

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

Database Management Systems (23CS3PCDBM)

Submitted by

SUMIT KUMAR MAHATO(1BF24CS305)

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 Dec-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 **Sumit Kumar Mahato(1BF24CS305)**, 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 2025. The Lab report has been approved as it satisfies the academic requirements in respect of a Database Management Systems (23CS3PCDBM) work prescribed for the said degree.

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

Index

Sl. No.	Date	Experiment Title	Page No.
1	25-9-2025	Insurance Database	4 - 11
2	9-10-2025	Bank Database	12 - 19
3	16-10-2025	More Queries on Bank Database	20- 23
4	30-10-2025	More Queries on Insurance Database	24-25
5	6-11-2025	Employee Database	26-35
6	13-11-2025	More Queries on Employee Database	36-37
7	20-11-2025	Supplier Database	38-45
8	27-11-2025	More Queries on Supplier Database	46-48
9	11-12-2025	NO SQL- Student Database	49-50
10	11-12-2025	NO SQL- Customer Database	51-54

Insurance Database

Experiment 1: Insurance Database

Specification of Insurance Database Application

The insurance database must maintain information about drivers, the cars they own, the accidents reported, and the participation of each driver and car in those accidents. Each driver in the system is uniquely identified by a driver ID, along with their name and address, and each car is

uniquely identified by its registration number together with details such as model and manufacturing year. The system must allow storing ownership information that links a driver to one or more cars, while also allowing a car to be linked to one or more drivers if shared ownership occurs; duplicate ownership

records for the same driver and car must not exist. Accident information must be stored using a unique report number assigned to each accident, along with the date on which the accident occurred and the

location where it happened. Every accident reported in the system must have at least one participation

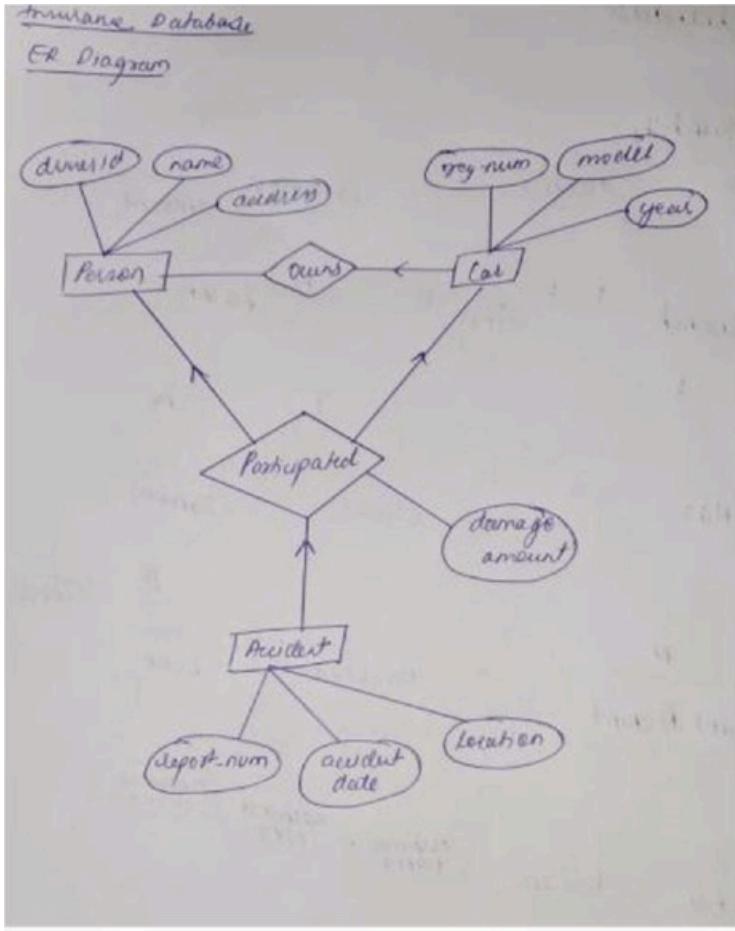
driver and car, and this participation is recorded by linking the driver, the involved car, and the accident

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

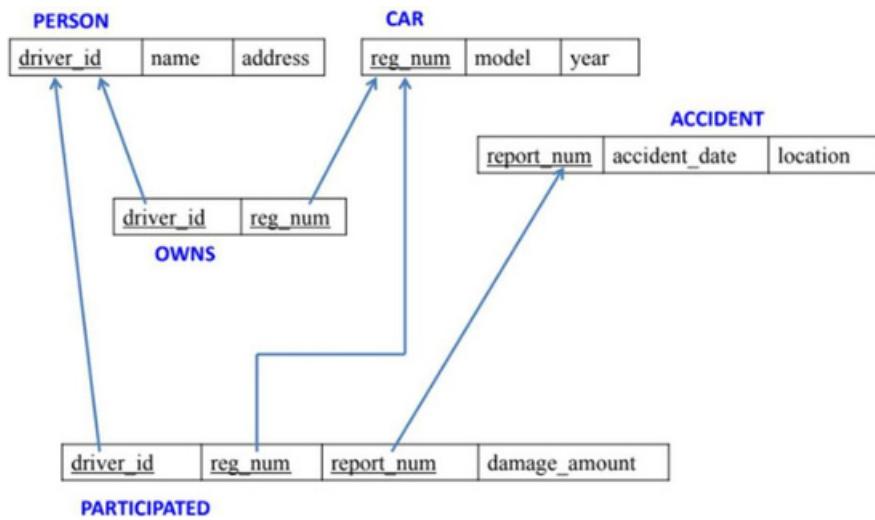
participation entries may repeat the same combination of driver, car, and accident report. The data base must ensure that damage amounts are non-negative, accident dates are valid calendar dates, and car manufacturing years fall within reasonable limits. It must also preserve referential integrity so that ownership or participation entries cannot exist without valid driver, car, and accident information already present in the system. Deletion policies must prevent

removal of drivers or cars that appear in past accident participation records unless historical consistency is preserved through controlled deletion rules or archival mechanisms. The system should maintain accurate links between drivers, cars, and accidents at all times, ensuring reliable retrieval of ownership histories, accident histories, and damage information for administrative, legal, and insurance-related purposes.

Entity Relationship Diagram



Schema Diagram



Create database

- CREATEDATABASE IF NOT EXISTS INSURANCE; SHOW DATABASES;
- USEINSURANCE;

Create table

- CREATE TABLE IF NOT EXISTS PERSON (driver_id VARCHAR(10) PRIMARY KEY, nameVARCHAR(50)NOTNULL,addressVARCHAR(100));
- CREATE TABLE IF NOT EXISTS CAR (reg_num VARCHAR(10) PRIMARY KEY, model VARCHAR(20),year INT);
- CREATE TABLE IF NOT EXISTS ACCIDENT(report_numINTPRIMARYKEY,accident_date DATE, location VARCHAR(50));
- CREATE TABLE IF NOT EXISTS OWNERSHIP (driver_id VARCHAR(10),reg_num VARCHAR(10), PRIMARY KEY (driver_id, reg_num),FOREIGN KEY (driver_id) REFERENCES PERSON(driver_id), FOREIGN KEY (reg_num) REFERENCES CAR(reg_num));
- CREATE TABLE IF NOT EXISTS PARTICIPATED (

driver_id	VARCHAR(10),
reg_num	VARCHAR(10),
report_num	INT,
damage_amount	INT,
PRIMARY KEY (driver_id, reg_num, report_num),	
FOREIGN KEY (driver_id) REFERENCES PERSON(driver_id), FOREIGN KEY (reg_num) REFERENCES CAR(reg_num),	
FOREIGN KEY (report_num) REFERENCES ACCIDENT(report_num));	

Structure of the table

Desc person;

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

desc accident;

	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 intoperson 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  
intoperson values("A05","John", "Hanumanth nagar");  
select* from person;
```

Result Grid	Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
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  
carvalues("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;
```

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

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

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

owns 22 X

```
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;
```

Queries :-

Display Accident date and location

```
66
67
68 •   select accident_date,location from accident;
69
70
```

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			

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

```
59
60 •   update participated set damage_amount=25000 where reg_num="KA053408" and report_num=12;
61
62 •   select * from participated;
63
```

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

Display Accident date and location

```
45 •   select * from accident;
46
47 •   create table participated(driver_id varchar(10), reg_num varchar(10),
48 |   report_num int, damage_amount int, primary key(driver_id, reg_num, report_num),
49 |   foreign key(driver_id) references person(driver_id),
50 |   foreign key(reg_num) references car(reg_num), foreign key(report_num) references accident(report_num));
51
52 •   insert into participated values("A01","KA052250",11,10000);
53 •   insert into participated values("A02","KA053408",12,50000);
54 •   insert into participated values("A03","KA095477",13,25000);
55 •   insert into participated values("A04","KA031181",14,3000);
```

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				
• NULL	NULL	NULL				

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

```
69
70
71 •  select driver_id from participated p, accident a where p.report_num=a.report_num and damage_amount>=25000;
72
73
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	driver_id			
▶	A02			
	A03			

More Queries on Insurance Database

Queries :-

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

```
78  
79  
80 •  select * from car order by year asc;  
81  
82
```

Result Grid			
	reg_num	model	year
▶	KA031181	Lancer	1957
	KA052250	Indica	1990
	KA095477	Toyota	1998
	KA041702	Audi	2005
	KA053408	Honda	2008
*	HULL	HULL	HULL

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

```
82  
83 •  select count(model) CNT from car c, participated p, accident a where c.model="Lancer" and c.reg_num=p.reg_num  
84     and p.report_num=a.report_num;  
85  
86
```

Result Grid	
	CNT
▶	1

Find the Average Damage Amount

```
75  
76  
77 •  select avg(damage_amount) from participated;  
78  
79
```

Result Grid	
	avg(damage_amount)
▶	13600.0000

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

```

86
87 • delete from participated where damage_amount < (select avg_damage from (select avg(damage_amount) as
88     avg_damage from participated)as t);
89
90 •   select * from participated;

```

Result Grid			
driver_id	reg_num	report_num	damage_amount
A02	KA053408	12	25000
A03	KA095477	13	25000
*	HULL	HULL	HULL

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

```

91
92
93 •   select name from person a, participated b where a.driver_id = b.driver_id and damage_amount >
94     (select avg(damage_amount) from participated);
95
-- 

```

Result Grid	
name	
Pradeep	
Smith	

Find Maximum Damage Amount.

```

92
93 •   select name from person a, participated b where a.driver_id = b.driver_id and damage_amount >
94     (select avg(damage_amount) from participated);
95
96
97 •   select max(damage_amount) from participated;

```

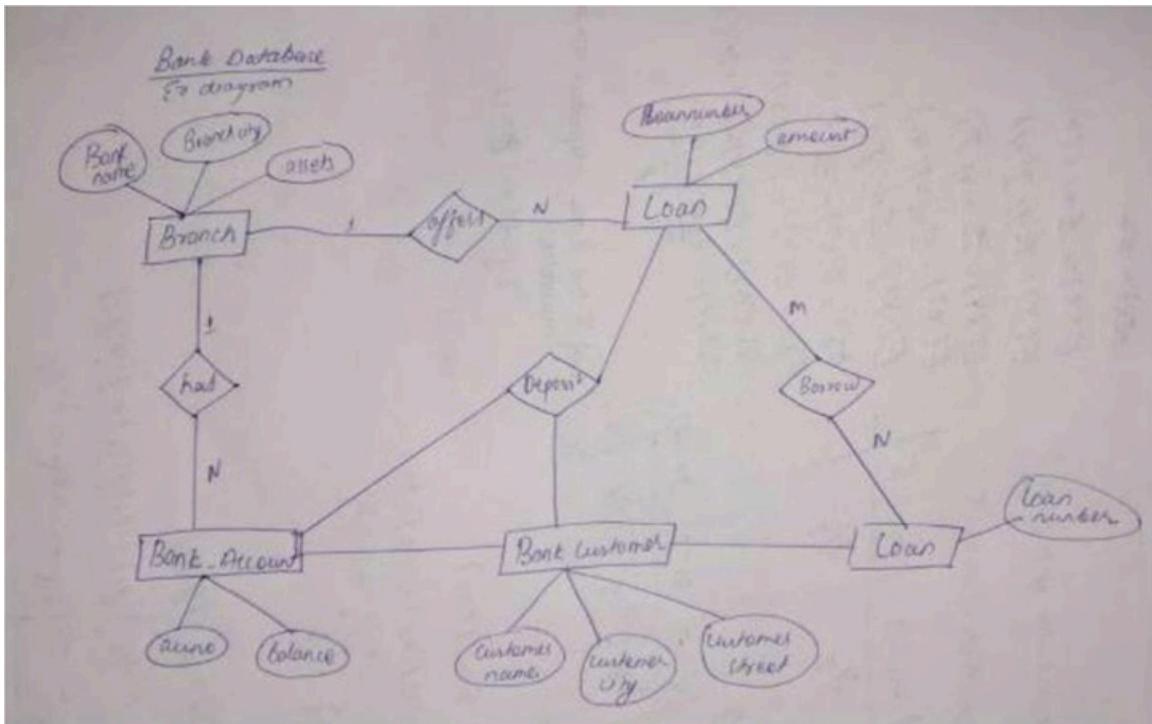
Result Grid	
max(damage_amount)	
25000	

Experiment 2: BANKING DATABASE

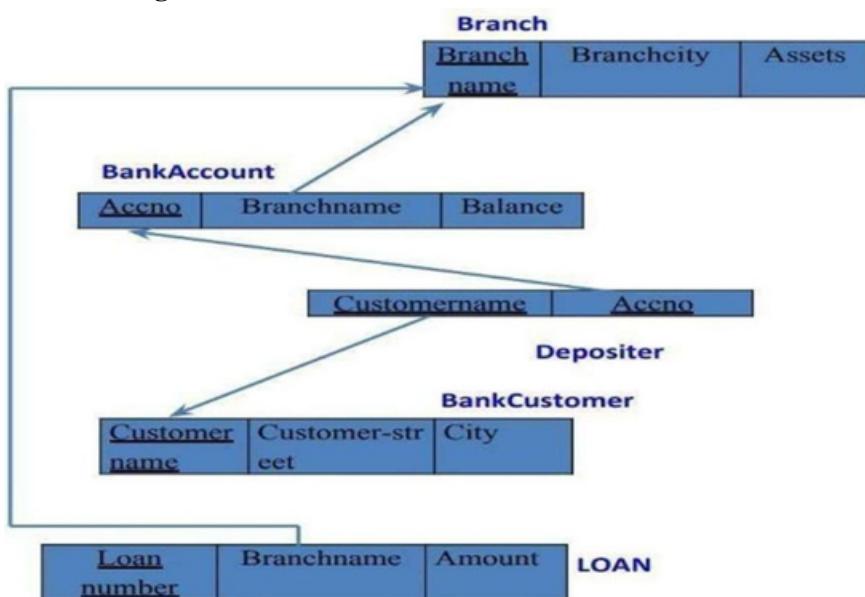
The banking system must store information about branches, bank accounts, customers, deposit relationships, and loans so that branch details (identified by branch name together with city and total assets) are linked to accounts and loans, each account (identified by an account number) records the branch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

Entity Relationship Diagram



Schema Diagram



Create Database

```
create database  
show bank_database;  
usebank_database;
```

Create Table

```
createtable branch( branch_name varchar(50), branch_city  
varchar(50), assets real, Primary key(branch_name)  
);  
create table bankAccount(  
accno int,  
branch_name varchar(50),  
balance real, primary key  
(accno), Foreign key(branch_name) references  
branch(branch_name)  
);  
create table bankCustomer( customer_name varchar(50),  
customer_street varchar(50), customer_city varchar(50) ,  
primary key (customer_name)  
);  
create table depositor( customer_name varchar(50), accno int,  
primary key (customer_name,accno), foreign  
key(customer_name) references  
bankCustomer(customer_name), foreign key(accno) references  
bankAccount(accno)  
);  
desc depositor; create table loan(  
loan_number int,  
branch_name varchar(50),  
amount real,  
primary key(loan_number),  
foreign key(branch_name) references branch(branch_name)  
);
```

Structure of the table

Desc branch;

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

Desc bankaccount;

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

Desc loan;

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

Desc bankcustomer;

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

Desc depositer;

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

Insertion of values into tables

insert into branch

```
values('SBI_chamrajpet','bangalore',50000); insert into
```

```
branch values('SBI_residencyroad','bangalore',10000);
```

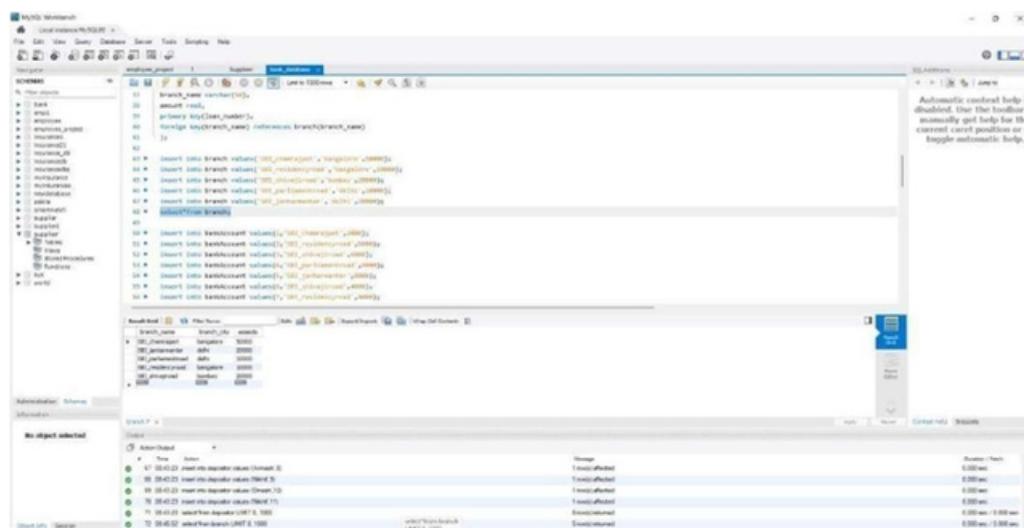
insert into branch

```
values('SBI_shivajiroad','bombay',20000); insert into
```

```
branch values('SBI_parliamentroad','delhi',10000); insert
```

```
into branch values('SBI_jantarmantar','delhi',20000);
```

select*from branch;



insert into bankAccount values(1,'SBI_chamrajpet',2000);

insert into bankAccount

```
values(2,'SBI_residencyroad',5000); insert into
```

```
bankAccount values(3,'SBI_shivajiroad',6000); insert into
```

```
bankAccount values(4,'SBI_parliamentroad',9000); insert
```

```
into bankAccount values(5,'SBI_jantarmantar',8000);
```

```
insert into bankAccount values(6,'SBI_shivajiroad',4000);
```

insert into bankAccount

```
values(7,'SBI_residencyroad',4000); insert into
```

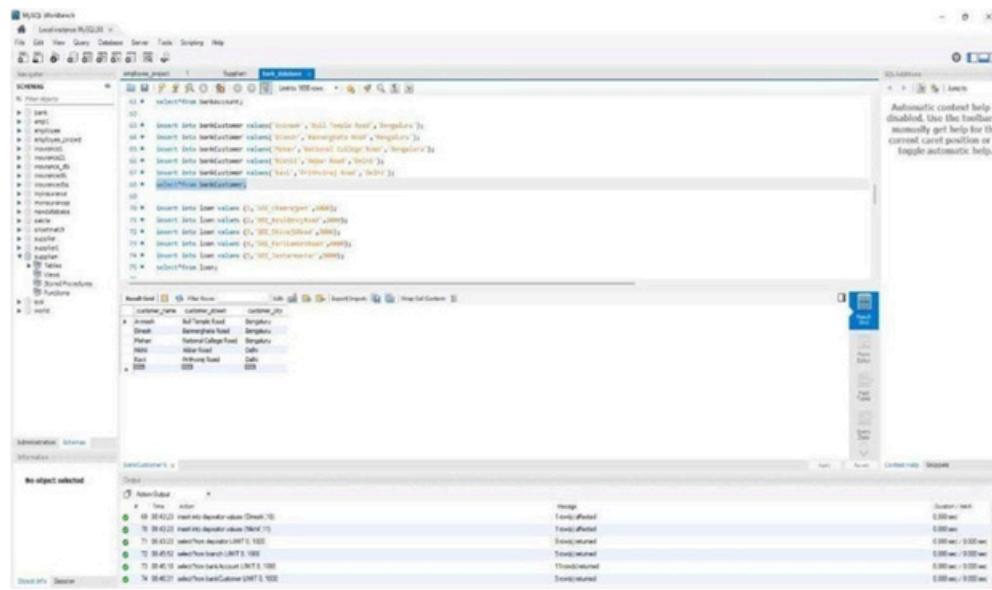
```
bankAccount values(8,'SBI_parliamentroad',3000);
```

```

insert into bankAccount
values(9,'SBI_residencyroad',5000); insert into
bankAccount values(10,'SBI_jantarmantar',2000); insert
into bankAccount values(11,'SBI_jantarmantar',2000);

insert into bankCustomer values('Avinash','Bull Temple
Road','Bengaluru'); insert into bankCustomer
values('Dinesh','Bannerghata Road','Bengaluru'); insert into
bankCustomer values('Mohan','National College Road','Bengaluru');
insert into bankCustomer values('Nikhil','Akbar Road','Delhi');
insert into bankCustomer values('Ravi','Prithviraj Road','Delhi'); select*from bankCustomer;

```



```

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

```

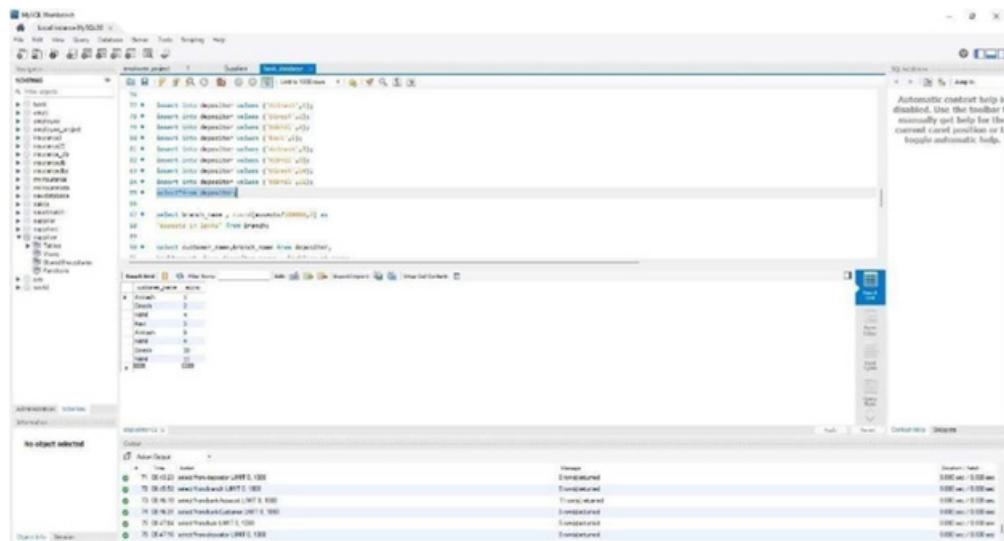
The screenshot shows the MySQL Workbench interface with two panes. The top pane displays the results of five INSERT statements into the 'loan' table, each with a timestamp and a value of 1000, 2000, 3000, 4000, and 5000 respectively. The bottom pane displays the results of a SELECT * statement from the 'loan' table, showing the same five rows.

Timestamp	Value
2023-08-07 10:45:10	1000
2023-08-07 10:45:10	2000
2023-08-07 10:45:10	3000
2023-08-07 10:45:10	4000
2023-08-07 10:45:10	5000

```

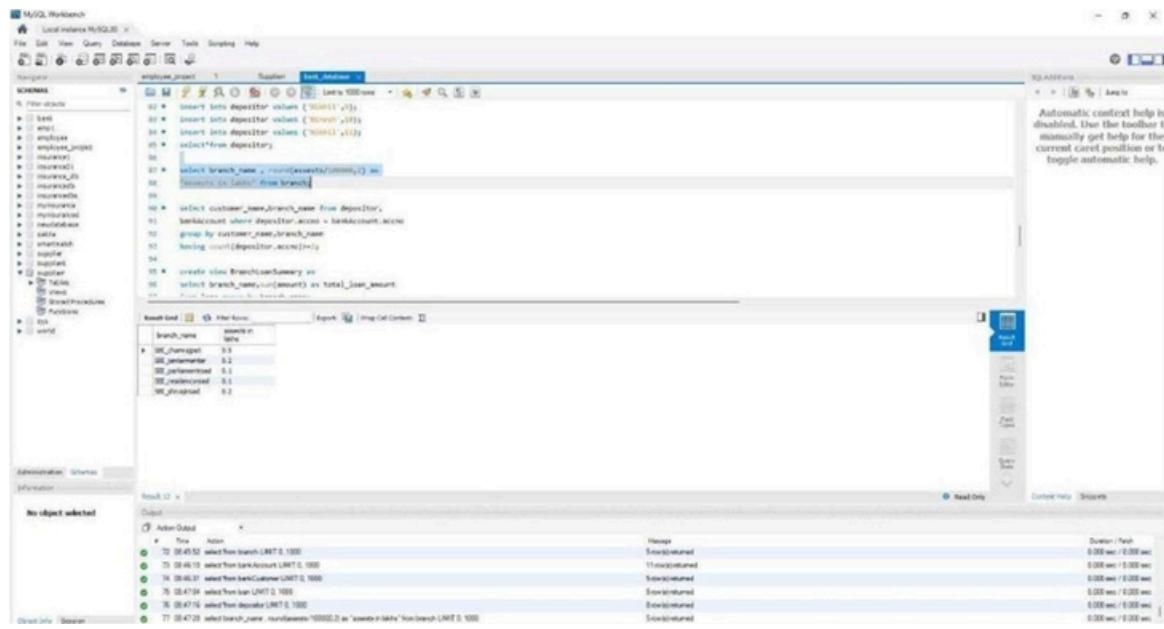
insert into depositor values
('Avinash',1); insert into depositor
values ('Dinesh',2); insert into depositor
values ('Nikhil',4); insert into depositor
values ('Ravi',5);
insert into depositor values ('Avinash',8);
insert into depositor values ('Nikhil',9);
inserts (if Dinesh'db) positoinsert into
depositor values ('Nikhil',11); Select *
from depositor

```



Queries :-

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



```

MySQL Workbench - Local instance MySQL 8.0
File Edit View Query Database Server Tools Scripting Help
Navigation: emploee_proj 1 Supplier Task Database
Result Grid: Filter Rows Export Other Get Context
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid: Filter Rows Export Other Get Context
branch_name assets_in_lakhs
SBI_chennai 3.5
SBI_pune 5.2
SBI_mumbai 4.1
SBI_reddypur 5.1
SBI_gurgaon 5.2

```

The screenshot shows the MySQL Workbench interface with a query editor containing the following SQL code:

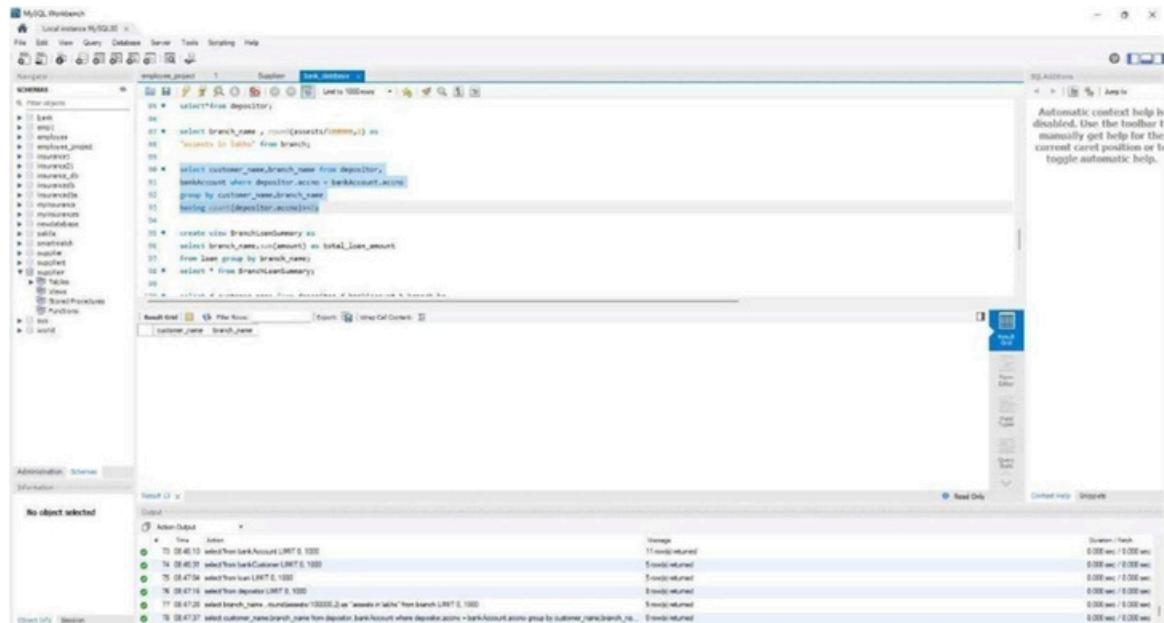
```

-- Insert data into depositor table
01 * insert into depositor values ('W001','x1');
02 * insert into depositor values ('W002','x1');
03 * insert into depositor values ('W003','x1');
04 * select*from depositor;
05 *
06 * select branch_name + 'assets_in_lakhs' as
07 * assets_in_lakhs from branch;
08 *
09 * select customer_name,branch_name from depositor,
10 * bankaccount where depositor.acno = bankaccount.acno
11 * group by customer_name,branch_name
12 * having count(depositor.acno)>1;
13 *
14 * create view BranchAssetsSummary as
15 * select branch_name,(count) as total_loan_amount
16 * from bankaccount;
17 *
18 * select branch_name ,round(count(*)*1000000.0) as "assets in lakhs" from branch LIMIT 0,1000

```

The results grid displays the branch names and their corresponding assets in lakhs.

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



The screenshot shows the MySQL Workbench interface with a query editor containing the same SQL code as the previous screenshot, but with a different result set:

```

MySQL Workbench - Local instance MySQL 8.0
File Edit View Query Database Server Tools Scripting Help
Navigation: emploee_proj 1 Supplier Task Database
Result Grid: Filter Rows Export Other Get Context
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid: Filter Rows Export Other Get Context
customer_name branch_name

```

The results grid displays the customer names and the branches they have accounts in.

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

The screenshot shows the MySQL Workbench interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, Help. The left sidebar shows the 'Schemas' tree with 'employees_project' selected, containing tables like bank, branch, employee, and loan. The main area has tabs for 'employees_project', 'Supplier', and 'test_database'. A large query editor window displays a complex SELECT statement:

```
10: bankaccount where depositor.acco = bankaccount.acco
11: group by customer_name,branch_name
12: Having count(depositor.acco)>1
13:
14: ┌─ create view BranchLoanSummary as
15: ┌─ select branch_name, count(acco) as Total_Loan_Amount
16: ┌─ from loan group by branch_name
17: ┌─ select * from BranchLoanSummary
18:
19: ┌─ select A.customer_name from depositor d,bankaccount b,branch br
20: ┌─ where d.acco=d.depositor_acco
21: ┌─ and b.Branch_Number=branch_name
22: ┌─ and br.Branch_City='Orissa'
23: ┌─ and d.acco=b.acco
24: ┌─ Having count(d.acco)=(select br.branch_name)-(select count(*) from
25: ┌─ loan where branch_name=br.branch_name)
```

The results of this query are shown in a table below:

branch_name	Total_Loan_Amount
BR_Jharsuguda	5000
BR_Bhubaneswar	5000
BR_Puri	5000
BR_Rasalguda	5000
BR_Durgapada	5000

At the bottom, the 'Information' tab is open, showing 'No object selected'. The status bar at the bottom right indicates 'Read Only' and 'Context Help'.

Experiment 3: More Queries on Bank Database

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

```

USE employee_project;
CREATE VIEW branchCustomerSummary AS
SELECT branch_name, COUNT(*) AS total_branches
FROM branch
GROUP BY branch_name;
SELECT d.customer_name FROM depositor d
JOIN account a ON d.depositor_id = a.depositor_id
JOIN branch b ON a.branch_id = b.branch_id
WHERE d.customer_name IN (
    SELECT customer_name FROM branchCustomer
    WHERE branch_name = 'Delhi'
);
SELECT customer_name FROM bankCustomer
WHERE customer_name NOT IN (SELECT customer_name FROM depositor);
  
```

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

```

USE employee_project;
CREATE VIEW branchCustomerSummary AS
SELECT branch_name, COUNT(*) AS total_branches
FROM branch
GROUP BY branch_name;
SELECT d.customer_name FROM depositor d
JOIN account a ON d.depositor_id = a.depositor_id
JOIN branch b ON a.branch_id = b.branch_id
WHERE d.customer_name IN (
    SELECT customer_name FROM branchCustomer
    WHERE branch_name = 'Delhi'
);
SELECT customer_name FROM bankCustomer
WHERE customer_name NOT IN (SELECT customer_name FROM depositor);
  
```

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

```

USE `bank`;
CREATE VIEW BranchCustomerSummary AS
SELECT branch_name,(count(*)) AS total_loan_account
FROM loan GROUP BY branch_name;
SELECT * FROM BranchCustomerSummary;

SELECT d.customer_name FROM depositor d, bankAccount b, branch br
WHERE d.account_no = b.account_no
AND br.branch_name = b.branch_no
AND br.branch_city = 'Bengaluru'
GROUP BY d.customer_name
HAVING COUNT(DISTINCT br.branch_name) = (SELECT COUNT(*) FROM branch WHERE branch_city = 'Bengaluru');

SELECT customer_name FROM bankCustomer WHERE
customer_name NOT IN (SELECT customer_name FROM depositor);

SELECT DISTINCT d.customer_name FROM depositor d, bankAccount b,
(SELECT l FROM loan WHERE account_no = d.account_no AND
branch_name = b.branch_no) AS l,
(SELECT branch_name FROM branch WHERE branch_city = 'Bengaluru') AS b
WHERE b.branch_name = l.branch_name AND
l.branch_name IN (SELECT branch_name FROM branch WHERE branch_city = 'Bengaluru');

SELECT branch_name FROM branch WHERE assets > ALL (SELECT assets
FROM branch WHERE branch_city = 'Bengaluru');

DELETE FROM bankCustomer WHERE branch_name IN (
SELECT branch_name FROM branch WHERE branch_city = 'Bengaluru');

UPDATE bankAccount
SET balance = balance * .05;

SET @branch_name = 'Bengaluru';
SET @branch_id = (SELECT branch_id FROM branch WHERE branch_name = @branch_name);
SET @customer_name = (SELECT customer_name FROM bankCustomer WHERE branch_name = @branch_name);
SET @account_no = (SELECT account_no FROM depositor WHERE customer_name = @customer_name AND branch_id = @branch_id);
SET @loan_id = (SELECT loan_id FROM loan WHERE account_no = @account_no AND branch_name = @branch_name);

INSERT INTO transaction (transaction_id, amount, date, branch_id, account_no)
VALUES (1, 1000, '2018-01-01', @branch_id, @account_no);

SELECT * FROM transaction;

```

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

```

USE `bank`;
CREATE VIEW BranchCustomerSummary AS
SELECT branch_name,(count(*)) AS total_loan_account
FROM loan GROUP BY branch_name;
SELECT * FROM BranchCustomerSummary;

SELECT customer_name FROM bankCustomer WHERE
customer_name NOT IN (SELECT customer_name FROM depositor);

SELECT DISTINCT d.customer_name FROM depositor d, bankAccount b,
(SELECT l FROM loan WHERE account_no = d.account_no AND
branch_name = b.branch_no) AS l,
(SELECT branch_name FROM branch WHERE branch_city = 'Mumbai') AS b
WHERE b.branch_name = l.branch_name AND
l.branch_name IN (SELECT branch_name FROM branch WHERE branch_city = 'Mumbai');

SELECT branch_name FROM branch WHERE assets > ALL (SELECT assets
FROM branch WHERE branch_city = 'Mumbai');

DELETE FROM bankCustomer WHERE branch_name IN (
SELECT branch_name FROM branch WHERE branch_city = 'Mumbai');

UPDATE bankAccount
SET balance = balance * .05;

SET @branch_name = 'Mumbai';
SET @branch_id = (SELECT branch_id FROM branch WHERE branch_name = @branch_name);
SET @customer_name = (SELECT customer_name FROM bankCustomer WHERE branch_name = @branch_name);
SET @account_no = (SELECT account_no FROM depositor WHERE customer_name = @customer_name AND branch_id = @branch_id);
SET @loan_id = (SELECT loan_id FROM loan WHERE account_no = @account_no AND branch_name = @branch_name);

INSERT INTO transaction (transaction_id, amount, date, branch_id, account_no)
VALUES (1, 1000, '2018-01-01', @branch_id, @account_no);

SELECT * FROM transaction;

```

Experiment 5: Employee Database

The employee database must record each employee's identifying number, name, manager reference, hire date, salary, and department affiliation while also tracking departmental details, project

assignments (including the role an employee plays on a project), and any incentive payments given to employees. Every employee is represented by a unique employee number and has a hire date and

salary that must be valid; the manager field is a self-referencing link that must, if present, point to an existing employee and must never create a circular management chain or reference the employee

themselves. 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

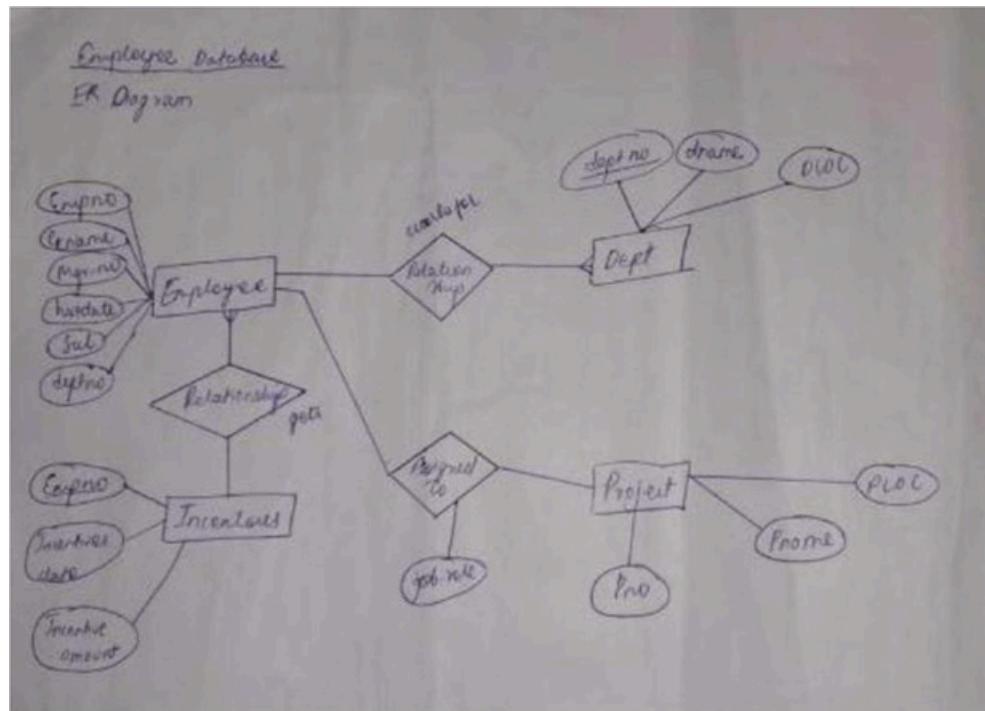
assignment and incentive records cannot exist without corresponding employee, project, or department records as appropriate. Salary, incentive amounts, and any monetary fields must be constrained to valid numeric ranges and hire/ incentive dates must be valid calendar dates (and typically not future-dated 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 themselves, disallowing duplicate project-assignments, requiring that incentive dates fall within the employee's employment window, and optionally requiring at least one project assignment or at least one incentive record depending on policy for reporting. Implementation should use primary-key and foreign-key constraints for identity and linkage, unique constraints to

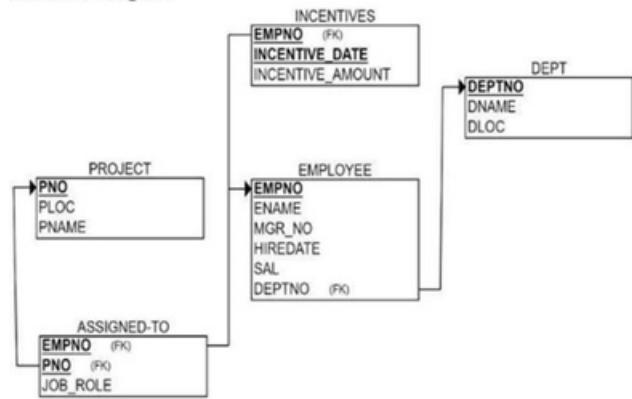
prevent duplicate assignments, check constraints for monetary and date ranges, and application logic or triggers for complex temporal or graph constraints (like cycle detection in management relationships and enforcing non-overlap or other schedule-related rules if assignments gain temporal attributes later). The system must therefore reliably support queries such as employee reporting lines, department staffing lists, project rosters with job roles, incentive payment histories, salary analyses, and audit reports while maintaining data integrity, preventing inconsistent deletions, and preserving a complete historical record for HR and compliance needs.

Entity Relationship Diagram



Schema Diagram

Schema Diagram



Create Database

```
create database Employee_project;
```

```
show databases;
```

```
useemployee_project;
```

Create table

```
CREATE TABLE DEPT (
DEPTNO INT PRIMARY KEY,
DNAME VARCHAR(30) NOT NULL,
DLOC VARCHAR(30) NOT NULL
);
```

```
descdept;
```

```
CREATE TABLE EMPLOYEE (
EMPNO INT PRIMARY KEY,
ENAME VARCHAR(30) NOT NULL,
MGR_NO INT,
HIREDATE DATE,
SAL DECIMAL(10,2),
DEPTNO INT,
FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)
);
descemployee;
```

```
CREATE TABLE PROJECT (
PNO INT PRIMARY KEY,
PLOC VARCHAR(30) NOT NULL,
PNAME VARCHAR(30) NOT NULL
);
```

```
descproject;
```

```
CREATE TABLE ASSIGNED_TO (
EMPNO INT,
PNO INT,
JOB_ROLE VARCHAR(30),
PRIMARY KEY (EMPNO, PNO),
FOREIGN KEY (EMPNO) REFERENCES EMPLOYEE(EMPNO),
FOREIGN KEY (PNO) REFERENCES PROJECT(PNO)
);
desc assigned_to;
```

```

CREATE TABLE INCENTIVES (
    EMPNO INT,
    INCENTIVE_DATE DATE,
    INCENTIVE_AMOUNT DECIMAL(10,2),
    PRIMARY KEY (EMPNO, INCENTIVE_DATE),
    FOREIGN KEY (EMPNO) REFERENCES EMPLOYEE(EMPNO );

```

Structure of the table desc dept;
desc employee;

	Field	Type	Null	Key	Default	Extra
▶	empno	int	NO	PRI	NULL	
	ename	varchar(30)	YES		NULL	
	mgr_no	int	YES		NULL	
	hiredate	date	YES		NULL	
	sal	decimal(10,2)	YES		NULL	
	deptno	int	YES	MUL	NULL	

desc project;

	Field	Type	Null	Key	Default	Extra
▶	pno	int	NO	PRI	NULL	
	pname	varchar(30)	YES		NULL	
	dloc	varchar(30)	YES		NULL	

desc assigned;

	Field	Type	Null	Key	Default	Extra
▶	empno	int	NO	PRI	NULL	
	pno	int	NO	PRI	NULL	
	job_role	varchar(30)	YES		NULL	

desc incentives;

	Field	Type	Null	Key	Default	Extra
▶	empno	int	YES	MUL	NULL	
	incentives_date	date	YES		NULL	
	incentives_amount	decimal(10,2)	YES		NULL	

Insertion of values into Table

INSERTINTODEPTVALUES

```
(10, 'Sales','Bengaluru'),  
(20, 'Accounting', 'Hyderabad'),  
(30, 'Research', 'Mysuru'),  
(40, 'Operations', 'Chennai'),  
(50, 'HR', 'Mumbai'),  
(60, 'IT', 'Delhi');  
select * from dept;
```

DEPTNO	DNAME	LOC
10	Sales	Bengaluru
20	Accounting	Hyderabad
30	Research	Mysuru
40	Operations	Chennai
50	HR	Mumbai
60	IT	Delhi
70	Marketing	Chennai

INSERT INTO EMPLOYEE VALUES

```
(1001, 'Alice', NULL, '2019-02-10', 75000, 10),  
(1002, 'Bob', 1001, '2020-06-01', 65000, 20),  
(1003, 'Charlie', 1001, '2018-09-15', 80000, 30),  
(1004, 'Diana', 1002, '2021-01-20', 55000, 40),  
(1005, 'Ethan', 1003, '2022-07-12', 60000, 50),  
(1006, 'Fay', 1001, '2023-03-05', 52000, 10);  
Select * from employee;
```

EMPNO	ENAME	JOB	HIREDATE	Sal	Deptno
1001	Alice	Analyst	2019-02-10	75000	10
1002	Bob	Analyst	2020-06-01	65000	20
1003	Charlie	Analyst	2018-09-15	80000	30
1004	Diana	Analyst	2021-01-20	55000	40
1005	Ethan	Analyst	2022-07-12	60000	50
1006	Fay	Analyst	2023-03-05	52000	10

```

INSERT INTO PROJECT VALUES
(200, 'Bengaluru', 'Alpha'), (201,
'Hyderabad', 'Beta'), (202,
'Mysuru', 'Gamma'), (203,
'Chennai', 'Delta'),
'Mumbai','Epsilon'), (205,
'Pune','Zeta'); select * from project;

```

PROJ_ID	PLC	NAME
200	Bengaluru	Alpha
201	Hyderabad	Beta
202	Mysuru	Gamma
203	Chennai	Delta
204	Mumbai	Epsilon
205	Pune	Zeta

```

INSERT INTO ASSIGNED_TO
VALUES (1001, 200, 'Lead'),
(1002, 201, 'Analyst'),
(1003, 202, 'Senior'),
(1004, 203, 'Support'),
(1005, 204, 'Recruiter'),
(1006, 200, 'Developer'),
(1002, 203, 'Tester'),
(1003, 200, 'Consultant');
select * from
assigned_to;

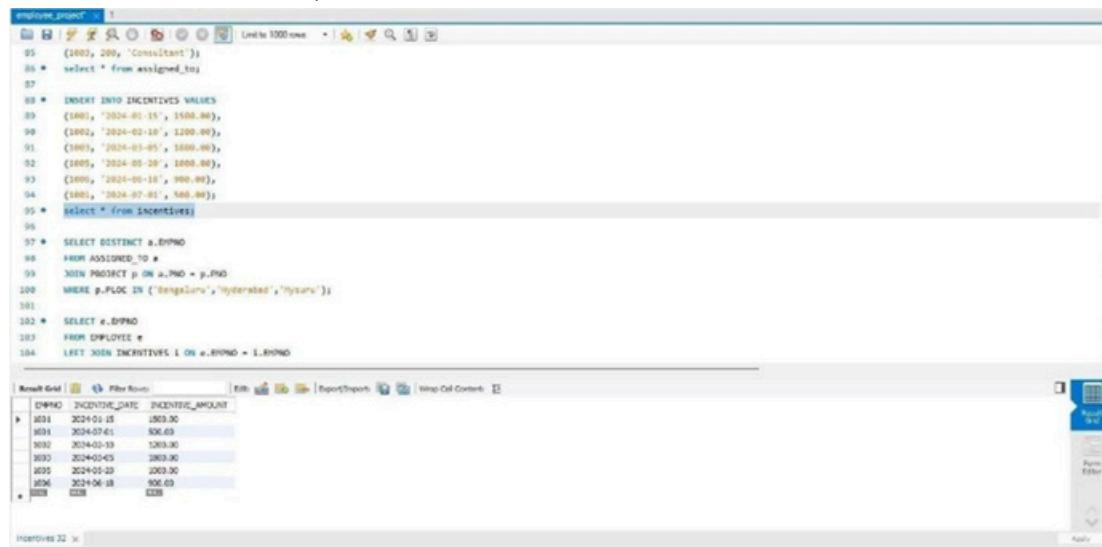
```

ASSIGN_ID	EMP_ID	PROJ_ID
1001	1001	200
1002	1002	201
1003	1003	202
1004	1004	203
1005	1005	204
1006	1006	200
1007	1002	203
1008	1003	200

```

INSERT INTO INCENTIVES VALUES
(1001, '2024-01-15', 1500.00),
(1002, '2024-02-10', 1200.00), (1003,
'2024-03-05', 1800.00),
(1005, '2024-05-20', 1000.00),
(1006, '2024-06-18', 900.00),
(1001,'2024-07-01', 500.00);
select * from incentives;

```



The screenshot shows the Oracle SQL Developer interface. At the top, there is a script editor window titled 'employee_project' containing the provided SQL code. Below it is a results window titled 'Incentives 32' displaying the data from the 'INCENTIVES' table.

INCENTIVE_ID	INCENTIVE_DATE	INCENTIVE_AMOUNT
1001	2024-01-15	1500.00
1001	2024-07-01	500.00
1002	2024-02-10	1200.00
1003	2024-03-05	1800.00
1005	2024-05-20	1000.00
1006	2024-06-18	900.00
1008	2024-07-01	500.00

Queries

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

The screenshot shows a database interface with a SQL query editor and a results grid. The query retrieves distinct employee numbers (EPMNO) from the assigned_to table, joined with the project table (p) on EPMNO = PRO. It filters the results by project location (PLOC) being in ('Bengaluru', 'Hyderabad', 'Mysuru'). The results grid displays four rows of employee numbers: 1001, 1003, 1006, and 1002.

```
employee_project > 1
86 * select * from assigned_to;
87
88 * INSERT INTO INCENTIVES VALUES
89 {1001, '2024-01-15', 1500.00},
90 {1002, '2024-02-10', 1200.00},
91 {1003, '2024-03-05', 1800.00},
92 {1005, '2024-05-20', 1000.00},
93 {1006, '2024-06-10', 900.00},
94 {1001, '2024-07-01', 500.00};
95 * select * from incentives;
96
97 * SELECT DISTINCT a.EPMNO
98 FROM ASSIGNED_TO a
99 JOIN PROJECT p ON a.PRO = p.PRO
100 WHERE p.PLOC IN ('Bengaluru', 'Hyderabad', 'Mysuru');
101
102 * SELECT e.EPMNO
103 FROM EMPLOYEE e
104 LEFT JOIN INCENTIVES i ON e.EPMNO = i.EPMNO
105 WHERE i.EPMNO IS NULL;
```

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

EPMNO
1001
1003
1006
1002

Result 33 < [Close]

Read Only

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

The screenshot shows a database interface with a SQL query editor and a results grid. The query retrieves distinct employee numbers (EPMNO) from the assigned_to table, joined with the project table (p) on EPMNO = PRO. It filters the results by project location (PLOC) being in ('Bengaluru', 'Hyderabad', 'Mysuru'). The results grid displays one row of an employee number: 1004.

```
employee_project > 1
86 * select * from assigned_to;
87
88 * INSERT INTO INCENTIVES VALUES
89 {1001, '2024-01-15', 1500.00},
90 {1002, '2024-02-10', 1200.00},
91 {1003, '2024-03-05', 1800.00},
92 {1005, '2024-05-20', 1000.00},
93 {1006, '2024-06-10', 900.00},
94 {1001, '2024-07-01', 500.00};
95 * select * from incentives;
96
97 * SELECT DISTINCT a.EPMNO
98 FROM ASSIGNED_TO a
99 JOIN PROJECT p ON a.PRO = p.PRO
100 WHERE p.PLOC IN ('Bengaluru', 'Hyderabad', 'Mysuru');
101
102 * SELECT e.EPMNO
103 FROM EMPLOYEE e
104 LEFT JOIN INCENTIVES i ON e.EPMNO = i.EPMNO
105 WHERE i.EPMNO IS NULL;
```

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

EPMNO
1004

Result 34 < [Close]

Read Only

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.

The screenshot shows a MySQL Workbench window titled "employees_project". The SQL editor contains the following query:

```

184 {1001, '2024-07-01', 500.00}
185 *
186 select * from incentives;
187 *
188 SELECT DISTINCT e.EMPNO
189 FROM ASSIGNED_TO a
190 JOIN PROJECT p ON a.PRO = p.PRO
191 WHERE p.PLOC IN ('Bengaluru', 'Hyderabad', 'Mysuru');
192 *
193 SELECT e.EMPNO,
194       e.ENAME, e.EMPNO, e.DEPTNO, e.JOB_ROLE,
195       d.DLOC AS DEPT_LOCATION, p.PLOC AS PROJECT_LOCATION
196 FROM EMPLOYEE e
197 JOIN DEPT d ON e.DEPTNO = d.DEPTNO
198 JOIN ASSIGNED_TO a ON e.EMPNO = a.EMPNO
199 JOIN PROJECT p ON a.PRO = p.PRO
200 WHERE d.DLOC = p.PLOC;

```

The results grid displays the following data:

EMPNO	ENAME	DEPTNO	JOB_ROLE	DEPT_LOCATION	PROJECT_LOCATION
1001	Alice	10	Lead	Bengaluru	Bengaluru
1002	Bob	20	Analyst	Hyderabad	Hyderabad
1003	Charlie	30	Senior	Mysuru	Mysuru
1004	Diana	40	Support	Chennai	Chennai
1005	Ethan	50	Recruiter	Mumbai	Mumbai
1006	Fay	10	Developer	Bengaluru	Bengaluru

Experiment 6: More Queries on Employee Database

List the name of the managers with the maximum employees.

The screenshot shows the MySQL Workbench interface with the following details:

- File Menu:** File, Edit, View, Query, Database, Server, Tools, Sourcing, Help.
- Schema Tree:** Shows the database structure with the `employees_project` schema selected.
- Query Editor:** Contains a complex SQL script for managing employee projects. The script includes joins between `EMPLOYEE`, `DEPT`, `DEPTINFO`, `PROJECT`, and `EMPLOYEE_PROJECT` tables, along with various aggregate functions and subqueries to calculate manager names and project counts.
- Result Grid 1:** Displays the output of the first part of the query, showing columns `manager_name` and `num_employees`. One row is shown with a value of 3.
- Result Grid 2:** Displays the output of the second part of the query, showing a single row with a value of 1.
- Log:** Shows the execution log with 7 entries, detailing the execution time (e.g., 2.0824.55), the action (e.g., "show databases"), the message (e.g., "2 databases returned"), and the duration (e.g., 0.000 sec / 0.000 sec).

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

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

The screenshot shows the MySQL Workbench interface with a query editor containing a multi-step SQL script. The script uses common table expressions (CTEs) to calculate manager counts and then joins these with employee data to find second-level managers. It also includes a subquery to find employees with missing manager numbers and a final join to incentives data.

```
111 * select e.Pname as Manager_Name
112 from employee e
113 where e.EmpNo in (select distinct Mgr_No from employee where Mgr_No is not null);
114 *
115 select e.Ename
116 from employee e
117 where e.Mgr_No = e.EmpNo;
118 *
119 *
120 * select e.Ename as second_level_manager,d.Dname as department
121 from employee e
122 join dept d on e.deptno = d.deptno
123 where e.Mgr_No is (
124 select EmpNo from employee where Mgr_No is null);
125 *
126 *
127 * select empno,incentive_date,incentive_amount
128 from incentives
129 where year(incentive_date)=2010 and month(incentive_date)=1
130 order by incentive_amount desc;
```

The results of the first query are displayed in a table:

second_level_manager	department
Var	Sales
Bob	Accounting
Charlie	Research

The results of the second query are displayed in a table:

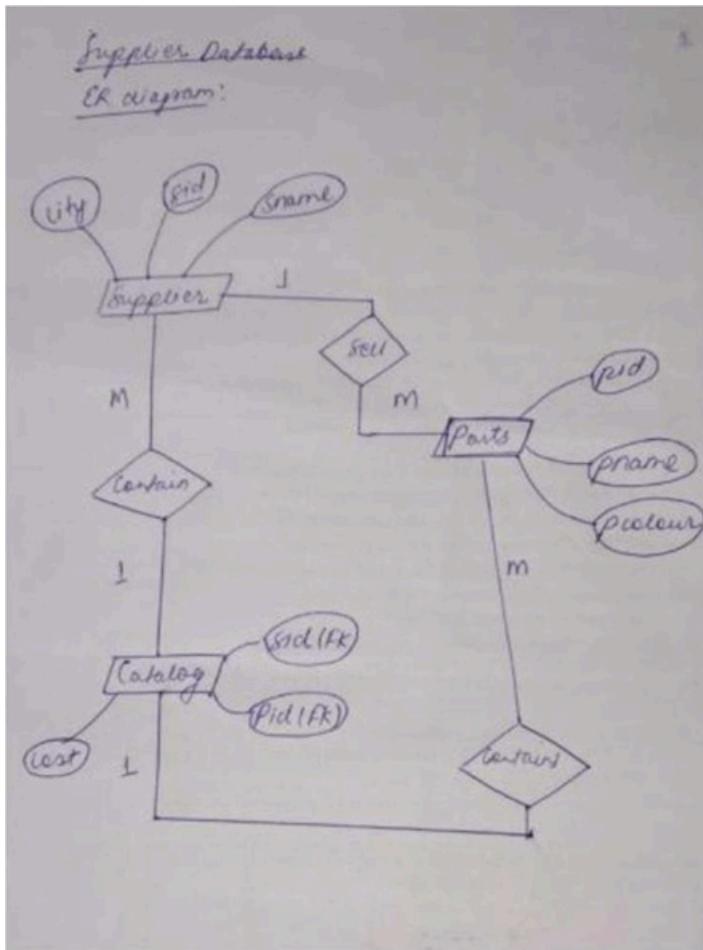
Action	Time	Action	Time
11 00:33:10	select e.Ename as second_level_manager,d.Dname as department from employee e join dept d on e.deptno = d.deptno where e.Mgr_No is not null;	11 00:33:10	InnoDB returned
12 00:36:01	select empno,incentive_amount,Incentive_date from incentives where year(incentive_date)=2010 and month(incentive_date)=1 order by incentive_amount desc;	12 00:36:01	InnoDB returned
13 00:44:05	select e.Ename as employee_name from employee e join employee e1 on e.Mgr_No = e1.EmpNo;	13 00:44:05	Error Code: 1054 Unknown column 'InName' in 'field list'
14 00:48:27	select e.Ename as employee_name from employee e join employee e1 on e.Mgr_No = e1.EmpNo;	14 00:48:27	InnoDB returned
15 00:48:36	select empno,incentive_amount,Incentive_date from incentives where year(incentive_date)=2010 and month(incentive_date)=1 order by incentive_amount desc;	15 00:48:36	InnoDB returned
16 00:49:41	select e.Ename as second_level_manager,d.Dname as department from employee e join dept d on e.deptno = d.deptno where e.Mgr_No is not null;	16 00:49:41	InnoDB returned

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

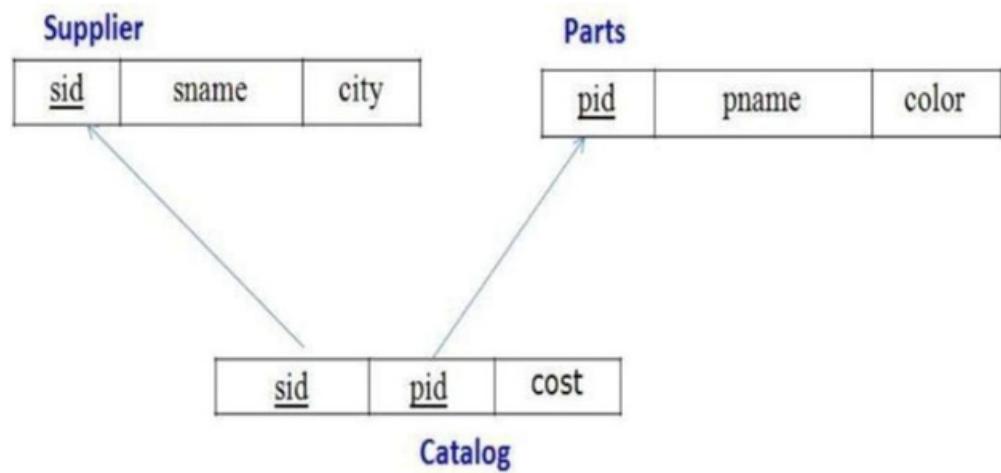
Experiment 7: Supplier Database

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

```
createdatabaseif not exists Supplierr ; use supplierr;
```

Create Table

```
CREATE TABLE Supplier      city VARCHAR(50)
( sid INT PRIMARY KEY,
  sname VARCHAR(50),
);
CREATE TABLE Parts
( pid INT PRIMARY
KEY, pname
VARCHAR(50),
color VARCHAR(20)
);
CREATE TABLE Catalog (
  sid INT,
  pid INT,
  cost INT,
  PRIMARY KEY (sid, pid),
  FOREIGN KEY (sid) REFERENCES Supplier(sid),
  FOREIGN KEY (pid) REFERENCES Parts(pid)
);
desc supplier;
```

Result Grid						
	Field	Type	Null	Key	Default	Extra
▶	sid	int	NO	PRI	NULL	
	sname	varchar(40)	YES		NULL	
	city	varchar(40)	YES		NULL	

```
desc parts;
```

Result Grid						
	Field	Type	Null	Key	Default	Extra
▶	pid	int	NO	PRI	NULL	
	pname	varchar(40)	YES		NULL	
	colour	varchar(20)	YES		NULL	

desc catalog;

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

Insertion of values into the table

```
INSERT INTO Supplier VALUES  
(10001, 'Acme Widget', 'Bangalore'),  
(10002, 'Johns', 'Kolkata'),  
(10003, 'Vimal', 'Mumbai'),  
(10004, 'Reliance', 'Delhi');
```

The screenshot shows the MySQL Workbench interface. The left sidebar displays the database structure for 'supplier'. The main area contains a query editor window with the following SQL script:

```
use supplier;
insert into parts values
(1, 'CPU', 100, 'Intel', 'High', 1),
(2, 'GPU', 200, 'NVIDIA', 'High', 1),
(3, 'SSD', 50, 'Western Digital', 'Medium', 1),
(4, 'Motherboard', 150, 'ASUS', 'Medium', 1),
(5, 'RAM', 30, 'Corsair', 'Medium', 1),
(6, 'Power Supply', 80, 'Corsair', 'Medium', 1),
(7, 'Case', 120, 'NZXT', 'Medium', 1),
(8, 'CPU Cooler', 20, 'Noctua', 'Low', 1),
(9, 'GPU Cooler', 40, 'MSI', 'Low', 1),
(10, 'SSD Cooler', 10, 'Western Digital', 'Low', 1),
(11, 'Motherboard Cooler', 15, 'ASUS', 'Low', 1),
(12, 'RAM Cooler', 5, 'Corsair', 'Low', 1),
(13, 'Power Supply Cooler', 10, 'Corsair', 'Low', 1),
(14, 'Case Cooler', 20, 'NZXT', 'Low', 1);

insert into supplier values
(1, 'John', 'Doe', 'Supplier 1'),
(2, 'Jane', 'Doe', 'Supplier 2'),
(3, 'Vimal', 'Hariharan', 'Supplier 3'),
(4, 'Kiran', 'Kumar', 'Supplier 4');

select * from parts;
select * from supplier;
```

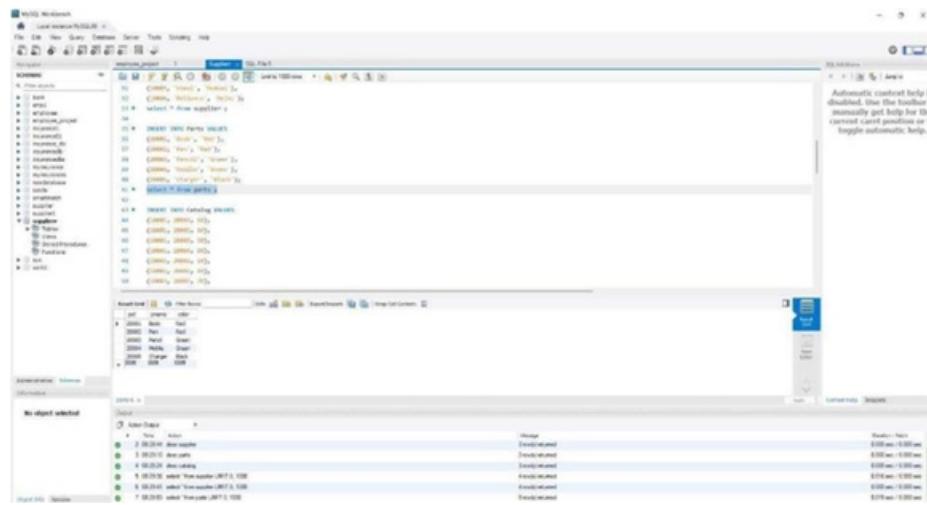
The results grid below the query editor shows the data inserted into the 'parts' and 'supplier' tables:

ID	Name	Unit Price	Supplier ID	Category	Rating
1	CPU	100	1	Processor	High
2	GPU	200	1	Processor	High
3	SSD	50	1	Storage	Medium
4	Motherboard	150	1	Motherboard	Medium
5	RAM	30	1	RAM	Medium
6	Power Supply	80	1	Power Supply	Medium
7	Case	120	1	Case	Medium
8	CPU Cooler	20	1	Cooler	Low
9	GPU Cooler	40	1	Cooler	Low
10	SSD Cooler	10	1	Cooler	Low
11	Motherboard Cooler	15	1	Cooler	Low
12	RAM Cooler	5	1	Cooler	Low
13	Power Supply Cooler	10	1	Cooler	Low
14	Case Cooler	20	1	Cooler	Low

The bottom status bar indicates 'Duration: 1ms'.

INSERT INTO Parts VALUES

```
(20001, 'Book', 'Red'),  
(20002, 'Pen', 'Red'),  
(20003, 'Pencil', 'Green'),  
(20004, 'Mobile', 'Green'),  
(20005, 'Charger', 'Black');
```



Queries

Find the pnames of parts for which there is some supplier

Find the names of suppliers who supply every part

The screenshot shows the MySQL Workbench interface with the following details:

- Top Bar:** MySQL Workbench, Local instance MySQL 8.0, File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Schemas:** employee_project
- SQL Editor:** Contains a complex multi-step query for an employee project. The steps include:
 - Step 1: SELECT e.employee FROM Suppliers x WHERE NOT EXISTS (SELECT p.pid FROM Parts p WHERE p.colour = 'Red' AND p.pid NOT IN (SELECT c.pid FROM Catalog c WHERE c.pid = x.pid))
 - Step 2: SELECT e.employee FROM Suppliers x WHERE NOT EXISTS (SELECT p.pid FROM Parts p WHERE p.colour = 'Red' AND p.pid NOT IN (SELECT c.pid FROM Catalog c WHERE c.pid = x.pid))
 - Step 3: SELECT e.employee FROM Suppliers x WHERE NOT EXISTS (SELECT p.pid FROM Parts p WHERE p.colour = 'Red' AND p.pid NOT IN (SELECT c.pid FROM Catalog c WHERE c.pid = x.pid))
- Results Tab:** Shows the results of the query, returning 0 rows.
- Execution Plan:** Displays the execution plan for the query, showing 26 steps. The first step is a SELECT from Suppliers x. Subsequent steps involve joins between Suppliers, Catalog, and Parts tables based on pid and colour.

Find the snames of suppliers who supply every red part.

```

MySQL Workbench - Local instance MySQL80 [1]
File Edit View Query Database Server Tools Scripting Help
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 389 390 391 392 393 394 395 396 397 398 399 399 400

```

The screenshot shows the MySQL Workbench interface with a query editor containing the following SQL code:

```

SELECT c.sname
FROM Catalog c
WHERE c.cid = s.cid
;

SELECT s.sname
FROM Supplier s
WHERE NOT EXISTS (
    SELECT p.pid
    FROM Parts p
    WHERE p.color = 'red'
    AND p.pid NOT IN (
        SELECT c.pid
        FROM Catalog c
        WHERE c.cid = s.cid
    )
)
;
SELECT p.pname

```

The results pane shows a single row: "Acme Widget".

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

```

MySQL Workbench - Local instance MySQL80 [1]
File Edit View Query Database Server Tools Scripting Help
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 109 110 111 112 113 114 115 116 117 118 119 119 120 121 122 123 124 125 126 127 128 129 129 130 131 132 133 134 135 136 137 138 139 139 140 141 142 143 144 145 146 147 148 149 149 150 151 152 153 154 155 156 157 158 159 159 160 161 162 163 164 165 166 167 168 169 169 170 171 172 173 174 175 176 177 178 179 179 180 181 182 183 184 185 186 187 188 189 189 190 191 192 193 194 195 196 197 198 199 199 200 201 202 203 204 205 206 207 208 209 209 210 211 212 213 214 215 216 217 218 219 219 220 221 222 223 224 225 226 227 228 229 229 230 231 232 233 234 235 236 237 238 239 239 240 241 242 243 244 245 246 247 248 249 249 250 251 252 253 254 255 256 257 258 259 259 260 261 262 263 264 265 266 267 268 269 269 270 271 272 273 274 275 276 277 278 279 279 280 281 282 283 284 285 286 287 288 289 289 290 291 292 293 294 295 296 297 298 299 299 300 301 302 303 304 305 306 307 308 309 309 310 311 312 313 314 315 316 317 318 319 319 320 321 322 323 324 325 326 327 328 329 329 330 331 332 333 334 335 336 337 338 339 339 340 341 342 343 344 345 346 347 348 349 349 350 351 352 353 354 355 356 357 358 359 359 360 361 362 363 364 365 366 367 368 369 369 370 371 372 373 374 375 376 377 378 379 379 380 381 382 383 384 385 386 387 388 389 389 390 391 392 393 394 395 396 397 398 399 399 400

```

The screenshot shows the MySQL Workbench interface with a query editor containing the following SQL code:

```

SELECT p.pname
FROM Parts p
JOIN Catalog c ON p.pid = c.pid
JOIN Supplier s ON c.cid = s.cid
WHERE s.sname = 'Acme Widget' AND p.pid NOT IN (
    SELECT p.pid
    FROM Catalog c
    WHERE c.cid = s.cid
)
;
SELECT DISTINCT c.cid
FROM Catalog c
WHERE c.cid NOT IN (
    SELECT c.cid
    FROM Catalog c
    WHERE s.sname = 'Acme Widget'
)
;
SELECT c.cid
FROM Catalog c
WHERE c.cid NOT IN (
    SELECT c.cid
    FROM Catalog c
    WHERE s.sname = 'Acme Widget'
)
;

```

The results pane shows a single row: "Mobile".

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).

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

Lab8-MORE QUERIES ON SUPPLIER DATABASE

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

Select p.pname,s.sname,c.cost from catalog_202 c join parts_202 p on c.pid=p.pid join supplier_202 s on c.sid=s.sid where c.cost=(select max(cost) from catalog);

	Pname	Sname	Cost
▶	Pencil	Reliance	40

- Find suppliers who do NOT supply any red parts.

Select s.sname from supplier_202 s where s.sid not in (select c.sid from catalog_202 c join parts_202 p on c.pid=p.pid where p.color='Red');

	Sname
▶	Vimal
	Reliance

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

select s.sname,sum(c.cost) as total_value from supplier_202 s join catalog_202 c on s.sid=c.sid group by s.sname

	Sname	Total_Value
▶	Acme Widget	70
	Johns	30
	Vimal	30
	Reliance	40

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

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

	Sname
▶	Acme Widget

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

select p.pname, s.sname, c.cost from parts p join catalog c on p.pid = c.pid join supplier s on c.sid = s.sid where c.cost = (select min(c1.cost) from catalog c1 where c1.pid = p.pid);

	Pname	Sname	Cost
▶	Book	Acme Widget	10
	Pen	Acme Widget	10
	Pencil	Acme Widget	30
	Mobile	Acme Widget	10
	Charger	Acme Widget	10
	Book	Johns	10
	Pencil	Vimal	30

- **Create a view showing suppliers and the total number of parts they supply.**
 create view supplierpartcount_view as select s.sname, count(c.pid) as part_count from supplier s join catalog c on s.sid = c.sid group by s.sname;
 select * from SupplierPartCount_View;

	Sname	Part_Count
▶	Acme Widget	5
	Johns	2
	Vimal	1
	Reliance	1

- **Create a view of the most expensive supplier for each part.**
 create view mostexpensivesupplier_view as select p.pname, s.sname, c.cost from parts p join catalog c on p.pid = c.pid join supplier s on c.sid = s.sid where c.cost = (select max(c1.cost) from catalog c1 where c1.pid = p.pid); select * from MostExpensiveSupplier_View;

	Pname	Sname	Cost
▶	Book	Acme ...	10
	Mobile	Acme ...	10
	Charger	Acme ...	10
	Book	Johns	10
	Pen	Johns	20
	Pencil	Reliance	40

- **Create a Trigger to prevent inserting a Catalog cost below 1.**
 delimiter //
 create trigger min_cost_check_new
 before insert on catalog
 for each row
 begin
 if new.cost < 1 then
 signal sqlstate '45000' set message_text = 'catalog cost cannot be less than 1.';

```
end if;
end//  
delimiter ;  
insert into catalog values (101, 201, 0);
```

Error Code: 1644. Catalog cost cannot be less than 1.

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

```
delimiter //  
create trigger set_default_cost_new  
before insert on catalog  
for each row  
begin  
    if new.cost is null then  
        set new.cost = 10;  
    end if;  
end//  
delimiter ;  
insert into catalog values (10002, 20005, null);  
select * from catalog where sid=10002 and pid= 20005;
```

	sid	pid	cost
▶	10002	20005	10
*	NULL	NULL	NULL

Experiment 9: Nosql lab1(Student)

Create a database “Student” with the following attributes Rollno, Age, ContactNo, EmailId.

```
>_MONGOSH
> use Student
< switched to db Student
Student> |
```

Insert appropriate values.

```
switched to db Students_Table
Students_Table> db.student.insertMany([
...   { Rollno: 10, Age: 20, ContactNo: "9876543210", EmailId: "rahull0@gmail.com", Name: "RAHUL" },
...   { Rollno: 11, Age: 21, ContactNo: "9123456780", EmailId: "raj11@gmail.com", Name: "RAJ" },
...   { Rollno: 12, Age: 22, ContactNo: "9988776655", EmailId: "rajvardhan12@gmail.com", Name: "RAJVARDHAN" }
... ])
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6927c9b61771b37f9963b112'),
    '1': ObjectId('6927c9b61771b37f9963b113'),
    '2': ObjectId('6927c9b61771b37f9963b114')
  }
}
```

The screenshot shows the MongoDB Compass interface with the 'student' collection selected. The top navigation bar includes tabs for 'student' (selected), 'Students_Table', and a '+' button. Below the navigation is a search bar with the query 'localhost:27017 > student1 > student'. The main area has tabs for 'Documents' (selected), 'Aggregations', 'Schema', 'Indexes', and 'Validation'. Under 'Documents', there is a search input 'Type a query: { field: 'value' } or [Generate query](#)' and buttons for '+ ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. Three document cards are listed:

- Document 1:**

```
_id: ObjectId('6927c6f526e0825c3463b112')
Rollno : 10
Age : 20
ContactNo : "9876543210"
EmailId : "newemail10@gmail.com"
Name : "ABC"
```
- Document 2:**

```
_id: ObjectId('6927c6f526e0825c3463b113')
Rollno : 11
Age : 21
ContactNo : "9123456780"
EmailId : "abc11@gmail.com"
Name : "ABC"
```
- Document 3:**

```
_id: ObjectId('6927c6f526e0825c3463b114')
Rollno : 12
Age : 22
ContactNo : "9988776655"
EmailId : "abc12@gmail.com"
Name : "XYZ"
```

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

```
Students_Table> db.student.updateOne(  
...   { Rollno: 10 },  
...   { $set: { EmailId: "raman10@gmail.com" } }  
... )
```

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

```
Students_Table> db.student.updateOne(  
...   { Rollno: 10 },  
...   { $set: { EmailId: "raman10@gmail.com" } }  
... )  
...  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 1,  
  modifiedCount: 1,  
  upsertedCount: 0  
}  
Students_Table> db.student.updateOne(  
...   { Rollno: 11, Name: "RAJ" },  
...   { $set: { Name: "RAJU" } }  
... )  
...  
{  
  acknowledged: true,  
  insertedId: null,
```

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

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

Export the created table into local file system.

```
2025-02-20T16:22:13.543+0530      connected to: localhost  
2025-02-20T16:22:13.678+0530      exported 3 records
```

Experiment 10 : Nosql lab2(Customer)

Create a collection by name Customers with the following attributes. Cust_id, Acc_Bal, Acc_Type

```
>_MONGOSH

> use CustomerDB

< switched to db CustomerDB
> db.Customers.insertMany([

    { Cust_id: 101, Acc_Bal: 1500, Acc_Type: "Z" },

    { Cust_id: 102, Acc_Bal: 900,   Acc_Type: "A" },

    { Cust_id: 101, Acc_Bal: 1800, Acc_Type: "Z" },

    { Cust_id: 103, Acc_Bal: 1300, Acc_Type: "Z" },

    { Cust_id: 104, Acc_Bal: 700,   Acc_Type: "B" }

])

< {

    acknowledged: true,
    insertedIds: {
        '0': ObjectId('693a3695a2d039c4696aa0a8'),
        '1': ObjectId('693a3695a2d039c4696aa0a9'),
        '2': ObjectId('693a3695a2d039c4696aa0aa'),
        '3': ObjectId('693a3695a2d039c4696aa0ab'),
        '4': ObjectId('693a3695a2d039c4696aa0ac')
    }
}
```

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

```
>_MONGOSH
    }
}
> db.Customers.find(
  { Acc_Type: "Z", Acc_Bal: { $gt: 1200 } }
)
< [
  {
    _id: ObjectId('693a3695a2d039c4696aa0a8'),
    Cust_id: 101,
    Acc_Bal: 1500,
    Acc_Type: 'Z'
  },
  {
    _id: ObjectId('693a3695a2d039c4696aa0aa'),
    Cust_id: 101,
    Acc_Bal: 1800,
    Acc_Type: 'Z'
  },
  {
    _id: ObjectId('693a3695a2d039c4696aa0ab'),
    Cust_id: 103,
    Acc_Bal: 1300,
    Acc_Type: 'Z'
  }
]
```

Determine Minimum and Maximum account balance for each customer_id.

```
>_MONGOSH
    Acc_Type: 'Z'
}
{
  _id: ObjectId('693a3695a2d039c4696aa0ab'),
  Cust_id: 103,
  Acc_Bal: 1300,
  Acc_Type: 'Z'
}
> db.Customers.aggregate([
  {
    $group: {
      _id: "$Cust_id",
      Min_Balance: { $min: "$Acc_Bal" },
      Max_Balance: { $max: "$Acc_Bal" }
    }
  }
])
< [
  {
    _id: 104,
    Min_Balance: 700,
    Max_Balance: 700
  },
  {
    _id: 102,
    Min_Balance: 900,
    Max_Balance: 900
  },
  {
    _id: 101,
    Min_Balance: 1500,
    Max_Balance: 1800
  },
  {
    _id: 103,
    Min_Balance: 1300,
    Max_Balance: 1300
  }
]
```

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

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

Export the created collection into local file system.

```
2025-02-20T16:55:12.543+0530      connected to: localhost
2025-02-20T16:55:12.684+0530      exported 5 records
```