

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

Database Management Systems (23CS3PCDBM)

Submitted by

Crevan Neil Fernandes(1BM23CS082)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Sep-2024 to Jan-2025

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “Database Management Systems (23CS3PCDBM)” carried out by **Crevan Neil Fernandes(1BM23CS082)**, 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 2024. 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.

Ms. Sheetal V A Assistant Professor Department of CSE, BMSCE	Dr. Kavita Sooda Professor HOD Department of CSE, BMSCE
--	---

Index

Sl. No.	Experiment Title	Page No.
1	Insurance Database	4-8
2	More Queries on Insurance Database	9-12
3	Bank Database	12-17
4	More Queries on Bank Database	17-20
5	Employee Database	20-28
6	More Queries on Employee Database	28-30
7	Supplier Database	30-36
8	NO SQL - Student Database	37-38
9	NO SQL - Customer Database	38-40
10	NO SQL – Restaurant Database	40-44

Insurance Database

Question (Week 1)

Consider the Insurance database given below.

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) i.

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

ii. Enter at least five tuples for each relation.

iii. Display Accident date and location.

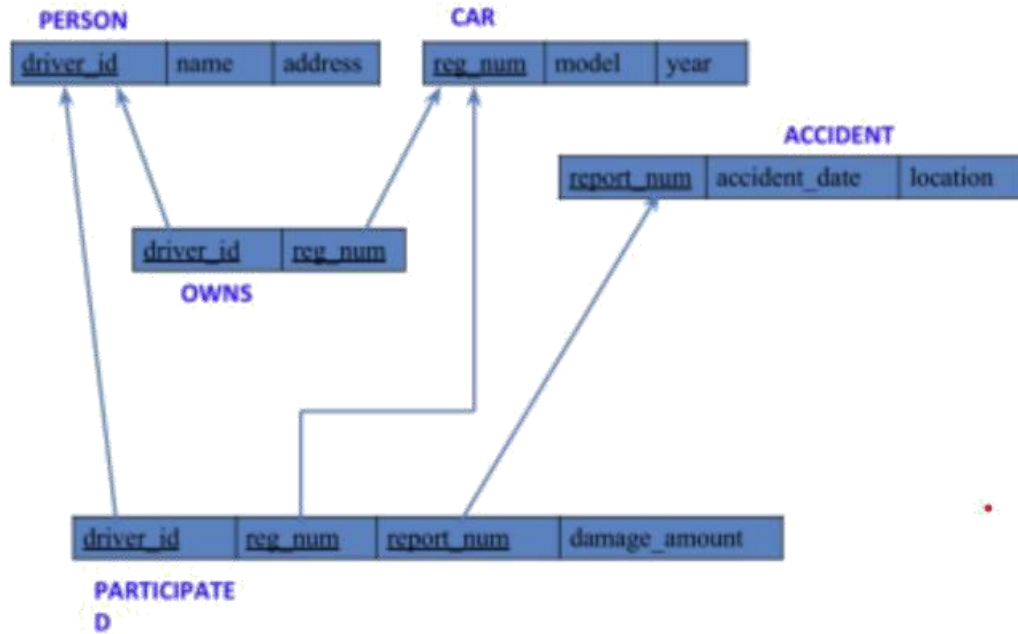
iv. Update the damage amount to 25000 for the car with a specific reg_num (example 'K A053408') for which the accident report number was 12.

v. Add a new accident to the database.

vi. Display Accident date and location.

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

Schema



```
CREATE DATABASE insurance; USE insurance;
```

```
create table person( driver_id varchar(25) PRIMARY KEY, name varchar(25), address varchar(25));
```

```
create table car( reg_num varchar(25) Primary key, model varchar(25), year int);
```

```
create table accident( report_num int, accident_date date, location varchar(25));
```

```
create table owns( driver_id varchar(25) , reg_num varchar(25),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(25), reg_num varchar(25) , report_num int, damage_amount int, primary
key(driver_id,reg_num), foreign key(driver_id) references person(driver_id) , foreign key (reg_num) references
car(reg_num));
```




Structure of the table

```
desc person;
```

Result Grid						
		Filter Rows:			Export:	
	Field	Type	Null	Key	Default	Extra
►	driver_id	varchar(25)	NO	PRI	NULL	
desc accident;	name	varchar(25)	YES		NULL	
	address	varchar(25)	YES		NULL	

Result Grid		Filter Rows:		Exports:		
	Field	Type	Null	Key	Default	Extra
▶	driver_id	varchar(25)	NO	PRI	NULL	
	name	varchar(25)	YES		NULL	
	address	varchar(25)	YES		NULL	

desc participated;

Result Grid		 Filter Rows:			Export:	
	Field	Type	Null	Key	Default	Extra
	driver_id	varchar(25)	NO	PRI	NULL	
	name	varchar(25)	YES		NULL	
	address	varchar(25)	YES		NULL	

desc car;

Field	Type	Null	Key	Default	Extra
reg_num	varchar(25)	NO	PRI	NULL	
model	varchar(25)	YES		NULL	
year	int	YES		NULL	

desc own;

Field	Type	Null	Key	Default	Extra
reg_num	varchar(25)	NO	PRI	NULL	
model	varchar(25)	YES		NULL	
year	int	YES		NULL	

Insertion of values in the table:

insert into person values ("A01","richard","srinivas nagar"), ("A02","Pradeep","Rajaji nagar"), ("A03","Smith","Ashok nagar"), ("A04","Venu","N R Colony"), ("A05","Jhon","Hanumanth nagar");

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

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

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

insert into owns_204 values ("A01", "KA052250"),("A02", "KA031181"),("A03", "KA095477"),("A04", "KA053408"),("A05", "KA041702");

select * from person;

select * from participated;

select * from car;

select * from accident;

select * from owns;

	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

	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

	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

	report_num	accident_date	location
▶	11	2003-01-01	Mysore Road
	12	2004-02-02	South end Circle
	13	2003-01-21	Bull temple Road
	14	2008-02-17	Mysore Road
	15	2004-03-05	Kanakpura Road

	driver_id	reg_num
▶	A02	KA03118
	A05	KA041702
	A01	KA052250
	A04	KA053408
	A03	KA095477
•	NULL	NULL

Queries:

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

update participated SET damage_amount=25000 where reg_num="KA053408"
AND report_num=12;

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

Add a new accident to the database

insert into accident values(16,'2008-03-08','Domlur');

	report_num	accident_date	location
▶	11	2003-01-01	Mysore Road
	12	2004-02-02	South end Circle
	13	2003-01-21	Bull temple Road
	14	2008-02-17	Mysore Road
	15	2004-03-05	Kanakpura Road
	16	2008-03-08	Domlur

Display Accident date and location

select accident_date,location
from accident;

	accident_date	location
▶	2003-01-01	Mysore Road
	2004-02-02	South end Circle
	2003-01-21	Bull temple Road
	2008-02-17	Mysore Road
	2004-03-05	Kanakpura Road
	2008-03-08	Domlur

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

PROGRAM 2. More Queries on Insurance Database

PERSON (driver_id: String, name: String, address: String)

CAR (reg_num: String, model: String, year: int) A

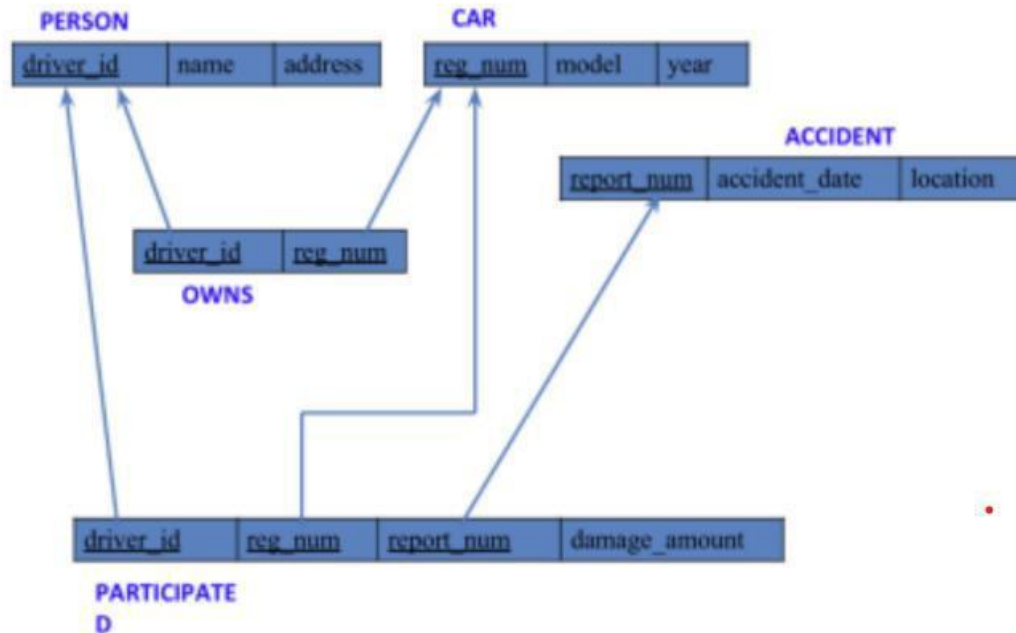
CCIDENT (report_num: int, accident_date: date, location: String)

OWNS (driver_id: String, reg_num: String)

PARTICIPATED (driver_id: String, reg_num: String, report_num: int, damage_amount: int) Create the above tables by properly specifying the primary keys and the foreign keys as done in "Program-1 " week's lab and Enter at least five tuples for each relation.

- i. Display the entire CAR relation in the ascending order of manufacturing year.
- ii. Find the number of accidents in which cars belonging to a specific model (example 'Lancer') were involved.
- iii. Find the total number of people who owned cars that involved in accidents in 2008.
- iv. List the entire participated relation in the Descending Order of Damage Amount.
- v. Find the Average Damage Amount.
- vi. Delete the tuple whose Damage Amount is below the Average Damage Amount
- vii. List the name of drivers whose Damage is Greater than the Average Damage Amount.
- viii. Find Maximum Damage Amount.

Schema:



Queries:

Display the entire CAR relation in the ascending order of manufacturing year. `select * from car order by year asc;`

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

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

`select model, count(model) from participated, car where participated.reg_num = car.reg_num group by model;`

	model	count(model)
►	Lancer	1
	Audi	1
	Indica	1
	Honda	1
	Toyota	1

List the entire participated relation in the Descending Order of Damage Amount.

```
select * from participated
```

```
order by damage_amount;
```

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

Find the Average Damage Amount.

```
select avg(damage_amount) as Avg_Damage_Amount from participated;
```

	Avg_Damage_Amount
▶	13600.0000

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

```
delete from participated where damage_amount < (select * from  
(select avg(damage_amount) from participated) as average);
```

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

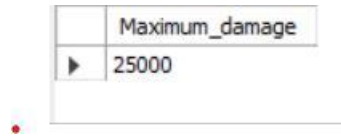
List the name of drivers whose Damage is Greater than the Average Damage Amount.

```
select name from person join participated on participated.driver_id = person.driver_id  
where damage_amount >= (select avg(damage_amount) from participated);
```

	name
▶	Pradeep
	Smith

Find Maximum Damage Amount.

select max(damage_amount) as Maximum_damage from participated;



Maximum_damage
25000

PROGRAM 3: Bank Database

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

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

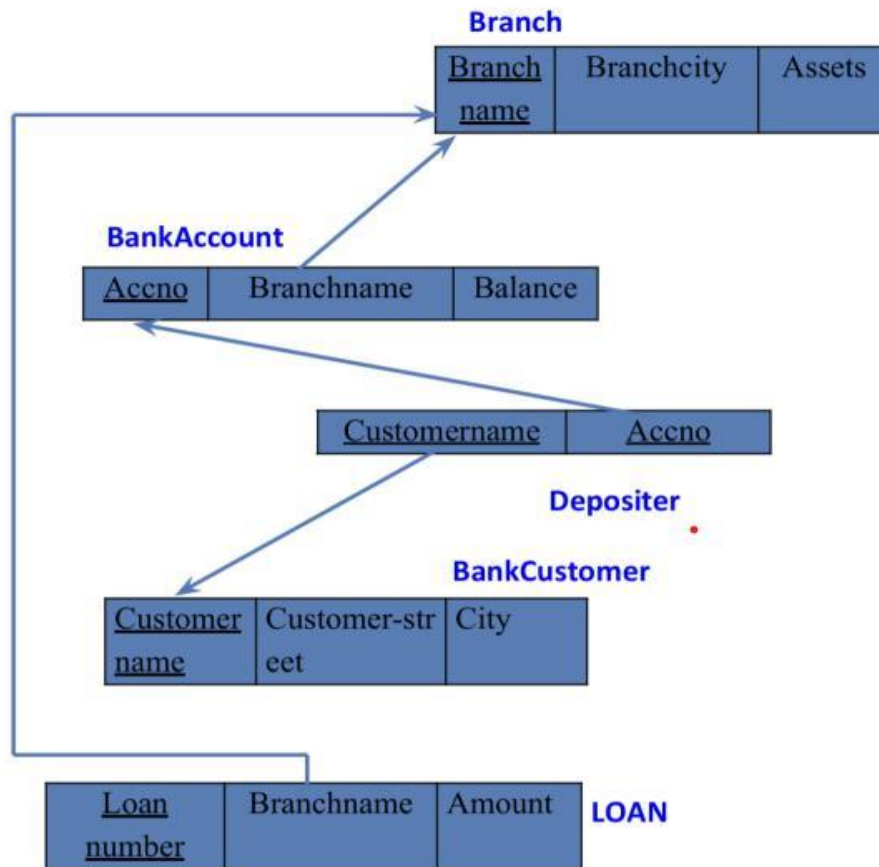
BankCustomer (customer-name: String, customer-street: String, customer-city: String)

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

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

- i. Create the above tables by properly specifying the primary keys and the foreign keys.
- ii. Enter at least five tuples for each relation.
- iii. Display the branch name and assets from all branches in lakhs of rupees and rename the assets column to 'assets in lakhs'.
- iv. Find all the customers who have at least two accounts at the same branch (ex. SBI_ResidencyRoad).
- v. Create a view which gives each branch the sum of the amount of all the loans at the branch.

Schema



```
create database bank;
```

```
use bank;
```

```
create table branch (branch_name varchar(25), branch_city varchar(25), assets
real, primary key(branch_name));
```

```
create table bank_account (accno int, branch_name varchar(25), balance real,
primary key(accno), foreign key (branch_name) references branch(branch_name));
```

```
create table bankcustomer (customer_name varchar(25), customer_street
varchar(25), customer_city varchar(25), primary key (customer_name));
```

```
create table depositer (customer_name varchar(25), accno int, primary key(accno),
foreign key (customer_name) references bankcustomer(customer_name),foreign key
(accno) references bank_account(accno));
```

create table loan (loan_number int, branch_name varchar(25), amount real, primary key(loan_number), foreign key (branch_name) references branch(branch_name));

Structure of Table:

desc bank_account;

	Field	Type	Null	Key	Default	Extra
►	accno	int	NO	PRI	NULL	
	branch_name	varchar(25)	YES	MUL	NULL	
	balance	double	YES		NULL	

desc bankcustomer;

	Field	Type	Null	Key	Default	Extra
►	customer_name	varchar(25)	NO	PRI	NULL	
	customer_street	varchar(25)	YES		NULL	
	customer_city	varchar(25)	YES		NULL	

desc branch;

	Field	Type	Null	Key	Default	Extra
►	branch_name	varchar(25)	NO	PRI	NULL	
	branch_city	varchar(25)	YES		NULL	
	assets	double	YES		NULL	

desc depositer;

	Field	Type	Null	Key	Default	Extra
►	customer_name	varchar(25)	YES	YES	NULL	
	accno	int	NO	PRI	NULL	

desc loan;

	Field	Type	Null	Key	Default	Extra
►	loan_number	int	NO	PRI	NULL	
	branch_name	varchar(25)	YES	MUL	NULL	
	amount	double	YES		NULL	

Inserting values:

```
insert into branch values ("SBI_chamrajpet","Bangalore",50000),
("SBI_ResidencyROad","Bangalore",10000), ("SBI_ShivajiRoad","Bombay",20000),
("SBI_ParliamentRoad","Delhi",10000), ("SBI_Jantarmanatar","Delhi",20000);
```

```
insert into bank_account values (1,"SBI_Chamrajpet",2000), (2,"SBI_ResidencyRoad",5000),
(3,"SBI_SHivajiRoad",6000), (4,"SBI_ParliamentRoad",9000), (5,"SBI_Jantarmanatar",8000),
(6,"SBI_ShivajiRoad",4000), (8,"SBI_ResidencyRoad",4000), (9,"SBI_ParliamentRoad",3000),
(10,"SBI_ResidencyRoad",5000), (11,"SBI_Jantarmanatar",2000);
```

```
insert into bankcustomer values ("Avinash","Bull_Temple_road","Bangalore"),
("Dinesh","Bannergatta_Road","Bangalore"), ("Mohan","NationalCollege_Road","Bangalore"),
("Nikhil","Akbar_Road","Delhi"), ("Ravi","Prithviraj_Road","Delhi");
```

```
insert into Depositer values ("Avinash",1), ("Dinesh",2), ("Nikhil",4), ("Ravi",5),
("Avinash",8), ("Nikhil",9), ("Dinesh",10), ("Nikhil",11);
```

insert into loan values

```
(1,"SBI_Chamrajpet",1000),(2,"SBI_ResidencyRoad",2000),(3,"SBI_ShivajiRoad",3000),(4,"SBI_Parli
amentRoad",4000),(5,"SBI_Jantarmanatar",5000);
```

select * from branch

	branch_name	branch_city	assets
▶	SBI_chamrajpet	Bangalore	50000
	SBI_Jantarmanatar	Delhi	20000
	SBI_ParliamentRoad	Delhi	10000
	SBI_ResidencyROad	Bangalore	10000
	SBI_ShivajiRoad	SBI_ShivajiRoad	20000
;	NULL	NULL	NULL

select * from depositer;

	customer_name	accno
▶	Avinash	1
	Avinash	8
	Dinesh	2
	Dinesh	10
	Nikhil	4
	Nikhil	9
	Nikhil	11
	Ravi	5

select * from loan;

	loan_number	branch_name	amount
▶	1	SBI_Chamrajpet	1000 1000
	2	SBI_ResidencyRoad	2000
	3	SBI_ShivajiRoad	3000
	4	SBI_ParliamentRoad	4000
	5	SBI_Jantarmanatar	5000
★	NULL	NULL	NULL

select * from bankcustomer;

	customer_name	customer_street	customer_city
▶	Avinash	Bull_Temple_road	Bangalore
	Dinesh	Bannergatta_Road	Bangalore
	Mohan	NationalCollege_Road	Bangalore
	Nikhil	Akbar_Road	Delhi
	Ravi	Prithviraj_Road	Delhi
★	NULL	NULL	NULL

select * from bank_account;

	accno	branch_name	balance
▶	1	SBI_Chamrajpet	2000
	2	SBI_ResidencyRoad	5000
	3	SBI_ShivajiRoad	6000
	4	SBI_ParliamentRoad	9000
	5	SBI_Jantarmanatar	8000
	6	SBI_ShivajiRoad	4000
	8	SBI_ResidencyRoad	4000
	9	SBI_ParliamentRoad	3000
	10	SBI_ResidencyRoad	5000
	11	SBI_Jantarmanatar	2000
★	NULL	NULL	NULL

Queries

Display the branch name and assets from all branches 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	0.5
	SBI_Jantarmanatar	0.2
	SBI_ParliamentRoad	0.1
	SBI_ResidencyRoad	0.1
	SBI_ShivajiRoad	0.2

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

```
select customer_name from (select customer_name,branch_name from
depositer join bank_account on depositer.accno=bank_account.accno)
as tab1 where branch_name="SBI_ResidencyRoad" group by
customer_name having count(customer_name)>1;
```

	customer_name
▶	Dinesh

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

```
create view Total_Loan_Per_Branch as select branch_name,sum(amount) as
Total_Loan from loan group by branch_name;
```

```
select * from Total_Loan_Per_Branch;
```

	branch_name	Total_Loan
▶	SBI_Chamrajpet	1000
	SBI_Jantarmanatar	5000
	SBI_ParliamentRoad	4000
	SBI_ResidencyRoad	2000
	SBI_ShivajiRoad	3000

PROGRAM 4: More Queries on Bank Database

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

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

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

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

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

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

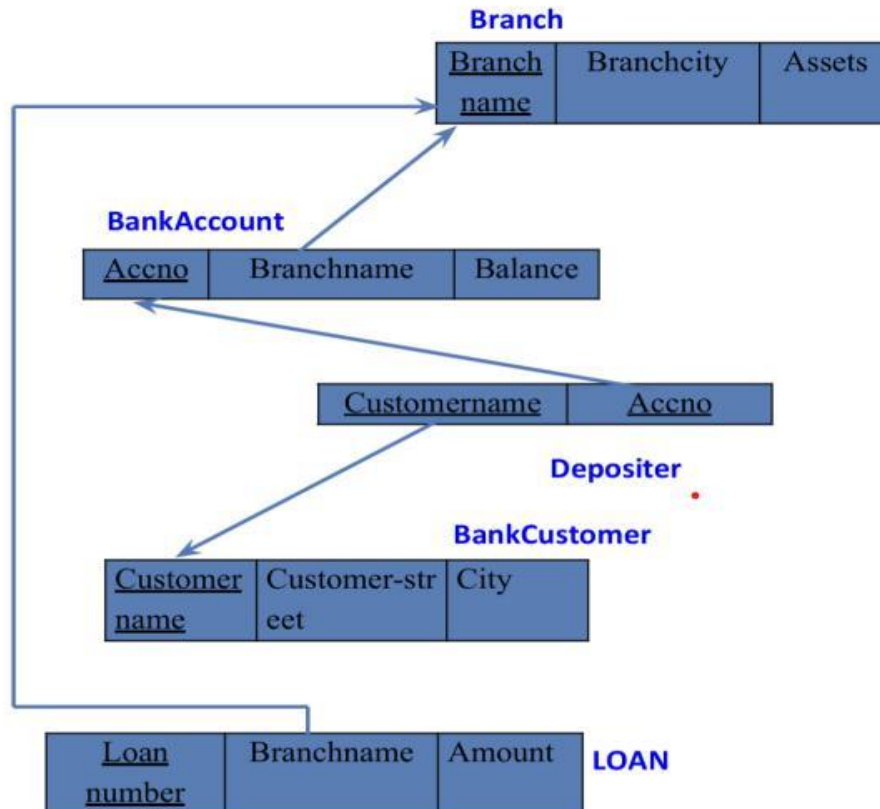
ii. Find all customers who have a loan at the bank but do not have an account.

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

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

v. Demonstrate how you delete all account tuples at every branch located in a specific city (Ex. Bombay). vi. Update the Balance of all accounts by 5%.

Schema:



```
create table borrower( customer_name varchar(20), loan_number int,
foreign key(customer_name) references bankcustomer(customer_name),
foreign key(loan_number) references loan(loan_number) );
```

```
insert into branch values ("SBI_MantriMarg", "Delhi", 200000);
```

```
insert into bank_account values (12, "SBI_MantriMarg", 2000);
```

```
insert into depositor values("Nikhil", 12);
```

```
insert into borrower values ("Avinash", 1), ("Dinesh", 2), ("Mohan", 3), ("Nikhil", 4),
("Ravi", 5);
```

Queries:

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

```
select d.customer_name from branch b, deposits d, bank_account ba where
b.branch_city='Delhi' and d.accno=ba.acc_no and b.branchname=ba.branchname
group by d.customername having count(distinct b.branchname)= (select count(distinct
b.branchname) from branch_204 b where b.branch_city='Delhi');
```

customer_name
Nikhil

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

```
select customer_name, loan.loan_no from (borrower right outer join loan on loan.loan_no
=borrower.loan_no) where customer_name not in (select customer_name from
deposits bank_account where deposits.acc_no = bank_account.accno group by
customer_name, branch_name);
```

customer_name	loan_number
Mohan	3

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

Branch.

```
select distinct customer_name from depositer where customer_name in (select
depositer.customer_name from branch, bank_account, deposits where
branch.branch_city = "Banglore" and branch.branch_name = bank_account.branch_name
and bank_account.acc_no = depositer.acc_no) and customer_name in (select
customer_name from borrower, loan where branch_name in (select branch_name from
branch where branch_city = "Banglore"));
```

	customer_name
▶	Avinash
	Dinesh

Refresh data

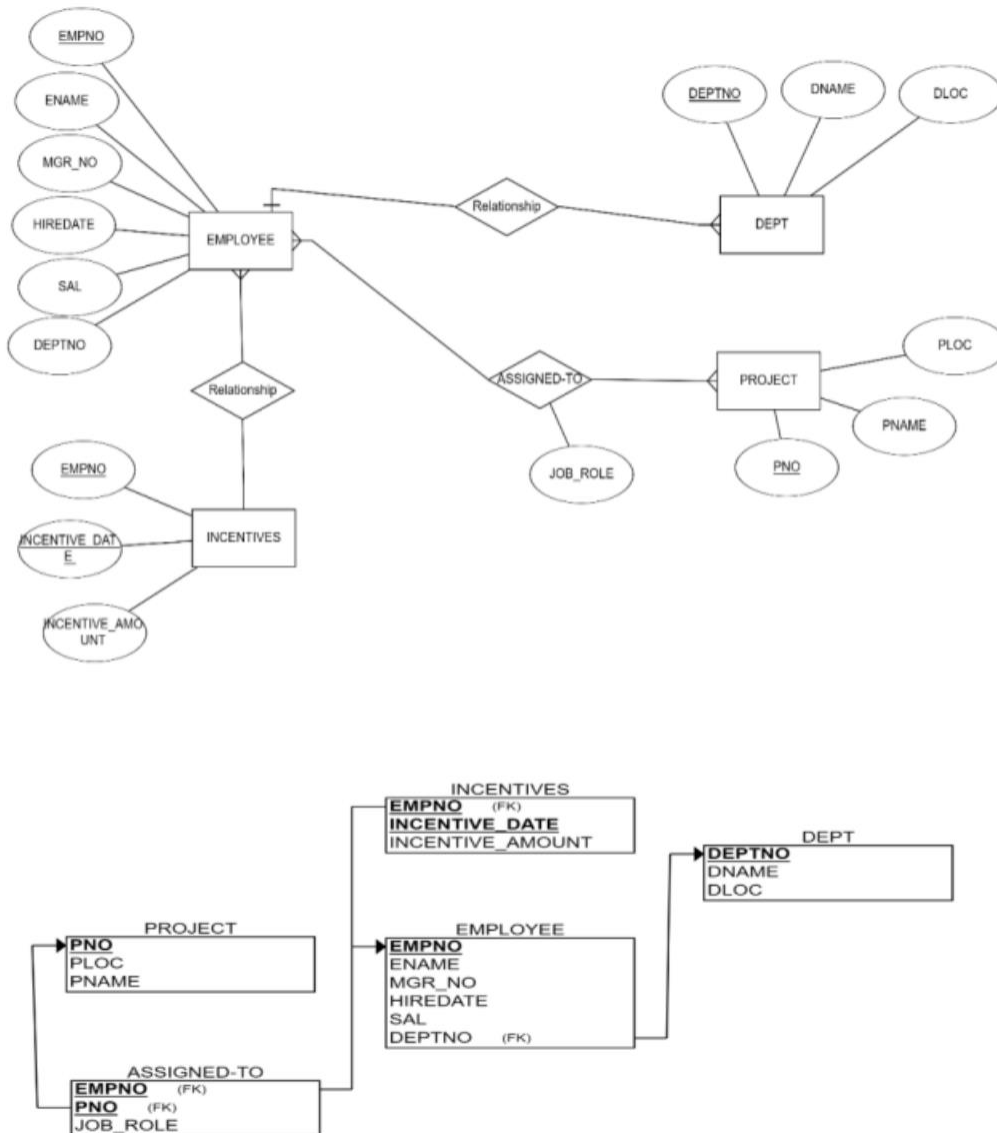
Update the Balance of all accounts by 5% .

```
update bank_account set balance = 1.05*balance;
```

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

```
delete from bank_account where branch_name in (select branch_name from
branch where branch_city = "Bombay");
```

PROGRAM 5: Employee Database



- Using Scheme diagram, create tables by properly specifying the primary keys and the foreign keys.
- Enter greater than five tuples for each table.
- Retrieve the employee numbers of all employees who work on project located in Bengaluru, Hyderabad, or Mysuru.
- Get Employee ID's of those employees who didn't receive incentives.

v. 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.

Create Database and Table:

```
CREATE DATABASE employee;
```

```
USE employee;
```

```
CREATE TABLE employee (
```

```
    empno INT,
```

```
    mgrno INT,
```

```
    hiredate DATE,
```

```
    sal INT,
```

```
    deptno INT,
```

```
    PRIMARY KEY(empno)
```

```
);
```

```
CREATE TABLE incentives (
```

```
    empno INT,
```

```
    incentive_date DATE,
```

```
    incentive_amount INT,
```

```
    PRIMARY KEY(empno, incentive_date),
```

```
    FOREIGN KEY(empno) REFERENCES employee(empno)
```

```
);
```

```
CREATE TABLE project (
```

```
    pno INT,
```

```
    ploc VARCHAR(26),
```

```

    pname VARCHAR(25),
    PRIMARY KEY(pno)
);

CREATE TABLE assigendto (
    empno INT,
    pno INT,
    jobrole VARCHAR(25),
    PRIMARY KEY(empno, pno),
    FOREIGN KEY(empno) REFERENCES employee(empno),
    FOREIGN KEY(pno) REFERENCES project(pno)
);

CREATE TABLE dept (
    deptno INT NOT NULL UNIQUE,
    dname VARCHAR(25),
    dloc VARCHAR(25),
    PRIMARY KEY(deptno)
);

```

Structure of Tables:

```
desc employee;
```

	Field	Type	Null	Key	Default	Extra
►	empno	int	NO	PRI	NULL	
	mgrno	int	YES		NULL	
	hiredate	date	YES		NULL	
	sal	int	YES		NULL	
	deptno	int	YES		NULL	

```
desc incentives;
```

	Field	Type	Null	Key	Default	Extra
►	empno	int	NO	PRI	NULL	
	incentive_date	date	NO	PRI	NULL	
	incentive_amount	int	YES		NULL	

desc project;

	Field	Type	Null	Key	Default	Extra
►	pno	int	NO	PRI	NULL	
	ploc	varchar(26)	YES		NULL	
	pname	varchar(25)	YES		NULL	

desc assignedto;

	Field	Type	Null	Key	Default	Extra
►	empno	int	NO	PRI	NULL	
	pno	int	NO	PRI	NULL	
	jobrole	varchar(25)	YES		NULL	

desc dept;

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

Inserting values :

```
INSERT INTO employee (empno, mgrno, hiredate, sal, deptno) VALUES
(101, 201, '2020-01-15', 50000, 1), (102, 201, '2021-06-10', 45000, 2),
(103, 202, '2019-03-05', 60000, 3), (104, 203, '2022-09-01', 55000, 2),
(105, 203, '2023-02-14', 40000, 1);
```

```
INSERT INTO incentives (empno, incentive_date, incentive_amount)
VALUES (101, '2023-06-01',5000),(102,'2023-05-15',3000),(103, '2023-
07-20', 4500), (104, '2023-08-10', 4000), (105, '2023-09-01', 2500);
```



```
INSERT INTO project (pno, ploc, pname) VALUES (1, 'New York', 'Project Alpha'), (2, 'London', 'Project Beta'), (3, 'Mumbai', 'Project Gamma'), (4, 'Berlin', 'Project Delta'), (5, 'Tokyo', 'Project Epsilon');
```

```
INSERT INTO assigendto (empno, pno, jobrole) VALUES (101, 1, 'Developer'), (102, 2, 'Tester'), (103, 3, 'Manager'), (104, 4, 'Analyst'), (105, 5, 'Intern');
```

```
INSERT INTO dept (deptno, dname, dloc) VALUES (1, 'HR', 'New York'), (2, 'Finance', 'London'), (3, 'Engineering', 'Mumbai'), (4, 'Marketing', 'Berlin'), (5, 'Sales', 'Tokyo');
```

```
select * from employee;
```

	empno	mgrno	hiredate	sal	deptno
▶	101	201	2020-01-15	50000	1
	102	201	2021-06-10	45000	2
	103	202	2019-03-05	60000	3
	104	203	2022-09-01	55000	2
	105	203	2023-02-14	40000	1
	111	205	2021-01-15	50400	1
	NULL	NULL	NULL	NULL	NULL

```
select * from assigendto;
```

	empno	pno	jobrole
▶	101	1	Developer
	102	2	Tester
	103	3	Manager
	104	4	Analyst
	105	5	Intern
*	NULL	NULL	NULL

```
select * from dept;
```

	deptno	dname	dloc
▶	1	HR	New York
	2	Finance	London
	3	Engineering	Mumbai
	4	Marketing	Berlin
	5	Sales	Tokyo
*	NULL	NULL	NULL

select * from incentives;

	empno	incentive_date	incentive_amount
▶	101	2023-06-01	5000
	102	2023-05-15	3000
	103	2023-07-20	4500
	104	2023-08-10	4000
	105	2023-09-01	2500
*	NULL	NULL	NULL

select * from project;

	pno	ploc	pname
▶	1	New York	Project Alpha
	2	London	Project Beta
	3	Mumbai	Project Gamma
	4	Berlin	Project Delta
	5	Tokyo	Project Epsilon

QUERIES:

Retrieve the employee numbers of all employees who work on project located in Berlin,Tokyo,Mumbai.

select empno

from assigndto as a

join project as p

where a.pno= p.pno and ploc in("Mumbai","Tokyo","Berlin");

	empno
▶	103
	104
	105

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

```

insert into employee values( 111, 205, '2021-01-15', 50400, 1);

select empno
from employee
where empno not in(select empno from incentives);

```



	empno
▶	111
✱	NULL

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.

```

create view a as
select empno,dname,dloc
from employee
join dept
where employee.deptno=dept.deptno;

create view b as
select empno, jobrole,ploc
from assigendto
join project
where project.pno=assigendto.pno;

select a.empno,dname,jobrole,dloc,ploc
from a
join b
where a.dloc=b.ploc and a.empno=b.empno;

```

	empno	dname	jobrole	dloc	ploc
▶	101	HR	Developer	New York	New York
	102	Finance	Tester	London	London
	103	Engineering	Manager	Mumbai	Mumbai
	104	Marketing	Analyst	Berlin	Berlin
	105	Sales	Intern	Tokyo	Tokyo

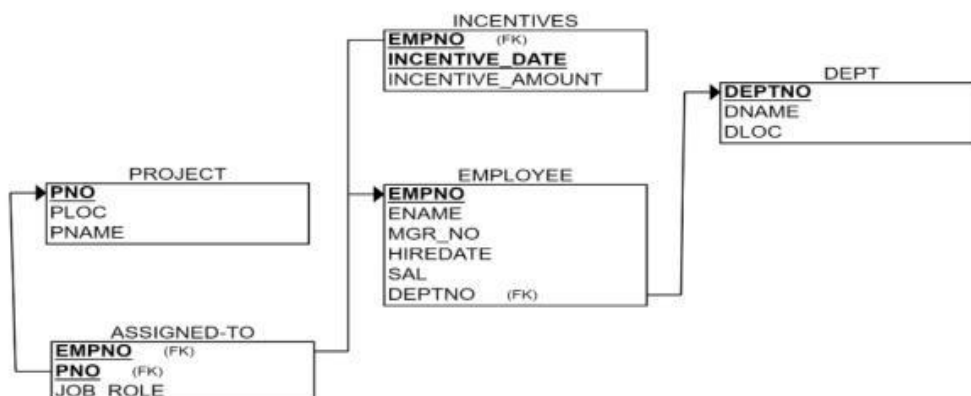
PROGRAM 6: More Queries on Employee Database

Using Scheme diagram (under Program-5),

Create tables by properly specifying the primary keys and the foreign keys.

- ii. Enter greater than five tuples for each table.
- iii. List the name of the managers with the maximum employees
- iv. Display those managers name whose salary is more than average salary of his employee.
- v. Find the name of the second top level managers of each department.
- vi. Find the employee details who got second maximum incentive in January 2019.
- vii. Display those employees who are working in the same department where his manager is working.

Schema:



Queries:

List the name of the managers with the maximum employees

```

create view mgre as
select mgrno,count(empno ) as e
from employee
group by mgrno ;
select mgrno
from mgre
where e=
(select max(e)
from mgre);

```



	mgrno
▶	201
	203

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

```

SELECT e1.empno AS Manager_ID, e1.sal AS Manager_Salary FROM employee e1 WHERE
e1.sal > ( SELECT AVG(e2.sal) FROM employee e2 WHERE e2.mgrno = e1.empno );

```

Find the name of the second top level managers of each department.

```

SELECT empno, incentive_date, incentive_amount FROM incentives WHERE
incentive_date BETWEEN '2019-01-01' AND '2019-01-31' ORDER BY incentive_amount
DESC LIMIT 1 OFFSET 1; -- This retrieves the second maximum incentive

```

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

```

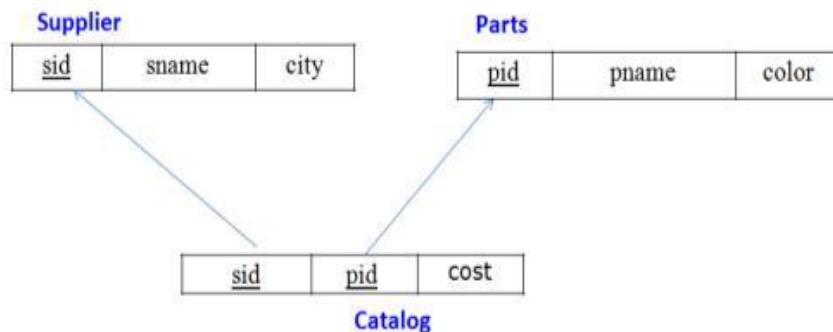
SELECT e1.empno AS Employee_ID, e1.deptno AS Department_ID, e1.mgrno AS
Manager_ID FROM employee e1 JOIN employee e2 ON e1.mgrno = e2.empno
WHERE e1.deptno = e2.deptno;

```

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

```
WITH TopLevelManagers AS (  
    SELECT empno AS Manager_ID  
    FROM employee  
    WHERE mgrno IS NULL  
)  
  
SecondLevelManagers AS (  
    SELECT e1.empno AS Manager_ID, e1.deptno  
    FROM employee e1  
    JOIN TopLevelManagers t ON e1.mgrno = t.Manager_ID  
)  
  
SELECT e2.empno, e2.deptno, e2.customername AS  
SecondTopManager FROM employee e2  
JOIN SecondLevelManagers s ON e2.empno = s.Manager_ID;
```

PROGRAM 7: Supplier Database



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

- ii. Insert appropriate records in each table.
- iii. Find the pnames of parts for which there is some supplier.
- iv. Find the snames of suppliers who supply every part
- v. Find the snames of suppliers who supply every red part.
- vi. Find the pnames of parts supplied by Acme Widget Suppliers and by no one else.
- vii. 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).
- viii. For each part, find the sname of the supplier who charges the most for that part.

Create Database and Table:

create database supplier;

use supplier;

CREATE TABLE suppliers (

 sid INT PRIMARY KEY,

 sname VARCHAR(50),

 city VARCHAR(50)

);

CREATE TABLE parts (

 pid INT PRIMARY KEY,

 pname VARCHAR(50),

 color VARCHAR(20)

);

CREATE TABLE catalog (

```

sid INT,

pid INT,

cost DECIMAL(10, 2),

PRIMARY KEY (sid, pid),

FOREIGN KEY (sid) REFERENCES suppliers(sid),

FOREIGN KEY (pid) REFERENCES parts(pid)

);

```

Structure Of tables:

desc catalog;

	Field	Type	Null	Key	Default	Extra
►	sid	int	NO	PRI	NULL	
	pid	int	NO	PRI	NULL	
	cost	decimal(10,2)	YES		NULL	

desc parts;

	Field	Type	Null	Key	Default	Extra
►	pid	int	NO	PRI	NULL	
	pname	varchar(50)	YES		NULL	
	color	varchar(20)	YES		NULL	

desc suppliers;

	Field	Type	Null	Key	Default	Extra
►	sid	int	NO	PRI	NULL	
	sname	varchar(50)	YES		NULL	
	city	varchar(50)	YES		NULL	

Inserting values:

```
INSERT INTO suppliers VALUES (1, 'Acme Widget Suppliers', 'New York');
```



```

INSERT INTO suppliers VALUES (2, 'Global Industries', 'Los
Angeles'); INSERT INTO suppliers VALUES (3, 'Tech Supplies
Co.', 'Chicago') INSERT INTO parts VALUES (101, 'Bolt', 'Red');
INSERT INTO parts VALUES (102, 'Nut', 'Blue');
INSERT INTO parts VALUES (103, 'Screw', 'Red');
INSERT INTO parts VALUES (104, 'Washer', 'Green');
INSERT INTO catalog VALUES (1, 101, 50.00);
INSERT INTO catalog VALUES (1, 102, 30.00);
INSERT INTO catalog VALUES (2, 103, 60.00);
INSERT INTO catalog VALUES (3, 104, 40.00);
INSERT INTO catalog VALUES (2, 101, 55.00);

```

Select * from catalog;

	sid	pid	cost
▶	1	101	50.00
	1	102	30.00
	2	101	55.00
	2	103	60.00
	3	104	40.00
*	NULL	NULL	NULL

Select * from parts;

	pid	pname	color
▶	101	Bolt	Red
	102	Nut	Blue
	103	Screw	Red
	104	Washer	Green
*	NULL	NULL	NULL

Select * from suppliers;

	sid	sname	city
▶	1	Acme Widget Suppliers	New York
	2	Global Industries	Los Angeles
	3	Tech Supplies Co.	Chicago
*	NULL	NULL	NULL

Queries:

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

```
SELECT DISTINCT pname
```

```
FROM parts
```

```
WHERE pid IN (SELECT pid FROM catalog);
```

	pname
▶	Bolt
	Nut
	Screw
	Washer

Find the snames of suppliers who supply every red part.

```
SELECT sname
```

```
FROM suppliers
```

```
WHERE NOT EXISTS (
```

```
    SELECT pid
```

```
    FROM parts
```

```
    WHERE color = 'Red' AND pid NOT IN (
```

```
        SELECT pid
```

```
        FROM catalog
```

```
WHERE catalog.sid = suppliers.sid  
  
)  
);
```

	sname
▶	Global Industries

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

```
SELECT pname  
  
FROM parts  
  
WHERE pid IN (  
  
    SELECT pid  
  
    FROM catalog  
  
    WHERE sid = (SELECT sid FROM suppliers WHERE sname = 'Acme Widget Suppliers')  
  
)  
  
AND pid NOT IN (  
  
    SELECT pid  
  
    FROM catalog  
  
    WHERE sid != (SELECT sid FROM suppliers WHERE sname = 'Acme Widget Suppliers')  
  
);
```

	pname
▶	Nut

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 c1.sid
FROM catalog c1
WHERE c1.cost > (
    SELECT AVG(c2.cost)
    FROM catalog c2
    WHERE c1.pid = c2.pid
);

```

	sid
▶	2

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

```

SELECT p.pname, s.sname, c.cost
FROM catalog c
JOIN suppliers s ON c.sid = s.sid
JOIN parts p ON c.pid = p.pid
WHERE c.cost = (
    SELECT MAX(cost)
    FROM catalog c2
    WHERE c2.pid = c.pid
);

```

	pname	sname	cost
▶	Nut	Acme Widget Suppliers	30.00
	Bolt	Global Industries	55.00
	Screw	Global Industries	60.00
	Washer	Tech Supplies Co.	40.00

PROGRAM 8: NoSQL Student Database

Perform the following DB operations using MongoDB.

- i. Create a database “Student” with the following attributes Rollno, Age, ContactNo, Email-Id.
- ii. Insert appropriate values
- iii. Write query to update Email-Id of a student with rollno 10.
- iv. Replace the student name from “ABC” to “FEM” of rollno 11.
- v. Export the created table into local file system
- vi. Drop the table.
- vii. Import a given csv dataset from local file system into mongodb collection.

Create database, table and insert values:

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

```
db.students.insertMany([ { Rollno: 10, Name: "John", Age: 20, ContactNo: "9876543210",  
EmailId: "john@example.com" }, { Rollno: 11, Name: "ABC", Age: 21, ContactNo:  
"9876543221", EmailId: "abc@example.com" }, { Rollno: 12, Name: "Jane", Age: 22,  
ContactNo: "9876543232", EmailId: "jane@example.com" } ])
```

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

```
db.students.updateOne( { Rollno: 10 }, { $set: { EmailId: "newemail@example.com" } } )
```

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

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

Export the created table into local file system

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

Drop the table.

```
db.students.drop()
```

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

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

```
Atlas atlas-qvfsgw-shard-0 [primary] Student> db.createCollection("students")
{ ok: 1 }
Atlas atlas-qvfsgw-shard-0 [primary] Student> use Student
already on db Student
Atlas atlas-qvfsgw-shard-0 [primary] Student> db.students.insertMany([
...   { Rollno: 10, Name: "John", Age: 20, ContactNo: "9876543210", EmailId: "john@example.com" },
...   { Rollno: 11, Name: "ABC", Age: 21, ContactNo: "9876543221", EmailId: "abc@example.com" },
...   { Rollno: 12, Name: "Jane", Age: 22, ContactNo: "9876543232", EmailId: "jane@example.com" }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67670cfe2c6914392a4eeb86'),
    '1': ObjectId('67670cfe2c6914392a4eeb87'),
    '2': ObjectId('67670cfe2c6914392a4eeb88')
  }
}
Atlas atlas-qvfsgw-shard-0 [primary] Student> db.students.updateOne(
...   { Rollno: 10 },
...   { $set: { EmailId: "newemail@example.com" } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-qvfsgw-shard-0 [primary] Student> db.students.updateOne(
...   { Rollno: 11 },
...   { $set: { Name: "FEM" } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

PROGRAM 9: NoSQL Customer Database

Perform the following DB operations using MongoDB.

- i. Create a collection by name Customers with the following attributes. Cust_id, Acc_Bal, Acc_Type
- ii. Insert at least 5 values into the table.
- iii. Write a query to display those records whose total account balance is greater than 1200 of account type 'Z' for each customer_id.
- iv. Determine Minimum and Maximum account balance for each

v. Export the created collection into local file system.

vi. Drop the table. vii. Import a given csv dataset from local file system into mongodb collection.

Create Database,Table and insert Values:

```
use CustomerDB
```

```
db.createCollection("Customers")
```

```
db.Customers.insertMany([ { Cust_id: 1, Acc_Bal: 1500, Acc_Type: 'Z' }, { Cust_id: 2, Acc_Bal: 1300, Acc_Type: 'Z' }, { Cust_id: 3, Acc_Bal: 1100, Acc_Type: 'Z' }, { Cust_id: 4, Acc_Bal: 2000, Acc_Type: 'A' }, { Cust_id: 5, Acc_Bal: 1700, Acc_Type: 'Z' } ])
```

Queries:

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

```
db.Customers.find({ Acc_Bal: { $gt: 1200 }, Acc_Type: 'Z' })
```

Determine Minimum and Maximum account balance for each customer_id.

```
db.Customers.aggregate([ { $group: { _id: "$Cust_id", min_balance: { $min: "$Acc_Bal" }, max_balance: { $max: "$Acc_Bal" } } } ])
```

Export the created collection into local file system.

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

Drop the table.

```
db.Customers.drop()
```

```

Atlas atlas-qvfsgw-shard-0 [primary] Student> use CustomerDB
switched to db CustomerDB
Atlas atlas-qvfsgw-shard-0 [primary] CustomerDB> db.createCollection("Customers")
{ ok: 1 }
Atlas atlas-qvfsgw-shard-0 [primary] CustomerDB> db.Customers.insertMany([
...   { Cust_id: 1, Acc_Bal: 1500, Acc_Type: 'Z' },
...   { Cust_id: 2, Acc_Bal: 1300, Acc_Type: 'Z' },
...   { Cust_id: 3, Acc_Bal: 1100, Acc_Type: 'Z' },
...   { Cust_id: 4, Acc_Bal: 2000, Acc_Type: 'A' },
...   { Cust_id: 5, Acc_Bal: 1700, Acc_Type: 'Z' }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('676710332c6914392a4eeb89'),
    '1': ObjectId('676710332c6914392a4eeb8a'),
    '2': ObjectId('676710332c6914392a4eeb8b'),
    '3': ObjectId('676710332c6914392a4eeb8c'),
    '4': ObjectId('676710332c6914392a4eeb8d')
  }
}
Atlas atlas-qvfsgw-shard-0 [primary] CustomerDB> db.Customers.find({ Acc_Bal: { $gt: 1200 }, Acc_Type: 'Z' })
[
  {
    _id: ObjectId('676710332c6914392a4eeb89'),
    Cust_id: 1,
    Acc_Bal: 1500,
    Acc_Type: 'Z'
  },
  {
    _id: ObjectId('676710332c6914392a4eeb8a'),
    Cust_id: 2,
    Acc_Bal: 1300,
    Acc_Type: 'Z'
  },
  {
    _id: ObjectId('676710332c6914392a4eeb8d'),
    Cust_id: 5,
    Acc_Bal: 1700,
    Acc_Type: 'Z'
  }
]
Atlas atlas-qvfsgw-shard-0 [primary] CustomerDB> db.Customers.aggregate([
...   { $group: { _id: "$Cust_id", min_balance: { $min: "$Acc_Bal" }, max_balance: { $max: "$Acc_Bal" } } }
... ])
[
  { _id: 2, min_balance: 1300, max_balance: 1300 },
  { _id: 3, min_balance: 1100, max_balance: 1100 },
  { _id: 4, min_balance: 2000, max_balance: 2000 },
  { _id: 5, min_balance: 1700, max_balance: 1700 },
  { _id: 1, min_balance: 1500, max_balance: 1500 }
]
Atlas atlas-qvfsgw-shard-0 [primary] CustomerDB> mongoexport --db CustomerDB --collection Customers --out customers.json

```

PROGRAM 10: NoSQL Restaurant Database

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'.

Create Database, Table and Inserting Values:

```
use RestaurantDB
```

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

```
db.restaurants.insertMany([
```

```
  { restaurant_id: 1, name: "Pizza Palace", town: "New York", cuisine: "Italian",  
    score: 8, zipcode: "10001", address: "123 Pizza St." },
```

```
  { restaurant_id: 2, name: "Sushi World", town: "San Francisco", cuisine:  
    "Japanese", score: 9, zipcode: "94105", address: "456 Sushi Ave." },
```

```
  { restaurant_id: 3, name: "Burger King", town: "Los Angeles", cuisine: "American",  
    score: 7, zipcode: "90001", address: "789 Burger Blvd." },
```

```
  { restaurant_id: 4, name: "Taco Bell", town: "Chicago", cuisine: "Mexican", score:  
    5, zipcode: "60601", address: "321 Taco Dr." },
```

```
  { restaurant_id: 5, name: "Pasta House", town: "Boston", cuisine: "Italian", score:  
    10, zipcode: "02101", address: "654 Pasta Rd." }  
])
```

Queries:

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

```
db.restaurants.find()
```

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

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

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.restaurants.find({ score: { $lte: 10 } }, { restaurant_id: 1, name: 1, town: 1, cuisine: 1 })
```

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

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

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

```
db.restaurants.find({ zipcode: /^10/ }, { name: 1, address: 1 })
```

```

Atlas atlas-qvfsgw-shard-0 [primary] restaurantdb> use abc
switched to db abc
Atlas atlas-qvfsgw-shard-0 [primary] abc> db.createCollection("restaurants")
{ ok: 1 }
Atlas atlas-qvfsgw-shard-0 [primary] abc> db.restaurants.insertMany([
... { restaurant_id: 1, name: "Pizza Palace", town: "New York", cuisine: "Italian", score: 8, zipcode: "10001", address: "123 Pizza St." },
... { restaurant_id: 2, name: "Sushi World", town: "San Francisco", cuisine: "Japanese", score: 9, zipcode: "94105", address: "456 Sushi Ave." },
... { restaurant_id: 3, name: "Burger King", town: "Los Angeles", cuisine: "American", score: 7, zipcode: "90001", address: "789 Burger Blvd." },
... { restaurant_id: 4, name: "Taco Bell", town: "Chicago", cuisine: "Mexican", score: 5, zipcode: "60601", address: "321 Taco Dr." },
... { restaurant_id: 5, name: "Pasta House", town: "Boston", cuisine: "Italian", score: 10, zipcode: "02101", address: "654 Pasta Rd." }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('676713222c6914392a4eeb93'),
    '1': ObjectId('676713222c6914392a4eeb94'),
    '2': ObjectId('676713222c6914392a4eeb95'),
    '3': ObjectId('676713222c6914392a4eeb96'),
    '4': ObjectId('676713222c6914392a4eeb97')
  }
}
Atlas atlas-qvfsgw-shard-0 [primary] abc> db.restaurants.find()
[
  {
    _id: ObjectId('676713222c6914392a4eeb93'),
    restaurant_id: 1,
    name: 'Pizza Palace',
    town: 'New York',
    cuisine: 'Italian',
    score: 8,
    zipcode: '10001',
    address: '123 Pizza St.'
  },
  {
    _id: ObjectId('676713222c6914392a4eeb94'),
    restaurant_id: 2,
    name: 'Sushi World',
    town: 'San Francisco',
    cuisine: 'Japanese',
    score: 9,
    zipcode: '94105',
    address: '456 Sushi Ave.'
  },
  {
    _id: ObjectId('676713222c6914392a4eeb95'),
    restaurant_id: 3,
    name: 'Burger King',
    town: 'Los Angeles',
    cuisine: 'American',
    score: 7,
    zipcode: '90001',
    address: '789 Burger Blvd.'
  },
  {
    _id: ObjectId('676713222c6914392a4eeb96'),
    restaurant_id: 4,
    name: 'Taco Bell',
    town: 'Chicago',
    cuisine: 'Mexican',
    score: 5,
    zipcode: '60601',
    address: '321 Taco Dr.'
  },
  {
    _id: ObjectId('676713222c6914392a4eeb97'),
    restaurant_id: 5,
    name: 'Pasta House',
    town: 'Boston',
    cuisine: 'Italian',
    score: 10,
    zipcode: '02101',
    address: '654 Pasta Rd.'
  }
]
Atlas atlas-qvfsgw-shard-0 [primary] abc> db.restaurants.find().sort({ name: -1 })
[
  {
    _id: ObjectId('676713222c6914392a4eeb96'),
    restaurant_id: 4,
    name: 'Taco Bell',
    town: 'Chicago',
    cuisine: 'Mexican',
    score: 5,
    zipcode: '60601',
    address: '321 Taco Dr.'
  },
  {
    _id: ObjectId('676713222c6914392a4eeb94'),
    restaurant_id: 2,
    name: 'Sushi World',
    town: 'San Francisco',
    cuisine: 'Japanese',
    score: 9,
    zipcode: '94105',
    address: '456 Sushi Ave.'
  },
  {
    _id: ObjectId('676713222c6914392a4eeb93'),
    restaurant_id: 1,
    name: 'Pizza Palace',
    town: 'New York',
    cuisine: 'Italian',
    score: 8,
    zipcode: '10001',
    address: '123 Pizza St.'
  },
  {
    _id: ObjectId('676713222c6914392a4eeb97'),
    restaurant_id: 5,
    name: 'Pasta House',
    town: 'Boston',
    cuisine: 'Italian',
    score: 10,
    zipcode: '02101',
    address: '654 Pasta Rd.'
  }
]

```

```

},
{
  _id: ObjectId('676713222c6914392a4eeb93'),
  restaurant_id: 1,
  name: 'Pizza Palace',
  town: 'New York',
  cuisine: 'Italian',
  score: 8,
  zipcode: '10001',
  address: '123 Pizza St.'
},
{
  _id: ObjectId('676713222c6914392a4eeb97'),
  restaurant_id: 5,
  name: 'Pasta House',
  town: 'Boston',
  cuisine: 'Italian',
  score: 10,
  zipcode: '02101',
  address: '654 Pasta Rd.'
},
{
  _id: ObjectId('676713222c6914392a4eeb95'),
  restaurant_id: 3,
  name: 'Burger King',
  town: 'Los Angeles',
  cuisine: 'American',
  score: 7,
  zipcode: '90001',
  address: '789 Burger Blvd.'
}
}

Atlas atlas-qvfsgw-shard-0 [primary] abc> db.restaurants.find({ score: { $lte: 10 } }, { restaurant_id: 1, name: 1, town: 1, cuisine: 1 })

{
  _id: ObjectId('676713222c6914392a4eeb93'),
  restaurant_id: 1,
  name: 'Pizza Palace',
  town: 'New York',
  cuisine: 'Italian'
},
{
  _id: ObjectId('676713222c6914392a4eeb94'),
  restaurant_id: 2,
  name: 'Sushi World',
  town: 'San Francisco',
  cuisine: 'Japanese'
},
},

{
  _id: ObjectId('676713222c6914392a4eeb95'),
  restaurant_id: 3,
  name: 'Burger King',
  town: 'Los Angeles',
  cuisine: 'American'
},
{
  _id: ObjectId('676713222c6914392a4eeb96'),
  restaurant_id: 4,
  name: 'Taco Bell',
  town: 'Chicago',
  cuisine: 'Mexican'
},
{
  _id: ObjectId('676713222c6914392a4eeb97'),
  restaurant_id: 5,
  name: 'Pasta House',
  town: 'Boston',
  cuisine: 'Italian'
}
}

Atlas atlas-qvfsgw-shard-0 [primary] abc> db.restaurants.aggregate([
... { $group: { _id: "$name", avg_score: { $avg: "$score" } } }
... ])

[
  { _id: 'Burger King', avg_score: 7 },
  { _id: 'Sushi World', avg_score: 9 },
  { _id: 'Taco Bell', avg_score: 5 },
  { _id: 'Pasta House', avg_score: 10 },
  { _id: 'Pizza Palace', avg_score: 8 }
]

Atlas atlas-qvfsgw-shard-0 [primary] abc> db.restaurants.find({ zipcode: /^10/ }, { name: 1, address: 1 })

[
  {
    _id: ObjectId('676713222c6914392a4eeb93'),
    name: 'Pizza Palace',
    address: '123 Pizza St.'
  }
]

Atlas atlas-qvfsgw-shard-0 [primary] abc> |

```

