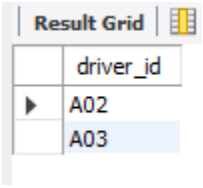
Result Grid | Filter Rows: | Edit:

driver_id	reg_num	report_num	damage_amount
A01	KA052250	11	10000
A02	KA053408	12	50000
A03	KA095477	13	25000
A04	KA031181	14	3000
A05	KA041702	15	5000
NULL	NULL	NULL	NULL

Queries (Question and answer should contain screen shot of the output)

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

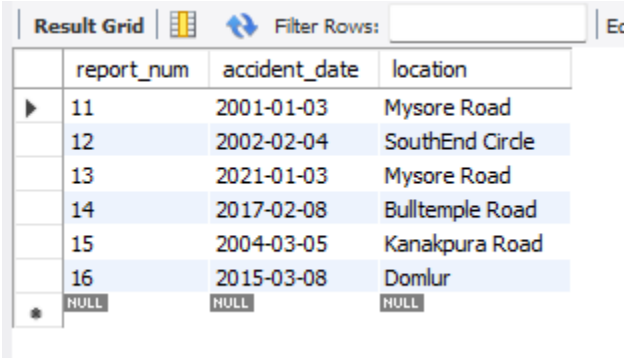
```
select driver_id
from participated
where damage_amount>=25000;
```



driver_id
A02
A03

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

```
insert into accident values (16,'15-03-08','Domlur');
select * from accident;
```

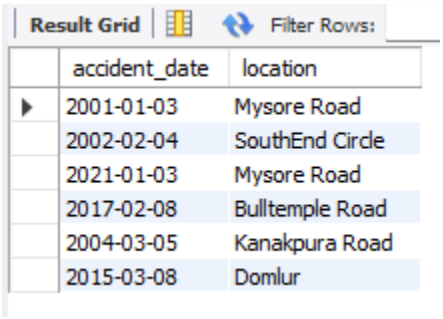


report_num	accident_date	location
11	2001-01-03	Mysore Road
12	2002-02-04	SouthEnd Circle
13	2021-01-03	Mysore Road
14	2017-02-08	Bulltemple Road
15	2004-03-05	Kanakpura Road
16	2015-03-08	Domlur
NULL	NULL	NULL

- **Display Accident date and location**

```
select accident_date, location

from accident;
```



Result Grid | Filter Rows:

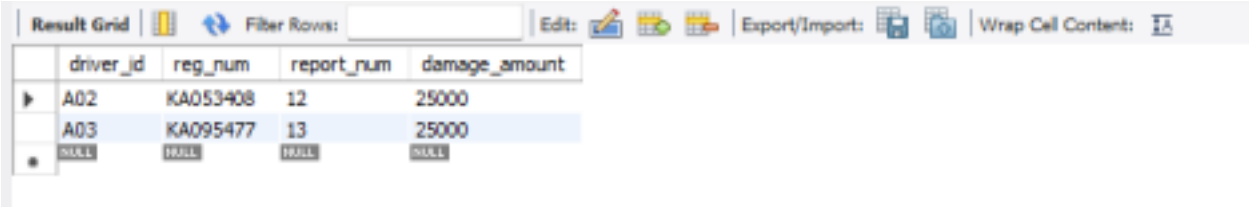
accident_date	location
2001-01-03	Mysore Road
2002-02-04	SouthEnd Circle
2021-01-03	Mysore Road
2017-02-08	Bulltemple Road
2004-03-05	Kanakpura Road
2015-03-08	Domlur

Experiment 2: More Queries on Insurance Database

Queries (Questions and output)

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

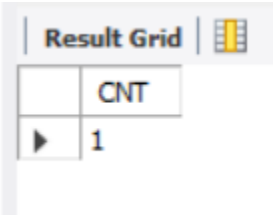
```
update participated
set damage_amount=25000
where reg_num='KA053408' and report_num=12;
```



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

- **Find the total number of people who owned cars that were involved in accidents in 2002.**

```
select count(distinct driver_id) CNT
from participated a, accident b
where a.report_num=b.report_num and b.accident_date like '2002%';
```



CNT
1

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

```
insert into accident values(16,'2008-03-08',"Domlur");
select * from accident;
```

Result Grid | Filter Rows:

report_num	accident_date	location
11	2001-01-03	Mysore Road
12	2002-02-04	SouthEnd Circle
13	2021-01-03	Mysore Road
14	2017-02-08	Bulltemple Road
15	2004-03-05	Kanakpura Road
16	2015-03-08	Domlur
NULL	NULL	NULL

- **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	KA053408	12	50000
A03	KA095477	13	25000
A01	KA052250	11	10000
A05	KA041702	15	5000
A04	KA031181	14	3000
NULL	NULL	NULL	NULL

- **FIND THE AVERAGE DAMAGE AMOUNT**

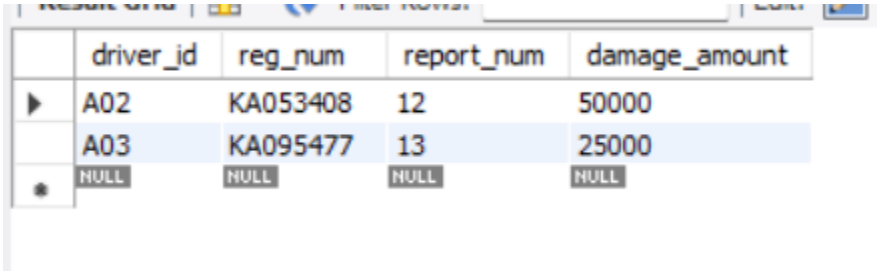
  select AVG(damage_amount)
  from participated;

AVG(damage_amount)
18600.0000

- **DELETE THE TUPLE FROM PARTICIPATED RELATION WHOSE DAMAGE AMOUNT IS BELOW THE AVERAGE DAMAGE AMOUNT**

  delete from participated

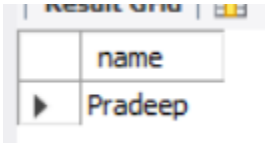
where damage_amount< (select avg_val from (select AVG(damage_amount) as avg_val from participated)as p);
select * from participated;



driver_id	reg_num	report_num	damage_amount
A02	KA053408	12	50000
A03	KA095477	13	25000
NULL	NULL	NULL	NULL

- **LIST THE NAME OF DRIVERS WHOSE DAMAGE IS GREATER THAN THE AVERAGE DAMAGE AMOUNT.**

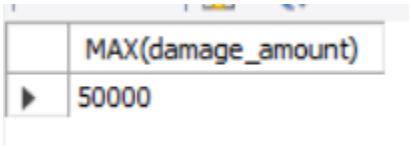
select name from person a,
participated b
where a.driver_id=b.driver_id AND damage_amount>(select AVG(damage_amount) from participated);



name
Pradeep

- **FIND MAXIMUM DAMAGE AMOUNT.**

select MAX(damage_amount)
from participated;

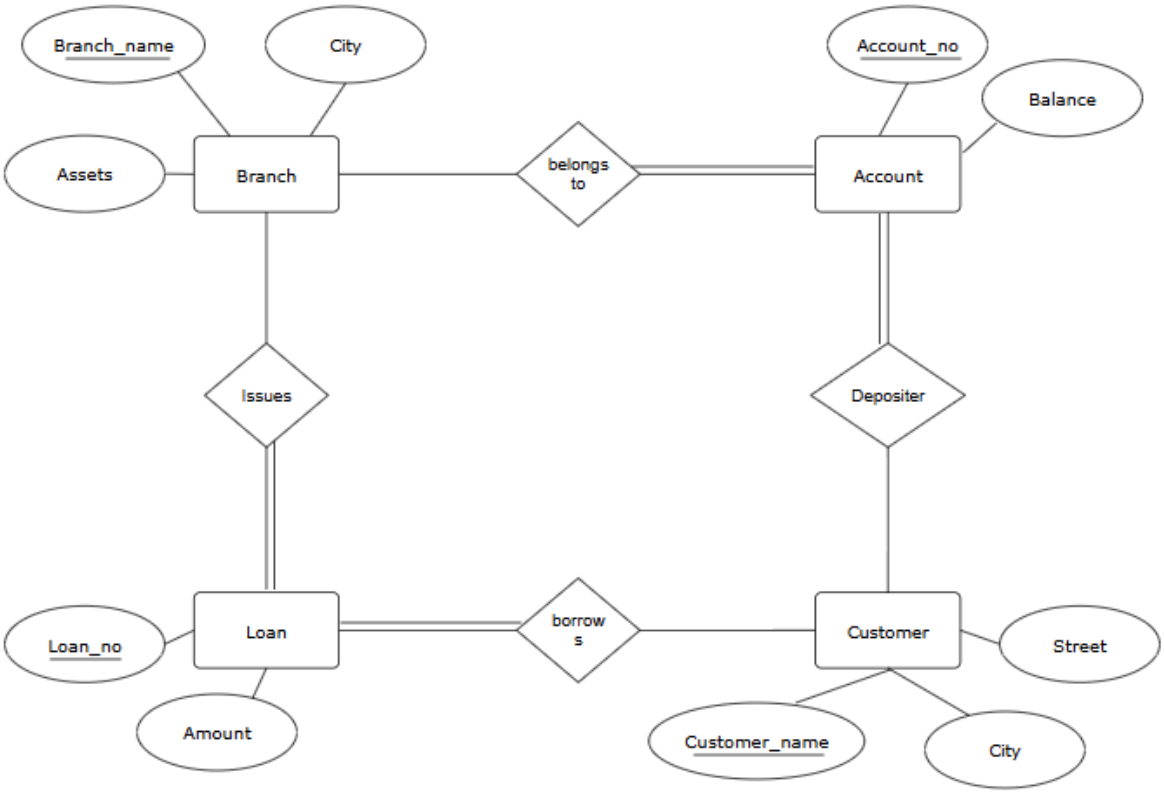


MAX(damage_amount)
50000

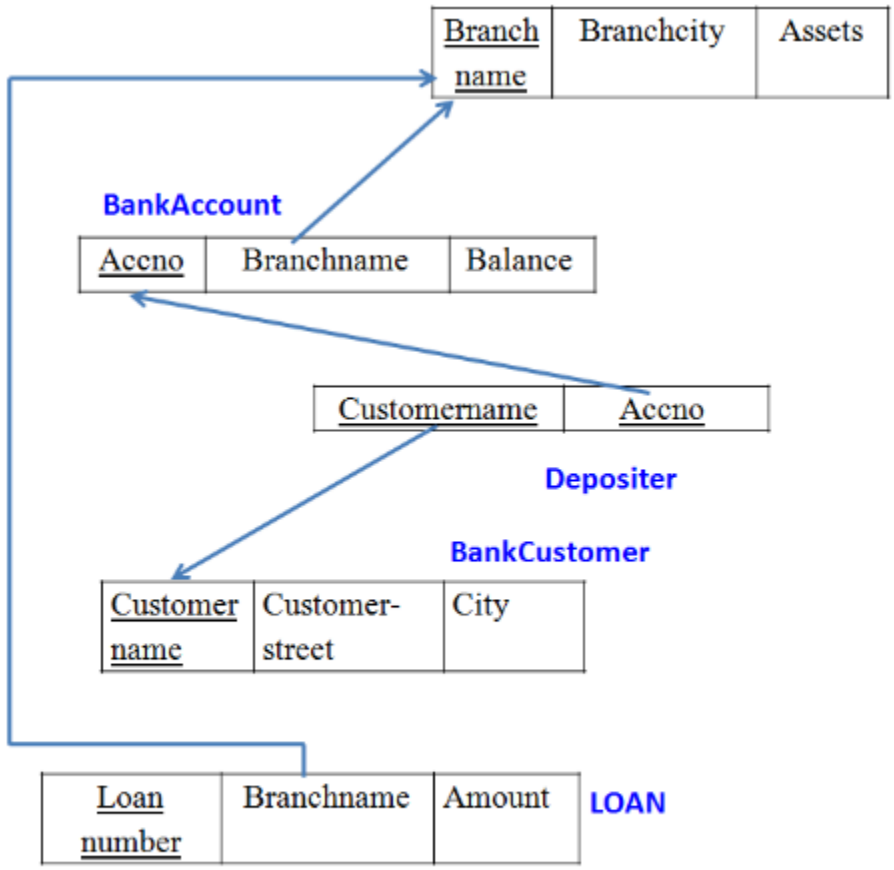
Experiment 3: Bank Database

Specifications of Insurance 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 modelled) 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 (Draw it in hand)



Schema Diagram



- BRANCH (branch_name: String, branch_city: String, assets: Int)
- BANK_ACCOUNT (acc_no: int, branch_name: String,balance: int)
- BANKCUSTOMER (customer_name: String,customer_street:String,customer_city: String)
- DEPOSITOR(customer_name: String, acc_no: int)
- LOAN (loan_no: int,branch_name: String, amount:real)

Create database

```
create database bankdatabase;
```

use bankdatabase;

Create table

```
create table branch(branch_name varchar(20) primary key,branch_city varchar(20),
assets int);

create table bank_account(account_no int primary key,
branch_name varchar(20),
balance int,
foreign key(branch_name) references branch(branch_name));

create table bankcustomer(customer_name varchar(30) primary key,
customer_street varchar(30),
customer_city varchar(30));

create table depositor(customer_name varchar(20),
account_no int,
foreign key(customer_name) references bankcustomer(customer_name),
foreign key(account_no) references bank_account(account_no));

create table loan(loan_no int,
branch_name varchar(30),
amount real,
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(20)	YES		NULL	
assets	int	YES		NULL	

desc bank_account;

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

desc bankcustomer;

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 depositor;

Field	Type	Null	Key	Default	Extra
customer_name	varchar(20)	YES	MUL	NULL	
account_no	int	YES	MUL	NULL	

desc loan;

Result Grid | Filter Rows: | Export: | Wrap

Field	Type	Null	Key	Default	Extra
loan_no	int	YES		NULL	
branch_name	varchar(30)	YES	MUL	NULL	
amount	double	YES		NULL	

Inserting Values to the table

insert into branch values('SBI_Chamrajpet','Bangalore', 50000);

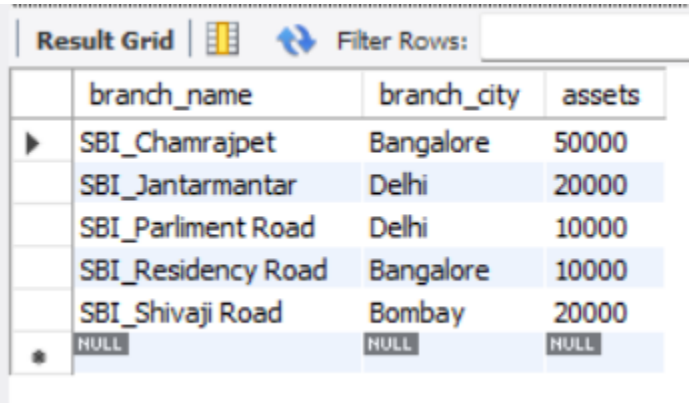
insert into branch values('SBI_Residency Road','Bangalore', 10000);

insert into branch values('SBI_Shivaji Road','Bombay', 20000);

insert into branch values('SBI_Parliment Road','Delhi', 10000);

insert into branch values('SBI_Jantarmantar','Delhi', 20000);

select * from branch;


Result Grid | Filter Rows:

branch_name	branch_city	assets
SBI_Chamrajpet	Bangalore	50000
SBI_Jantarmantar	Delhi	20000
SBI_Parliment Road	Delhi	10000
SBI_Residency Road	Bangalore	10000
SBI_Shivaji Road	Bombay	20000
NULL	NULL	NULL

insert into bank_account values(1,'SBI_Chamrajpet',2000);

insert into bank_account values(2,'SBI_Residency Road',5000);

insert into bank_account values(3,'SBI_Shivaji Road',6000);

insert into bank_account values(4,'SBI_Parliment Road',9000);

insert into bank_account values(5,'SBI_Jantarmantar',8000);

insert into bank_account values(6,'SBI_Shivaji Road',4000);

insert into bank_account values(8,'SBI_Residency Road',4000);

insert into bank_account values(9,'SBI_Parliment Road',3000);

insert into bank_account values(10,'SBI_Residency Road',5000);

insert into bank_account values(11,'SBI_Jantarmantar',2000);

select * from bank_account;

Result Grid | Filter Rows:

account_no	branch_name	balance
1	SBI_Chamrajpet	2000
2	SBI_Residency Road	5000
3	SBI_Shivaji Road	6000
4	SBI_Parliment Road	9000
5	SBI_Jantarmantar	8000
6	SBI_Shivaji Road	4000
8	SBI_Residency Road	4000
9	SBI_Parliment Road	3000
10	SBI_Residency Road	5000
11	SBI_Jantarmantar	2000
NULL	NULL	NULL

insert into bankcustomer values('Avinash','BullTempleRoad','Bangalore');

insert into bankcustomer values('Dinesh','BannergattaRoad','Bangalore');

insert into bankcustomer values('Mohan','NationalCollegeRoad','Bangalore');

insert into bankcustomer values('Nikil','AkbarRoad','Delhi');

insert into bankcustomer values('Ravi','PrithvirajRoad','Delhi');

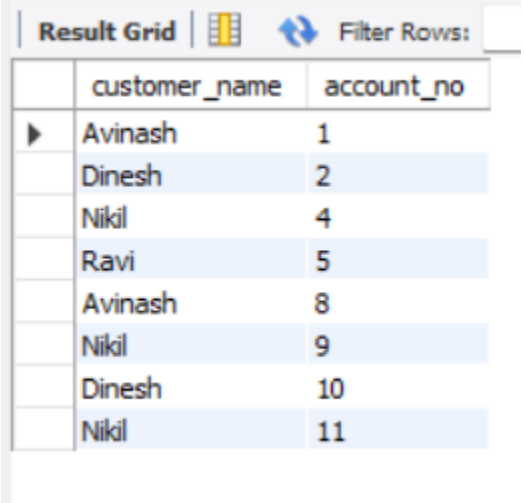
select * from bankcustomer;

Result Grid | Filter Rows: | Edit:

customer_name	customer_street	customer_city
Avinash	BullTempleRoad	Bangalore
Dinesh	BannergattaRoad	Bangalore
Mohan	NationalCollegeRoad	Bangalore
Nikil	AkbarRoad	Delhi
Ravi	PrithvirajRoad	Delhi
NULL	NULL	NULL

insert into depositor values('Avinash',1);

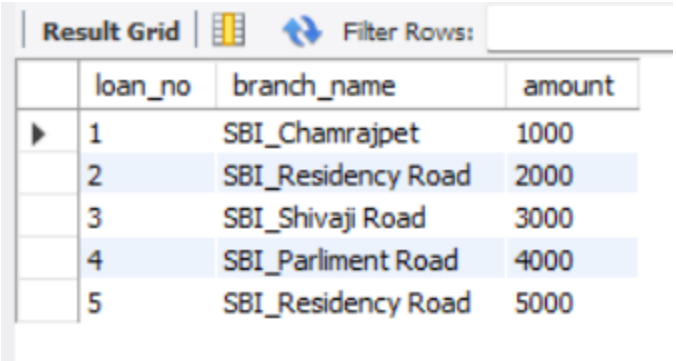
```
insert into depositor values('Dinesh',2);
insert into depositor values('Nikil',4);
insert into depositor values('Ravi',5);
insert into depositor values('Avinash',8);
insert into depositor values('Nikil',9);
insert into depositor values('Dinesh',10);
insert into depositor values('Nikil',11);
select * from depositor;
```



Result Grid | Filter Rows:

customer_name	account_no
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_Residency Road',2000);
insert into loan values(3,'SBI_Shivaji Road',3000);
insert into loan values(4,'SBI_Parliment Road',4000);
insert into loan values(5,'SBI_Residency Road',5000);
select * from loan;
```



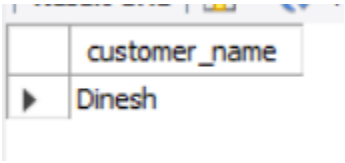
Result Grid | Filter Rows:

loan_no	branch_name	amount
1	SBI_Chamrajpet	1000
2	SBI_Residency Road	2000
3	SBI_Shivaji Road	3000
4	SBI_Parliment Road	4000
5	SBI_Residency Road	5000

Queries (Question and answer should contain screen shot of the output)

- **Find Find all the customers who have at least two deposits at the same branch (Ex. 'SBI_ResidencyRoad').**

```
select customer_name
from depositor, bank_account
where branch_name='SBI_Residency Road'AND
depositor.account_no=bank_account.account_no
group by customer_name
having count(customer_name)>=2;
```

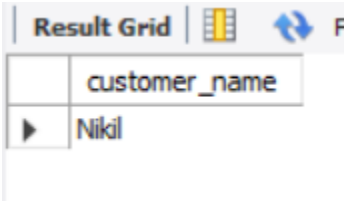


customer_name
Dinesh

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

```
select customer_name
from depositor, bank_account
```

```
where depositor.account_no=bank_account.account_no
AND bank_account.branch_name IN(
select branch_name
from branch
where branch_city='Delhi')
group by depositor.customer_name
HAVING count(distinct bank_account.branch_name)=(
select count(*)
from branch
where branch_city='Delhi');
```



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

```
delete account_no from bank_account
where (
select branch_name
from branch
where branch_city='Bombay');
```

Experiment 4: More Queries on Bank Database

Queries (Questions and output)

- **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, (assets/100000) as 'assets in lakhs' from branch;

branch_name	assets in lakhs
SBI_Chamrajpet	0.5000
SBI_Jantarmantar	0.2000
SBI_Parliment Road	0.1000
SBI_Residency Road	0.1000
SBI_Shivaji Road	0.2000

- **CREATE A VIEW WHICH GIVES EACH BRANCH THE SUM OF THE AMOUNT OF ALL THE LOANS AT THE BRANCH.**

create view Branch_total_loan(branch_name, total_loan) as select
branch_name, sum(amount) from loan
group by branch_name;

```
104 • create view branch_total_loan(branch_name,total_loan) as select branch_name,sum(amount) from Loan group by branch_name;
105 • drop view branch_total_loan;
```

- **Update THE ANNUAL INTEREST PAYMENTS ARE MADE AND ALL BRANCHES ARE TO BE INCREASED BY 5%.**

update bank_account set balance=balance*1.05;

Result Grid | Filter Rows:

account_no	branch_name	balance
1	SBI_Chamrajpet	2100
2	SBI_Residency Road	5250
3	SBI_Shivaji Road	6300
4	SBI_Parliment Road	9450
5	SBI_Jantarmantar	8400
6	SBI_Shivaji Road	4200
8	SBI_Residency Road	4200
9	SBI_Parliment Road	3150
10	SBI_Residency Road	5250
11	SBI_Jantarmantar	2100
NULL	NULL	NULL

- **LIST THE ENTIRE LOAN RELATION IN THE DESCENDING ORDER OF AMOUNT**
  select * from loan ORDER BY amount desc;

Result Grid | Filter Rows:

loan_no	branch_name	amount
5	SBI_Residency Road	5000
4	SBI_Parliment Road	4000
3	SBI_Shivaji Road	3000
2	SBI_Residency Road	2000
1	SBI_Chamrajpet	1000

- **2.Find all customers having a loan, an account or both at the bank.**
  select customer_name from depositer union (select customer_name from depositer,bankaccount where bankaccount.acc_no=depositer.acc_no);

Result Grid | Filt

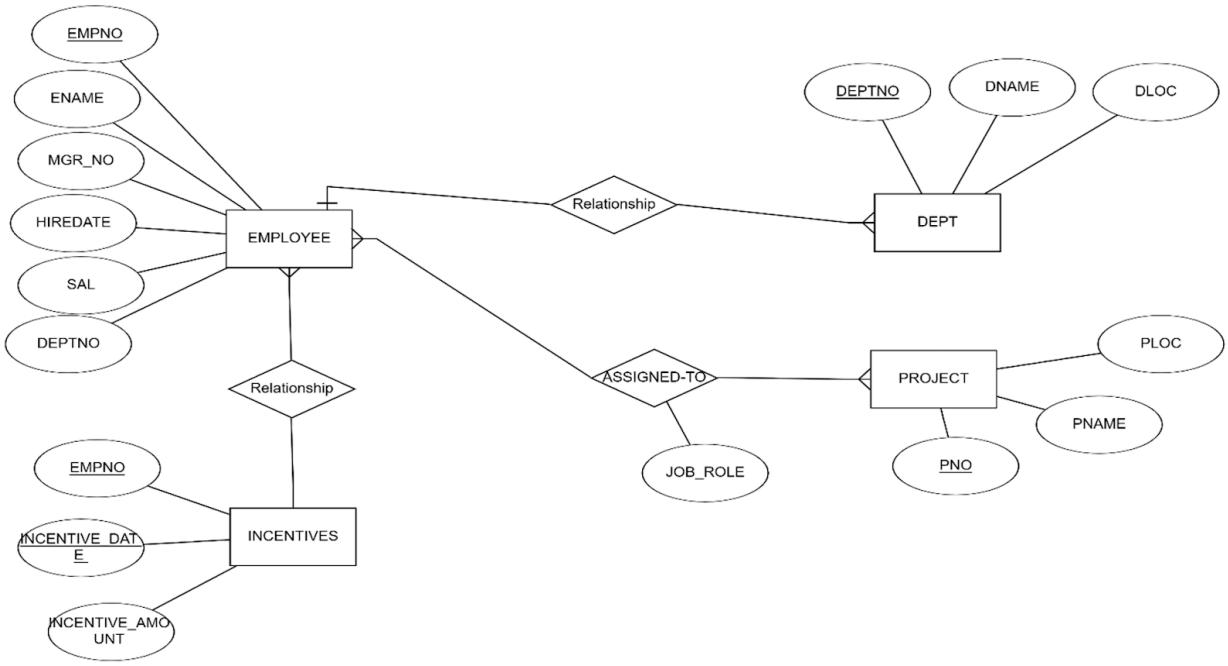
customer_name
Avinash
Dinesh
Nikil
Ravi

Experiment 5: Employee Database

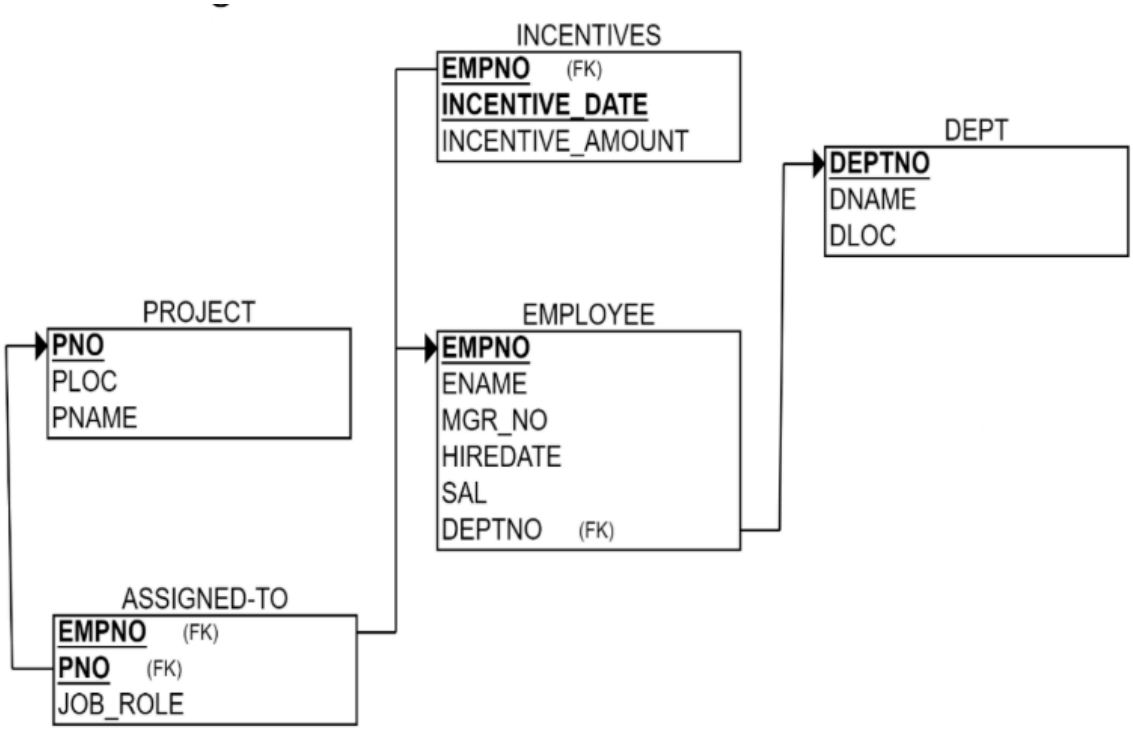
Specification of Employee Database Application:

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



- Project (pnumber: int, ploc: String, pname: String)

- Assigned_to(emp_no:decimal, number: int, job_role: String)

- Incentives (empno: decimal, incentive_date: date, incentive_amount: decimal)

- Emp(empno: decimal, ename: String, mgr_no:int, hire_date: date, sal: decimal, dept_no: int)

- dept (dept_no: int,dname: String, dloc: String)

Create database

create database Employeedatabase;

use Employeedatabase;

Create table

create table dept(deptno int primary key, dname varchar(25) default NULL,dloc varchar(20) default NULL);

```
create table project(pnumber int primary key,ploc varchar(20) default NULL,pname varchar(25)
default NULL);

create table emp(empno decimal(10,2) primary key,

 ename varchar(15) default NULL,

 mgr_no int default NULL,

hiredate date default NULL,

 sal decimal (7,2) default NULL,

 deptno int references dept(deptno) on delete cascade on update cascade);


create table incentives(empno decimal(10,2) references emp(empno) on delete cascade on update
        cascade,

incentive_date date ,

 incentive_amount decimal(10,2),

 primary key(empno, incentive_date) );



create table assigned_to(empno decimal(10,2) references emp(empno) on delete cascade on
        update cascade,

pnumber int references project(pnumber) on delete cascade on update cascade,

 job_role varchar(20),

 primary key(empno, pnumber));
```

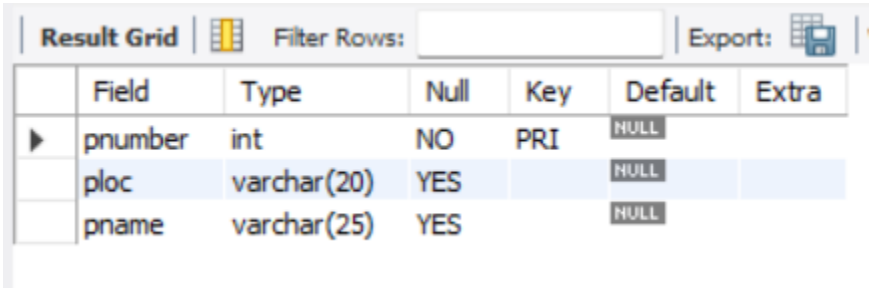
Structure of the table

desc dept;

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
deptno	int	NO	PRI	NULL	
dname	varchar(25)	YES		NULL	
dloc	varchar(20)	YES		NULL	

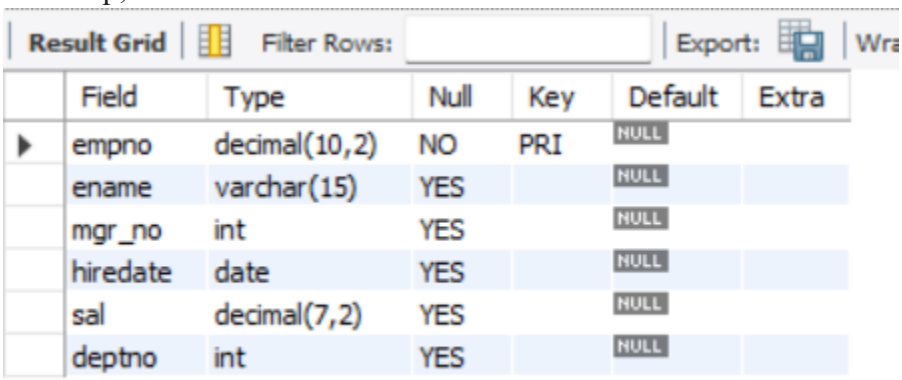
desc project;



Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
pnumber	int	NO	PRI	NULL	
ploc	varchar(20)	YES		NULL	
pname	varchar(25)	YES		NULL	

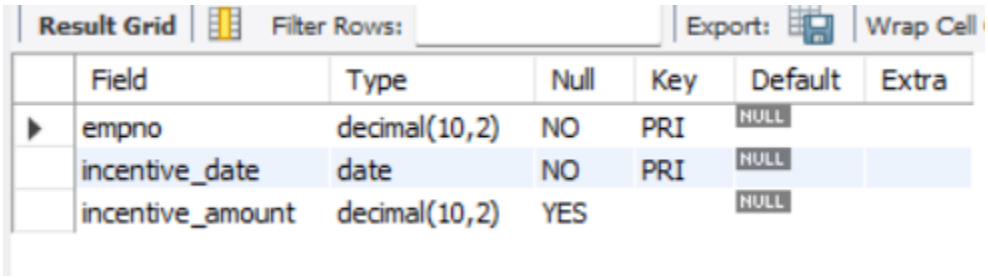
desc emp;



Result Grid | Filter Rows: | Export: | Wra

Field	Type	Null	Key	Default	Extra
empno	decimal(10,2)	NO	PRI	NULL	
ename	varchar(15)	YES		NULL	
mgr_no	int	YES		NULL	
hiredate	date	YES		NULL	
sal	decimal(7,2)	YES		NULL	
deptno	int	YES		NULL	

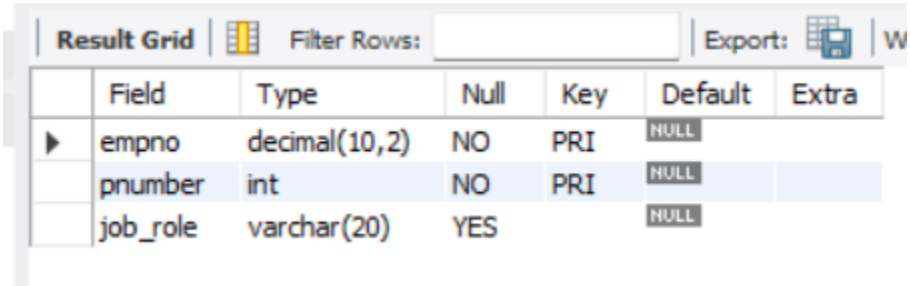
desc incentives;



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

Field	Type	Null	Key	Default	Extra
empno	decimal(10,2)	NO	PRI	NULL	
incentive_date	date	NO	PRI	NULL	
incentive_amount	decimal(10,2)	YES		NULL	

desc assigned_to;

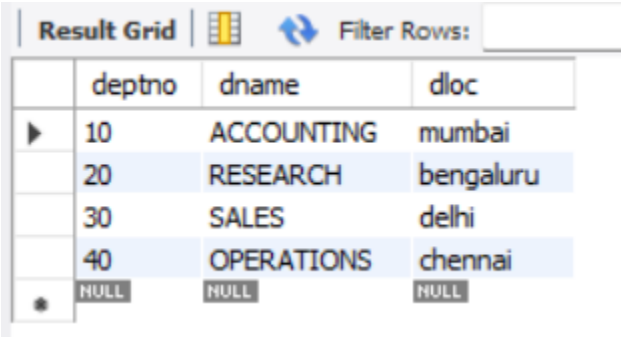


Result Grid | Filter Rows: | Export: | W

Field	Type	Null	Key	Default	Extra
empno	decimal(10,2)	NO	PRI	NULL	
pnumber	int	NO	PRI	NULL	
job_role	varchar(20)	YES		NULL	

Inserting Values to the table

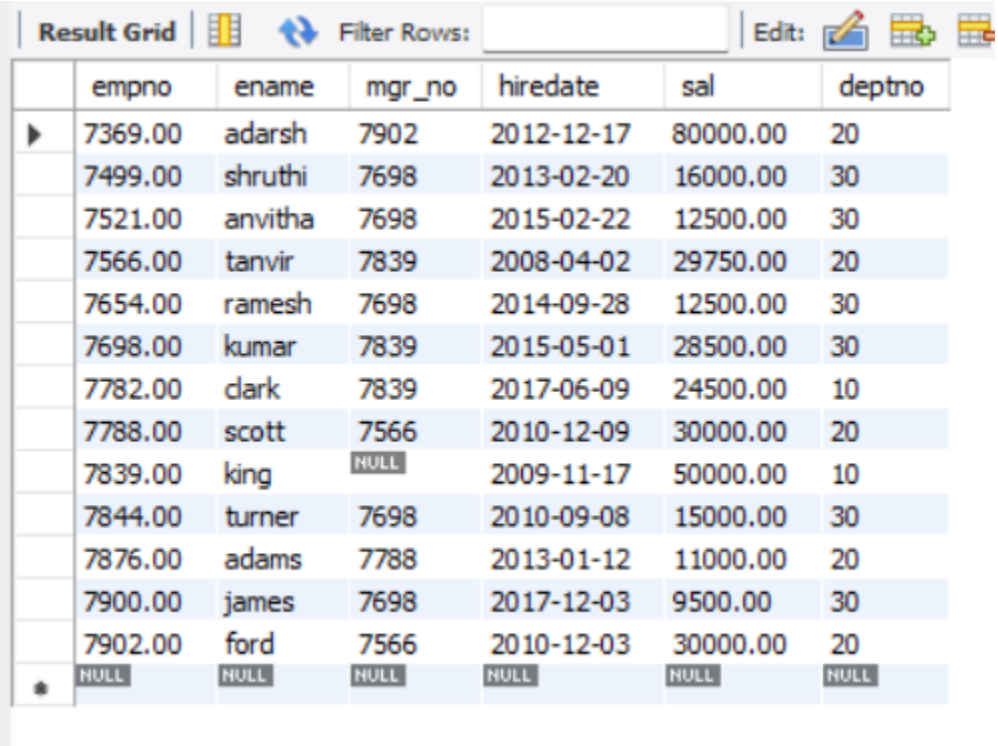
```
insert into dept values (10,'ACCOUNTING','mumbai'),
                        (20,'RESEARCH','bengaluru'),
                        (30,'SALES','delhi'),
                        (40,'OPERATIONS','chennai');
select * from dept;
```



Result Grid | Filter Rows:

deptno	dname	dloc
10	ACCOUNTING	mumbai
20	RESEARCH	bengaluru
30	SALES	delhi
40	OPERATIONS	chennai
NULL	NULL	NULL

```
insert into emp values (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');
select * from emp;
```



Result Grid | Filter Rows: | Edit:

empno	ename	mgr_no	hiredate	sal	deptno
7369.00	adarsh	7902	2012-12-17	80000.00	20
7499.00	shruthi	7698	2013-02-20	16000.00	30
7521.00	anvitha	7698	2015-02-22	12500.00	30
7566.00	tanvir	7839	2008-04-02	29750.00	20
7654.00	ramesh	7698	2014-09-28	12500.00	30
7698.00	kumar	7839	2015-05-01	28500.00	30
7782.00	clark	7839	2017-06-09	24500.00	10
7788.00	scott	7566	2010-12-09	30000.00	20
7839.00	king	NULL	2009-11-17	50000.00	10
7844.00	turner	7698	2010-09-08	15000.00	30
7876.00	adams	7788	2013-01-12	11000.00	20
7900.00	james	7698	2017-12-03	9500.00	30
7902.00	ford	7566	2010-12-03	30000.00	20
NULL	NULL	NULL	NULL	NULL	NULL

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

Result Grid | Filter Rows:

empno	incentive_date	incentive_amount
7499.00	2019-02-01	5000.00
7521.00	2019-02-01	8000.00
7521.00	2019-03-01	2500.00
7566.00	2022-02-01	5070.00
7654.00	2020-02-01	2000.00
7654.00	2022-04-01	879.00
7698.00	2019-03-01	500.00
7698.00	2020-03-01	9000.00
7698.00	2022-04-01	4500.00
NULL	NULL	NULL

insert into project values (101, 'ai project', 'bengaluru'),

(102, 'iot', 'hyderabad'),

(103, 'blockchain', 'bengaluru'),

(104, 'data science', 'mysuru'),

(105, 'autonomus systems', 'pune');

select * from project ;

Result Grid | Filter Rows:

pnumber	ploc	pname
101	ai project	bengaluru
102	iot	hyderabad
103	blockchain	bengaluru
104	data science	mysuru
105	autonomus systems	pune
NULL	NULL	NULL

insert into assigned_to values (7499, 101, 'software engineer'),

(7521, 101, 'software architect'),

(7566, 101, 'project manager'),

(7654, 102, 'sales'),

(7521, 102, 'software engineer'),

(7499, 102, 'software engineer'),

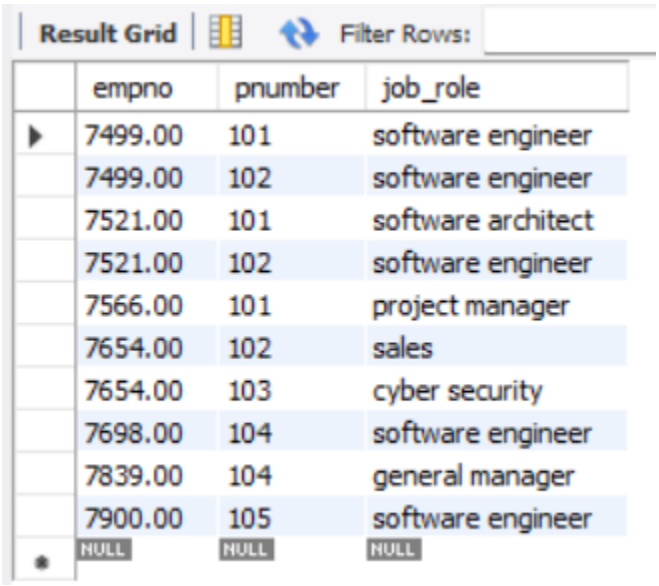
(7654, 103, 'cyber security'),

(7698, 104, 'software engineer'),

(7900, 105, 'software engineer'),

(7839, 104, 'general manager');

```
select * from assigned_to;
```



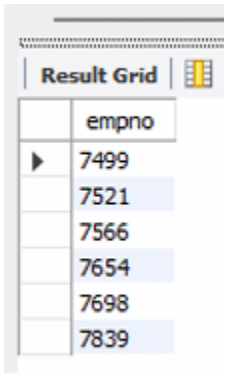
Result Grid | Filter Rows:

empno	pnumber	job_role
7499.00	101	software engineer
7499.00	102	software engineer
7521.00	101	software architect
7521.00	102	software engineer
7566.00	101	project manager
7654.00	102	sales
7654.00	103	cyber security
7698.00	104	software engineer
7839.00	104	general manager
7900.00	105	software engineer
NULL	NULL	NULL

Queries (Question and answer should contain screen shot of the output)

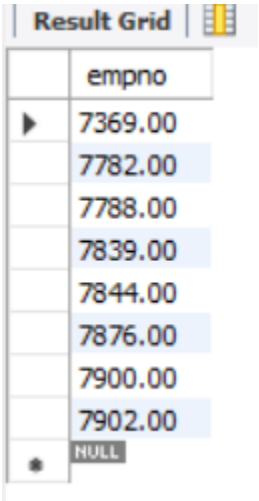
- **Retrieve the employee numbers of all employees who work on project located in Bengaluru, Hyderabad, or Mysuru**

```
select e.empno
from emp as e, project as p, assigned_to as a
where p.pnumber=a.pnumber AND e.empno=a.empno AND
p.ploc IN ('bengaluru', 'hyderabad','mysuru');
select empno
```



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

```
select empno from emp where empno NOT IN (select empno  from incentives );
```



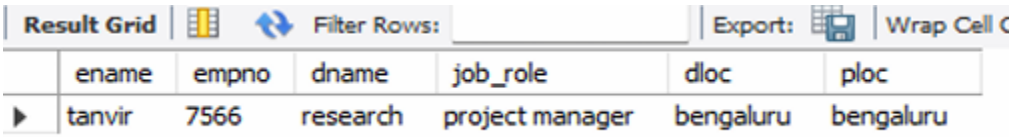
3)Write a SQL query to find the employees name, number, dept,job_role, department location and project location who are working for a project location same as his/her department location.

select distinct e.ename, e.empno, d.dname, a.job_role, d.dloc, p.ploc

from emp as e, project as p, assigned_to as a, dept as d

where e.deptno=d.deptno AND e.empno=a.empno AND a.pnumber=p.pnumber AND

d.dloc=p.ploc;



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

ename	empno	dname	job_role	dloc	ploc
tanvir	7566	research	project manager	bengaluru	bengaluru

Experiment 6: More Queries on Employee Database

Queries (Questions and output)

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

select m.ename, count(*)

from emp e, emp m

where e.mgr_no=m.empno

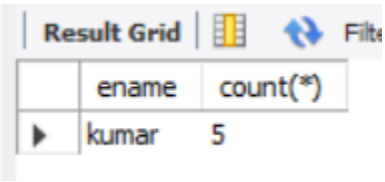
group by m.ename

having count(*)=(select MAX(my_count)

from (select count(*) my_count

from emp

group by mgr_no) as a);



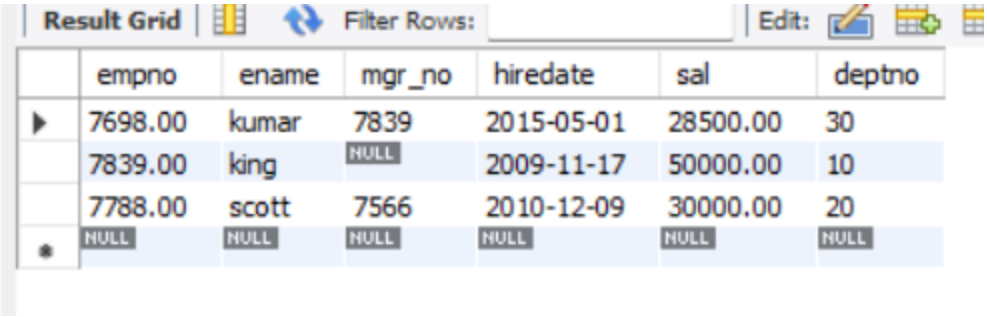
Result Grid | Filt

ename	count(*)
kumar	5

- **Display those managers name whose salary is more than average salary of his employee?**

select * from emp m

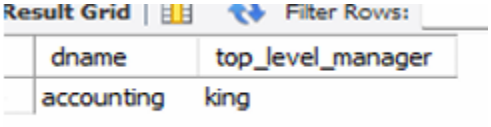
where m.empno in (select mgr_no from emp e where m.sal>(select avg(f.sal) from emp f where f.mgr_no=m.empno));



empno	ename	mgr_no	hiredate	sal	deptno
7698.00	kumar	7839	2015-05-01	28500.00	30
7839.00	king	NULL	2009-11-17	50000.00	10
7788.00	scott	7566	2010-12-09	30000.00	20
NULL	NULL	NULL	NULL	NULL	NULL

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

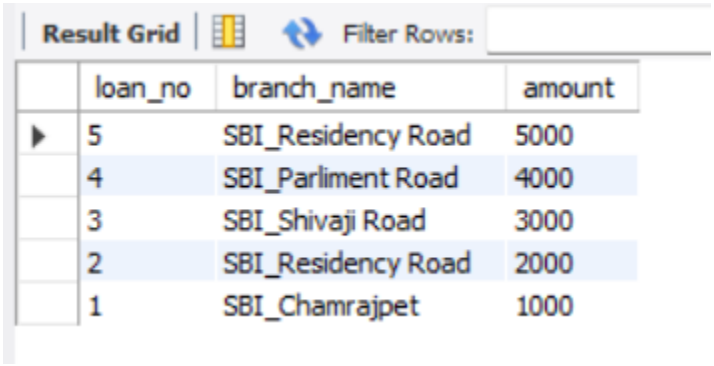
```
select d.dname, e.ename as top_level_manager
 from dept d
 join emp e on d.deptno=e.deptno
 where e.mgr_no=NULL;
```



dname	top_level_manager
accounting	king

- **LIST THE ENTIRE LOAN RELATION IN THE DESCENDING ORDER OF AMOUNT**

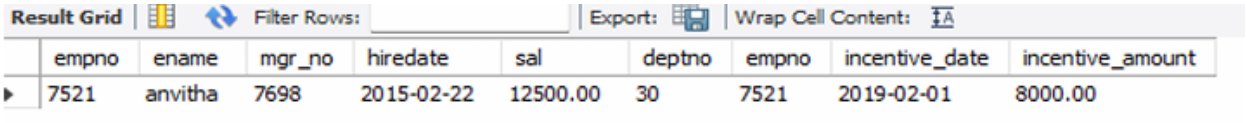
```
select * from loan ORDER BY amount desc;
```



loan_no	branch_name	amount
5	SBI_Residency Road	5000
4	SBI_Parliment Road	4000
3	SBI_Shivaji Road	3000
2	SBI_Residency Road	2000
1	SBI_Chamrajpet	1000

- **SQL Query to find the employee details who got second maximum incentive in Febrauary 2019.**

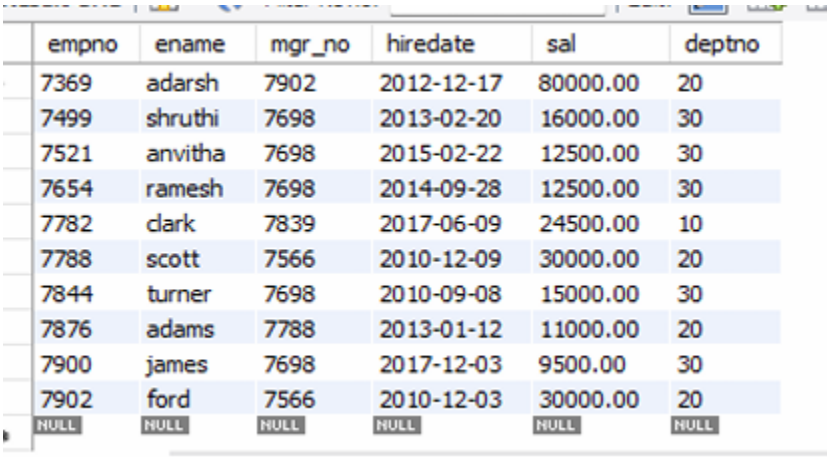
```
select * from employee e,incentives i where e.empno=i.empno and 2=(select count(*)
from incentives j where i.incentive_amount<=j.incentive_amount);
```



empno	ename	mgr_no	hiredate	sal	deptno	empno	incentive_date	incentive_amount
7521	anvitha	7698	2015-02-22	12500.00	30	7521	2019-02-01	8000.00

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

select * from employee e where e.deptno=(select e1.deptno from employee e1 where e1.empno=e.mgr_no);



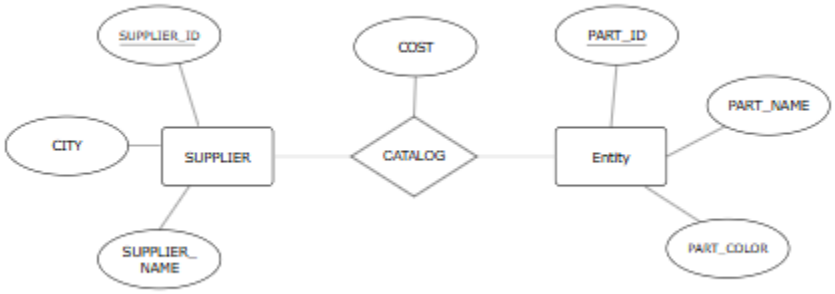
empno	ename	mgr_no	hiredate	sal	deptno
7369	adarsh	7902	2012-12-17	80000.00	20
7499	shruthi	7698	2013-02-20	16000.00	30
7521	anvitha	7698	2015-02-22	12500.00	30
7654	ramesh	7698	2014-09-28	12500.00	30
7782	clark	7839	2017-06-09	24500.00	10
7788	scott	7566	2010-12-09	30000.00	20
7844	turner	7698	2010-09-08	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

Experiment 7: Supplier Database

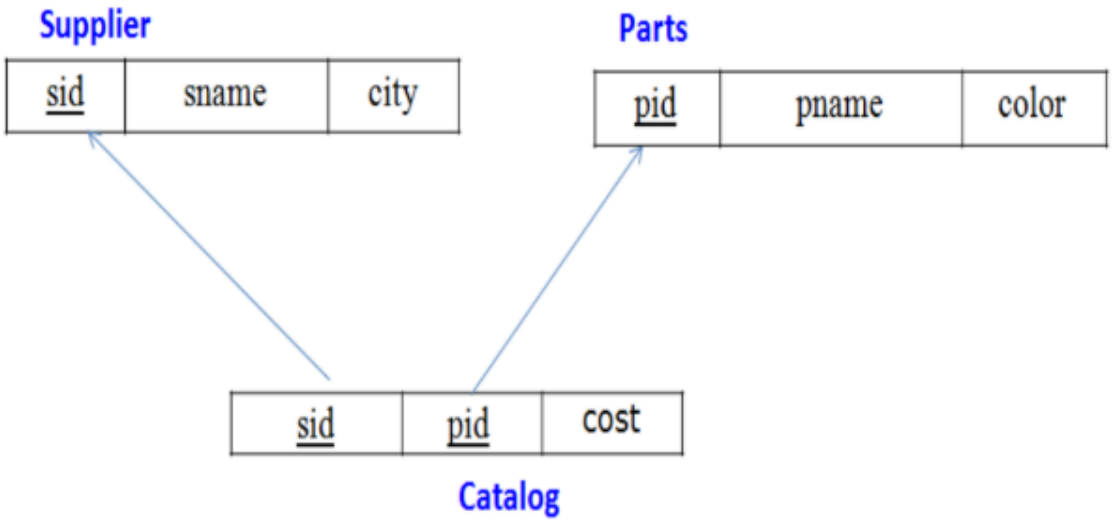
Specifications of Insurance Database Application:

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 colour. 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 “analyse 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



- Suppliers (sid: int, sname: String, city: String)
- Parts(pid:intl, pnamer: String, colour: String)
- Catalog(sid: int,pid: int, cost: Float)

Create database

create database supplier;

use supplier;

Create table

create table suppliers(sid int primary key, sname varchar(20), address varchar(20));

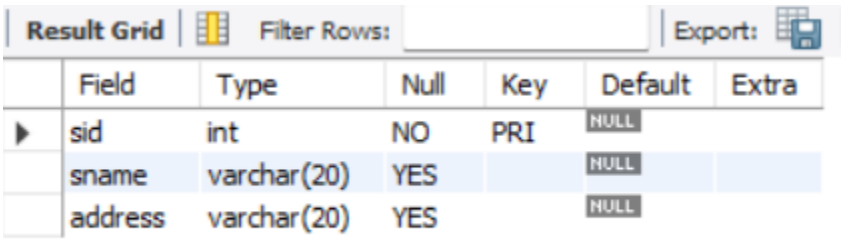
create table parts(pid int primary key, pname varchar(20), colour varchar(20));

create table catalog(sid int, pid int, cost real, foreign key(sid) references suppliers(sid),

foreign key(pid) references parts(pid), primary key(sid,pid));

Structure of the table

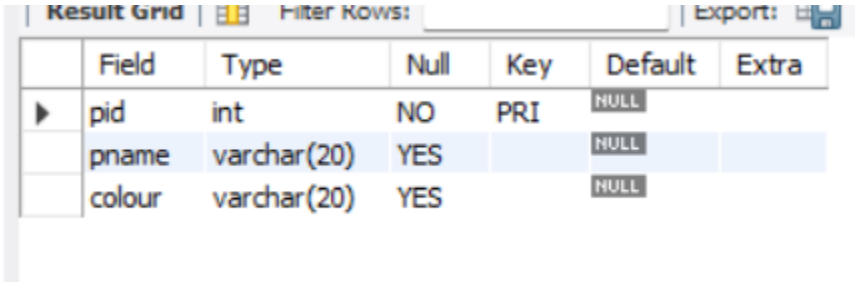
desc suppliers;



Result Grid | Filter Rows: | Export:

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

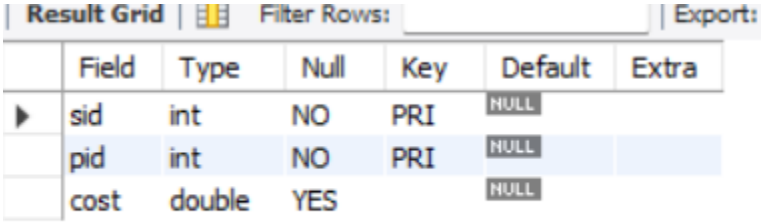
desc parts;



Result Grid | Filter Rows: | Export:

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

desc catalog ;



Result Grid | Filter Rows: | Export:

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

Inserting Values to the table

insert into suppliers values(10001, "Acme Widget", "Bangalore"),

(10002, "Johns", "Kolkata"),

(10003, "Vimal", "Mumbai"),

(10004, "Reliance", "Delhi");

select * from suppliers;

sid	sname	address
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	colour
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
NULL	NULL	NULL

Queries

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

  select pname from parts where pid in(select pid from catalog);

pname
Book
Pen
Pencil
Mobile
Charger

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

  select distinct s.sname

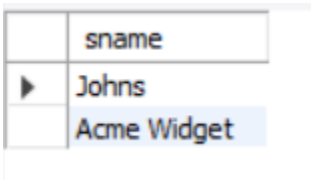
  from suppliers as s, parts as p, catalog as c

  where (select count(p.pid) from parts p)=(select count(c.pid) from catalog c where s.sid=c.sid);

sname
Acme Widget

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

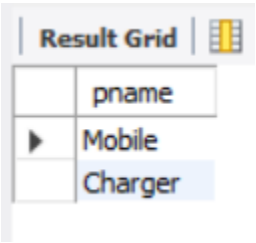
```
select distinct s.sname
from suppliers as s, parts as p, catalog as c
where c.sid=s.sid and c.pid IN (select p1.pid from parts as p1 where colour="Red");
```



sname
Johns
Acme Widget

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

```
select p.pname
from parts as p, suppliers as s, catalog as c
where c.sid=s.sid and c.pid=p.pid and s.sname="Acme Widget"
and not exists(select * from catalog c1, suppliers s1 where p.pid=c1.pid and c1.sid=s1.sid and
s1.sname<>"Acme Widget" );
```

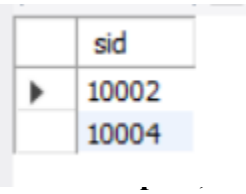


Result Grid

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).**

```
select distinct c.sid from parts as p, suppliers as s, catalog as c
where c.cost>(select avg(c1.cost)  from catalog c1 where c1.pid=c.pid);
```



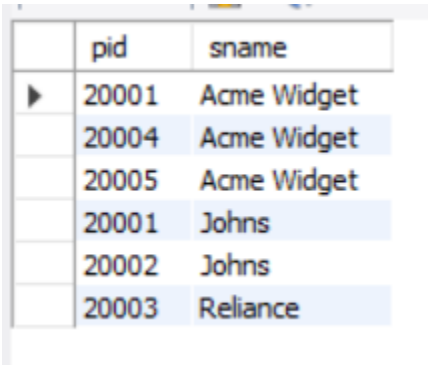
sid
10002
10004

- **Find the sname of the supplier who charges the most for that part.**

select p.pid, s.sname from parts p, suppliers s, catalog c

where c.pid=p.pid and c.sid=s.sid and c.cost=(select MAX(c1.cost) from catalog c1

where c1.pid=p.pid);



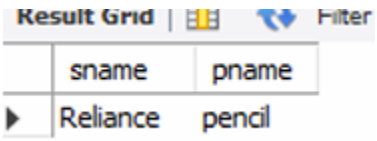
pid	sname
20001	Acme Widget
20004	Acme Widget
20005	Acme Widget
20001	Johns
20002	Johns
20003	Reliance

Experiment 8: More Queries on Supplier Database

Queries

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

select s.sname,p.pname from suppliers as s,parts as p,catalog as c where p.pid=c.pid and s.sid=c.sid and c.cost=(select max(cost) from catalog);

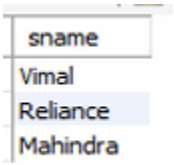


Result Grid | Filter

sname	pname
Reliance	pencil

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

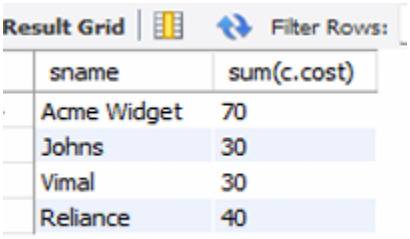
select distinct s.sname from suppliers as s,parts as p,catalog as c where s.sname not in (select s1.sname from suppliers as s1,parts as p1,catalog as c1 where s1.sid=c1.sid and p1.pid=c1.pid and p1.color='red');



sname
Vimal
Reliance
Mahindra

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

select s.sname, sum(c.cost) from suppliers s join catalog c on s.sid = c.sid group by s.sname;

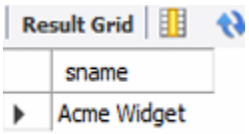


Result Grid | Filter Rows:

sname	sum(c.cost)
Acme Widget	70
Johns	30
Vimal	30
Reliance	40

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

select s.sname from suppliers s where (select count(*) from catalog c where c.sid = s.sid and c.cost < 20) >= 2;

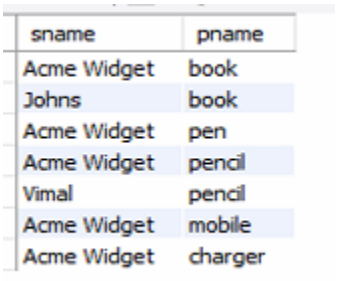


Result Grid

sname
Acme Widget

- **List suppliers who offer the cheapest cost for each part.**

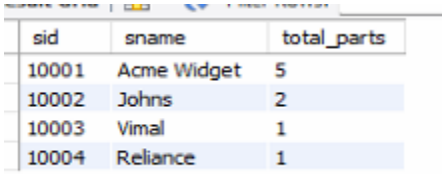
select distinct s.sname,p.pname from suppliers as s,parts as p,catalog as c where s.sid=c.sid and c.pid=p.pid and c.cost in (select min(c1.cost) from catalog as c1,suppliers as s1 where s.sid=s1.sid group by c1.pid);



sname	pname
Acme Widget	book
Johns	book
Acme Widget	pen
Acme Widget	pencil
Vimal	pencil
Acme Widget	mobile
Acme Widget	charger

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

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



sid	sname	total_parts
10001	Acme Widget	5
10002	Johns	2
10003	Vimal	1
10004	Reliance	1

- **Create a view of the most expensive supplier for each part**

create view mostexpensive as select s.sid, s.sname, c.pid, c.cost from suppliers s join catalog c on s.sid = c.sid where c.cost = (select max(cost) from catalog where pid = c.pid); select * from mostexpensive;

sid	sname	pid	cost
10001	Acme Widget	20001	10
10001	Acme Widget	20004	10
10001	Acme Widget	20005	10
10002	Johns	20001	10
10002	Johns	20002	20
10004	Reliance	20003	40

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

delimiter // create trigger preventlowcost before insert on catalog for each row begin if new.cost < 1 then signal sqlstate '45000' set message_text = 'cost must be at least 1'; end if; end; // delimiter ;

```
delimiter //
create trigger preventlowcost before insert on catalog
for each row
begin
  if new.cost < 1 then
    signal sqlstate '45000' set message_text = 'cost must be at least 1';
  end if;
end; //
delimiter ;
```

- **Create a trigger to set to default cost if not provided.**
  delimiter // create trigger setdefaultcost before insert on catalog for each row begin if new.cost is null then set new.cost = 1; end if; end; // delimiter ;

```
81    delimiter //
82    create trigger setdefaultcost before insert on catalog
83    for each row
84    begin
85      if new.cost is null then
86        set new.cost = 1;
87      end if;
88    end; //
89    delimiter ;
```

Experiment 9:

NoSQL Cloud Installation
(MongoDB Atlas)

Aim:
To install and configure a NoSQL database in the cloud using MongoDB Atlas and perform basic database operations.

Software Requirements:
- Web browser (Chrome / Firefox)
- Internet connection
- MongoDB Atlas account

Procedure:
1. Open a web browser and visit https://www.mongodb.com/atlas.
2. Click on **Sign Up** and create a MongoDB Atlas account using email or Google login.
3. After successful login, create a **new project**.
4. Click on **Build a Database** and choose the **Free Tier (M0)** cluster.
5. Select the cloud provider and region (preferably nearest region).
6. Create the cluster and wait for deployment to complete.
7. Configure **Database Access** by creating a database user with username and password.
8. Configure **Network Access** by allowing access from the current IP address (or 0.0.0.0/0).
9. Click **Connect** and obtain the MongoDB connection string.
10. Use MongoDB Compass or MongoDB Shell to connect to the cloud database using the provided connection string.
11. Create a database and collections to perform operations such as insert, find, update, and delete.

Result:
MongoDB Atlas cloud database was successfully created and connected. Basic NoSQL operations were performed on the cloud-hosted database.

Conclusion:
Thus, a NoSQL database was successfully installed and configured in the cloud using MongoDB Atlas, demonstrating cloud-based database deployment and access.

Experiment 10:
NoSQL Cloud - Student Database

Specification of NoSQL – Student Database Application:

The NoSQL student database must store and manage student information using a document-oriented data model in MongoDB. Each student record is uniquely identified by a roll number and includes attributes such as age, contact number, and email ID. The database must support insertion of appropriate student records and allow modification of existing data, including updating a student's email ID based on roll number and replacing a student name with a new value for a specified roll number. It should also support administrative operations such as exporting the student collection to the local file system for backup or data transfer purposes and importing data from external CSV files into MongoDB collections. Additionally, the system must allow safe deletion of collections when required. The database should ensure flexible schema handling, efficient document updates, and reliable data persistence while demonstrating fundamental NoSQL operations such as insert, update, replace, export, import, and drop.

Create Database:

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

Insert Appropriate Values:

Insert at least 4 documents each with RollNo, Name, Age, ContactNo and EmailID attributes.

```
> db.Student.insertMany ([
    {RollNo: 10, Name: "ABC", Age: 20, ContactNo: 1112223334, EmailID: "CSE10.cs24@bmsce.ac.in"},
    {RollNo: 11, Name: "DEF", Age: 20, ContactNo: 2223334445, EmailID: "CSE11.cs24@bmsce.ac.in"},
    {RollNo: 12, Name: "GHI", Age: 20, ContactNo: 3334445556, EmailID: "CSE12.cs24@bmsce.ac.in"},
    {RollNo: 13, Name: "JKL", Age: 20, ContactNo: 4445556667, EmailID: "CSE13.cs24@bmsce.ac.in"}
  ]);
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('693ff9c98e185b767bf90899'),
      '1': ObjectId('693ff9c98e185b767bf9089a'),
      '2': ObjectId('693ff9c98e185b767bf9089b'),
      '3': ObjectId('693ff9c98e185b767bf9089c')
    }
  }
```

Queries:

i. Write a query to update Email ID of a student with RollNo 10.

```
> db.Student.updateOne(
    {RollNo: 10},
    {$set: {EmailID: "CSE10Updated.cs24@bmsce.ac.in"}}
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

ii. Replace the student's name from "DEF" to "FEM" of RollNo 11.

```
> db.Student.updateOne (
    {RollNo: 11},
    {$set: {Name: "FEM"}}
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

iii. Export the created table into Local File System.

```
C:\Users\Sudhanva S M>mongoexport --db=Student --collection=Student --type=csv --fields=RollNo,Name,Age,ContactNo,EmailI
D --out=students.csv
2025-12-15T17:38:33.585+0530    connected to: mongodb://localhost/
2025-12-15T17:38:33.588+0530    exported 4 records
```

Or

a) Open your Collections Table.

b) Click on Export Data -> Export Full Collection.

c) Export File Type → CSV.

d) A dialog box will open. Choose file destination and click save.

	A	B	C	D	E
1	RollNo	Name	Age	ContactNo	EmailID
2	10	ABC	20	1112223334	CSE10Updated.cs24@bmsce.ac.in
3	11	FEM	20	2223334445	CSE11.cs24@bmsce.ac.in
4	12	GHI	20	3334445556	CSE12.cs24@bmsce.ac.in
5	13	[illegible]	[illegible]	[illegible]5556667	CSE13.cs24@bmsce.ac.in

iv. Drop the table.

```
> db.Student.drop();
< true
```

v. Import a given a CSV dataset from your Local File System into MongoDB collection.

a. Open your collections table.
b. Click Add Data → Import JSON or CSV file.
c. Select previously saved CSV file (students.csv).
d. Click Import.

Experiment 11:
NoSQL Cloud - Customer Database

Specification of NoSQL – Customer Database Application:

The NoSQL customer database is designed to store and manage customer account information using a document-based data model in MongoDB. Each customer record is uniquely identified by a customer ID and includes attributes such as account balance and account type. The database must support insertion of multiple customer records and enable querying of customers whose total account balance exceeds a specified value for a given account type. It should also allow aggregation operations to determine the minimum and maximum account balances for each customer. Additionally, the system must support administrative operations such as exporting the customer collection to the local file system for backup purposes, importing customer data from external CSV files, and safely dropping collections when required. The database should demonstrate efficient data storage, flexible querying, and basic aggregation capabilities of NoSQL databases.

Create Database:

```
> use customer
< switched to db customer
```

Insert Appropriate Values:

Insert at least 4 documents each with Cust_id, Acc_Bal, Acc_Type attributes.

```
> db.customer.insertMany ([
    {Cust_id: 101, Acc_Bal: 1234, Acc_Type: "Z"},
    {Cust_id: 101, Acc_Bal: 777, Acc_Type: "Y"},
    {Cust_id: 103, Acc_Bal: 1634, Acc_Type: "Z"},
    {Cust_id: 104, Acc_Bal: 987, Acc_Type: "Z"},
    {Cust_id: 104, Acc_Bal: 1009, Acc_Type: "Y"}
  ]);
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('6940081e104a74526bf2021c'),
      '1': ObjectId('6940081e104a74526bf2021d'),
      '2': ObjectId('6940081e104a74526bf2021e'),
      '3': ObjectId('6940081e104a74526bf2021f'),
      '4': ObjectId('6940081e104a74526bf20220')
    }
  }
```

Queries:

1. **Write a query to display those records whose total account balance is greater than 1200 of account type 'Z' for each Cust_id.**

```
> db.customer.aggregate ([
    {$match: {Acc_Type: 'Z'}},
    {$group: {
      _id: "$Cust_id",
      total_balance: {$sum: "$Acc_Bal"}
    }},
    {$match: {total_balance: {$gt: 1200}}}
  ]);
< {
    _id: 101,
    total_balance: 1234
  }
  {
    _id: 103,
    total_balance: 1634
  }
```

2. Determine Minimum and Maximum account balance for each customer_id.

```
> db.customer.aggregate([
    {$group: {
        _id: "$Cust_id",
        min_balance: { $min: "$Acc_Bal" },
        max_balance: { $max: "$Acc_Bal" }
      }
    }
  ]);
< {
    _id: 104,
    min_balance: 987,
    max_balance: 1009
  }
  {
    _id: 101,
    min_balance: 777,
    max_balance: 1234
  }
  {
    _id: 103,
    min_balance: 1634,
    max_balance: 1634
  }
```

3. Export the created table into Local File System.

```
C:\Users\Sudhanva S M>mongoexport --db=customer --collection=customer --type=csv --fields=Cust_id,Acc_Bal,Acc_Type --out
=customer.csv
2025-12-15T18:41:49.137+0530    connected to: mongodb://localhost/
2025-12-15T18:41:49.141+0530    exported 5 records
```

Or

a) Open your Collections Table.

b) Click on Export Data → Export Full Collection.

c) Export File Type → CSV.

d) A dialog box will open. Choose file destination and click save.

	A	B	C
1	Cust_id	Acc_Bal	Acc_Type
2	101	1234	Z
3	101	777	Y
4	103	1634	Z
5	104	987	Z
6	104	1009	Y

4. **Drop the table.**

```
> db.customer.drop()
< true
```

5. **Import a given a CSV dataset from your Local File System into MongoDB collection.**

    a) Open your collections table.

    b) Click Add Data → Import JSON or CSV file.

    c) Select previously saved CSV file (customer.csv).

    d) Click Import.

Experiment 12:
NoSQL Cloud - Restaurant Database

Specification of NoSQL – Restaurant Database Application:

The NoSQL restaurant database is designed to store and manage restaurant related information using a document-oriented data model in MongoDB. Each restaurant document contains details such as restaurant ID, name, location information, cuisine type, and inspection scores. The database must support retrieval of all restaurant records and allow sorting of restaurant data based on specified attributes. It should enable selective querying to extract specific fields such as restaurant ID, name, town, and cuisine for restaurants meeting given score criteria. Additionally, the system must support aggregation operations to compute the average inspection score for each restaurant and perform pattern-based queries to identify restaurants located in areas with specific zip code formats. The database demonstrates flexible querying, sorting, filtering, and aggregation capabilities of MongoDB for real-world data analysis.

Create Database:

```
> use restaurant
< switched to db restaurant
```

Insert Appropriate Values:

Insert at least 4 documents each with Restaurant_id, Name, Cuisine, Address (Town, Zipcode), Score attributes

```
> db.restaurant_details.insertOne([
   {restaurant_id: 1001,
    name: "Saffron & Sage",
    cuisine: "Modern Indian Fusion",
    address: {
      town: "Udaipur",
      zipcode: "101028"
    },
    score: 10.0
   }
 ]);
< {
   acknowledged: true,
   insertedId: ObjectId('69401559c7b448b946f298c3')
 }
```

```
> db.restaurant_details.insertOne([
   {restaurant_id: 1002,
    name: "The Gilded Noodle",
    cuisine: "Pan-Asian/Thai",
    address: {
      town: "Indore",
      zipcode: "452009"
    },
    score: 7.1
   }
 ]);
< {
   acknowledged: true,
   insertedId: ObjectId('69401566c7b448b946f298c4')
 }
```

```
> db.restaurant_details.insertOne([
   {restaurant_id: 1003,
    name: "Misty Malabar",
    cuisine: "South-Indian (Coastal)",
    address: {
      town: "Puducherry",
      zipcode: "105357"
    },
    score: 8.5
   }
 ]);
< {
   acknowledged: true,
   insertedId: ObjectId('6940156fc7b448b946f298c5')
 }
```

```
> db.restaurant_details.insertOne([
   {restaurant_id: 1004,
    name: "Velvet Vineyards",
    cuisine: "French-Italian",
    address: {
      town: "Mumbai",
      zipcode: "400001"
    },
    score: 18.5
   }
 ]);
< {
   acknowledged: true,
   insertedId: ObjectId('69401577c7b448b946f298c6')
 }
```

```
> db.restaurant_details.insertOne([
   {restaurant_id: 1005,
    name: "Indigo Masala",
    cuisine: "Indo-Mexican Fusion",
    address: {
      town: "Hyderabad",
      zipcode: "500108"
    },
    score: 12.5
   }
 ]);
< {
   acknowledged: true,
   insertedId: ObjectId('69401583c7b448b946f298c7')
 }
```

Queries:

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

```
> db.restaurant_details.find();
< {
   _id: ObjectId('69401559c7b448b946f298c3'),
   restaurant_id: 1001,
   name: 'Saffron & Sage',
   cuisine: 'Modern Indian Fusion',
   address: {
     town: 'Udaipur',
     zipcode: '101028'
   },
   score: 10
 }
```

```
{
  _id: ObjectId('69401566c7b448b946f298c4'),
  restaurant_id: 1002,
  name: 'The Gilded Noodle',
  cuisine: 'Pan-Asian/Thai',
  address: {
    town: 'Indore',
    zipcode: '452009'
  },
  score: 7.1
}
```

```
{
  _id: ObjectId('6940156fc7b448b946f298c5')
  restaurant_id: 1003,
  name: 'Misty Malabar',
  cuisine: 'South-Indian (Coastal)',
  address: {
    town: 'Puducherry',
    zipcode: '105357'
  },
  score: 8.5
}
```

```
{
  _id: ObjectId('69401577c7b448b946f298c6'),
  restaurant_id: 1004,
  name: 'Velvet Vineyards',
  cuisine: 'French-Italian',
  address: {
    town: 'Mumbai',
    zipcode: '400001'
  },
  score: 18.5
}
```

```
{
  _id: ObjectId('69401583c7b448b946f298c7'),
  restaurant_id: 1005,
  name: 'Indigo Masala',
  cuisine: 'Indo-Mexican Fusion',
  address: {
    town: 'Hyderabad',
    zipcode: '500108'
  },
  score: 12.5
}
```

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

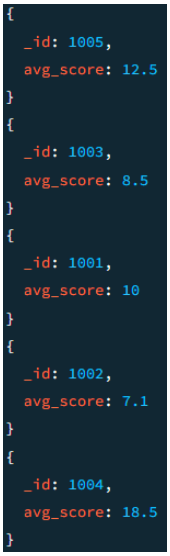
ii. Write a MongoDB query to find the restaurant Id, name, town and cuisine for those restaurants which achieved a score which is not more than 10.

```
> db.restaurant_details.find({score: {$lte: 10}});
< {
    _id: ObjectId('69401559c7b448b946f298c3'),
    restaurant_id: 1001,
    name: 'Saffron & Sage',
    cuisine: 'Modern Indian Fusion',
    address: {
      town: 'Udaipur',
      zipcode: '101028'
    },
    score: 10
  }
```

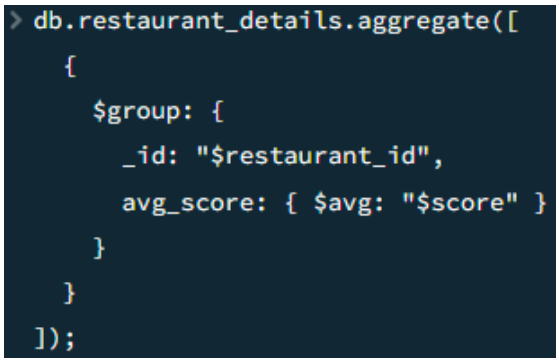
```
{
  _id: ObjectId('69401566c7b448b946f298c4'),
  restaurant_id: 1002,
  name: 'The Gilded Noodle',
  cuisine: 'Pan-Asian/Thai',
  address: {
    town: 'Indore',
    zipcode: '452009'
  },
  score: 7.1
}
```

```
{
  _id: ObjectId('6940156fc7b448b946f298c5'),
  restaurant_id: 1003,
  name: 'Misty Malabar',
  cuisine: 'South-Indian (Coastal)',
  address: {
    town: 'Puducherry',
    zipcode: '105357'
  },
  score: 8.5
}
```

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

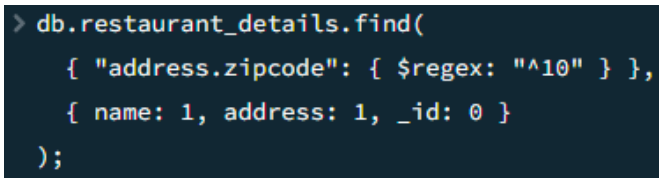


```
{
  _id: 1005,
  avg_score: 12.5
}
{
  _id: 1003,
  avg_score: 8.5
}
{
  _id: 1001,
  avg_score: 10
}
{
  _id: 1002,
  avg_score: 7.1
}
{
  _id: 1004,
  avg_score: 18.5
}
```



```
> db.restaurant_details.aggregate([
    {
      $group: {
        _id: "$restaurant_id",
        avg_score: { $avg: "$score" }
      }
    }
  ]);
```

iv. Write a MongoDB query to find the name and address of the restaurants that have a zip code that starts with '10'.



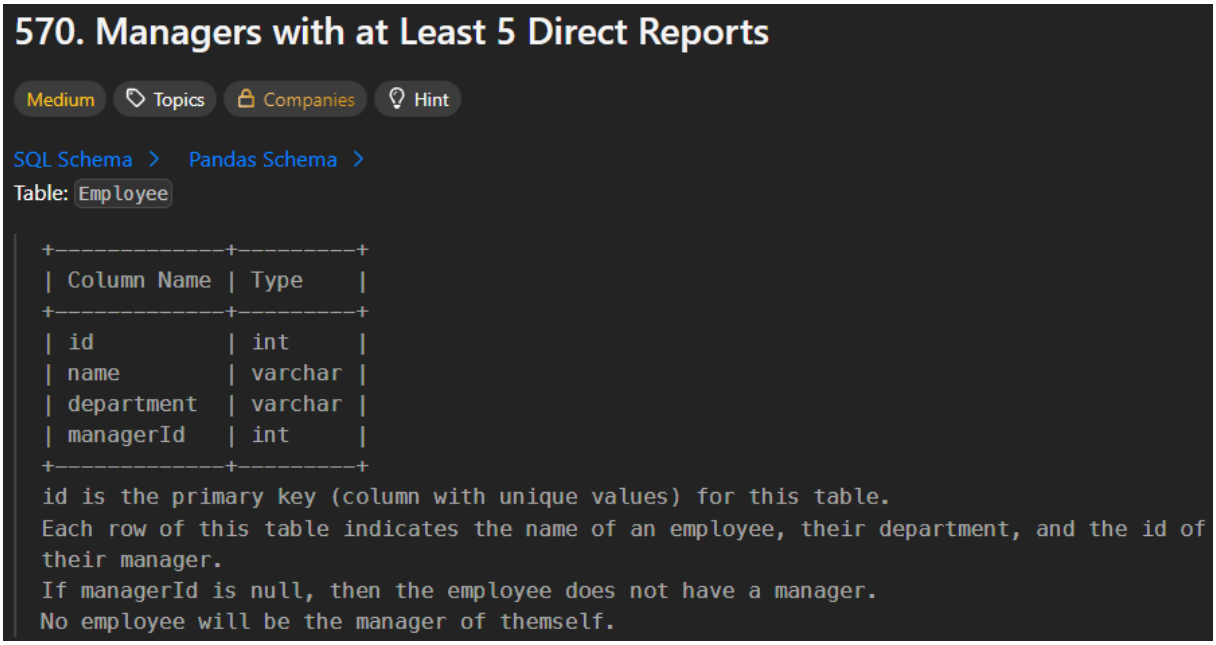
```
> db.restaurant_details.find(
    { "address.zipcode": { $regex: "^10" } },
    { name: 1, address: 1, _id: 0 }
  );
```

```
< {
    name: 'Saffron & Sage',
    address: {
      town: 'Udaipur',
      zipcode: '101028'
    }
  }
  {
    name: 'Misty Malabar',
    address: {
      town: 'Puducherry',
      zipcode: '105357'
    }
  }
```

Experiment 13:

LeetCode Practice - I

Problem – 570. Managers with at least 5 direct reports:



570. Managers with at Least 5 Direct Reports

Medium | Topics | Companies | Hint

SQL Schema > Pandas Schema >

Table: Employee

Column Name	Type
id	int
name	varchar
department	varchar
managerId	int

id is the primary key (column with unique values) for this table.
Each row of this table indicates the name of an employee, their department, and the id of their manager.
If managerId is null, then the employee does not have a manager.
No employee will be the manager of themself.

Query:

Write a solution to find managers with at least five direct reports.

```
SELECT e.name
FROM Employee e, Employee m
WHERE e.id = m.managerId
GROUP BY m.managerId
HAVING COUNT(m.managerId) >= 5;
```

Input	Output

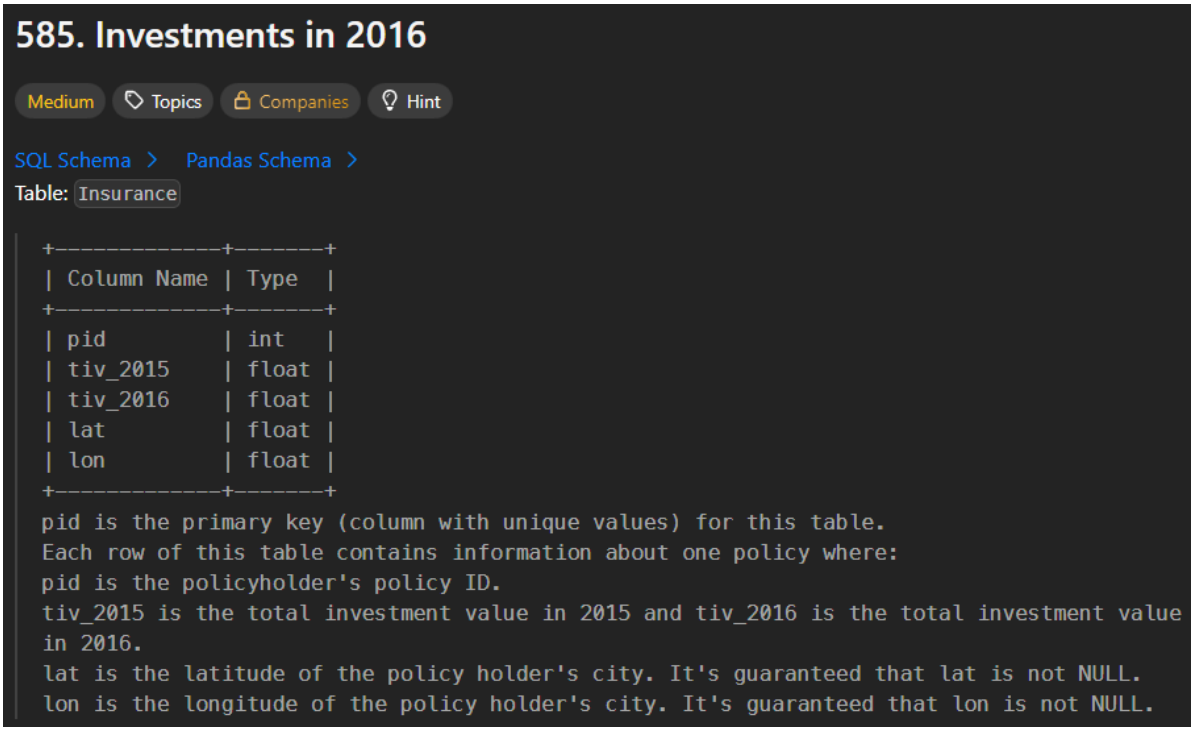
```
ployee =
id  | name  | department | managerId |
--- | ----- | ---------- | --------- |
101 | John  | A          | null      |
102 | Dan   | A          | 101       |
103 | James | A          | 101       |
104 | Amy   | A          | 101       |
105 | Anne  | A          | 101       |
106 | Ron   | B          | 101       |
```

```
| name |
| ---- |
| John |
```

Experiment 14:

LeetCode Practice - II

Problem – 585. Investments in 2016:



585. Investments in 2016

Medium | Topics | Companies | Hint

SQL Schema > Pandas Schema >

Table: Insurance

Column Name	Type
pid	int
tiv_2015	float
tiv_2016	float
lat	float
lon	float

pid is the primary key (column with unique values) for this table.
Each row of this table contains information about one policy where:
pid is the policyholder's policy ID.
tiv_2015 is the total investment value in 2015 and tiv_2016 is the total investment value in 2016.
lat is the latitude of the policy holder's city. It's guaranteed that lat is not NULL.
lon is the longitude of the policy holder's city. It's guaranteed that lon is not NULL.

Query:

Write a solution to report the sum of all total investment values in 2016 tiv_2016, for all policyholders who:

- Have the same tiv_2015 value as one or more other policyholders, and
- Are not located in the same city as any other policyholder (i.e., the (lat, lon) attribute pairs must be unique).

Round tiv_2016 to two decimal places.

```
SELECT ROUND(SUM(tiv_2016), 2) AS tiv_2016
FROM Insurance
WHERE (lat, lon) IN (
        SELECT lat, lon
        FROM Insurance
        GROUP BY lat, lon
        HAVING COUNT(*) = 1)
AND tiv_2015 IN (
        SELECT tiv_2015
        FROM Insurance
        GROUP BY tiv_2015
        HAVING COUNT(*) > 1
);
```

Input

Insurance =

pid	tiv_2015	tiv_2016	lat	lon
1	100	10	1	1
2	100	20	2	2
3	200	30	3	3
4	200	40	4	4
5	300	50	3	3

Output

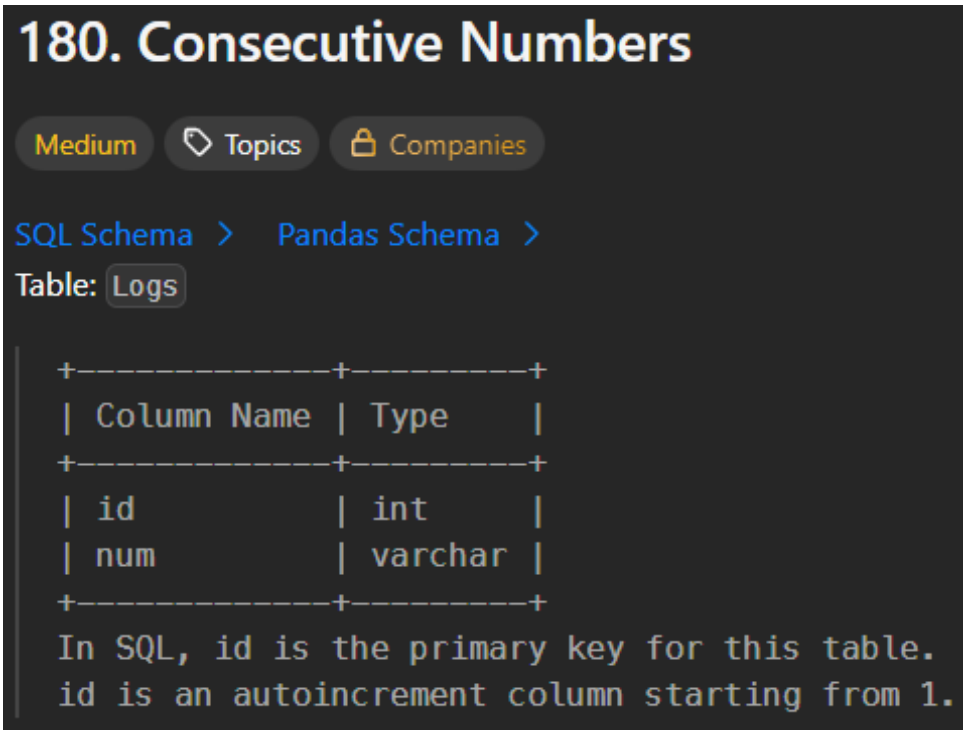
tiv_2016
70

Output is 70 because the location constraint is checked irrespective of tiv_2015. The pid 3 and pid 5 have same lat, lat pair – 3,3. Hence pid 3 is excluded from the output. Final output will only tiv_2016 from pid 1, 2 and 4 = 10 + 20 + 40 = 70.

Experiment 15:

LeetCode Practice - III

Problem – 180. Consecutive Numbers:



Query:

Find all numbers that appear at least three times consecutively.

```
SELECT DISTINCT l1.num AS ConsecutiveNums
FROM Logs l1, Logs l2, Logs l3
WHERE l1.num = l2.num AND
        l2.num = l3.num AND
        l1.id = l2.id – 1 AND
        l2.id = l3.id - 1;
```

Input

Logs =

id	num
1	1
2	1
3	1
4	2
5	2
6	2
7	3

Output

ConsecutiveNums
1
2