VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT on

Database Management Systems (23CS3PCDBM)

Submitted by

MANTRI RAMITHA (1BM24CS165)

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 **MANTRI RAMITHA (1BM24CS165),** 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. The Lab report has been approved as it satisfies the academic requirements in respect of a Database Management Systems (23CS3PCDBM) work prescribed for the said degree.

Rashmi.H Assistant Professor Department of CSE, BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

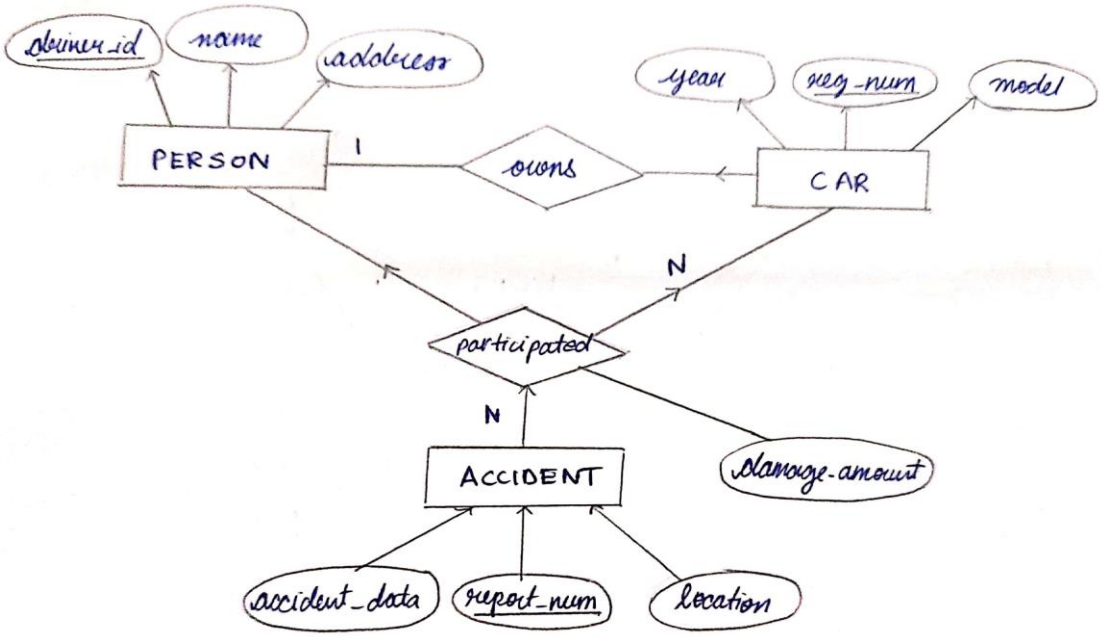
Index

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

Create database

```
create database insurance;

use insurance_dhiksha;
```

Create table

```
create table insurance.person(
driver_id varchar(20), name
varchar(30), address varchar(50),
PRIMARY KEY(driver_id)
);
create table insurance.car( reg_num
varchar(15),
model varchar(10),
year int,
PRIMARY KEY(reg_num)
);
create table insurance.owns(
driver_id varchar(20), 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 insurance.accident(
report_num int, accident_date date,
location varchar(50),
PRIMARY KEY(report_num)
);
create table insurance.participated(
driver_id varchar(20), reg_num
varchar(10), report_num int,
damage_amount int,
PRIMARY KEY(driver_id,reg_num,report_num),
FOREIGN KEY(driver_id) REFERENCES person(driver_id),
FOREIGN KEY(reg_num) REFERENCES car(reg_num),
FOREIGN KEY(report_num) REFERENCES accident(report_num)
);
```

Structure of the table

desc person;

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

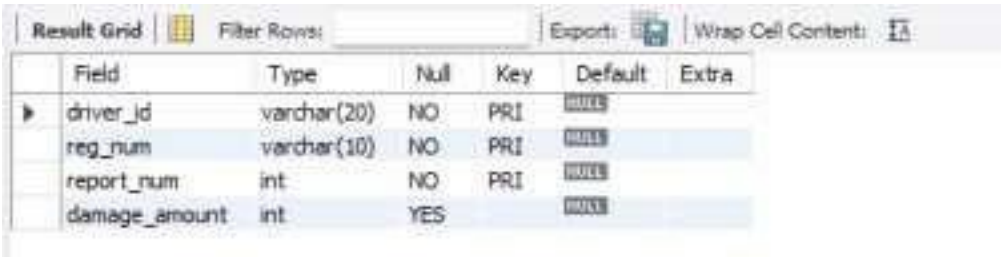
Field	Type	Null	Key	Default	Extra
driver_id	varchar(20)	NO	PRI	NULL	
reg_num	varchar(10)	NO	PRI	NULL	
report_num	int	NO	PRI	NULL	
damage_amount	int	YES		NULL	

desc accident;

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

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

desc participated;



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

desc car;

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

desc owns;

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

Inserting Values to the table

insert 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;

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

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

insert into owns values("A01","KA052250");

insert into owns values("A02","KA031181"); insert

into owns values("A03","KA095477"); insert into

owns values("A04","KA053408"); insert into owns

values("A05","KA041702");

select * from owns;

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

insert into accident values(11,'2003-01-01',"Mysore Road"); insert

into accident values(12,'2004-02-02',"South end Circle"); insert

into accident values(13,'2003-01-21',"Bull temple Road"); insert

into accident values(14,'2008-02-17',"Mysore Road"); insert into

accident values(15,'2004-03-05',"Kanakpura Road");

select * from accident;

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

insert into participated values("A01","KA052250",11,10000);

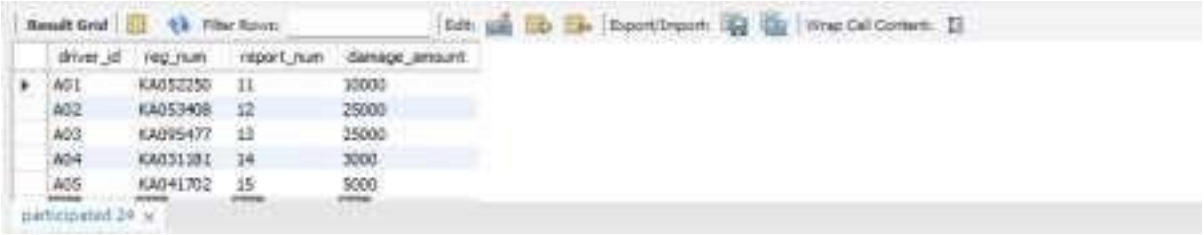
insert into participated values("A02","KA053408",12,50000);

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;



QUERIES:–

Query 1:

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

A) update participated set damage_amt=25000 where reg_num='KA053408' and report_num=12;

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

b) Add a new accident to the database.

A) insert into accident values(16, "2003-12-12" , "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	2003-12-12	Domlur

Query 2:
Display the entire CAR relation in the ascending order of manufacturing year.

A) select * from car order by yearE asc;

	vehicleno	carname	yearE
▶	KA031181	Lancer	1957
	KA052250	Indica	1995
	KA095477	Toyota	1998
	KA041702	Audi	2005
	KA053408	Honda	2008
*	NULL	NULL	NULL

Query 3:
Find the number of accidents in which cars belonging to a specific model (example Lancer) were involved.

select count(report_num) COUNT from car c,participated p where c.reg_num=p.reg_num and model="Lancer";

	COUNT
▶	1

Query 4:
Find the total number of people who owned cars that involved in accidents in 2008.

A) select count(distinct driver_id) cnt from participated,accident where participated.report_num = accident.report_num and accident.accident_date like '2008%';

	CNT
▶	1

Program 2: More Queries on Insurance Database

Queries (Questions and output)

Query 1:
List the entire participated relation in the descending order of damage amount.

A) select * from participated order by damage_amt desc;

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

Query 2:
Find the average damage amount.

A) select avg(damage_amt)AVG from participated;

	AVG
▶	13600.0000

Query 3:
Delete the tuple from participated relation whose damage amount is below the average damage amount.

A) DELETE FROM participated WHERE damage_amt<(select avg_damage from (select avg(damage_amt) as avg_damage from participated) as subquery_avg); select * from participated;

	driver_id	reg_num	report_num	damage_amt
▶	A02	KA053408	12	25000
	A03	KA095477	13	25000

Query 4:

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

A) select name from person a, participated b where a.driver_id =b.driver_id and damage_amt>(select avg(damage_amt) from participated);

	name
▶	Pradeep
	Smith

Query 5:

Find maximum damage amount.

A) Select MAX(damage_amt) from participated;

	MAX(damageamount)
▶	25000

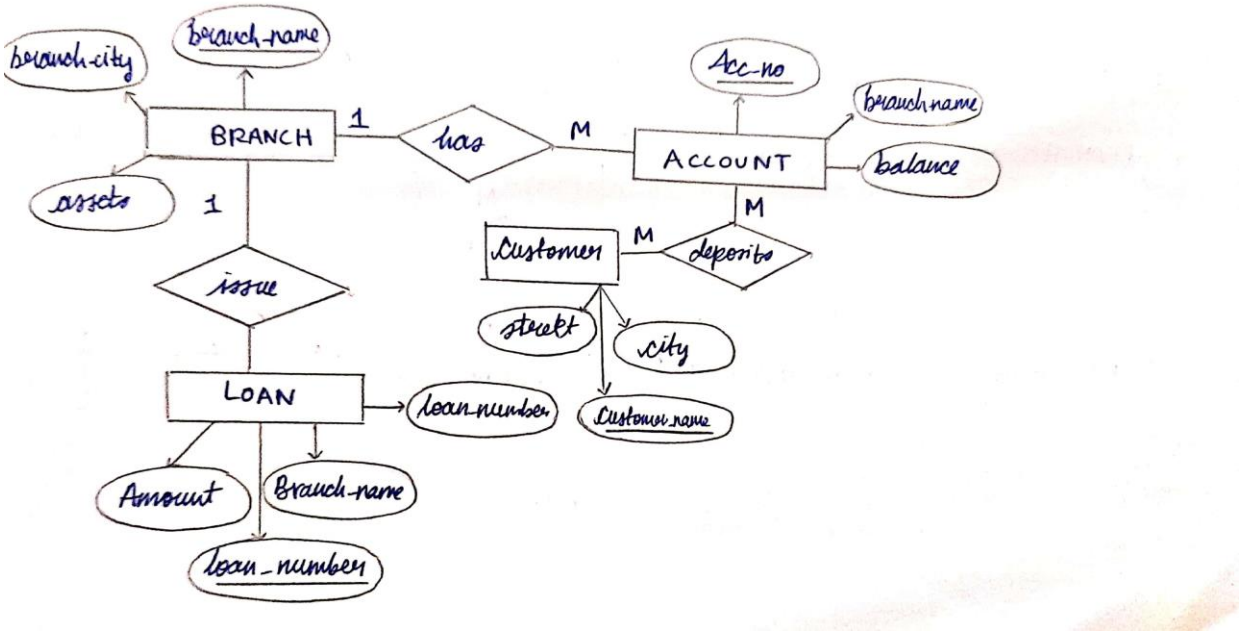
Program 3: Bank Database

Specification of Bank Database Application

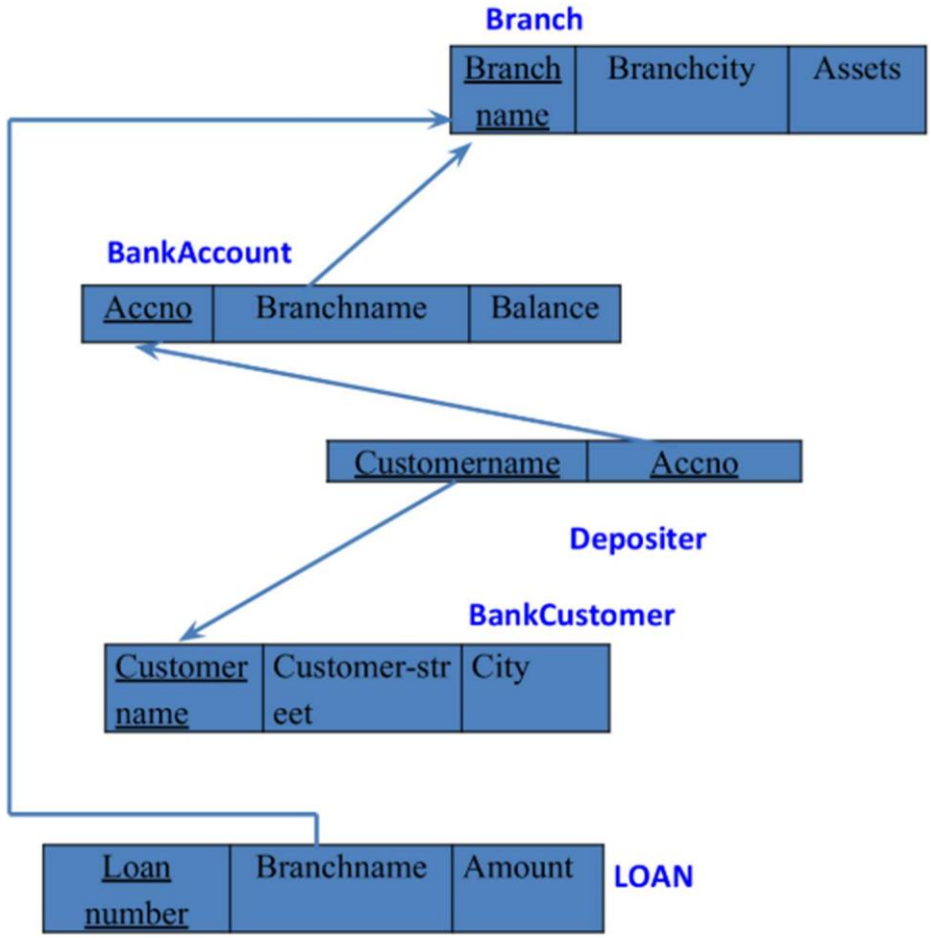
The bank database must maintain information about bank branches, customer accounts, bank customers, and the relationship between customers and the accounts they hold. Each branch in the system is uniquely identified by its branch name and stores additional information such as the city in which the branch is located and the total assets held by the branch. Branch asset values must be non-negative real numbers. Each bank account in the system is uniquely identified by an account number (accno) and is associated with exactly one branch. The account stores information such as the current balance, which must be a non-negative real value. Every account record must reference an existing branch, ensuring that no account can exist without a valid branch. The database must also maintain customer information. Each customer is uniquely identified by the customer name and includes details such as street and city of residence. Customer records must exist independently of account records. The relationship between customers and accounts is captured through the DEPOSITER relation. This relation links a customer to a bank account, representing ownership or deposit rights. A customer may hold multiple accounts, and an account may be jointly held by multiple customers. Duplicate depositer records for the same customer account combination must not exist. The database must enforce referential integrity so that depositer entries cannot exist unless the referenced customer and account records are already present. Deletion of customers or accounts that are referenced in depositer records must be restricted or handled using controlled cascading or archival policies to preserve account ownership history.

Overall, the system must maintain accurate and consistent relationships between branches, accounts, and customers, allowing reliable retrieval of branch details, customer information, account balances, and ownership relationships for banking operations, auditing, and customer services.

Entity Relationship Diagram



Schema Diagram



-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)

Create database

```
create database if not exists bank;
use bank;
```

Create tables

```
CREATE TABLE Branch (
 branchname VARCHAR(30) PRIMARY KEY,
 branchcity VARCHAR(30),
 assets REAL
);

CREATE TABLE BankAccount (
 accno INT PRIMARY KEY,
 branchname VARCHAR(30),
 balance REAL,
 FOREIGN KEY(branchname) REFERENCES Branch(branchname) ON DELETE CASCADE
);

CREATE TABLE BankCustomer (
 customername VARCHAR(30) PRIMARY KEY,
 customerstreet VARCHAR(30),
 customercity VARCHAR(30)
);

CREATE TABLE Depositer (
 customername VARCHAR(30),
 accno INT,
 PRIMARYKEY(customername, accno),
 FOREIGN KEY(customername) REFERENCES BankCustomer(customername) ON  DELETE
CASCADE,
 FOREIGN KEY(accno) REFERENCES BankAccount(accno) ON DELETE CASCADE
);

CREATE TABLE Loan(
 loannumber INT PRIMARY KEY,
 branchname VARCHAR(30),
 amount REAL,
 FOREIGN KEY(branchname) REFERENCES Branch(branchname) ON DELETE CASCADE
);
```

```
CREATE TABLE Borrower (
 customername VARCHAR(30),
 loannumber INT,
 PRIMARYKEY(customername, loannumber),
 FOREIGN KEY(customername) REFERENCES BankCustomer(customername) ON DELETE
CASCADE,
 FOREIGN KEY(loannumber) REFERENCES Loan(loannumber) ON DELETE CASCADE
);
```

Structure of the table

desc branch;

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

desc BankAccount;

	Field	Type	Null	Key	Default	Extra
▶	accno	int(11)	NO	PRI	NULL	
	branchname	varchar(30)	YES	MUL	NULL	
	balance	double	YES		NULL	

desc BankCustomer;

	Field	Type	Null	Key	Default	Extra
▶	customername	varchar(30)	NO	PRI	NULL	
	customerstreet	varchar(30)	YES		NULL	
	customercity	varchar(30)	YES		NULL	

desc depositer;

	Field	Type	Null	Key	Default	Extra
▶	customername	varchar(30)	NO	PRI	NULL	
	accno	int(11)	NO	PRI	NULL	

desc Loan;

	Field	Type	Null	Key	Default	Extra
▶	loannumber	int(11)	NO	PRI	NULL	
	branchname	varchar(30)	YES	MUL	NULL	
	amount	double	YES		NULL	

desc borrower;

	Field	Type	Null	Key	Default	Extra
▶	customername	varchar(30)	NO	PRI	NULL	
	loannumber	int(11)	NO	PRI	NULL	

Inserting Values to the table

insert into Branch values('SBI_Chamrajpet','Bangalore',50000);
insert into Branch values('SBI_ResidencyRoad','Bangalore',10000);

insert into Branch values('SBI_ShivajiRoad','Bombay',20000);
insert into Branch values('SBI_ParlimentRoad','Delhi',10000);
insert into Branch values('SBI_Jantarmantar','Delhi',20000);
select * from Branch;

branchname	branchcity	assets
SBI_Chamrajpet	Bangalore	50000
SBI_Jantarmantar	Delhi	20000
SBI_ParlimentRoad	Delhi	10000
SBI_ResidencyRoad	Bangalore	10000
SBI_ShivajiRoad	Bombay	20000
NULL	NULL	NULL

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

loannumber	branchname	amount
1	SBI_Chamrajpet	1000
2	SBI_ResidencyRoad	2000
3	SBI_ShivajiRoad	3000
4	SBI_ParlimentRoad	4000
5	SBI_Jantarmantar	5000
NULL	NULL	NULL

insert into BankCustomer (CUSTOMERNAME, CUSTOMERSTREET, CUSTOMERCITY) VALUES
('Avinash', 'Bull_Temple_Road', 'Bangalore'),
('Dinesh', 'Bannergatta_Road', 'Bangalore'),
('Mohan', 'NationalCollege_Road', 'Bangalore'),
('Nikil', 'Akbar_Road', 'Delhi'),
('Ravi', 'Prithviraj_Road', 'Delhi');
select * from BankCustomer;

customername	customerstreet	customercity
Avinash	Bull_Temple_Road	Bangalore
Dinesh	Bannergatta_Road	Bangalore
Mohan	NationalCollege_Road	Bangalore
Nikil	Akbar_Road	Delhi
Ravi	Prithviraj_Road	Delhi
NULL	NULL	NULL

```
insert into BankAccount (accno,branchname,balance) values
(1, 'SBI_Chamrajpet', 2000),
(2, 'SBI_ResidencyRoad', 5000),
(3, 'SBI_ShivajiRoad', 6000),
(4, 'SBI_ParlimentRoad', 9000),
(5, 'SBI_Jantarmantar', 8000),
(6, 'SBI_ShivajiRoad', 4000),
(7, 'SBI_ResidencyRoad', 4000),
(8, 'SBI_ParlimentRoad', 3000),
(9, 'SBI_ParlimentRoad', 5000),
(10, 'SBI_ResidencyRoad', 2000);
select * from BankAccount;
```

accno	branchname	balance
1	SBI_Chamrajpet	2000
2	SBI_ResidencyRoad	5000
3	SBI_ShivajiRoad	6000
4	SBI_ParlimentRoad	9000
5	SBI_Jantarmantar	8000
6	SBI_ShivajiRoad	4000
7	SBI_ResidencyRoad	4000
8	SBI_ParlimentRoad	3000
9	SBI_ParlimentRoad	5000
10	SBI_ResidencyRoad	2000
NULL	NULL	NULL

```
insert into Depositor (CUSTOMERNAME, ACCNO) VALUES
('Avinash', 1),
('Dinesh', 2),
('Nikil', 4),
('Ravi', 5),
('Avinash', 8),
('Nikil', 9),
('Dinesh', 10),
('Nikil', 3);
select * from Depositor;
```

	customername	accno
▶	Avinash	1
	Dinesh	2
	Nikil	4
	Ravi	5
	Avinash	8
	Nikil	9
	Dinesh	10
	Nikil	3

Queries (Questions and output)

Query 1:

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

A) select branch_name,concat(assets/100000,'lakhs')assets_in_lakhs from branch;

	branch_name	assets_in_lakhs
▶	SBI_Chamrajpet	0.5000lakhs
	SBI_Jantarmantar	0.2000lakhs
	SBI_ParliamentRoad	0.1000lakhs
	SBI_ResidencyRoad	0.1000lakhs
	SBI_ShivajiRoad	0.2000lakhs

Query 2:

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

A) select customername,count(customername)NUMBER
from Depositor d,BankAccount b
where b.Accno=d.Accno and branchname='SBI_ResidencyRoad'
group by customername having count(customername)>=2 ;

	Customername
▶	Dinesh

Query 3:

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

A) create view sum_of_loan
as select Branch_name, SUM(Balance)
from BankAccount
group by Branch_name;
select * from sum_of_loan;

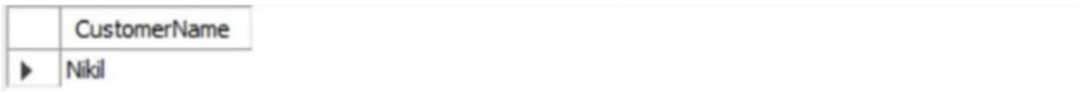
Branch_name	SUM(Balance)
SBI_Chamrajpet	2000
SBI_Jantarmantar	10000
SBI_ParliamentRoad	12000
SBI_ResidencyRoad	14000
SBI_ShivajiRoad	10000

Program 4: More Queries on Bank Database

Queries (Questions and output)

Query 1:
Find all the customers who have an account at all the branches located in a specific city (Ex. Delhi).

```
A ) select D.CustomerName
from Depositor D
JOIN BankAccount A ON D.AccNo = A.AccNo
JOIN Branch B ON A.BranchName = B.BranchName
where B.BranchCity = 'Delhi'
GROUP BY D.CustomerName
HAVING COUNT(DISTINCT B.BranchName) = (
   select COUNT(DISTINCT BranchName)
   from BranchS
   where BranchCity = 'Delhi'
```



	CustomerName
▶	Nikil

Query 2:
Find all customers who have a loan at the bank but do not have an account.

```
A)  select DISTINCT C.CustomerName
from BankCustomer C
JOIN Loan L ON C.CustomerCity = (
   SELECT BranchCity
   FROM Branch WHERE BranchName = L.BranchName)
where C.CustomerName NOT IN (
   select CustomerName from Depositor);
```

	CustomerName
▶	Mohan

Query 3:
Find all customers who have both an account and a loan at the Bangalore branch.

```
A)  select DISTINCT D.CustomerName
from Depositor D
JOIN BankAccount A ON D.AccNo = A.AccNo
JOIN Branch B1 ON A.BranchName = B1.BranchName
JOIN Loan L ON B1.BranchName = L.BranchName
JOIN Branch B2 ON L.BranchName = B2.BranchName
where B1.BranchCity = 'Bangalore' and B2.BranchCity = 'Bangalore';
```

	CustomerName
▶	Avinash
	Dinesh

Query 4:
Find the names of all branches that have greater assets than all branches located in Bangalore.

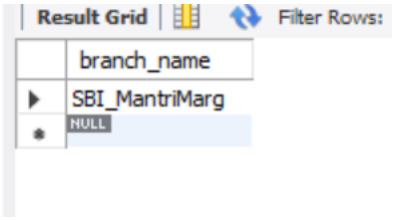
```
A)  select branch_name
from Branch
where assets > ALL (
   select assets FROM Branch WHERE branch_city='Bangalore'
);
```

	branch_name
▶	SBI_Jantarmantar
	SBI_MantriMarg
	SBI_ParlimentRoad
*	NULL

Query 5:

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

A) DELETE FROM BankAccount
where BranchName IN (
select BranchName
from Branch
where BranchCity = 'Bombay'
);



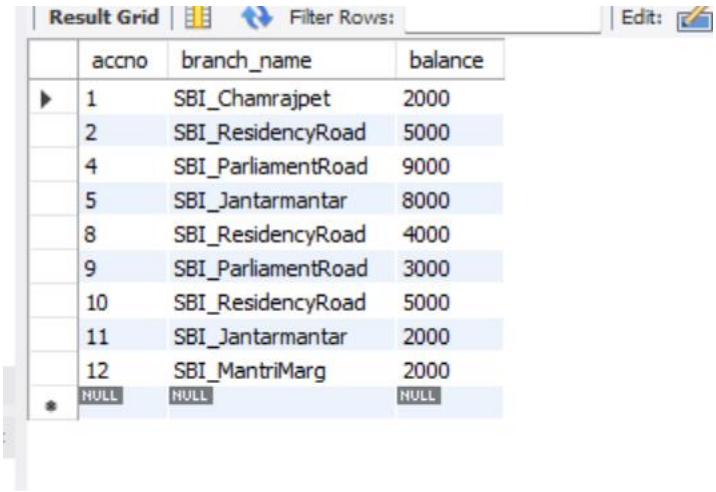
Result Grid | Filter Rows:

branch_name
SBI_MantriMarg
NULL

Query 6:

Update the Balance of all accounts by 5%

A) update BankAccount
SET Balance = Balance * 1.05
where AccNo > 0;



Result Grid | Filter Rows: | Edit:

accno	branch_name	balance
1	SBI_Chamrajpet	2000
2	SBI_ResidencyRoad	5000
4	SBI_ParliamentRoad	9000
5	SBI_Jantarmantar	8000
8	SBI_ResidencyRoad	4000
9	SBI_ParliamentRoad	3000
10	SBI_ResidencyRoad	5000
11	SBI_Jantarmantar	2000
12	SBI_MantriMarg	2000
NULL	NULL	NULL

Program 5: Employee Database

Specification of Employee Database Application

The employee database must maintain comprehensive information about employees, the departments they belong to, the projects they work on, and the incentives they receive. Each employee in the system is uniquely identified by an employee number (EMPNO), along with associated attributes such as employee name, hire date, salary, and department number. The system must also support a managerial hierarchy where an employee may act as a manager for other employees, represented through a recursive relationship in which the manager number (MGR_NO) references another existing employee.

Each department in the system is uniquely identified by a department number (DEPTNO) and stores additional information such as department name and department location. Every employee must be associated with a valid department, ensuring that no employee record can exist without a corresponding department already defined in the system. Referential integrity must be enforced so that department information cannot be removed while employees are still assigned to it.

The database must store project-related information, where each project is uniquely identified by a project number (PNO), along with its name and location. Employees may be assigned to one or more projects, and each project may have one or more employees working on it. This many-to-many relationship is represented through an ASSIGNED-TO association, which records the employee number, project number, and the job role performed by the employee in that project. Duplicate assignment records for the same employee–project combination must not exist.

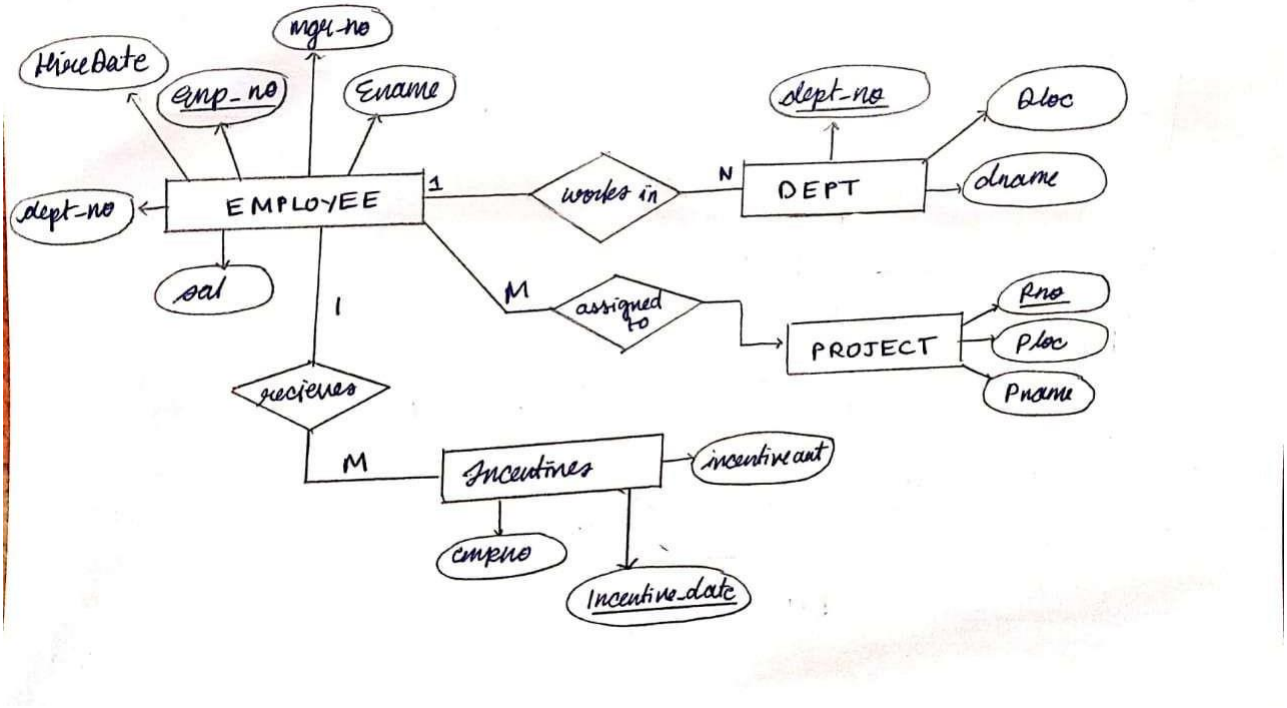
Incentive information must also be maintained for employees. Each incentive record is associated with a valid employee and is uniquely identified by the combination of employee number and incentive date. Additional details such as the incentive amount are stored. An employee may receive multiple incentives over time, but every incentive record must reference an existing employee. Incentive amounts must be non-negative, and incentive dates must be valid calendar dates.

The database must ensure that all foreign key relationships are properly enforced. Assignment records cannot exist unless both the referenced employee and project are present in the system, and incentive records cannot exist without a valid employee. Similarly, employees must reference valid departments, and managerial relationships must reference valid employee records.

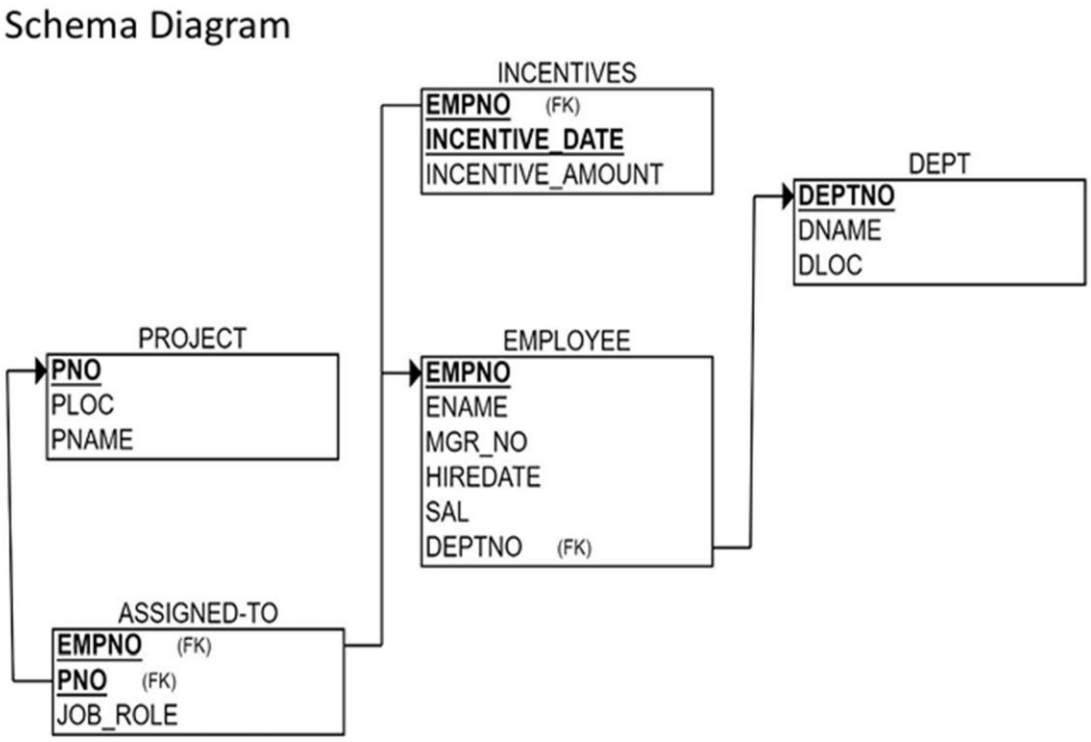
Deletion policies must preserve data consistency and historical correctness. Employees who are referenced as managers, assigned to projects, or associated with incentive records should not be deleted unless appropriate cascading rules or archival mechanisms are applied. Projects and departments should not be removed while they are still referenced by assignment or employee records, respectively.

Overall, the system must maintain accurate and consistent relationships between employees, departments, projects, and incentives, enabling reliable retrieval of employee details, departmental structures, project assignments, managerial hierarchies, and incentive histories for administrative, organizational, and payroll-related purposes.

Entity Relationship Diagram (Draw it in hand)



Schema Diagram



-Employee(emp_id, emp_name, emp_address, hire_date, salary, manager_id, dept_no)
-Department(dept_no, dept_name, dept_location, manager_id)
-Project(project_no, project_name, project_location)
-Assignment(emp_id, project_no, job_role, start_date, end_date)
-Incentive(incentive_id, emp_id, incentive_date, incentive_amount

Create database

```
create database employee;
use EMPLOYEE;
```

Create table

```
CREATE TABLE Employee (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(50) NOT NULL,
   emp_address VARCHAR(100),
   hire_date DATE NOT NULL,
   salary DECIMAL(10,2) CHECK (salary >= 0),
   dept_no INT
);
CREATE TABLE Department (
   dept_no INT PRIMARY KEY,
   dept_name VARCHAR(50) NOT NULL,
   dept_location VARCHAR(50),
   manager_id INT,
   CONSTRAINT fk_dept_manager
      FOREIGN KEY (manager_id) REFERENCES Employee(emp_id)
);
CREATE TABLE Project (
   project_no INT PRIMARY KEY,
   project_name VARCHAR(50),
   project_location VARCHAR(50)
);
CREATE TABLE Assignment (
   emp_id INT,
   project_no INT,
   job_role VARCHAR(50),
   start_date DATE NOT NULL,
   end_date DATE,
   PRIMARY KEY(emp_id, project_no),
   FOREIGN KEY(emp_id) REFERENCES Employee(emp_id),
   FOREIGN KEY(project_no) REFERENCES Project(project_no),
   CHECK (end_date IS NULL OR end_date >= start_date)
);
CREATE TABLE Incentive (
   incentive_id INT PRIMARY KEY,
   emp_id INT,
   incentive_date DATE NOT NULL,
   incentive_amount DECIMAL(10,2) CHECK (incentive_amount >= 0),
   FOREIGN KEY(emp_id) REFERENCES Employee(emp_id),
   CHECK (incentive_date >= '2000-01-01'));
```

Structure of table

DESC EMPLOYEE;

Field	Type	Null	Key	Default	Extra
emp_id	int	NO	PRI	NULL	
emp_name	varchar(50)	NO		NULL	
emp_address	varchar(100)	YES		NULL	
hire_date	date	NO		NULL	
salary	decimal(10,2)	YES		NULL	
dept_no	int	YES	MUL	NULL	
manager_id	int	YES		NULL	

DESC DEPARTMENT;

Field	Type	Null	Key	Default	Extra
dept_no	int	NO	PRI	NULL	
dept_name	varchar(50)	NO		NULL	
dept_location	varchar(50)	YES		NULL	
manager_id	int	YES	MUL	NULL	

DESC PROJECT;

Field	Type	Null	Key	Default	Extra
project_no	int	NO	PRI	NULL	
project_name	varchar(50)	YES		NULL	
project_location	varchar(50)	YES		NULL	

DESC ASSIGNMENT;

Field	Type	Null	Key	Default	Extra
emp_id	int	NO	PRI	NULL	
project_no	int	NO	PRI	NULL	
job_role	varchar(50)	YES		NULL	
start_date	date	NO		NULL	
end_date	date	YES		NULL	

DESC INCENTIVE;

	Field	Type	Null	Key	Default	Extra
▶	incentive_id	int	NO	PRI	NULL	
	emp_id	int	YES	MUL	NULL	
	incentive_date	date	NO		NULL	
	incentive_amount	decimal(10,2)	YES		NULL	

Insertion of values:

```
INSERT INTO Department (dept_no, dept_name, dept_location, manager_id)
VALUES
(10, 'HR', 'Bangalore', NULL),
(20, 'Finance', 'Chennai', NULL),
(30, 'Engineering', 'Mumbai', NULL);
SELECT * FROM DEPARTMENT;
```

	dept_no	dept_name	dept_location	manager_id
▶	10	HR	Bangalore	NULL
	20	Finance	Chennai	NULL
	30	Engineering	Mumbai	NULL
*	NULL	NULL	NULL	NULL

```
INSERT INTO Employee (emp_id, emp_name, emp_address, hire_date, salary, dept_no, manager_id)
VALUES
(1, 'Ravi', 'Bangalore', '2015-01-01', 60000, 10, NULL),
(2, 'Megha', 'Mysore', '2016-02-15', 55000, 10, 1),
(3, 'Arun', 'Chennai', '2014-03-10', 70000, 20, 1),
(4, 'Lata', 'Delhi', '2019-07-20', 45000, 20, 2),
(5, 'John', 'Mumbai', '2020-05-12', 40000, 30, 2);
SELECT * FROM EMPLOYEE;
```

	emp_id	emp_name	emp_address	hire_date	salary	dept_no	manager_id
▶	1	Ravi	Bangalore	2015-01-01	60000.00	10	NULL
	2	Megha	Mysore	2016-02-15	55000.00	10	1
	3	Arun	Chennai	2014-03-10	70000.00	20	1
	4	Lata	Delhi	2019-07-20	45000.00	20	2
	5	John	Mumbai	2020-05-12	40000.00	30	2
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
INSERT INTO Project VALUES
(100,'Migration','Bangalore'),
(101,'AI System','Hyderabad'),
(102,'Website','Chennai'),
(103,'Payroll','Mumbai');
SELECT * FROM PROJECT;
```

project_no	project_name	project_location
100	Migration	Bangalore
101	AI System	Hyderabad
102	Website	Chennai
103	Payroll	Mumbai
NULL	NULL	NULL

```
INSERT INTO Assignment VALUES
(1,100,'Manager','2020-01-01',NULL),
(2,100,'Developer','2020-01-10','2021-05-01'),
(3,101,'Analyst','2019-03-20',NULL),
(4,102,'Tester','2022-01-15',NULL),
(5,103,'Developer','2021-02-25',NULL);
SELECT  * FROM ASSIGNMENT;
```

emp_id	project_no	job_role	start_date	end_date
1	100	Manager	2020-01-01	NULL
2	100	Developer	2020-01-10	2021-05-01
3	101	Analyst	2019-03-20	NULL
4	102	Tester	2022-01-15	NULL
5	103	Developer	2021-02-25	NULL
NULL	NULL	NULL	NULL	NULL

```
INSERT INTO Incentive VALUES
(1,1,'2022-05-01',5000),
(2,2,'2023-06-10',4000),
(3,3,'2021-08-15',6000),
(4,4,'2022-12-11',3500),
(5,5,'2023-01-20',2000);
SELECT * FROM INCENTIVE;
```

	incentive_id	emp_id	incentive_date	incentive_amount
▶	1	1	2022-05-01	5000.00
	2	2	2023-06-10	4000.00
	3	3	2021-08-15	6000.00
	4	4	2022-12-11	3500.00
	5	5	2023-01-20	2000.00
*	NULL	NULL	NULL	NULL

Queries :-

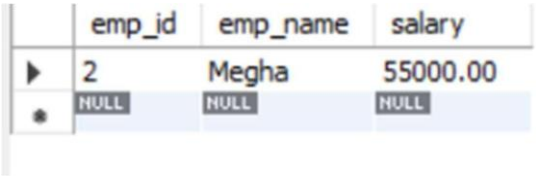
Query 1:
List the name of the managers with the most employees.

A)
```
SELECT manager_id, COUNT(*) AS total_employees
FROM Employee
WHERE manager_id IS NOT NULL
GROUP BY manager_id
HAVING COUNT(*) = (
    SELECT MAX(cnt)
    FROM (
        SELECT manager_id, COUNT(*) AS cnt
        FROM Employee
        WHERE manager_id IS NOT NULL
        GROUP BY manager_id
    ) AS x
);
```

	manager_id	total_employees
▶	1	2
	2	2

Query 2:
Display managers whose salary is MORE than the average salary of their employees

A) SELECT m.emp_id, m.emp_name, m.salary
FROM Employee m
WHERE m.emp_id IN (SELECT manager_id FROM Employee)
AND m.salary > (
    SELECT AVG(e.salary)
    FROM Employee e
    WHERE e.manager_id = m.emp_id
);



emp_id	emp_name	salary
2	Megha	55000.00
NULL	NULL	NULL

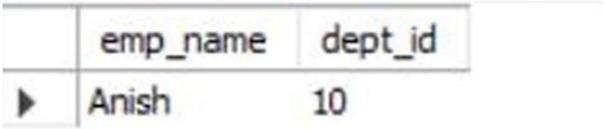
Query 3:
Get Employee ID's of those employees who didn't receive incentives.

A) select e.emp_id from employee e left join incentives i on e.emp_id = i.emp_id where i.emp_id is null;

emp_id

Query 4:
SQL Query to find the name of the top level manager of each department.(top-level manager is the employee whose manager_id is NULL ,in that department the top-level manager has no manager above them).

A) select m.emp_name from employee m join employee e on m.emp_id = e.manager_id group by m.emp_id, m.emp_name, m.salary having m.salary > avg(e.salary);



emp_name	dept_id
Anish	10

Query 5:

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

A) select e.* from employee e join incentives i on e.emp_id = i.emp_id where month(i.incentive_date) = 2 and year(i.incentive_date) = 2019 order by i.incentive_amount desc limit 1 offset 1;

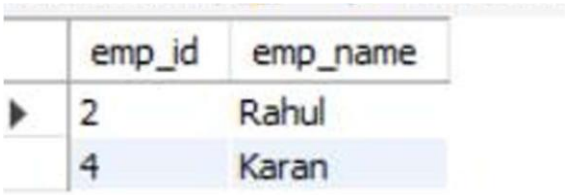
	emp_id	emp_name	dept_id	manager_id	job_role	salary	net_pay
▶	1	Anish	10	NULL	Manager	70000	68000

Program 6: More Queries on Employee Database

Queries (Questions and output)

Query 1:
Display those employees who are working in the same dept where his manager is work.

A) select e.emp_id, e.emp_name from employee e join employee m on e.manager_id = m.emp_id where e.dept_id = m.dept_id;



emp_id	emp_name
2	Rahul
4	Karan

Query 2:
Retrieve the employee numbers of all employees who work on project located in Bengaluru, Hyderabad, or Mysuru .

A) select distinct a.emp_id from assigned_to a join project p on a.project_id = p.project_id where p.project_location in ('BENGALURU', 'HYDERABAD', 'MYSURU');

emp_id
1
2
4
5

Query 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.

A) select e.emp_name, e.emp_id, e.dept_id, e.job_role, d.dept_location, p.project_location from employee e join department d on e.dept_id = d.dept_id join assigned_to a on e.emp_id = a.emp_id join project p on a.project_id = p.project_id where d.dept_location = p.project_location;

emp_name	emp_id	dept_id	job_role	dept_location	project_location
Anish	1	10	Manager	Bengaluru	Bengaluru
Rahul	2	10	Executive	Bengaluru	Bengaluru
Sita	3	20	Clerk	Mumbai	Mumbai
Karan	4	10	Analyst	Bengaluru	Bengaluru
Pooja	5	30	Associate	Hyderabad	Hyderabad

Query 4:
Write a SQL query to find those employees whose net pay are higher than or equal to the salary of any other employee in the company.

A) select e.emp_name, e.emp_id from employee e where e.net_pay >= any (select salary from employee);

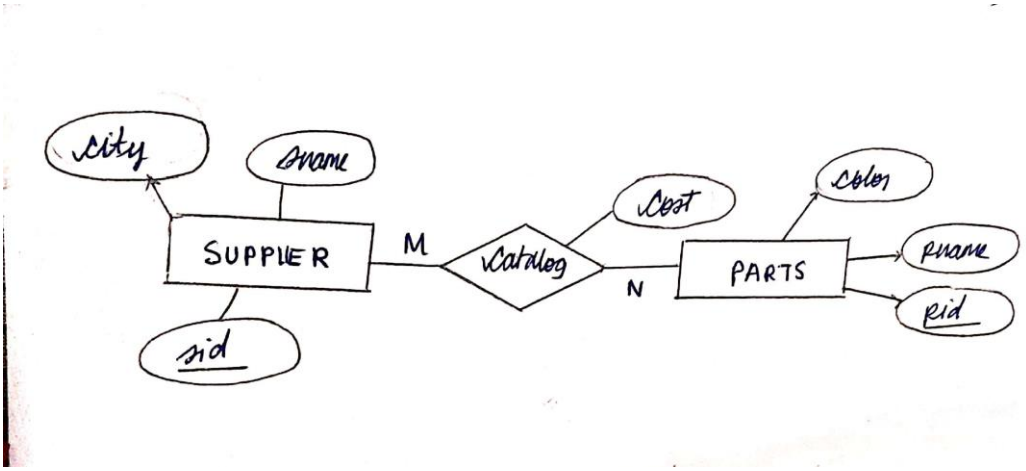
emp_name	emp_id
Anish	1
Rahul	2
Karan	4
Pooja	5

Program 7: Supplier Database

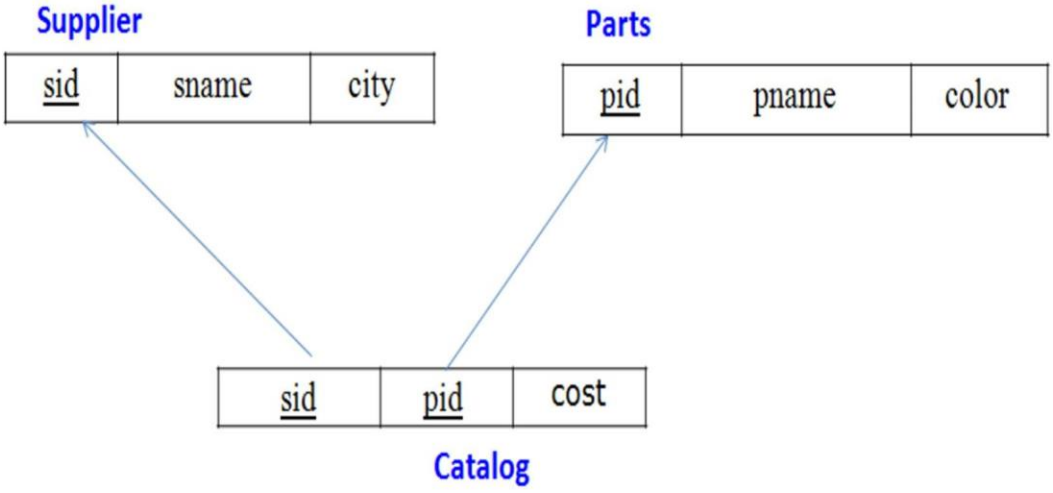
Specification of Insurance Supplier Application

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

Entity Relationship Diagram



Schema Diagram



Create Database

create database suppliers;
use suppliers;

Create Tables

```
create table suppliers(
    sid int,
    sname varchar(30),
    city varchar(30),
    primary key(sid)
);
create table parts(
    pid int,
    pname varchar(30),
    color varchar(30),
    primary key(pid)
);
create table catalog(
    sid int,
    pid int,
    cost int,
    foreign key(sid) references suppliers(sid),
    foreign key(pid) references parts(pid)
);
```

Structure of the table

desc suppliers;

	Field	Type	Null	Key	Default	Extra
▶	sid	int(11)	NO	PRI	NULL	
	sname	varchar(50)	YES		NULL	
	city	varchar(50)	YES		NULL	

desc parts;

	Field	Type	Null	Key	Default	Extra
▶	pid	int(11)	NO	PRI	NULL	
	pname	varchar(50)	YES		NULL	
	color	varchar(20)	YES		NULL	

desc catalog;

	Field	Type	Null	Key	Default	Extra
▶	sid	int(11)	YES	MUL	NULL	
	pid	int(11)	YES	MUL	NULL	
	cost	int(11)	YES		NULL	

Inserting values

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

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

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

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

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

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

QUERIES:–

Query 1:

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

A) select distinct pname from parts p, suppliers s, catalog c where p.pid=c.pid and s.sid=c.sid;

	pname
▶	Book
	Pen
	Pencil
	Mobile
	Charger

Query 2:
Find the snames of suppliers who supply every part.

A) SELECT s.sname FROM suppliers s
JOIN catalog c ON s.sid = c.sid
GROUP BY s.sid, s.sname
HAVING COUNT(DISTINCT c.pid) = (SELECT COUNT(*) FROM parts)

	SNAME
▶	Acme Widget

Query 3:
Find the snames of suppliers who supply every red part.

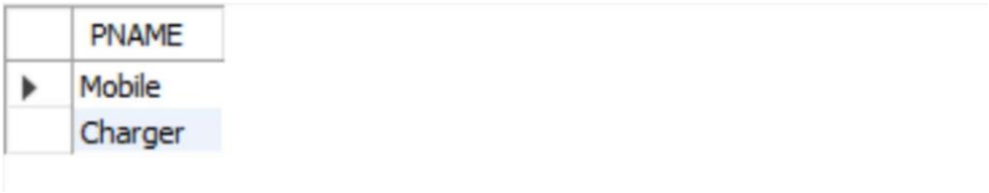
A) SELECT S.SNAME FROM SUPPLIERS S
WHERE NOT EXISTS(
SELECT P.PID FROM PARTS P
WHERE p.color= 'Red'
AND NOT EXISTS(
SELECT c.pid FROM catalog c
WHERE c.sid= s.sid AND c.pid = p.pid
));

	SNAME
▶	Acme Widget
	Johns

Query 4:

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

A) select p.pname from parts p where p.pid in (select pid from catalog where sid = 10001and pid
not in (select pid from catalog where sid <> 10001));



	PNAME
▶	Mobile
	Charger

Query 5:

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).

A) select distinct c.sid from catalog c join (select pid, avg(cost) as avg_cost from catalog group by pid) x on c.pid = x.pid where c.cost > x.avg_cost;

	SID
▶	10002
	10004

Query 6:

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

A) SELECT s.sname, c.pid, c.cost
FROM Suppliers s
JOIN Catalog c ON s.sid = c.sid
WHERE (c.pid, c.cost) IN (
SELECT pid, MAX(cost)
FROM Catalog
GROUP BY pid);

	sname	pid	cost
▶	Acme Widget	20001	10.00
	Acme Widget	20004	10.00
	Acme Widget	20005	10.00
	Johns	20002	20.00
	Reliance	20003	40.00

Program 8: More Queries on Supplier Database

Queries (Questions and output)

Query 1:

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

select s.sname, p.pname, 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);

	pname	cost	sname
▶	Pencil	40	Reliance

Query 2:

Find suppliers who do NOT supply any red parts.

select s.* from suppliers s where s.sid not in (select c.sid from catalog c join parts p on c.pid = p.pid where p.color = 'Red');

	sname
▶	Vimal
	Reliance

Query 3:

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

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

	sname	totalcost
▶	Acme Widget	70
	Johns	30
	Vimal	30
	Reliance	40

Query 4:

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

select s.sid, s.sname from suppliers s join catalog c on s.sid = c.sid where c.cost < 20 group by s.sid having count(c.pid) >= 2;

	sname
▶	Acme Widget

Query 5:

List suppliers who offer the cheapest cost for each part.

select s.sname, p.pname, 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 min(c2.cost) from catalog c2 where c2.pid = c.pid);

	pname	sname
▶	Book	Acme Widget
	Pen	Acme Widget
	Pencil	Acme Widget
	Mobile	Acme Widget
	Charger	Acme Widget
	Book	Johns
	Pencil	Vimal

Query 6:

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

create view supplier_part_count as select s.sid, s.sname, count(c.pid) as totalparts from suppliers s left join catalog c on s.sid = c.sid group by s.sid;

	sname	count(distinct c.pid)
▶	Acme Widget	5
	Johns	2
	Vimal	1
	Reliance	1

Query 7:

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

create view most_expensive_supplier as select s.sname, p.pname, 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(c2.cost) from catalog c2 where c2.pid = c.pid);

	sid	sname	pid	pname	cost
▶	10001	Acme Widget	20001	Book	10.00
	10001	Acme Widget	20004	Mobile	10.00
	10001	Acme Widget	20005	Charger	10.00
	10002	Johns	20002	Pen	20.00
	10004	Reliance	20003	Pencil	40.00

Query 8:

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

```
DELIMITER //
create trigger prevent_low_cost
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 ;
```

Query 9:

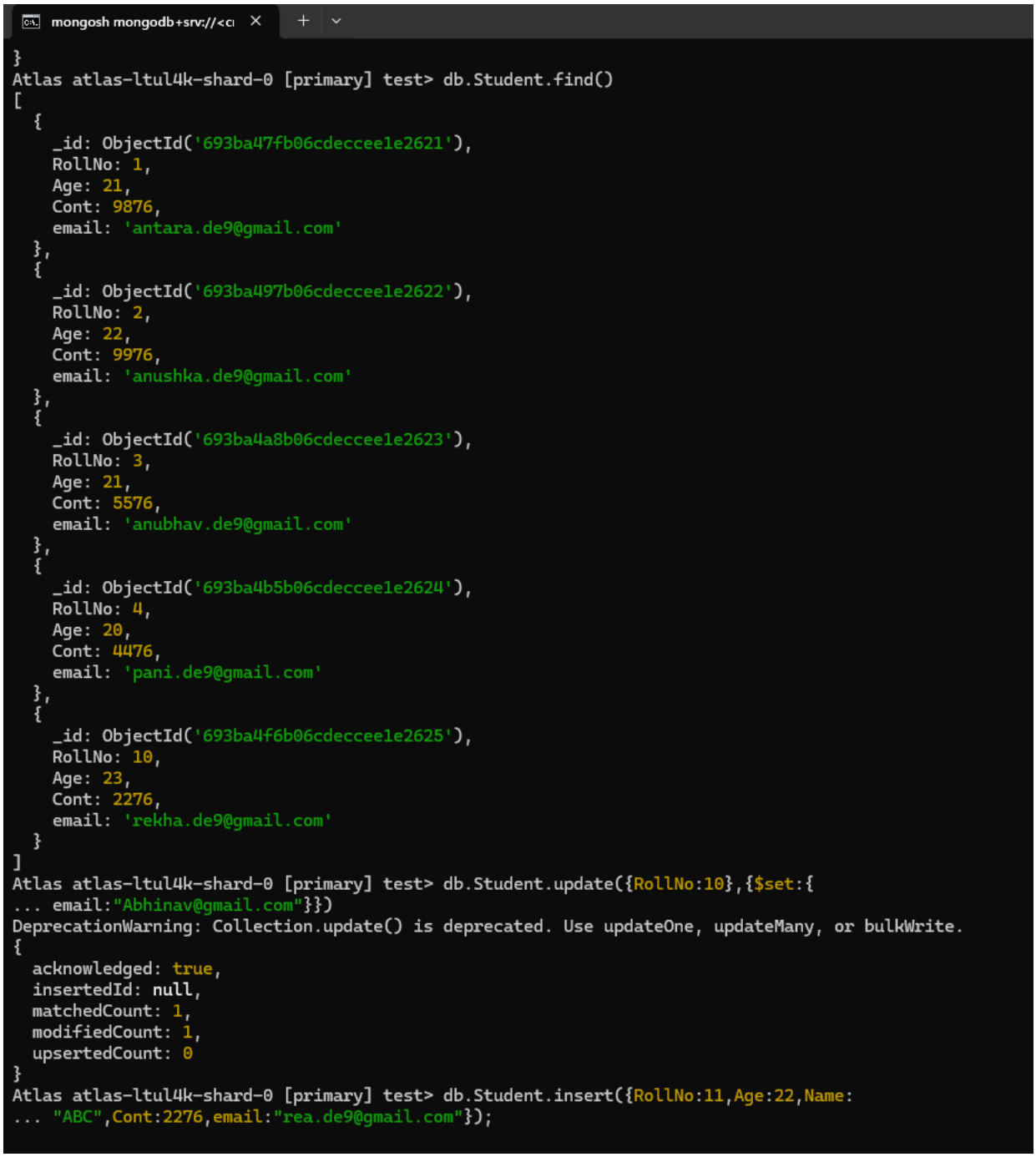
Create a trigger to set to default cost if not provided.

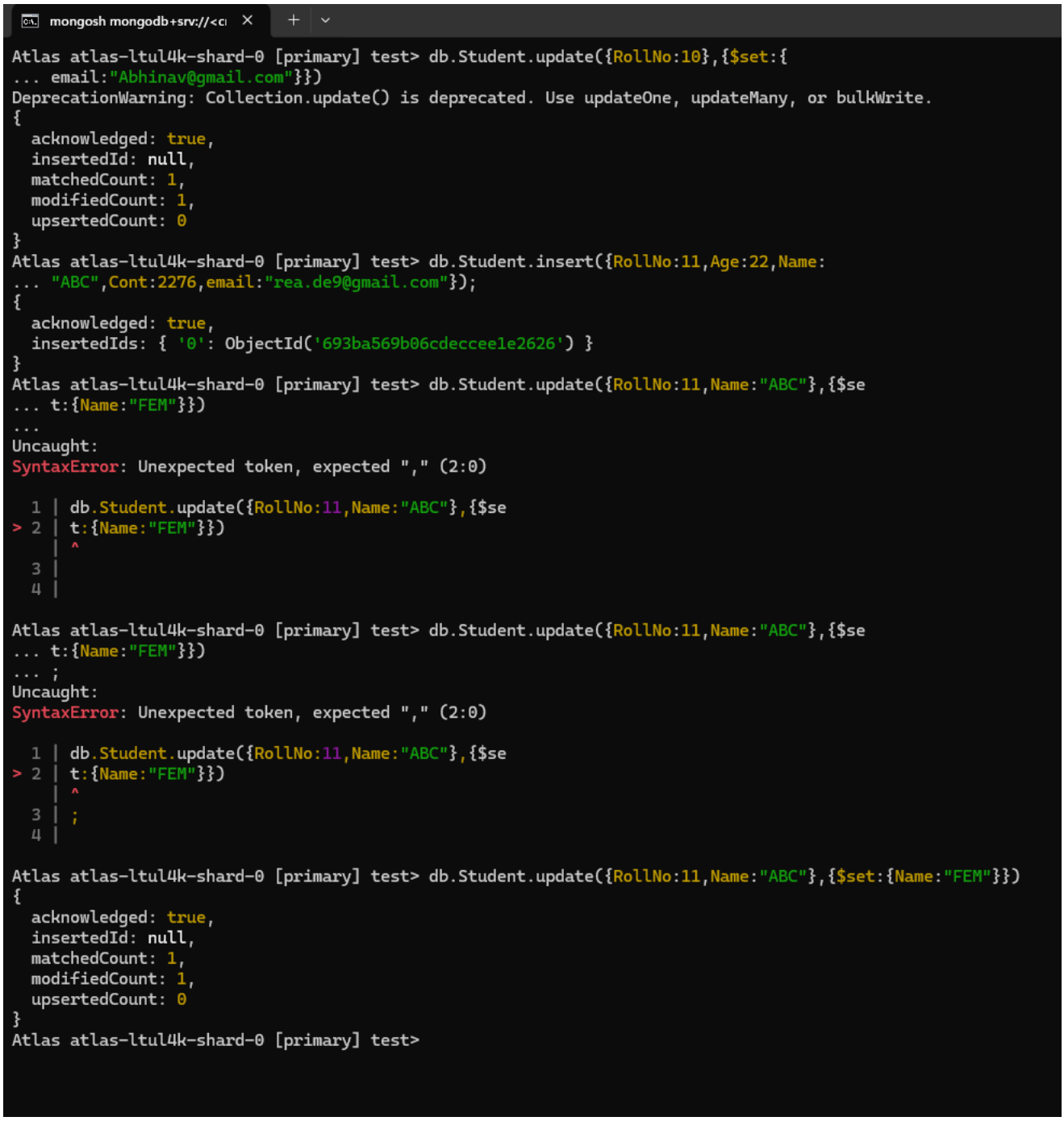
```
DELIMITER //
create trigger set_default_cost before insert on catalog
for each row begin if new.cost is null then set new.cost = 100;
end if;
end;
 //
DELIMITER ;
```

Program 9: NoSQL Student Database

Perform the following DB operations using MongoDB.
i.Create a database “Student” with the following attributesRollno, Age, ContactNo, EmailId.
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.

```
mongosh mongodb+srv://<cl
Microsoft Windows [Version 10.0.22631.6199]
(c) Microsoft Corporation. All rights reserved.

C:\Users\BMSCECSE-L3-36>cd C:\Program Files

C:\Program Files>mongosh "mongodb+srv://cluster0.iol4ckv.mongodb.net/" --apiVersion 1 --username poorvitcs24_db_user
Enter password: **********
Current Mongosh Log ID: 693ba43cb06cdeccee1e2620
Connecting to:          mongodb+srv://<credentials>@cluster0.iol4ckv.mongodb.net/?appName=mongosh+2.5.10
Using MongoDB:          8.0.16 (API Version 1)
Using Mongosh:          2.5.10

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

Atlas atlas-ltul4k-shard-0 [primary] test> db.createCollection("Student");
{ ok: 1 }
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.insert({RollNo:1,Age:21,Cont:9876,email:"antara.de9@gmail.com"});
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693ba47fb06cdeccee1e2621') }
}
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.insert({RollNo:2,Age:22,Cont:9976,email:"anushka.de9@gmail.com"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693ba497b06cdeccee1e2622') }
}
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.insert({RollNo:3,Age:21,Cont:5576,email:"anubhav.de9@gmail.com"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693ba4a8b06cdeccee1e2623') }
}
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.insert({RollNo:4,Age:20,Cont:4476,email:"pani.de9@gmail.com"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693ba4b5b06cdeccee1e2624') }
}
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.insert({RollNo:10,Age:23,Cont:2276,email:"rekha.de9@gmail.com"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693ba4f6b06cdeccee1e2625') }
}
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.find()
[
  {
    _id: ObjectId('693ba47fb06cdeccee1e2621'),
    RollNo: 1,
```


mongosh mongodb+srv://<cl
}
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.find()
[
{
_id: ObjectId('693ba47fb06cdeccee1e2621'),
RollNo: 1,
Age: 21,
Cont: 9876,
email: 'antara.de9@gmail.com'
},
{
_id: ObjectId('693ba497b06cdeccee1e2622'),
RollNo: 2,
Age: 22,
Cont: 9976,
email: 'anushka.de9@gmail.com'
},
{
_id: ObjectId('693ba4a8b06cdeccee1e2623'),
RollNo: 3,
Age: 21,
Cont: 5576,
email: 'anubhav.de9@gmail.com'
},
{
_id: ObjectId('693ba4b5b06cdeccee1e2624'),
RollNo: 4,
Age: 20,
Cont: 4476,
email: 'pani.de9@gmail.com'
},
{
_id: ObjectId('693ba4f6b06cdeccee1e2625'),
RollNo: 10,
Age: 23,
Cont: 2276,
email: 'rekha.de9@gmail.com'
}
]
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.update({RollNo:10},{$set:{
... email:"Abhinav@gmail.com"}})
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
{
acknowledged: true,
insertedId: null,
matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0
}
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.insert({RollNo:11,Age:22,Name:
... "ABC",Cont:2276,email:"rea.de9@gmail.com"});


mongosh mongodb+srv://<cl
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.update({RollNo:10},{$set:{
... email:"Abhinav@gmail.com"}})
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
{
acknowledged: true,
insertedId: null,
matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0
}
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.insert({RollNo:11,Age:22,Name:
... "ABC",Cont:2276,email:"rea.de9@gmail.com"});
{
acknowledged: true,
insertedIds: { '0': ObjectId('693ba569b06cdeccee1e2626') }
}
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.update({RollNo:11,Name:"ABC"},{$se
... t:{Name:"FEM"}})
...
Uncaught:
SyntaxError: Unexpected token, expected "," (2:0)
1 | db.Student.update({RollNo:11,Name:"ABC"},{$se
> 2 | t:{Name:"FEM"}})
| ^
3 |
4 |
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.update({RollNo:11,Name:"ABC"},{$se
... t:{Name:"FEM"}})
... ;
Uncaught:
SyntaxError: Unexpected token, expected "," (2:0)
1 | db.Student.update({RollNo:11,Name:"ABC"},{$se
> 2 | t:{Name:"FEM"}})
| ^
3 | ;
4 |
Atlas atlas-ltul4k-shard-0 [primary] test> db.Student.update({RollNo:11,Name:"ABC"},{$set:{Name:"FEM"}})
{
acknowledged: true,
insertedId: null,
matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0
}
Atlas atlas-ltul4k-shard-0 [primary] test>

Program 10: NoSQL Restaurant Database

Perform the following DB operations using MongoDB.

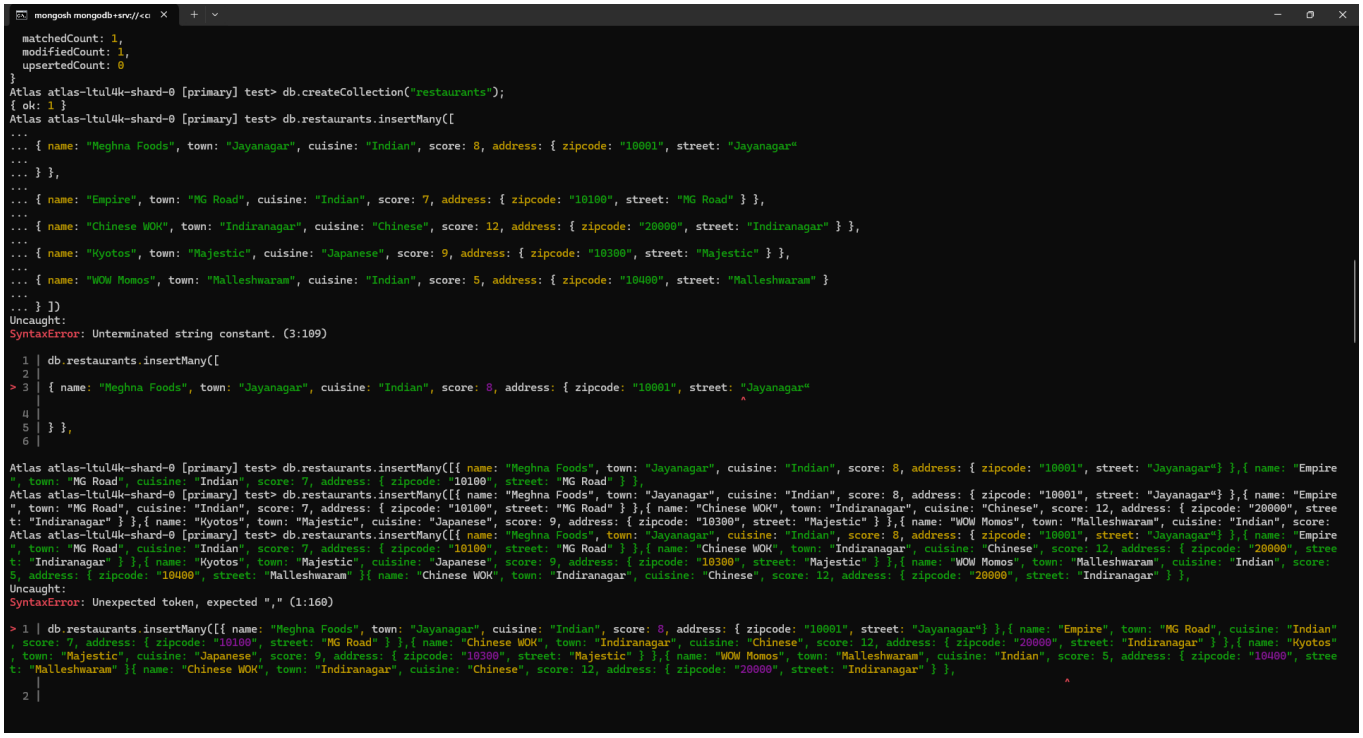
i. Write NoSQL Queries on “Restaurant” collection.

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.

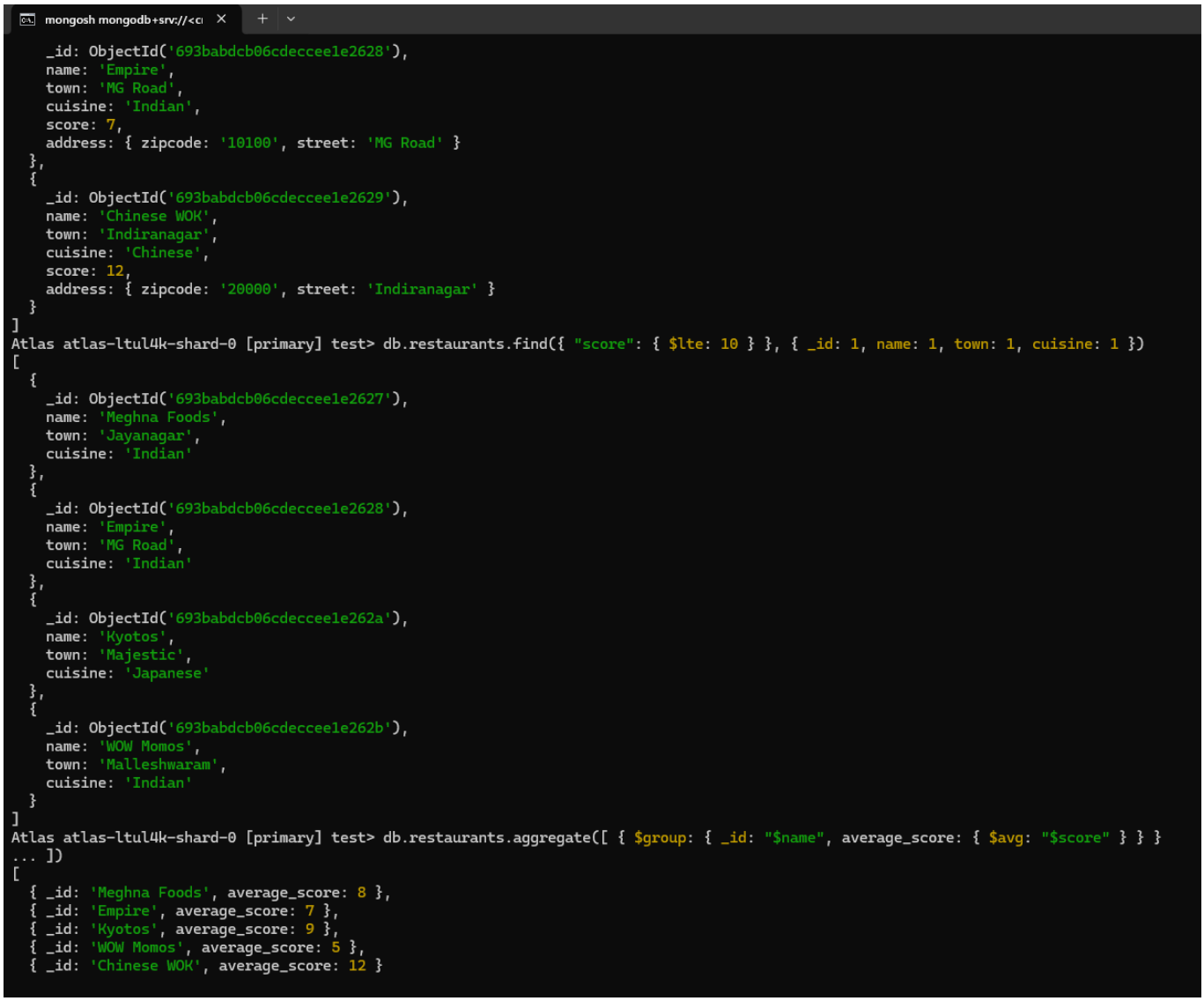
iv. Write a MongoDB query to find the restaurant Id, name, town and cuisine for those restaurants which achieved a score which is not more than 10.

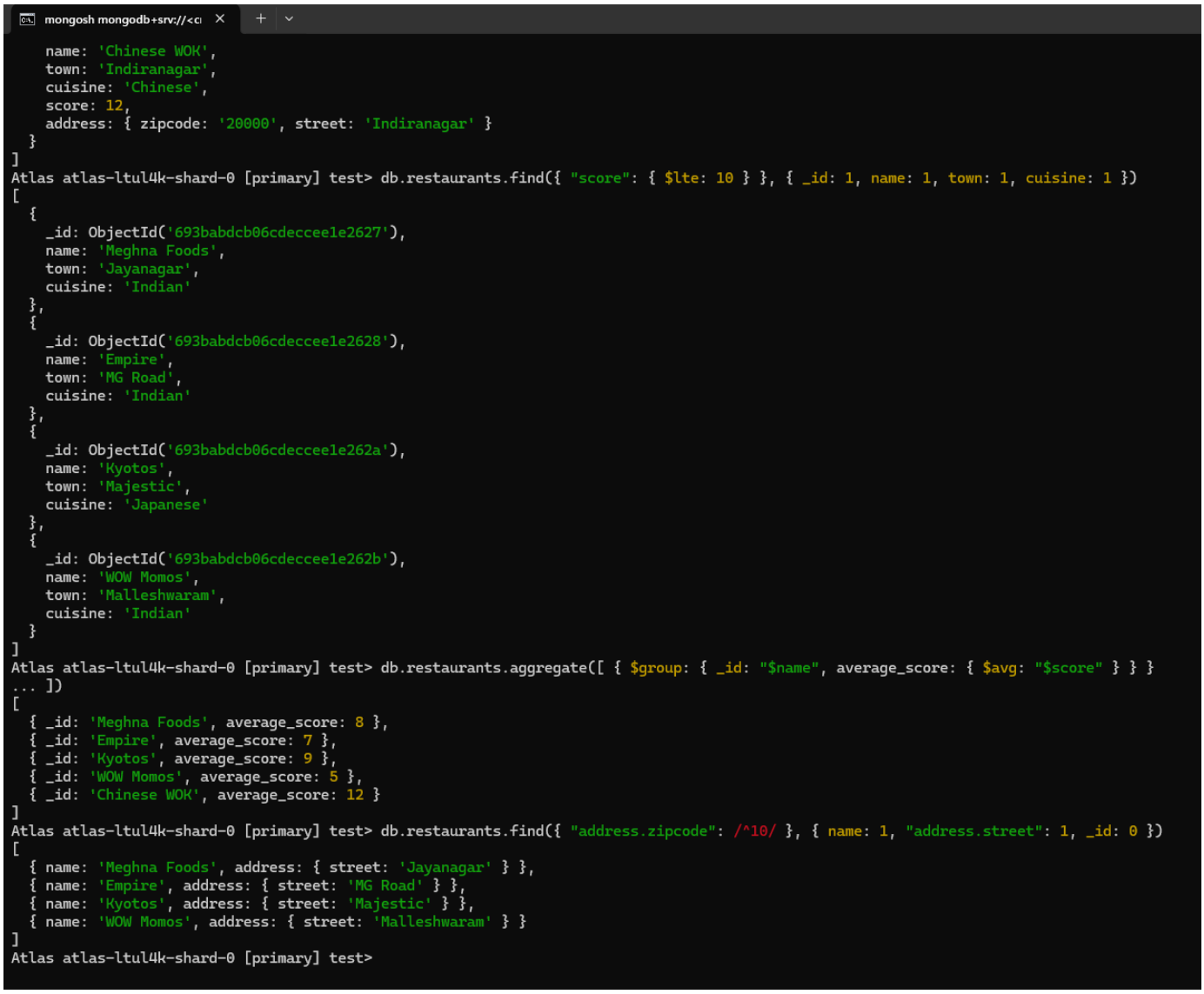
v. Write a MongoDB query to find the average score for each restaurant.



```
  6 |

Atlas atlas-ltul4k-shard-0 [primary] test> db.restaurants.insertMany([
...
... { name: "Meghna Foods", town: "Jayanagar", cuisine: "Indian", score: 8, address: { zipcode: "10001", street: "Jayanagar"} },
...
... { name: "Empire", town: "MG Road", cuisine: "Indian", score: 7, address: { zipcode: "10100", street: "MG Road" } },
...
... { name: "Chinese WOK", town: "Indiranagar", cuisine: "Chinese", score: 12, address: { zipcode: "20000", street: "Indiranagar" } },
...
... { name: "Kyotos", town: "Majestic", cuisine: "Japanese", score: 9, address: { zipcode: "10300", street: "Majestic" } },
...
... { name: "WOW Momos", town: "Malleshwaram", cuisine: "Indian", score: 5, address: { zipcode: "10400", street: "Malleshwaram" }
...
... } ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('693babdcb06cdecceele2627'),
    '1': ObjectId('693babdcb06cdecceele2628'),
    '2': ObjectId('693babdcb06cdecceele2629'),
    '3': ObjectId('693babdcb06cdecceele262a'),
    '4': ObjectId('693babdcb06cdecceele262b')
  }
}
Atlas atlas-ltul4k-shard-0 [primary] test> db.restaurants.find({})
[
  {
    _id: ObjectId('693babdcb06cdecceele2627'),
    name: 'Meghna Foods',
    town: 'Jayanagar',
    cuisine: 'Indian',
    score: 8,
    address: { zipcode: '10001', street: 'Jayanagar' }
  },
  {
    _id: ObjectId('693babdcb06cdecceele2628'),
    name: 'Empire',
    town: 'MG Road',
    cuisine: 'Indian',
    score: 7,
    address: { zipcode: '10100', street: 'MG Road' }
  },
  {
    _id: ObjectId('693babdcb06cdecceele2629'),
    name: 'Chinese WOK',
    town: 'Indiranagar',
    cuisine: 'Chinese',
    score: 12,
    address: { zipcode: '20000', street: 'Indiranagar' }
```


mongosh mongodb+srv://<cl
cuisine: 'Chinese',
score: 12,
address: { zipcode: '20000', street: 'Indiranagar' }
},
{
_id: ObjectId('693babdcb06cdeccee1e262a'),
name: 'Kyotos',
town: 'Majestic',
cuisine: 'Japanese',
score: 9,
address: { zipcode: '10300', street: 'Majestic' }
},
{
_id: ObjectId('693babdcb06cdeccee1e262b'),
name: 'WOW Momos',
town: 'Malleshwaram',
cuisine: 'Indian',
score: 5,
address: { zipcode: '10400', street: 'Malleshwaram' }
}
]
Atlas atlas-ltul4k-shard-0 [primary] test> db.restaurants.find({}).sort({ name: -1 })
[
{
_id: ObjectId('693babdcb06cdeccee1e262b'),
name: 'WOW Momos',
town: 'Malleshwaram',
cuisine: 'Indian',
score: 5,
address: { zipcode: '10400', street: 'Malleshwaram' }
},
{
_id: ObjectId('693babdcb06cdeccee1e2627'),
name: 'Meghna Foods',
town: 'Jayanagar',
cuisine: 'Indian',
score: 8,
address: { zipcode: '10001', street: 'Jayanagar' }
},
{
_id: ObjectId('693babdcb06cdeccee1e262a'),
name: 'Kyotos',
town: 'Majestic',
cuisine: 'Japanese',
score: 9,
address: { zipcode: '10300', street: 'Majestic' }
},
{
_id: ObjectId('693babdcb06cdeccee1e2628'),
name: 'Empire',
town: 'MG Road',


mongosh mongodb+srv://<cl
_id: ObjectId('693babdcb06cdecceele2628'),
name: 'Empire',
town: 'MG Road',
cuisine: 'Indian',
score: 7,
address: { zipcode: '10100', street: 'MG Road' }
},
{
_id: ObjectId('693babdcb06cdecceele2629'),
name: 'Chinese WOK',
town: 'Indiranagar',
cuisine: 'Chinese',
score: 12,
address: { zipcode: '20000', street: 'Indiranagar' }
}
]
Atlas atlas-ltul4k-shard-0 [primary] test> db.restaurants.find({ "score": { $lte: 10 } }, { _id: 1, name: 1, town: 1, cuisine: 1 })
[
{
_id: ObjectId('693babdcb06cdecceele2627'),
name: 'Meghna Foods',
town: 'Jayanagar',
cuisine: 'Indian'
},
{
_id: ObjectId('693babdcb06cdecceele2628'),
name: 'Empire',
town: 'MG Road',
cuisine: 'Indian'
},
{
_id: ObjectId('693babdcb06cdecceele262a'),
name: 'Kyotos',
town: 'Majestic',
cuisine: 'Japanese'
},
{
_id: ObjectId('693babdcb06cdecceele262b'),
name: 'WOW Momos',
town: 'Malleshwaram',
cuisine: 'Indian'
}
]
Atlas atlas-ltul4k-shard-0 [primary] test> db.restaurants.aggregate([{ $group: { _id: "$name", average_score: { $avg: "$score" } } }
...])
[
{ _id: 'Meghna Foods', average_score: 8 },
{ _id: 'Empire', average_score: 7 },
{ _id: 'Kyotos', average_score: 9 },
{ _id: 'WOW Momos', average_score: 5 },
{ _id: 'Chinese WOK', average_score: 12 }


mongosh mongodb+srv://<c
name: 'Chinese WOK',
town: 'Indiranagar',
cuisine: 'Chinese',
score: 12,
address: { zipcode: '20000', street: 'Indiranagar' }
}
]
Atlas atlas-ltul4k-shard-0 [primary] test> db.restaurants.find({ "score": { $lte: 10 } }, { _id: 1, name: 1, town: 1, cuisine: 1 })
[
{
_id: ObjectId('693babdcb06cdeccee1e2627'),
name: 'Meghna Foods',
town: 'Jayanagar',
cuisine: 'Indian'
},
{
_id: ObjectId('693babdcb06cdeccee1e2628'),
name: 'Empire',
town: 'MG Road',
cuisine: 'Indian'
},
{
_id: ObjectId('693babdcb06cdeccee1e262a'),
name: 'Kyotos',
town: 'Majestic',
cuisine: 'Japanese'
},
{
_id: ObjectId('693babdcb06cdeccee1e262b'),
name: 'WOW Momos',
town: 'Malleshwaram',
cuisine: 'Indian'
}
]
Atlas atlas-ltul4k-shard-0 [primary] test> db.restaurants.aggregate([{ $group: { _id: "$name", average_score: { $avg: "$score" } } }
...])
[
{ _id: 'Meghna Foods', average_score: 8 },
{ _id: 'Empire', average_score: 7 },
{ _id: 'Kyotos', average_score: 9 },
{ _id: 'WOW Momos', average_score: 5 },
{ _id: 'Chinese WOK', average_score: 12 }
]
Atlas atlas-ltul4k-shard-0 [primary] test> db.restaurants.find({ "address.zipcode": /^10/ }, { name: 1, "address.street": 1, _id: 0 })
[
{ name: 'Meghna Foods', address: { street: 'Jayanagar' } },
{ name: 'Empire', address: { street: 'MG Road' } },
{ name: 'Kyotos', address: { street: 'Majestic' } },
{ name: 'WOW Momos', address: { street: 'Malleshwaram' } }
]
Atlas atlas-ltul4k-shard-0 [primary] test>