VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

Database Management Systems (23CS3PCDBM)

Submitted by

SHRIHARI VISWANATHAN (1BF24CS288)

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

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 SHRIHARI VISWANATHAN (1BF24CSCS288), who is bonafide student of B. M. S. College of Engineering. It is in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a Database Management Systems (23CS3PCDBM) work prescribed for the said degree.

Lab faculty Incharge Name Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

•

Index

•

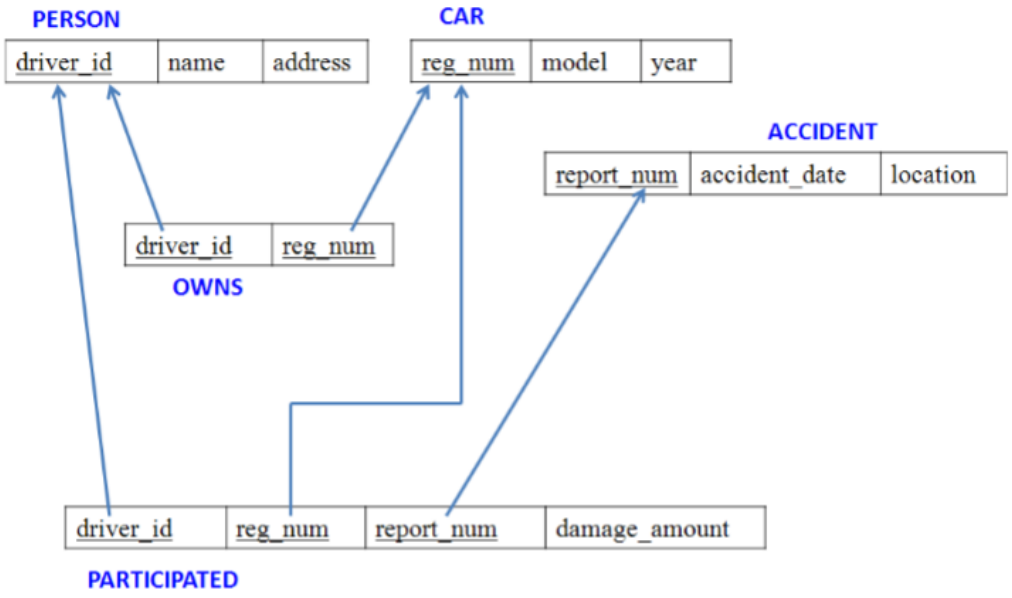
INSURANCE DATABASE

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.

•

Schema Diagram



CREATING AND INSERTING INTO CAR :

```
CREATE TABLE car(
    reg_num VARCHAR(10) PRIMARY KEY,
    model VARCHAR(10),
    year INT
);

INSERT INTO car(reg_num, model, year)
VALUES
    ("KA052250", "Indica", 1990),
    ("KA031181", "Lancer", 1957),
    ("KA095477", "Toyota", 1998),
    ("KA053408", "Honda", 2008),
    ("KA041702", "Audi", 2005);
```

-

CREATING AND INSERTING INTO PERSON

```
25 • CREATE TABLE person(
26       driver_id VARCHAR(10) PRIMARY KEY NOT NULL,
27       name VARCHAR(20) NOT NULL,
28       address VARCHAR(30) NOT NULL
29   );
30
31 •  INSERT INTO person(driver_id, name, address)
32   VALUES
33       ("A01", "Richard", "Srinivas nagar"),
34       ("A02", "Pradeep", "Rajaji nagar"),
35       ("A03", "Smith", "Ashok nagar"),
36       ("A04", "Venu", "N R Colony "),
37       ("A05", "Jhon", "Hanumanth nagar");
38
39
```

- **CREATING AND INSERTING INTO ACCIDENT**

```
41 • CREATE TABLE accident(
42       report_num INT PRIMARY KEY NOT NULL ,
43       accident_date DATE,
44       location VARCHAR(20)
45   );
46
47 •  INSERT INTO accident(report_num, accident_date, location)
48   VALUES
49       (11, "2003-01-01", "Mysore Road"),
50       (12, "2004-02-02", "South end Circle"),
51       (13, "2003-01-21", "Bull temple Road"),
52       (14, "2008-02-17", "Mysore Road"),
53       (15, "2005-03-04", "Kanakpura Road");
54
55
```

- **CREATING AND INSERTING INTO OWNS**

```
57 • CREATE TABLE owns(
58       driver_id VARCHAR(10) ,
59       reg_num VARCHAR(10) ,
60       PRIMARY KEY(reg_num, driver_id),
61       FOREIGN KEY(driver_id) REFERENCES person(driver_id),
62       FOREIGN KEY(reg_num) REFERENCES car(reg_num)
63   );
64
65 •  INSERT INTO owns(driver_id, reg_num)
66   VALUES
67       ("A01", "KA052250"),
68       ("A02", "KA053408"),
69       ("A03", "KA031181"),
70       ("A04", "KA095477"),
71       ("A05", "KA041702");
72
```

-

CREATING AND INSERTING INTO PARTICIPATED

```
75 • CREATE TABLE participated(
76       driver_id VARCHAR(10),
77       reg_num VARCHAR(10),
78       report_num INT,
79       damage_amount INT,
80       PRIMARY KEY(driver_id, reg_num, report_num),
81       FOREIGN KEY(driver_id) REFERENCES person(driver_id),
82       FOREIGN KEY(reg_num) REFERENCES car(reg_num),
83       FOREIGN KEY(report_num) REFERENCES accident(report_num)
84   );
85
86 • INSERT INTO participated(driver_id, reg_num, report_num, damage_amount)
87   VALUES
88       ("A01", "KA052250", 11, 10000),
89       ("A02", "KA053408", 12, 50000),
90       ("A03", "KA095477", 13, 25000),
91       ("A04", "KA031181", 14, 3000),
92       ("A05", "KA041702", 15, 5000);
93
```

- **DISPLAY ALL THE DATA IN THE TABLE PARTICIPATED**

```
96 • SELECT * FROM participated;
```

Result Grid | Filter Rows: | Edit:

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

25000

INSERT A NEW ROW INTO ACCIDENTS TABLE

```
102 • INSERT INTO accident
103   VALUES(16, "2008-03-15", "Domlur");
```

```
105 • SELECT * FROM accident;
106
```

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

report_num	accident_date	location
11	2003-01-01	Mysore Road
12	2004-02-02	South end Circle
13	2003-01-21	Bull temple Road
14	2008-02-17	Mysore Road
15	2005-03-04	Kanakpura Road
16	2008-03-15	Domlur
NULL	NULL	NULL

-

- **SELECTING GENERAL**

```
108 •  SELECT accident_date, location FROM accident;
109
```

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

accident_date	location
2003-01-01	Mysore Road
2004-02-02	South end Circle
2003-01-21	Bull temple Road
2008-02-17	Mysore Road
2005-03-04	Kanakpura Road
2008-03-15	Domlur

FIND ALL THE DRIVERS ID WHO HAVE CAUSED DAMAGE OF ATLEAST 25000

```
111 •  SELECT driver_id FROM participated
112    WHERE damage_amount >= 25000;
113
```

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

driver_id
A02
A03

- **RENAMING THE TABLE**

```
ALTER TABLE accident RENAME TO accidents;
ALTER TABLE accidents RENAME TO accident;
```

- **Find the maximum damage_amount**

```
120 •  SELECT MAX(damage_amount) FROM participated;
121
```

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

MAX(damage_amount)
25000

-

- **FIND THE TOTAL NO.OF PEOPLE WHO OWNED CARS THAT INVOLVED IN AN ACCIDENT IN 2008**

```
123 •  SELECT COUNT(DISTINCT driver_id) AS COUNT_PEOPLE_WHO_OWNED_CARS_THAT_INVOLVED_AN_ACCIDENT_IN_2008 FROM participated
124    WHERE report_num IN (SELECT report_num FROM accident WHERE YEAR(accident_date) = "2008");
125
```

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

COUNT_PEOPLE_WHO_OWNED_CARS_THAT_INV
1

DISPLAY THE DETAILS IN THE PARTICIPATED IN THE DESCENDING ORDER OF damage_amount

```
130 •  SELECT * FROM PARTICIPATED ORDER BY damage_amount DESC;
131
```

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

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

- **Find the average of damage_amount**

```
133 •  SELECT AVG(damage_amount) FROM participated;
134
```

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

AVG(damage_amount)
13600.0000

INSURANCE DATABASE -> ADDITIONAL QUERIES

LIST THE ENTIRE PARTICIPATED RELATION IN THE DESCENDING ORDER OF DAMAGE AMOUNT

```
36 •  SELECT * FROM PARTICIPATED ORDER BY damage_amount DESC;
```

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

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

•

FIND THE AVERAGE DAMAGE AMOUNT

```
39 • SELECT AVG(damage_amount) FROM participated;
```

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

AVG(damage_amount)
13600.0000

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

```
144 • DELETE FROM PARTICIPATED WHERE damage_amount  < (SELECT AVG(damage_amount) FROM PARTICIPATED);
145 • SELECT * FROM damage_amount
146   ORDER BY damage_amount desc;
```

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

driver_id	reg_num	report_num	damage_amount
A01	KA052250	11	10000
A04	KA031181	14	3000
A05	KA041702	15	5000

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

```
143 • SELECT name
144   FROM Person A, Participated B
145   WHERE
146       A.driver_id = B.driver_id
147       AND
148       damage_amount > (
149           SELECT AVG(damage_amount)
150           FROM Participated);
```

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

name
Pradeep
Smith

FIND MAXIMUM DAMAGE AMOUNT.

```
155 • SELECT MAX(damage_amount)
156   FROM participated;
```

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

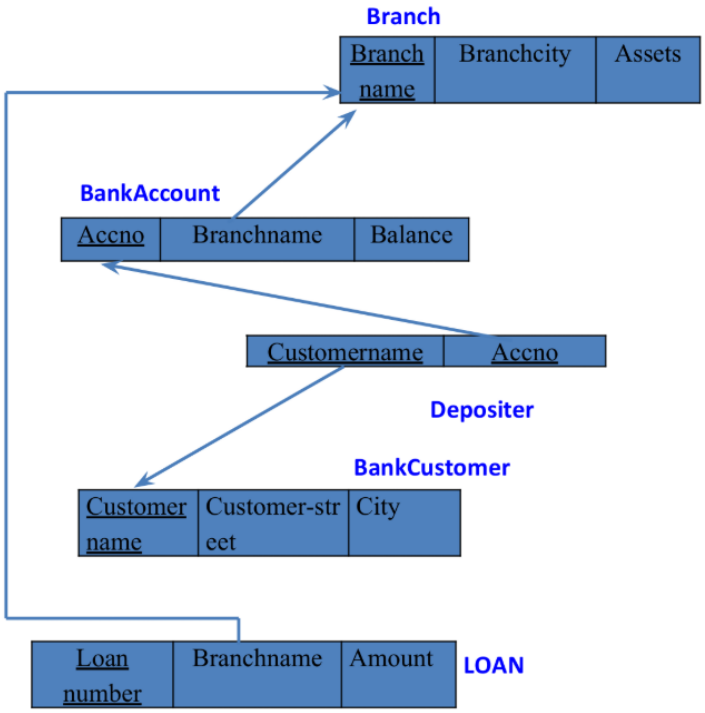
MAX(damage_amount)
10000

BANK ACCOUNT DATABASE

Specification of Bank Database Application:

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

Schema Diagram:



QUERY 1 AND 2: CREATE THE TABLE AND INSERT DATA INTO THEM •

DESC branch

```
 6 • CREATE TABLE branch(
 7       branch_name VARCHAR(30) PRIMARY KEY,
 8       branch_city VARCHAR(30),
 9       assets REAL
10   );
11
12 • desc branch;
```

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

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

```
53 • INSERT INTO branch VALUES
54       ("SBI_Chamrajpet", "Bangalore", 50000),
55       ("SBI_Residency_Road", "Bangalore", 10000),
56       ("SBI_Shivaji_Road", "Bangalore", 20000),
57       ("SBI_Parliament_Road", "Bangalore", 10000),
58       ("SBI_Jantar_Mantar", "Delhi", 20000);
59
```

- **DESC bank_account**

```
15 • CREATE TABLE bank_account(
16       account_no INTEGER PRIMARY KEY,
17       branch_name VARCHAR(30),
18       balance REAL,
19       FOREIGN KEY(branch_name) REFERENCES branch(branch_name)
20   );
21
22 • desc bank_account;
```

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

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

```
61 • INSERT INTO bank_account VALUES
62       (1, "SBI_Chamrajpet", 2000),
63       (2, "SBI_Residency_Road", 5000),
64       (3, "SBI_Shivaji_Road", 6000),
65       (4, "SBI_Parliament_Road", 9000),
66       (5, "SBI_Jantar_Mantar", 8000),
67       (6, "SBI_Shivaji_Road", 4000),
68       (8, "SBI_Residency_Road", 5000),
69       (9, "SBI_Parliament_Road", 3000),
70       (10, "SBI_Residency_Road", 5000),
71       (11, "SBI_Jantar_Mantar", 2000);
72
```

- **DESC bank_customer**

```
CREATE TABLE bank_customer(
    customer_name VARCHAR(30) PRIMARY KEY,
    customer_street VARCHAR(30),
    customer_city VARCHAR(30)
);

DESC bank_customer;
```

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

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

```
74 • INSERT INTO bank_customer VALUES
75       ("Avinash", "Bull_Temple_Road", "Bangalore"),
76       ("Dinesh", "Bannergatta_Road", "Bangalore"),
77       ("Mohan", "National_College_Road", "Bangalore"),
78       ("Nikhil", "Akbar_Road", "Delhi"),
79       ("Ravi", "Prithviraj_Road", "Delhi");
```

•

DESC depositor

```
33 • CREATE TABLE depositor(
34       customer_name VARCHAR(30),
35       account_no INTEGER,
36       PRIMARY KEY (customer_name, account_no),
37       FOREIGN KEY(customer_name) REFERENCES bank_customer(customer_name),
38       FOREIGN KEY(account_no) REFERENCES bank_account(account_no)
39   );
40
41 • DESC depositor;
```

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

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

```
81 • INSERT INTO depositor VALUES
82       ("Avinash", 1),
83       ("Avinash", 8),
84       ("Dinesh", 2),
85       ("Nikhil", 4),
86       ("Nikhil", 9),
87       ("Dinesh", 10),
88       ("Nikhil", 11),
89       ("Ravi", 5);
```

DESC loan;

```
44 • CREATE TABLE loan(
45       loan_number INTEGER PRIMARY KEY,
46       branch_name VARCHAR(30),
47       amount REAL,
48       FOREIGN KEY(branch_name) REFERENCES branch(branch_name)
49   );
50
51 • DESC loan;
```

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

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

```
91 • INSERT INTO loan VALUES
92       (1, 'SBI_Chamrajpet', 1000),
93       (2, 'SBI_Residency_Road', 2000),
94       (3, 'SBI_Residency_Road', 3000),
95       (4, 'SBI_Shivaji_Road', 4000),
96       (5, 'SBI_Jantar_Mantar', 5000);
97
```

QUERY 3 Find all the customers who have atleast two deposits at the same branch

```
99 • SELECT d.customer_name, count(d.customer_name) AS  Number_of_Acoounts, ba.branch_name
00   FROM depositor d
01   NATURAL JOIN bank_account ba
02   GROUP BY d.customer_name, ba.branch_name
03   HAVING COUNT(d.account_no) >= 2;
```

Result Grid | Filter

customer_name
Dinesh
Nikhil

QUERY 4 Find all the customers who have an account at all the branches located in specific city ex: Delhi

```
SELECT bc.customer_name
FROM bank_customer bc
WHERE NOT EXISTS (
    SELECT branch_name
    FROM branch
    WHERE branch_city = "Delhi"
    AND branch_name NOT IN (
        SELECT ba.branch_name
        FROM depositor d
        JOIN bank_account ba ON d.account_no = ba.account_no
        WHERE d.customer_name = bc.customer_name
    )
);
```

Result Grid | Filter Rows:

customer_name
Nikhil
Ravi
NULL

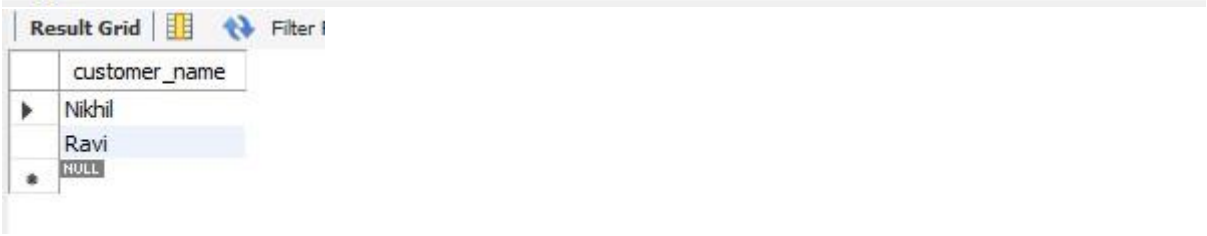
QUERY 5 Demonstrate how you delete all the account tuples at a branch located at ex: Shivaji Road

```
DELETE FROM bank_account
WHERE branch_name IN (
    SELECT branch_name
    FROM branch
    where branch_name = "SBI_Shivaji_Road"
);
```

BANK ACCOUNT DATABASE -> ADDITIONAL QUERIES

- **Find all customers who have an account at all the branches located in a specific city ex: Delhi**

```sql
SELECT bc.customer_name
FROM bank_customer bc
WHERE NOT EXISTS (

    SELECT b.branch_name
    FROM branch b
    WHERE b.branch_city = 'Delhi'
    AND NOT EXISTS (
        SELECT 1
        FROM depositor d
        JOIN bank_account ba ON d.account_no = ba.account_no
        WHERE d.customer_name = bc.customer_name
        AND ba.branch_name = b.branch_name
    )
);
```



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

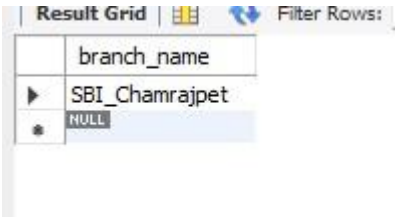
```sql
SELECT DISTINCT lc.customer_name
FROM loan l
JOIN branch b ON l.branch_name = b.branch_name
LEFT JOIN bank_account ba ON l.branch_name = ba.branch_name
LEFT JOIN depositor d ON ba.account_no = d.account_no
JOIN bank_customer lc ON d.customer_name = lc.customer_name
WHERE d.customer_name IS NULL;
```

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

```
SELECT DISTINCT customer_name
FROM loan l JOIN bank_account ba ON l.branch_name = ba.branch_name
JOIN depositor d ON ba.account_no = d.account_no
WHERE l.branch_name = "Bangalore" AND ba.branch_name = "Bangalore";
```

- **Find the names of all the branches that have greater assets than all the branches located in Delhi**

```
SELECT branch_name FROM branch
WHERE assets > all
(SELECT assets FROM branch WHERE branch_city = "Delhi");
```



- **Demonstrate how you delete all the account tuples at every branch located in a specific city ex: Delhi**

```
DELETE FROM bank_account
WHERE branch_name IN (
    SELECT branch_name
    FROM branch
    WHERE branch_city = 'Delhi'
);
```

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

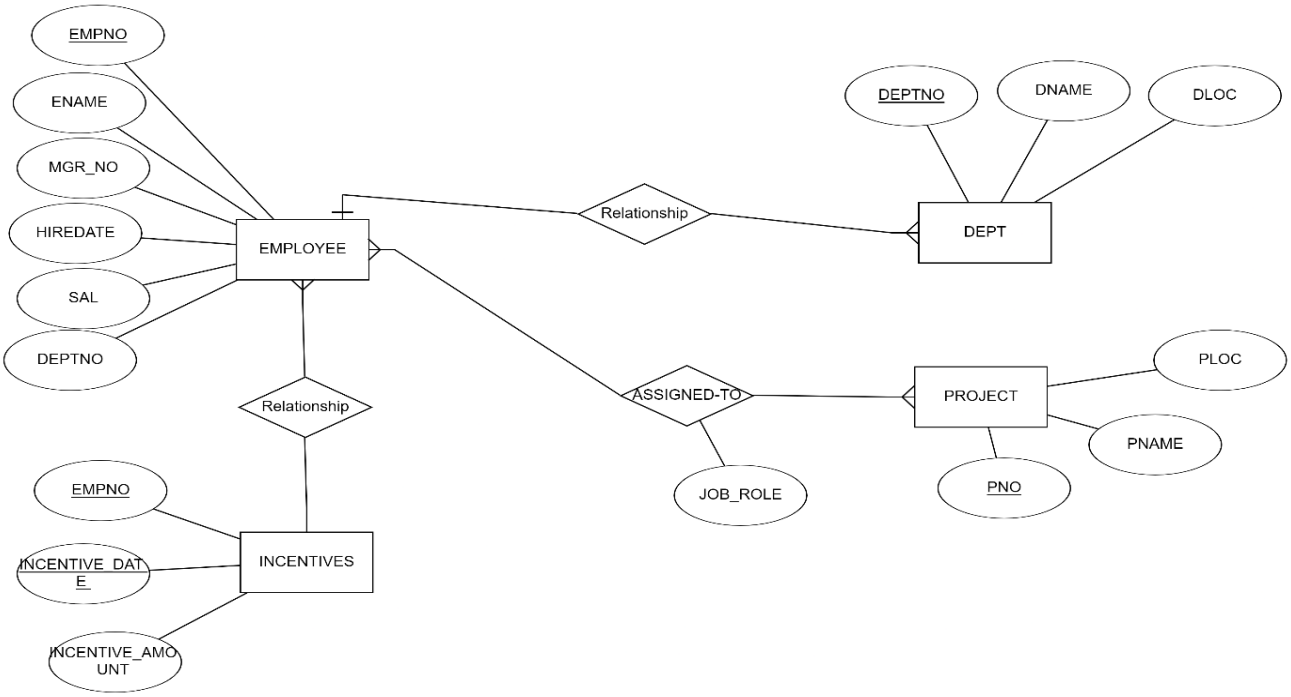
```
ALTER TABLE bank_account
ADD COLUMN interest_rate REAL DEFAULT 0.05;

UPDATE bank_account
SET balance = balance + (balance * interest_rate);

UPDATE branch
SET assets = assets * 1.05;
```

EMPLOYEE DATABASE

Specification of Employee Database Application:

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

Schema Diagram:




```sql
CREATE DATABASE EMP;

USE EMP;

CREATE TABLE dept(
    deptno DECIMAL(2, 0) PRIMARY KEY,
    dname VARCHAR(14) DEFAULT NULL,
    dloc VARCHAR(14) DEFAULT NULL
);


CREATE TABLE emp (
    empno decimal(4,0) primary key,
    ename varchar(10) default NULL,
    mgr_no decimal(4,0) default NULL,
    hiredate date default NULL,
    sal decimal(7,2) default NULL,
    deptno decimal(2,0) references dept(deptno) on delete cascade on update cascade
);

```

```sql
create table incentives (
     empno decimal(4,0) references emp(empno) on delete cascade on update cascade,
     incentive_date date,
     incentive_amount decimal(10,2),
     primary key(empno,incentive_date)
);


Create table project (
     pno int primary key,
     pname varchar(30) not null,
     ploc varchar(30)
);


Create table assigned_to (
     empno decimal(4,0) references emp(empno) on delete cascade on update cascade,
     pno int references project(pno) on delete cascade on update cascade,
     job_role varchar(30),
     primary key(empno,pno)
);
```

```sql
INSERT INTO dept VALUES (10, 'ACCOUNTING', 'MUMBAI');
INSERT INTO dept VALUES (20, 'RESEARCH', 'BENGALURU');
INSERT INTO dept VALUES (30, 'SALES', 'DELHI');
INSERT INTO dept VALUES (40, 'OPERATIONS', 'CHENNAI');

INSERT INTO emp VALUES (7369, 'Adarsh', 7902, '2012-12-17', '80000.00', '20');
INSERT INTO emp VALUES (7499, 'Shruthi', 7698, '2013-02-20', '16000.00', '30');
INSERT INTO emp VALUES (7521, 'Anvitha', 7698, '2015-02-22', '12500.00', '30');
INSERT INTO emp VALUES (7566, 'Tanvir', 7839, '2008-04-02', '29750.00', '20');
INSERT INTO emp VALUES (7654, 'Ramesh', 7698, '2014-09-28', '12500.00', '30');
INSERT INTO emp VALUES (7698, 'Kumar', 7839, '2015-05-01', '28500.00', '30');
INSERT INTO emp VALUES (7782, 'CLARK', 7839, '2017-06-09', '24500.00', '10');
INSERT INTO emp VALUES (7788, 'SCOTT', 7566, '2010-12-09', '30000.00', '20');
INSERT INTO emp VALUES (7839, 'KING', NULL, '2009-11-17', '5000.00', '10');
INSERT INTO emp VALUES (7844, 'TURNER', 7698, '2010-09-08', '15000.00', '30');
INSERT INTO emp VALUES (7876, 'ADAMS', 7788, '2013-01-12', '11000.00', '20');
INSERT INTO emp VALUES (7900, 'JAMES', 7698, '2017-12-03', '9500.00', '30');
INSERT INTO emp VALUES (7902, 'FORD', 7566, '2010-12-03', '30000.00', '20');

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

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

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

```
95 •   SELECT * FROM DEPT;
96
97
```

Result Grid | Filter Rows: | Edit:

deptno	dname	dloc
10	ACCOUNTING	MUMBAI
20	RESEARCH	BENGALURU
30	SALES	DELHI
40	OPERATIONS	CHENNAI
NULL	NULL	NULL

```
98 •  select * from emp;
```

Result Grid | Filter Rows: | Edit:

empno	ename	mgr_no	hiredate	sal	deptno
7566	Tanvir	7839	2008-04-02	29750.00	20
7654	Ramesh	7698	2014-09-28	12500.00	30
7698	Kumar	7839	2015-05-01	28500.00	30
7782	CLARK	7839	2017-06-09	24500.00	10
7788	SCOTT	7566	2010-12-09	30000.00	20
7839	KING	NULL	2009-11-17	5000.00	10
7844	TURNER	7698	2010-09-08	15000.00	30
7876	ADAMS	7788	2013-01-12	11000.00	20
7900	JAMES	7698	2017-12-03	9500.00	30
7902	FORD	7566	2010-12-03	30000.00	20
NULL	NULL	NULL	NULL	NULL	NULL

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

```
100 •  select e.empno
101    from emp e, assigned_to a, project p
102    where e.empno=a.empno and a.pno=p.pno and
103    p.ploc in ("Bengaluru", "Hyderabad", "Mysuru");
```

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

empno
7499
7499
7521
7521
7566
7654
7654
7698
7839

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

```
106 •  SELECT e.empno
107    FROM emp e
108    WHERE NOT EXISTS (
109        SELECT 1
110        FROM incentives i
111        WHERE i.empno = e.empno
112    );
```

Result Grid | Filter Rows: | Edit: | Export/Import:

empno
7369
7782
7788
7839
7844
7876
7900
7902
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.**

```
116 •  select e.empno, e.ename, d.dname, d.dloc, a.job_role, p.ploc
117    from emp e, dept d, assigned_to a, project p
118    where
119        e.deptno = d.deptno and
120        e.empno = a.empno and
121        a.pno = p.pno and d.dloc = p.ploc;
```

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

empno	ename	dname	dloc	job_role	ploc
7566	Tanvir	RESEARCH	BENGALURU	Project Manager	BENGALURU

EMPLOYEE DATABASE -> ADDITIONAL QUERIES

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

```
26  SELECT m.ename, count(*)
27  FROM emp e,emp m
28  WHERE e.mgr_no = m.empno
29  GROUP BY m.ename
30  HAVING count(*) =(SELECT MAX(mycount)
31  from (SELECT COUNT(*) mycount
32  FROM emp
33  GROUP BY mgr_no) a);
```

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

ename	count(*)
Kumar	5

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

```
36  SELECT *
37  FROM emp m
38  WHERE m.empno IN
39  (SELECT mgr_no
40  FROM emp)
41  AND m.sal >
42  (SELECT avg(e.sal)
43  FROM emp e
44  WHERE e.mgr_no = m.empno );
```

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

empno	ename	mgr_no	hiredate	sal	deptno
7698	Kumar	7839	2015-05-01	28500.00	30
7788	SCOTT	7566	2010-12-09	30000.00	20
NULL	NULL	NULL	NULL	NULL	NULL

- **SQL Query to find the employee details who got second maximum**

  incentive in Febrauary 2019.

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

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

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

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

```
select distinct m.mgr_no from emp e, emp m
where e.mgr_no = m.mgr_no and e.deptno = m.deptno and e.empno in
    (select distinct m.mgr_no from emp e, emp m
    where e.mgr_no = m.mgr_no and e.deptno = m.deptno);
```

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

mgr_no
7839
7566

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

```
SELECT *
FROM EMP E
WHERE E.DEPTNO = (SELECT E1.DEPTNO
FROM EMP E1
WHERE E1.EMPNO=E.MGR_NO);
```

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

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

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

```
69  SELECT distinct e.ename
70  FROM emp e,incentives i
71  WHERE (SELECT max(sal+incentive_amount)
72  FROM emp,incentives) >= ANY
73  (SELECT sal
74  FROM emp e1
75  where e.deptno=e1.deptno);
```

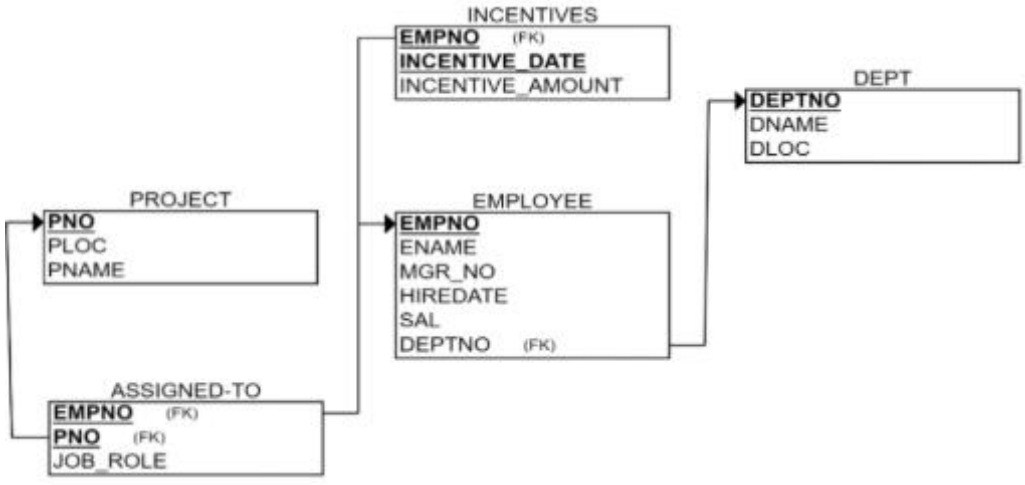
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ename
Adarsh
Shruthi
Anvitha
Tanvir
Ramesh
Kumar
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD

SUPPLIER DATABASE

Specification of Supplier Database Application:

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



```
create table SUPPLIERS(sid integer(5) primary key, sname varchar(20), city
varchar(20));

create table PARTS(pid integer(5) primary key, pname varchar(20), color varchar(10));

create table CATALOG(sid integer(5), pid integer(5), foreign key(sid)
references SUPPLIERS(sid), foreign key(pid) references PARTS(pid), cost
float(6), primary key(sid, pid));
```

```
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");
insert into suppliers values(10005, "Mahindra","Mumbai");
```

```
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");
```

```
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);
```

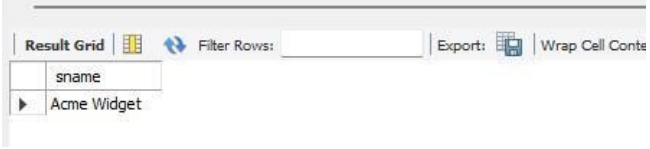
i) **Find the pnames of parts for which there is some supplier.**

```
40 •  SELECT DISTINCT P.pname
41    FROM Parts P, Catalog C
42    WHERE P.pid = C.pid;
```

pname
Book
Pen
Pencil
Mobile
Charger

ii) **Find the snames of suppliers who supply every part.**

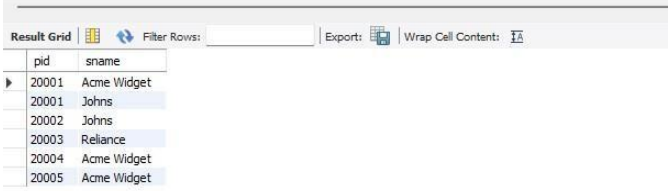
```
45 •  SELECT S.sname
46    FROM Suppliers S
47    WHERE
48    (( SELECT count(P.pid)
49    FROM Parts P ) =
50    ( SELECT count(C.pid)
51    FROM Catalog C
52    WHERE C.sid = S.sid ));
```



sname
Acme Widget

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

```
90 •  SELECT P.pid, S.sname
91    FROM Parts P, Suppliers S, Catalog C
92    WHERE C.pid = P.pid
93    AND C.sid = S.sid
94    AND C.cost = (SELECT MAX(C1.cost)
95    FROM Catalog C1
96    WHERE C1.pid = P.pid);
```



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

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

```
57 • SELECT S.sname
58   FROM Suppliers S
59   WHERE
60   (( SELECT count(P.pid)
61   FROM Parts P where color="Red" ) =
62   ( SELECT count(C.pid)
63   FROM Catalog C, Parts P
64   WHERE C.sid = S.sid AND
65   C.pid = P.pid AND P.color = "Red" ));
66
```

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

sname
Acme Widget
Johns

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

```
71 • SELECT P.pname
72   FROM Parts P, Catalog C, Suppliers S
73   WHERE P.pid = C.pid AND C.sid = S.sid
74   AND S.sname = "Acme Widget"
75   AND NOT EXISTS ( SELECT *
76   FROM Catalog C1, Suppliers S1
77   WHERE P.pid = C1.pid AND C1.sid = S1.sid AND
78   S1.sname != "Acme Widget");
79
80
81
```

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

pname
Mobile
Charger

v) **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).**

```
83 • SELECT DISTINCT C.sid FROM Catalog C
84   WHERE C.cost > ( SELECT AVG (C1.cost)
85   FROM Catalog C1
86   WHERE C1.pid = C.pid );
87
```

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

sid
10002
10004

SUPPLIER DATABASE -> ADDITIONAL QUERIES

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

```
103 •   SELECT P.pid, P.pname, S.sid, S.sname, C.cost
104     FROM Catalog C
105     JOIN Parts P ON C.pid = P.pid
106     JOIN Suppliers S ON C.sid = S.sid
107     WHERE C.cost = (SELECT MAX(cost) FROM Catalog);
108
```

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

pid	pname	sid	sname	cost
20003	Pencil	10004	Reliance	40

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

```
110 •   SELECT S.sid, S.sname
111     FROM Suppliers S
112     WHERE S.sid NOT IN (
113         SELECT C.sid
114         FROM Catalog C
115         JOIN Parts P ON C.pid = P.pid
116         WHERE P.color = 'Red'
117     );
```

Result Grid | Filter Rows:

sid	sname
10003	Vimal
10004	Reliance
10005	Mahindra
NULL	NULL

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

```
122 •  SELECT S.sid, S.sname, SUM(C.cost) AS total_value
123    FROM Suppliers S
124    LEFT JOIN Catalog C ON S.sid = C.sid
125    GROUP BY S.sid, S.sname;
126
```

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

sid	sname	total_value
10001	Acme Widget	70
10002	Johns	30
10003	Vimal	30
10004	Reliance	40
10005	Mahindra	NULL

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

```
130 •  SELECT S.sid, S.sname
131    FROM Suppliers S
132    JOIN Catalog C ON S.sid = C.sid
133    WHERE C.cost < 20
134    GROUP BY S.sid, S.sname
135    HAVING COUNT(*) >= 2;
```

Result Grid | Filter Rows: | Exp

sid	sname
10001	Acme Widget

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

```
140 •  SELECT P.pid, P.pname, S.sid, S.sname, C.cost
141    FROM Catalog C
142    JOIN Parts P ON C.pid = P.pid
143    JOIN Suppliers S ON C.sid = S.sid
144    WHERE C.cost = (
145        SELECT MIN(C2.cost)
146        FROM Catalog C2
147        WHERE C2.pid = C.pid
148    );
```

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

pid	pname	sid	sname	cost
20001	Book	10001	Acme Widget	10
20001	Book	10002	Johns	10
20002	Pen	10001	Acme Widget	10
20003	Pencil	10001	Acme Widget	30
20003	Pencil	10003	Vimal	30
20004	Mobile	10001	Acme Widget	10
20005	Charger	10001	Acme Widget	10

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

```
CREATE VIEW SupplierPartCount AS
SELECT S.sid, S.sname, COUNT(C.pid) AS total_parts
FROM Suppliers S
LEFT JOIN Catalog C ON S.sid = C.sid
GROUP BY S.sid, S.sname;
```

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

```
CREATE VIEW MostExpensiveSupplierPerPart AS
SELECT P.pid, P.pname, S.sid, S.sname, C.cost
FROM Catalog C
JOIN Parts P ON C.pid = P.pid
JOIN Suppliers S ON C.sid = S.sid
WHERE C.cost = (
    SELECT MAX(C2.cost)
    FROM Catalog C2
    WHERE C2.pid = C.pid
);
```

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

```
DELIMITER $$
CREATE TRIGGER check_cost_before_insert
BEFORE INSERT ON Catalog
FOR EACH ROW
BEGIN
    IF NEW.cost < 1 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Cost cannot be below 1';
    END IF;
END$$
DELIMITER ;
```

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

```
DELIMITER $$
CREATE TRIGGER set_default_cost
BEFORE INSERT ON Catalog
FOR EACH ROW
BEGIN
    IF NEW.cost IS NULL THEN
        SET NEW.cost = 10;
    END IF;
END$$
DELIMITER ;
```

NO SQL STUDENT DATABASE

Specification of NoSQL – Student Database Application:

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

Create Database:

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

Insert Appropriate Values:

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

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

Queries

1. Write a query to update EmailID of a student with RollNo 10.

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

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

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

3. Export the created table into Local File System.

	A	B	C	D	E
1	RollNo	Name	Age	ContactNo	EmailID
2	10	ABC	20	1112223334	CSE10Updated.cs24@bmsce.ac.in
3	11	FEM	20	2223334445	CSE11.cs24@bmsce.ac.in
4	12	GHI	20	3334445556	CSE12.cs24@bmsce.ac.in
5	13	JKL	20	4445556667	CSE13.cs24@bmsce.ac.in

4. Drop the table.

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

NO SQL – Customer Database

Specification of NoSQL – Customer Database Application:

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

Create Database:

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

Insert Appropriate Values:

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

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

Queries:

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

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

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

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

3. Export the created table into Local File System.

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

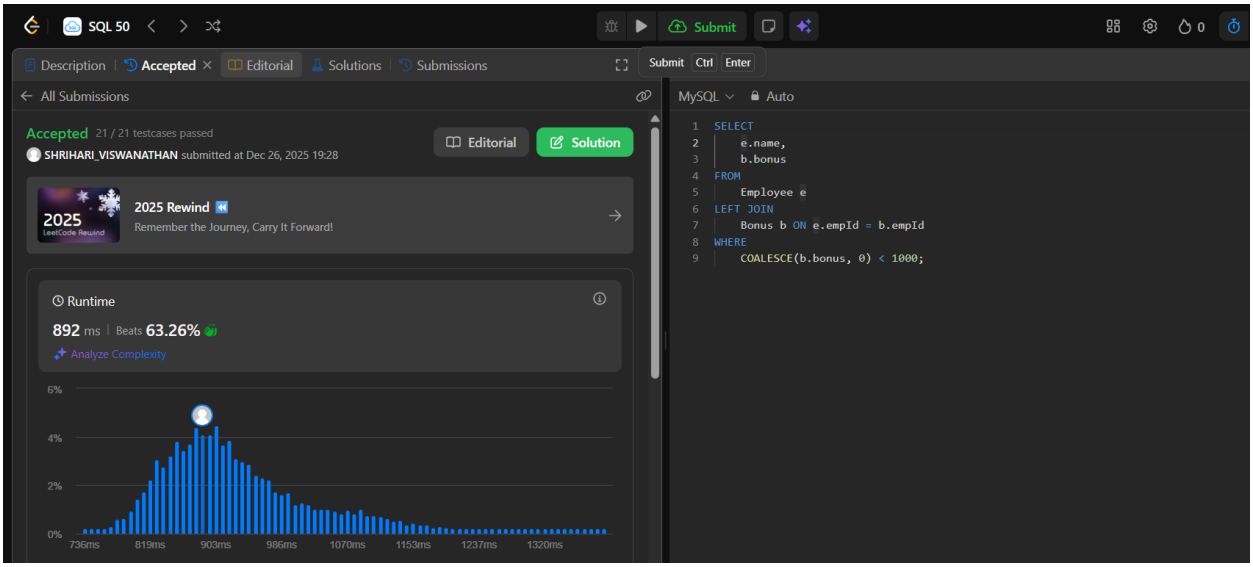
4. Drop the table.

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

Leetcode – 1

577. Employee Bonus

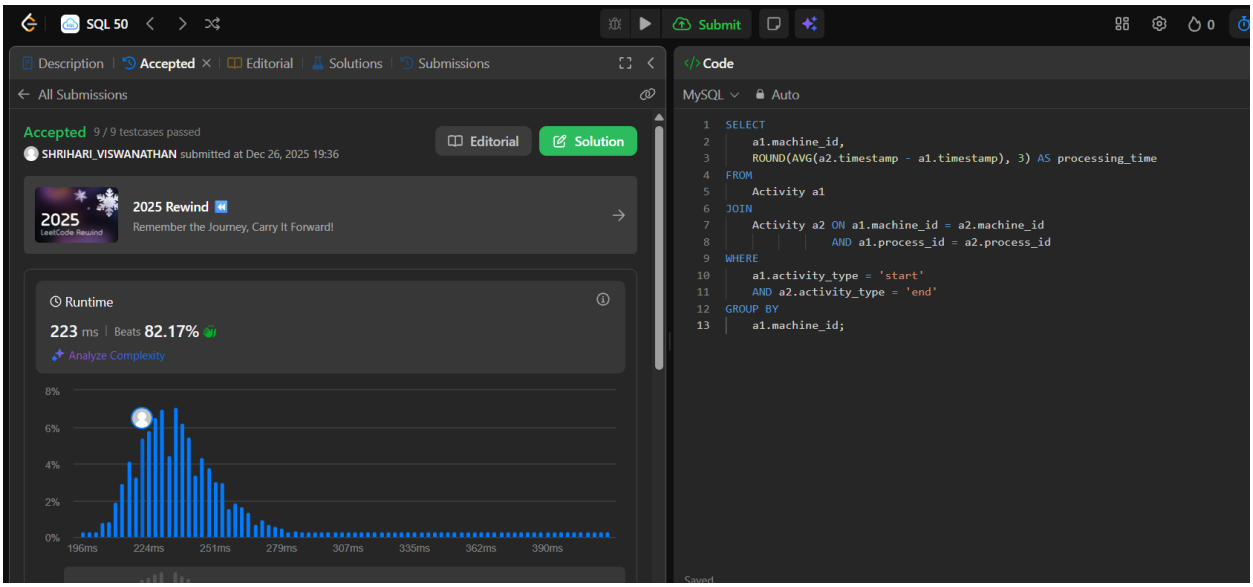
https://leetcode.com/problems/employee-bonus/description/?envType=study-plan-v2&envId=top-sql-50



Leetcode -2

1661. Average Time of Process per Machine

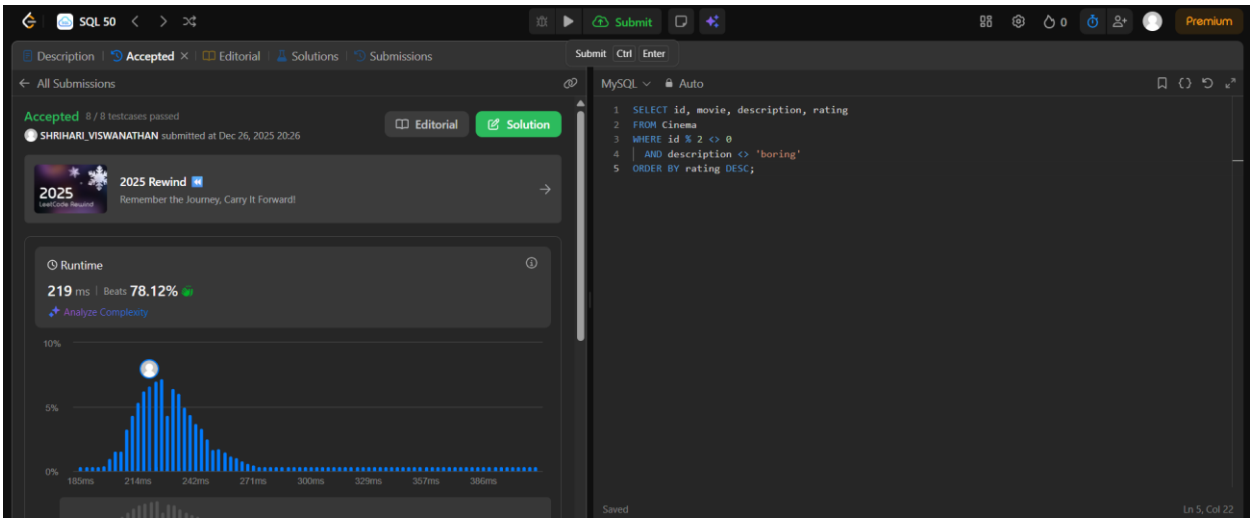
https://leetcode.com/problems/average-time-of-process-per-machine/description/?envType=study-plan-v2&envId=top-sql-50



Leetcode – 3

620. Not Boring Movies

https://leetcode.com/problems/not-boring-movies/description/?envType=study-plan-v2&envId=top-sql-50



Leetcode – 4

1978. Employees Whose Manager Left the Company

https://leetcode.com/problems/employees-whose-manager-left-the-company/?envType=study-plan-v2&envId=top-sql-50