VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Database Management Systems (23CS3PCDBM)

Submitted by

MOUNESHWAR(1BM24CS175)

in partial fulfilment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING BENGALURU-560019
(Autonomous Institution under VTU)
Sep-2025 to Jan-2026

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Database Management Systems (23CS3PCDBM)" carried out by **Mouneshwar(1BM24CS175),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a Database Management Systems (23CS3PCDBM) work prescribed for the said degree.

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

Index

Sl. No.	Date	Experiment Title	Page No.
1		Insurance Database	
2		More Queries on Insurance Database	

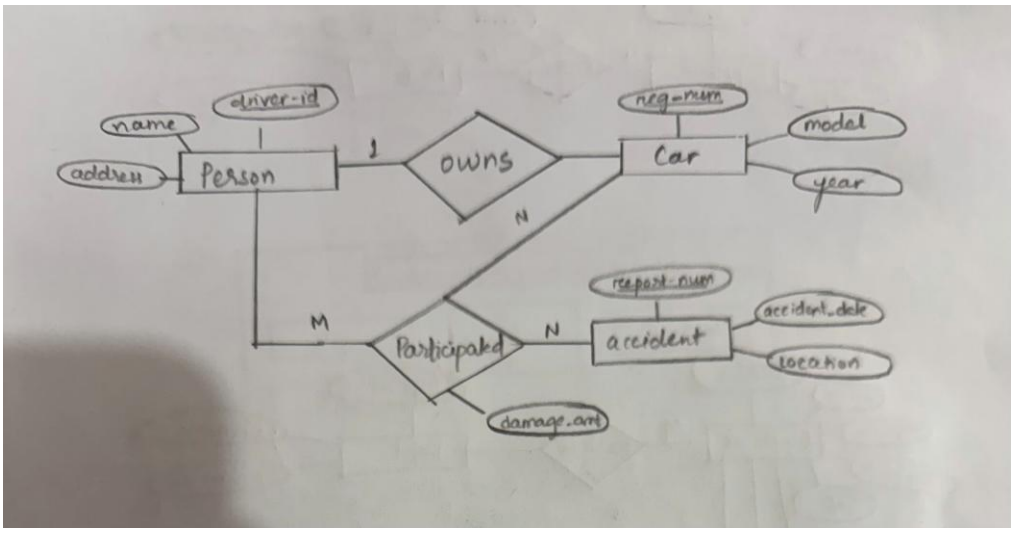
3		Bank Database	
4		More Queries on Bank Database	
5		Employee Database	
6		More Queries on Employee Database	
7		Supplier Database	
8		More Queries on Supplier Database	
9		NOSQL Installation in Cloud	
10		NO SQL - Student Database	
11		NO SQL - Customer Database	
12		NO SQL – Restaurant Database	
13		LeetCode Practice	
14		LeetCode Practice	

Experiment 1: Insurance Database

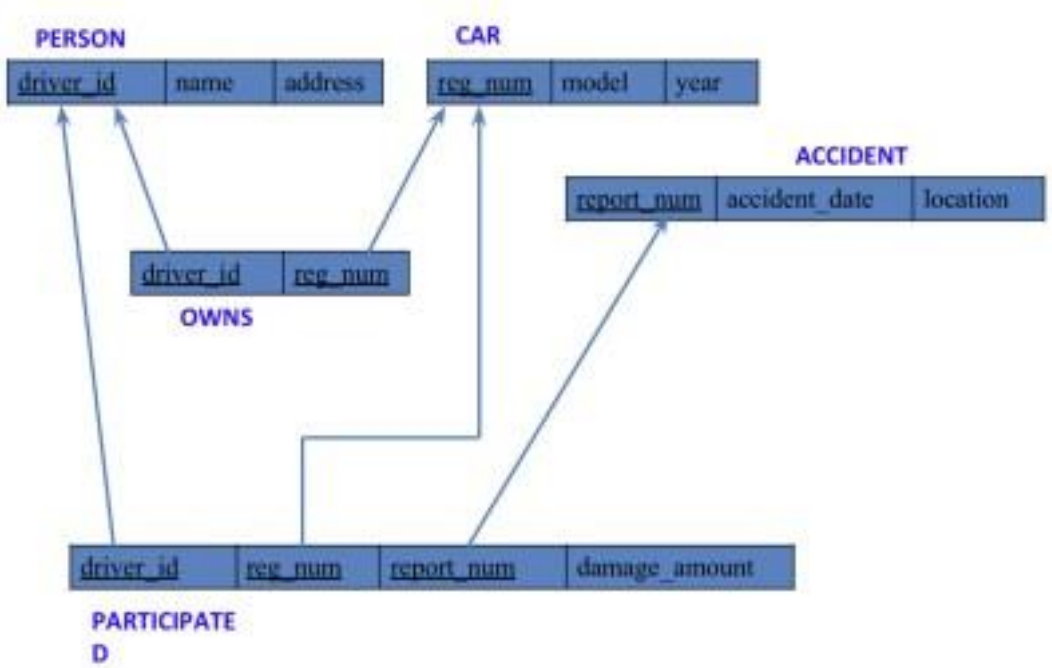
Specification of Insurance Database Application

The insurance database must maintain information about drivers, the cars they own, the accidents reported, and the participation of each driver and car in those accidents. Each driver in the system is uniquely identified by a driver ID, along with their name and address, and each car is uniquely identified by its registration number together with details such as model and manufacturing year. The system must allow storing ownership information that links a driver to one or more cars, while also allowing a car to be linked to one or more drivers if shared ownership occurs; duplicate ownership records for the same driver and car must not exist. Accident information must be stored using a unique report number assigned to each accident, along with the date on which the accident occurred and the location where it happened. Every accident reported in the system must have at least one participating driver and car, and this participation is recorded by linking the driver, the involved car, and the accident report together with the corresponding damage amount for that particular involvement. A participation record must reference an existing driver, an existing car, and an existing accident, and no two participation entries may repeat the same combination of driver, car, and accident report. The database must ensure that damage amounts are non-negative, accident dates are valid calendar dates, and car manufacturing years fall within reasonable limits. It must also preserve referential integrity so that ownership or participation entries cannot exist without valid driver, car, and accident information already present in the system. Deletion policies must prevent removal of drivers or cars that appear in past accident participation records unless historical consistency is preserved through controlled deletion rules or archival mechanisms. The system should maintain accurate links between drivers, cars, and accidents at all times, ensuring reliable retrieval of ownership histories, accident histories, and damage information for administrative, legal, and insurance-related purposes.

Entity Relationship Diagram



Schema Diagram



- PERSON (driver_id: String, name: String, address: String)
- CAR (reg_num: String, model: String, year: int)
- ACCIDENT (report_num: int, accident_date: date, location: String)
- - OWNS (driver_id: String, reg_num: String)

- PARTICIPATED (driver_id: String,reg_num: String, report_num: int, damage_amount: int)
- Create the above tables by properly specifying the primary keys and the foreign keys. - Enter at least five tuples for each relation

Create database

```
create database insurance_barsha;
use
insurance_barsha;
```

Create table create table

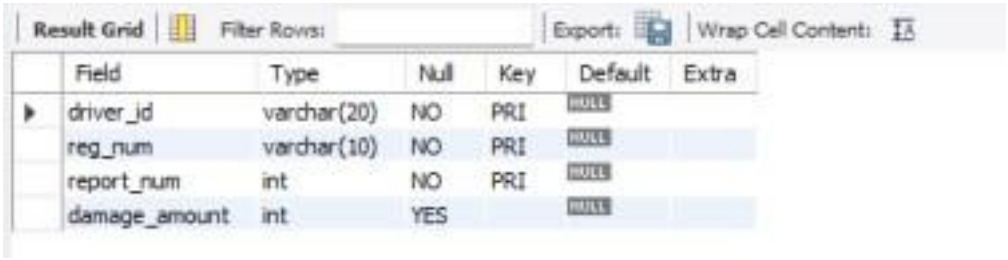
```
person (     driver_id
varchar(10),     name
varchar(30),     address
varchar(50),     primary
key (driver_id)
);

create table car (
reg_num varchar(15),
model varchar(20),
year int,
   primary key (reg_num)
);
```

```
create table accident (
report_num int,
accident_date date,
location varchar(30),
primary key
(report_num) );

create table owns (     driver_id varchar(10),
reg_num varchar(15),     primary key
(driver_id, reg_num),     foreign key
(driver_id) references person(driver_id),
foreign key (reg_num) references
car(reg_num)
);

create table participated (
driver_id varchar(10),
reg_num varchar(15),
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;

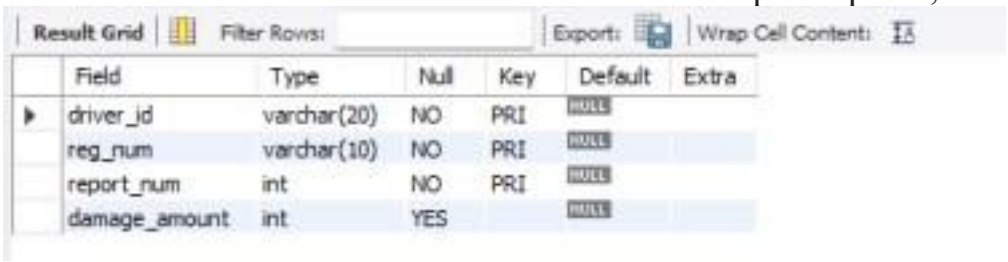


Field	Type	Null	Key	Default	Extra
driver_id	varchar(20)	NO	PRI	NULL	
reg_num	varchar(10)	NO	PRI	NULL	
report_num	int	NO	PRI	NULL	
damage_amount	int	YES		NULL	

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 into person values

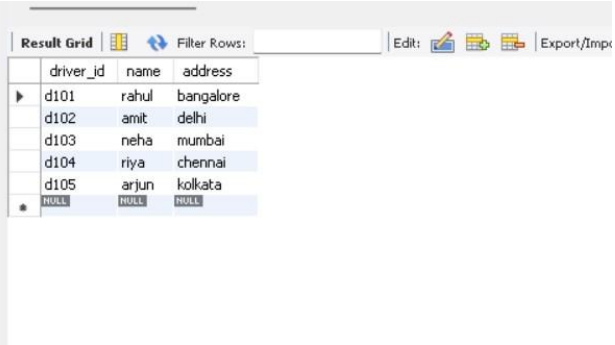
('d101','rahul','bangalore'),

('d102','amit','delhi'),

('d103','neha','mumbai'),
('d104','riya','chennai'),

('d105','arjun','kolkata');

Select * from person;



driver_id	name	address
d101	rahul	bangalore
d102	amit	delhi
d103	neha	mumbai
d104	riya	chennai
d105	arjun	kolkata
NULL	NULL	NULL

insert into car values

('ka01ab1234','lancer',200
5),

('ka02cd5678','swift',2008
),

('mh03ef9012','creta',2010

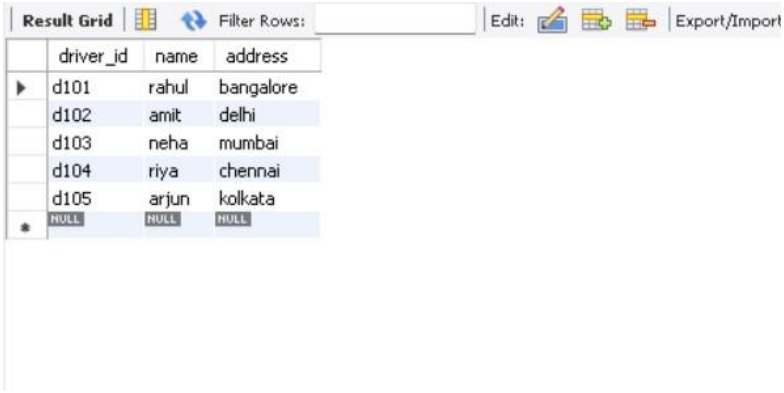
),

('dl04gh3456','verna',2007

),

('tn05ij7890','lancer',2008

); select * from car;



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

driver_id	name	address
d101	rahul	bangalore
d102	amit	delhi
d103	neha	mumbai
d104	riya	chennai
d105	arjun	kolkata
NULL	NULL	NULL

insert into accident values

(101,'2008-02-
12','bangalore'),

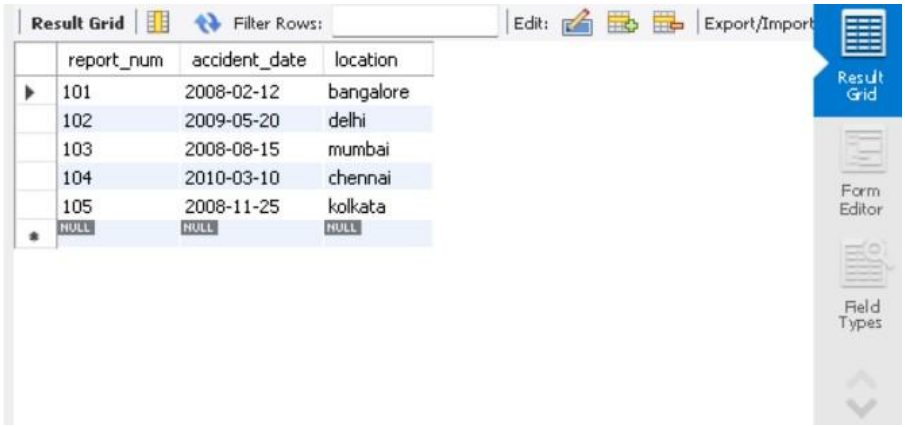
(102,'2009-05-20','delhi'),

(103,'2008-08-
15','mumbai'),

(104,'2010-03-
10','chennai'),

(105,'2008-11-
25','kolkata');

Select * from accident;



report_num	accident_date	location
101	2008-02-12	bangalore
102	2009-05-20	delhi
103	2008-08-15	mumbai
104	2010-03-10	chennai
105	2008-11-25	kolkata
NULL	NULL	NULL

insert into owns values

('d101','ka01ab1234'),

('d102','ka02cd5678'),

('d103','mh03ef9012'),

('d104','dl04gh3456')

,

('d105','tn05ij7890');

select * from owns

driver_id	reg_num
d104	dl04gh3456
d101	ka01ab1234
d102	ka02cd5678
d103	mh03ef9012
d105	tn05ij7890
NULL	NULL

insert into participated values

('d101','ka01ab1234',101,30000),

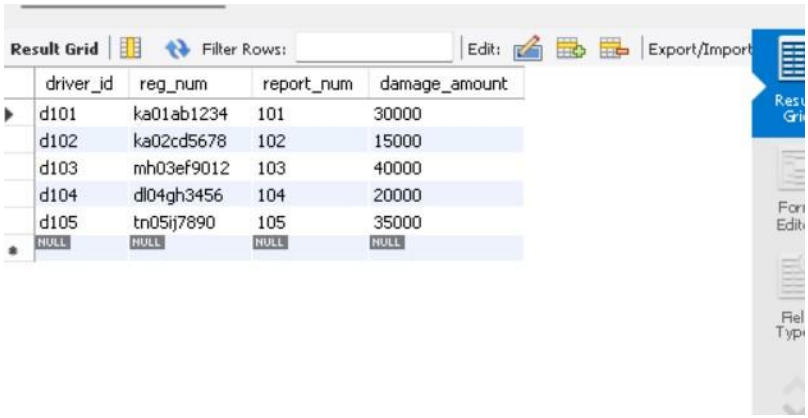
('d102','ka02cd5678',102,15000),

('d103','mh03ef9012',103,40000),

('d104','dl04gh3456',104,20000),

('d105','tn05ij7890',105,35000);

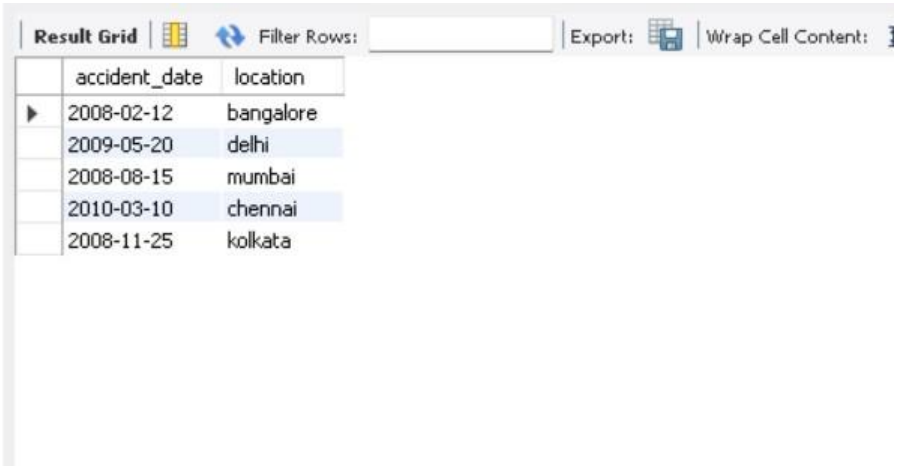
Select * from participated;



driver_id	reg_num	report_num	damage_amount
d101	ka01ab1234	101	30000
d102	ka02cd5678	102	15000
d103	mh03ef9012	103	40000
d104	dl04gh3456	104	20000
d105	tn05ij7890	105	35000
NULL	NULL	NULL	NULL

Queries Display accident date and location

select accident_date, location from accident;



accident_date	location
2008-02-12	bangalore
2009-05-20	delhi
2008-08-15	mumbai
2010-03-10	chennai
2008-11-25	kolkata

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

Update participated set damage_amount = 25000 where reg_num = ‘KA053408’ and

report_num=12 **Add a new accident to the database**

insert into accident values (15, '2023-06-18', ‘Hyderabad');

Display accident date and location

Select accident_date,location from accident;

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

select distinct driver_id from participated where damage_amount >= 25000;

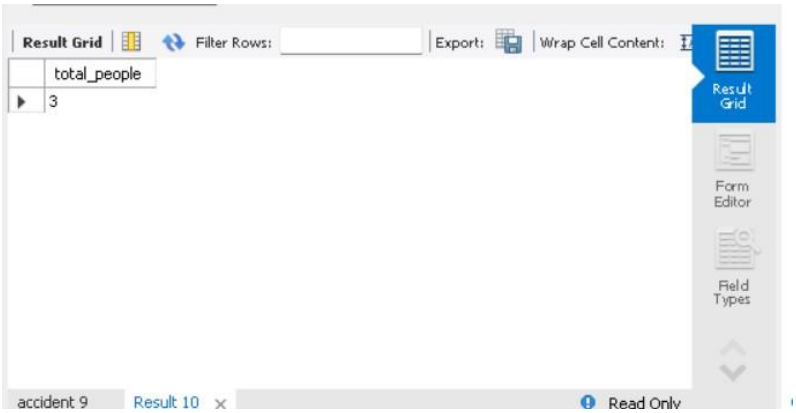

Result Grid
Filter Rows:
Export:
Wrap Cell Content:
driver_id
d101
d103
d105
Result Grid
Form Editor
Field Types

Experiment 2: More Queries on Insurance Database

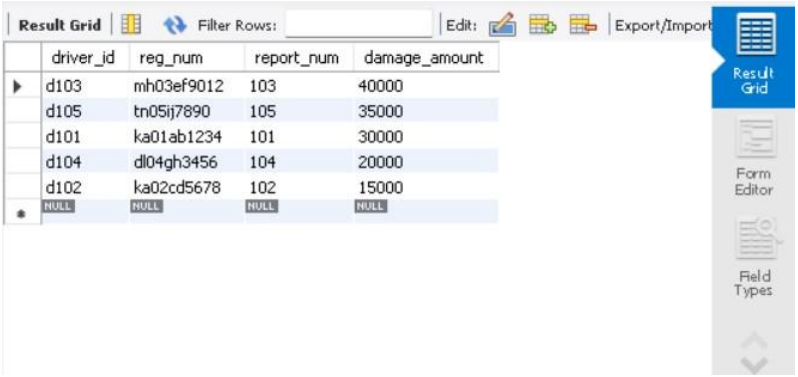
Display the entire car relation in ascending order of manufacturing year select * from car order by year asc;

Find the number of accidents involving cars of a specific model (example: 'lancer') select count(*) as total_accidents from participated p, car where p.reg_num = c.reg_num and c.model = 'lancer';

find the total number of people who owned cars involved in accidents in 2008 select count(distinct o.driver_id) as total_people from owns o, participated p, accident a where o.reg_num = p.reg_num
and p.report_num = a.report_num and extract(year from a.accident_date) = 2008;

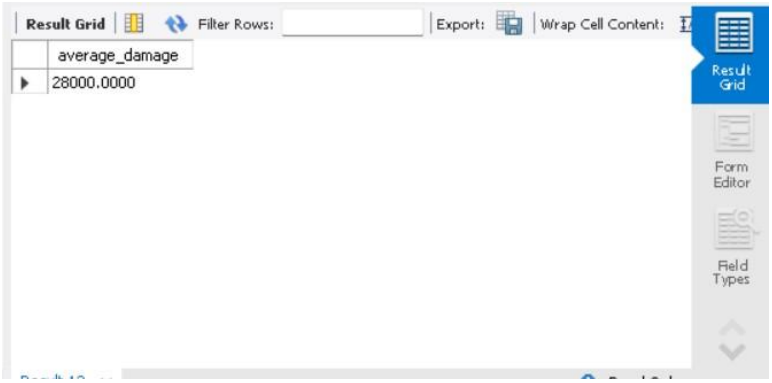


List the entire participated relation in descending order of damage amount select * from participated order by damage_amount desc;



find the average damage amount

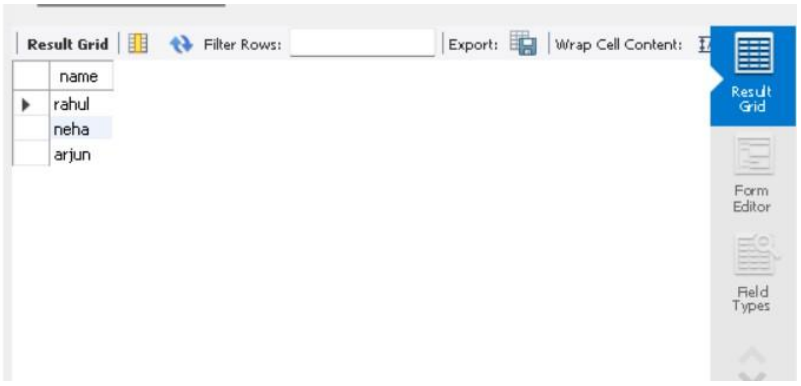
select avg(damage_amount) as average_damage from participated;



Delete the tuples whose damage amount is below the average damage amount

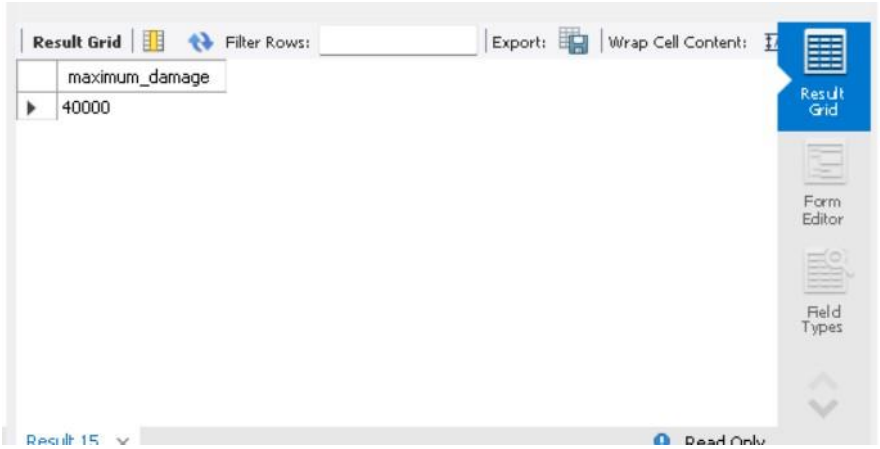
delete from participated where damage_amount
< (select avg(damage_amount)
from participated
);

List the names of drivers whose damage is greater than the average damage amount select distinct p.name from person p, participated pa
where p.driver_id = pa.driver_id and pa.damage_amount > (
select avg(damage_amount) from participated);



Find the maximum damage amount

select max(damage_amount) as maximum_damage from participated;


Result Grid
Filter Rows:
Export:
Wrap Cell Content:
maximum_damage
40000
Result Grid
Form Editor
Field Types
Result 15
Read Only

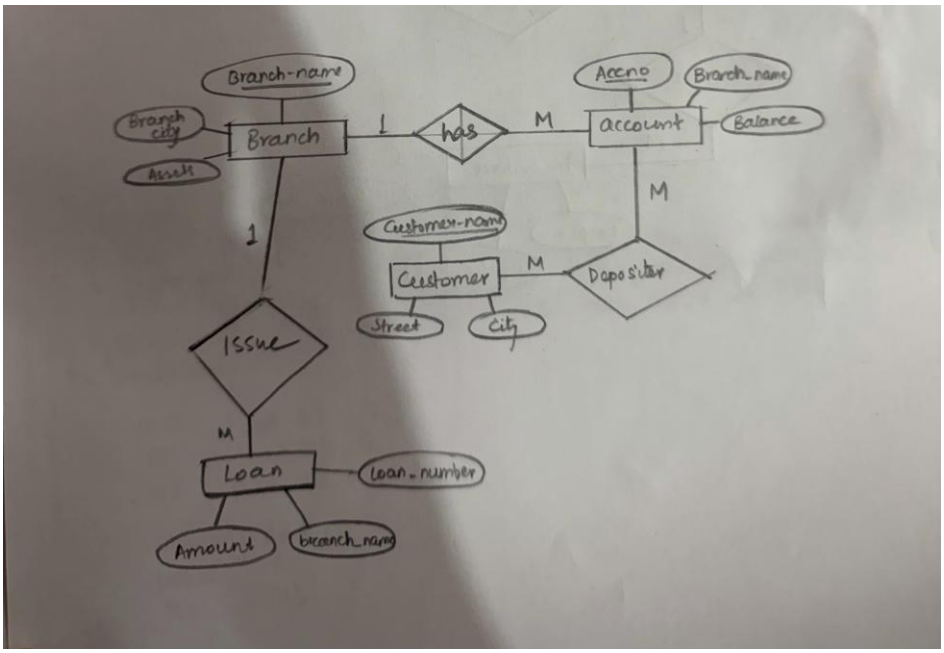
EXPERIMENT 3: BANK DATABASE

Specification of Bank Database Application

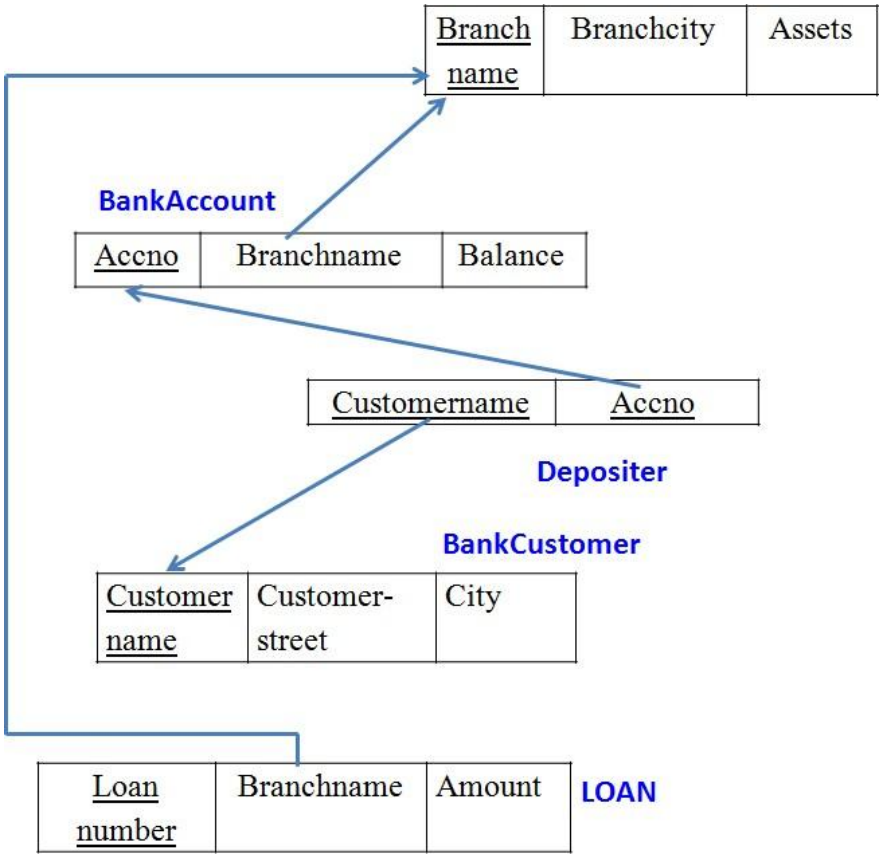
The banking system must store information about branches, bank accounts, customers, deposit relationships, and loans so that branch details (identified by branch name together with city and total assets) are linked to accounts and loans, each account (identified by an account number) records the branch 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 modeled) 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:



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

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

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

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

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

i. Create the above tables by properly specifying the primary keys and the foreign keys. ii. Enter at least five tuples for each relation.

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

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

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

CREATION OF DATABASE:

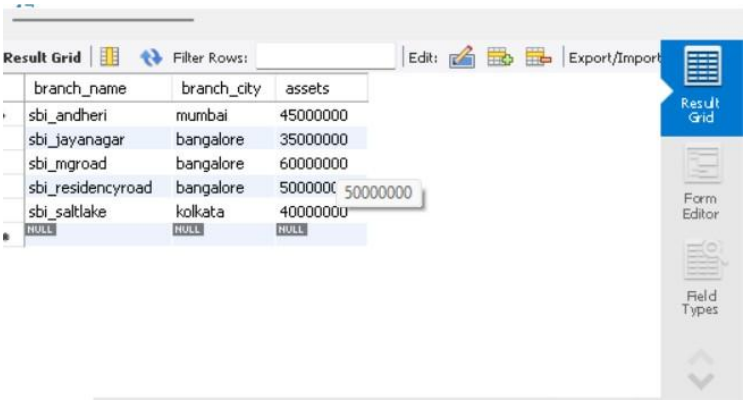
create databse
bank_db; use bank_db;

1.Creation of table with primary key and foreign key

```
create table branch (
branch_name
varchar(30),
branch_city varchar(30),
assets real,     primary
key (branch_name)
);

create table bankaccount
(     accno int,
branch_name
varchar(30),
   balance real,     primary key (accno),     foreign key
(branch_name) references branch(branch_name)
);

create table bankcustomer (
customer_name varchar(30),
customer_street varchar(30),
customer_city varchar(30),
   primary key (customer_name)
);
create table depositer (
   customer_name varchar(30),
   accno int,
   primary key (customer_name, accno),     foreign key
(customer_name) references bankcustomer(customer_name),
foreign key (accno) references bankaccount(accno)
);

create table loan (     loan_number int,
branch_name varchar(30),     amount real,     primary
key (loan_number),     foreign key (branch_name)
references branch(branch_name)
);
```

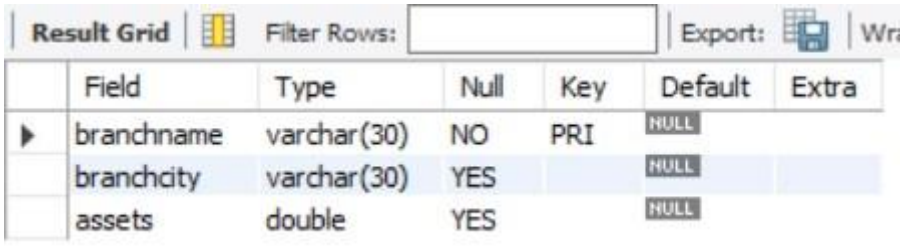
ii. inserting at least five tuples into each relation

insert into branch values
('sbi_residencyroad','bangalore',50000000),
('sbi_jayanagar','bangalore',35000000),
('sbi_mgroad','bangalore',60000000),
('sbi_andheri','mumbai',45000000),
('sbi_saltlake','kolkata',40000000);
Select * from branch;

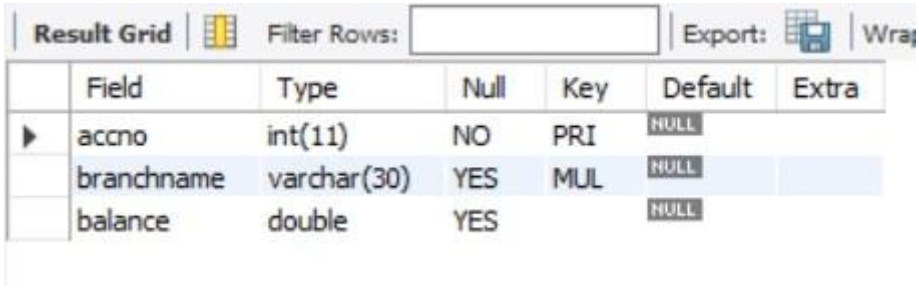


Structure of the table

desc branch;



desc bankaccount;



desc bankcustomer;

Result Grid | Filter Rows: | Export: | Wrap

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

desc depositer;

Result Grid | Filter Rows: | Export: | Wrap

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

desc loan;

Result Grid | Filter Rows: | Export: | Wrap

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

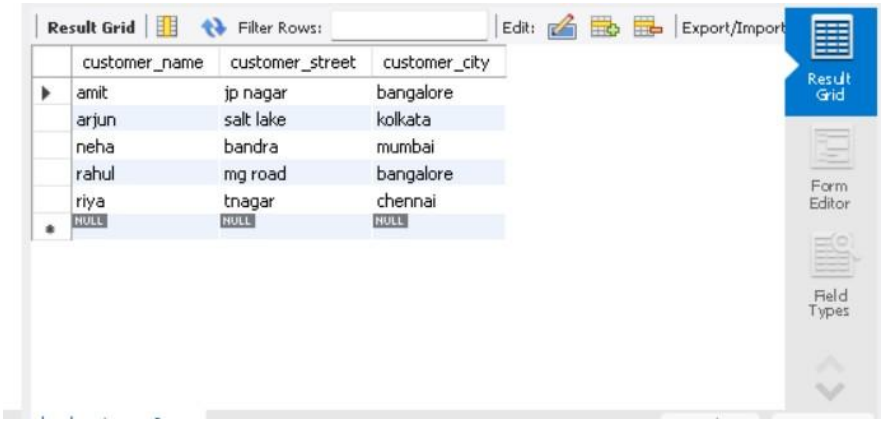
insert into bankaccount values
(101,'sbi_residencyroad',250000),
(102,'sbi_residencyroad',180000),
(103,'sbi_jayanagar',300000),
(104,'sbi_mgroad',400000),
(105,'sbi_andheri',220000);
Select from * bankaccount;

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

accno	branch_name	balance
101	sbi_residencyroad	250000
102	sbi_residencyroad	180000
103	sbi_jayanagar	300000
104	sbi_mgroad	400000
105	sbi_andheri	220000
NULL	NULL	NULL

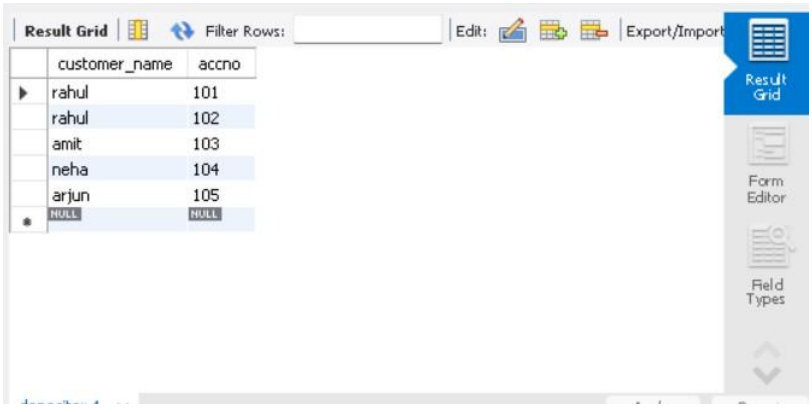
Result Grid | Form Editor | Field Types

insert into bankcustomer values
('rahul','mg road','bangalore'),
('amit','jp nagar','bangalore'),
('neha','bandra','mumbai'),
('riya','tnagar','chennai'),
('arjun','salt lake','kolkata');
Select * from bankcustomer;



customer_name	customer_street	customer_city
amit	jp nagar	bangalore
arjun	salt lake	kolkata
neha	bandra	mumbai
rahul	mg road	bangalore
riya	tnagar	chennai
NULL	NULL	NULL

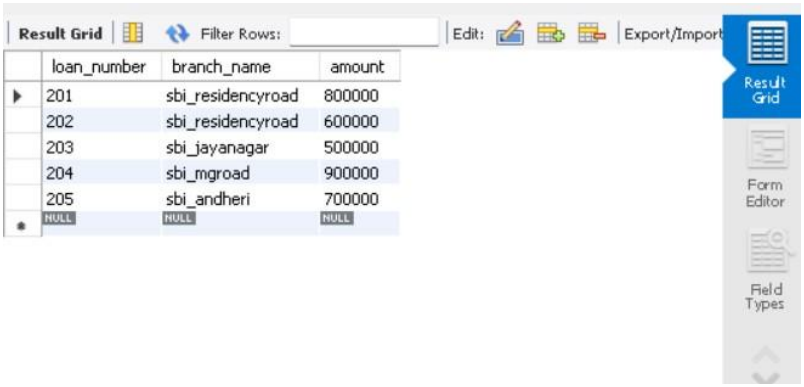
insert into depositer values
('rahul',101),
('rahul',102),
('amit',103),
('neha',104),
('arjun',105);
Select from depositor values;



customer_name	accno
rahul	101
rahul	102
amit	103
neha	104
arjun	105
NULL	NULL

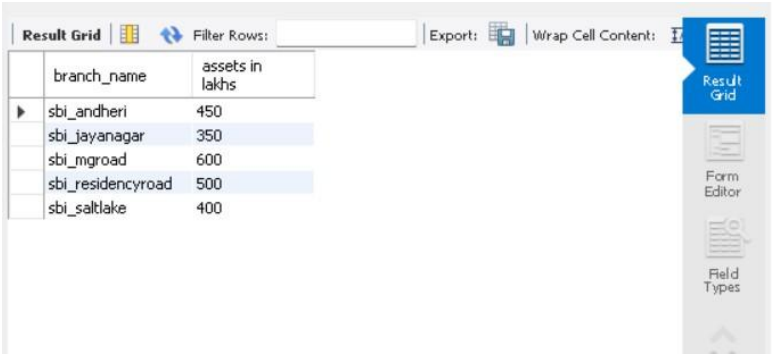
insert into loan values
(201,'sbi_residencyroad',800000),
(202,'sbi_residencyroad',600000),

(203,'sbi_jayanagar',500000),
(204,'sbi_mgroad',900000),
(205,'sbi_andheri',700000);
Select * from loan;



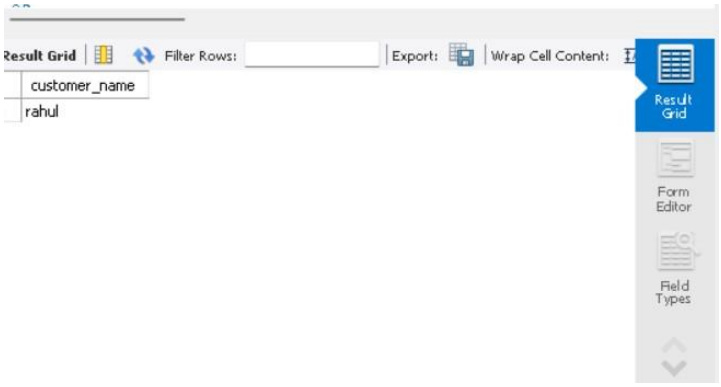
loan_number	branch_name	amount
201	sbi_residencyroad	800000
202	sbi_residencyroad	600000
203	sbi_jayanagar	500000
204	sbi_mgroad	900000
205	sbi_andheri	700000
NULL	NULL	NULL

Display branch name and assets in lakhs of rupees select branch_name, assets/100000 as "assets in lakhs" from branch;



branch_name	assets in lakhs
sbi_andheri	450
sbi_jayanagar	350
sbi_mgroad	600
sbi_residencyroad	500
sbi_saltlake	400

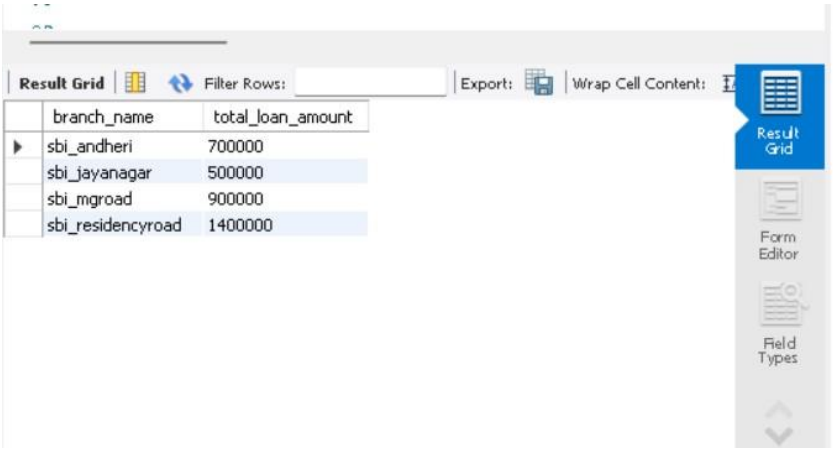
Find customers who have at least two accounts at the same branch select d.customer_name from depositer d, bankaccount b where d.accno = b.accno and b.branch_name = 'sbi_residencyroad' group by d.customer_name having count(d.accno) >= 2;



customer_name
rahul

Create a view that shows each branch and the total loan amount at that branch create view branch_loan_total as select branch_name, sum(amount) as total_loan_amount from loan group by branch_name;

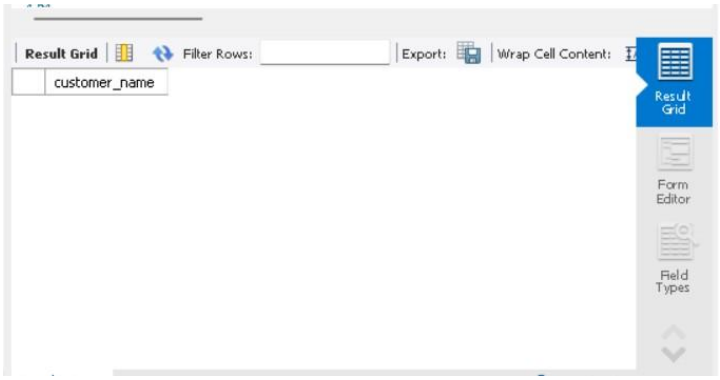
To display the view: select
* from branch_loan_total;



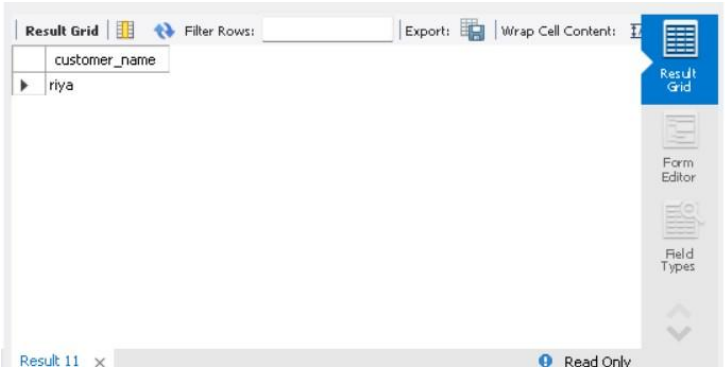
EXPERIMENT 4: MORE QUERIES ON BANK DATABASE

i)Find all customers who have an account at all branches located in a specific city (example: delhi)

```
select d.customer_name
from depositer d, bankaccount b,
branch br where d.accno = b.accno
and b.branch_name =
br.branch_name
and br.branch_city = ‘delhi' group
by d.customer_name having
count(distinct b.branch_name) = (
select count(*)
from branchwhere branch_city = 'delhi'
                    );
```



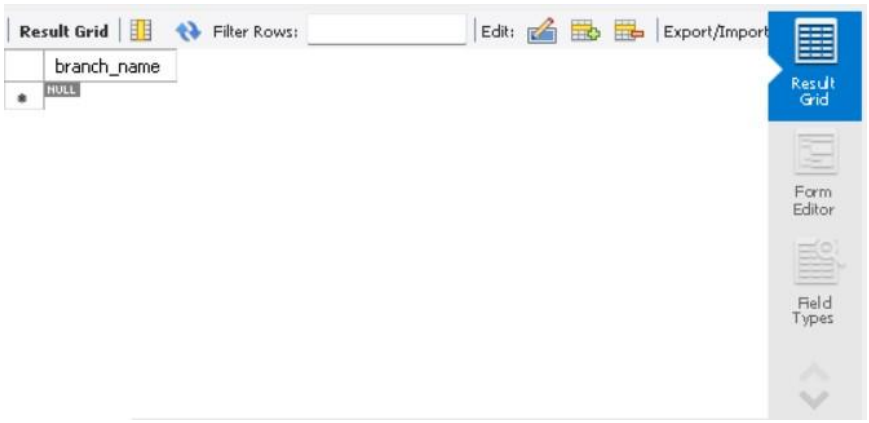
ii)Find all customers who have a loan but do not have an account select distinct c.customer_name from bankcustomer c, loan l where c.customer_name not in (
select d.customer_name
    from depositer d
);



iii)Find all customers who have both an account and a loan at the bangalore branch select distinct d.customer_name from depositer d, bankaccount b, loan l, branch br
where d.accno = b.accno and
b.branch_name =
br.branch_name and
l.branch_name =
br.branch_name and
br.branch_city = 'bangalore';

iv. find names of branches whose assets are greater than all branches located in Bangalore

select branch_name from
branch where assets > all (
select assets from branch
where branch_city =
'bangalore'
);



v)Delete all account tuples at branches located in a specific city (example: bombay)

```
delete from bankaccount
where branch_name in (
select branch_name     from
branch     where
branch_city = 'bombay'
);
```

vi)update the balance of all accounts by 5%

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

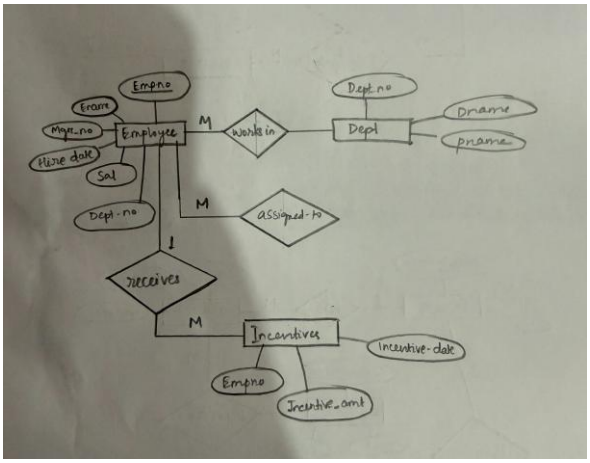
EXPERIMENT 5: EMPLOYEE DATABASE

Specification of Employee Database Application

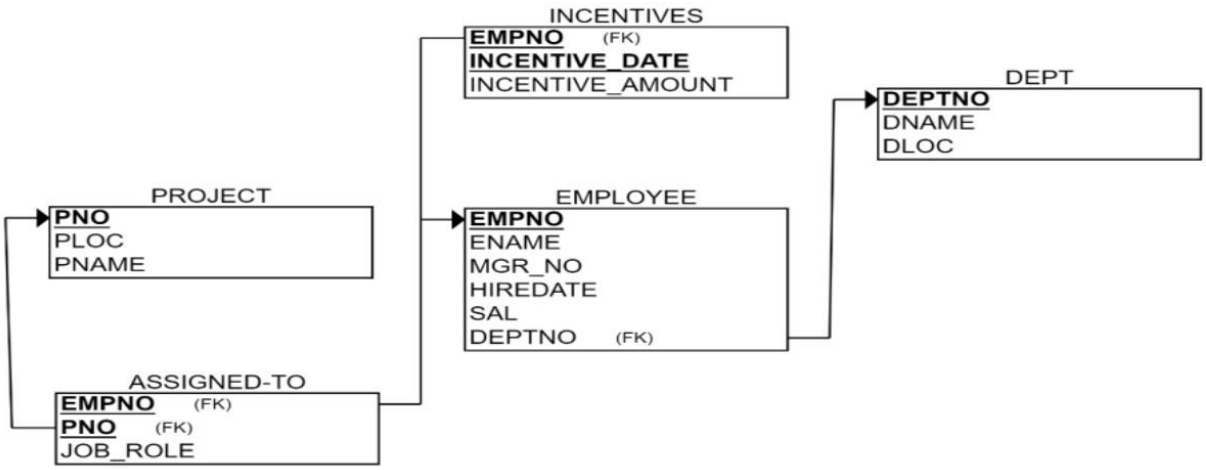
The employee database must record each employee's identifying number, name, manager reference, hire date, salary, and department affiliation while also tracking departmental details, project assignments (including the role an employee plays on a project), and any incentive payments given to employees.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

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

ENTITY RELATIONSHIP DIAGRAM:



SCHEMA DIAGRAM:



CREATION OF DATABASE:

Create database
employee; Use employee;

CREATION OF TABLE:

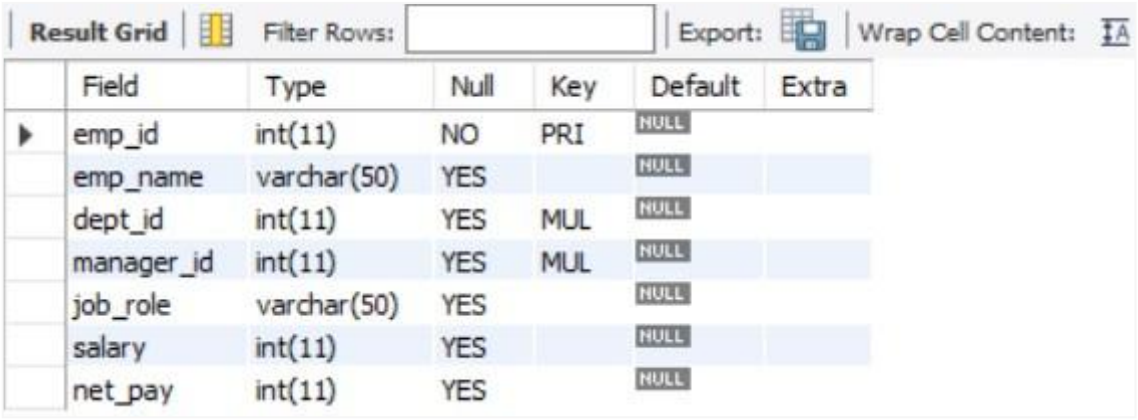
Using Scheme diagram, create tables by properly specifying the primary keys and the foreign keys

```
create table dept (
deptno int primary
key,    dname
varchar(50),
   dloc varchar(50)
);

create table employee (    empno int
primary key,    ename varchar(50),
mgr_no int,    hiredate date,    sal
decimal(10,2),    deptno int,    foreign
key (deptno) references dept(deptno)
);

create table project (
pno int primary key,
ploc varchar(50),
   pname varchar(50)
```

```
);

create table assigned_to (     empno int,
pno int,     job_role varchar(50),     primary
key (empno, pno),     foreign key (empno)
references employee(empno),     foreign key
(pno) references project(pno)
);
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

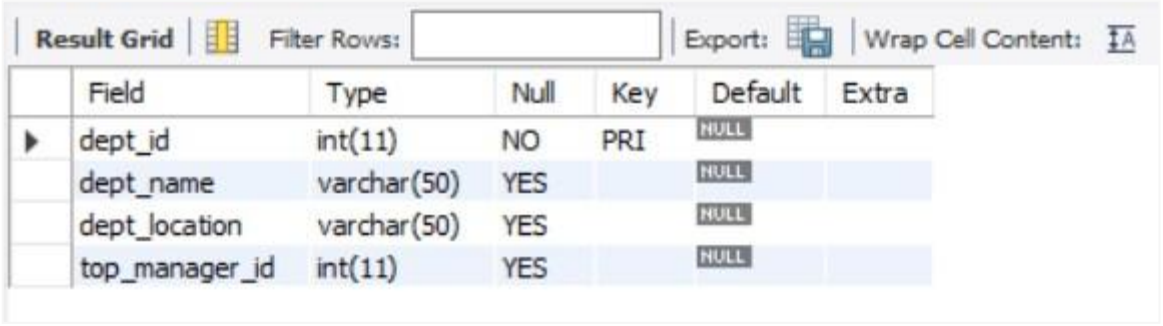
Desc employee;



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

Field	Type	Null	Key	Default	Extra
emp_id	int(11)	NO	PRI	NULL	
emp_name	varchar(50)	YES		NULL	
dept_id	int(11)	YES	MUL	NULL	
manager_id	int(11)	YES	MUL	NULL	
job_role	varchar(50)	YES		NULL	
salary	int(11)	YES		NULL	
net_pay	int(11)	YES		NULL	

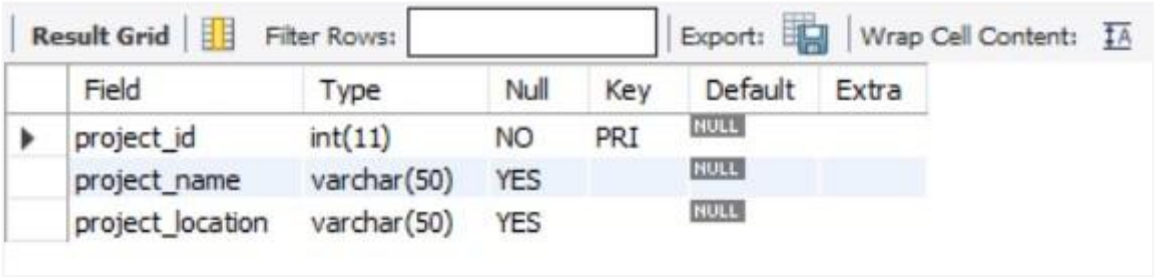
Desc department;



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

Field	Type	Null	Key	Default	Extra
dept_id	int(11)	NO	PRI	NULL	
dept_name	varchar(50)	YES		NULL	
dept_location	varchar(50)	YES		NULL	
top_manager_id	int(11)	YES		NULL	

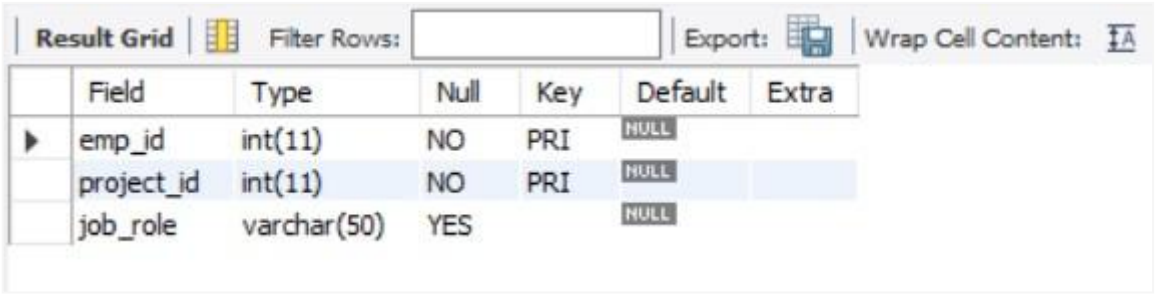
desc project;



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

Field	Type	Null	Key	Default	Extra
project_id	int(11)	NO	PRI	NULL	
project_name	varchar(50)	YES		NULL	
project_location	varchar(50)	YES		NULL	

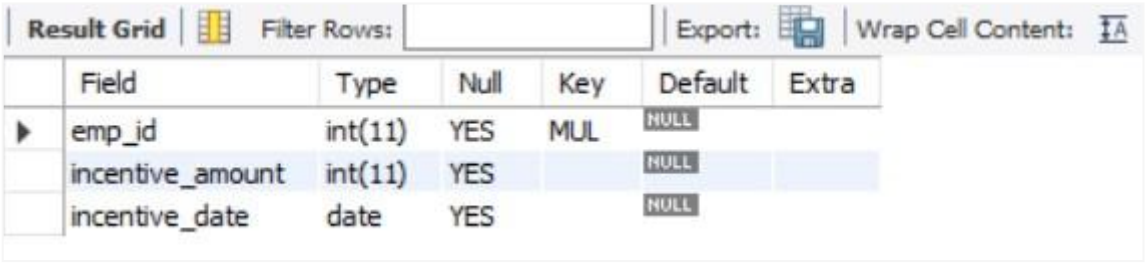
desc assigned_to;



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

Field	Type	Null	Key	Default	Extra
emp_id	int(11)	NO	PRI	NULL	
project_id	int(11)	NO	PRI	NULL	
job_role	varchar(50)	YES		NULL	

desc incentives;

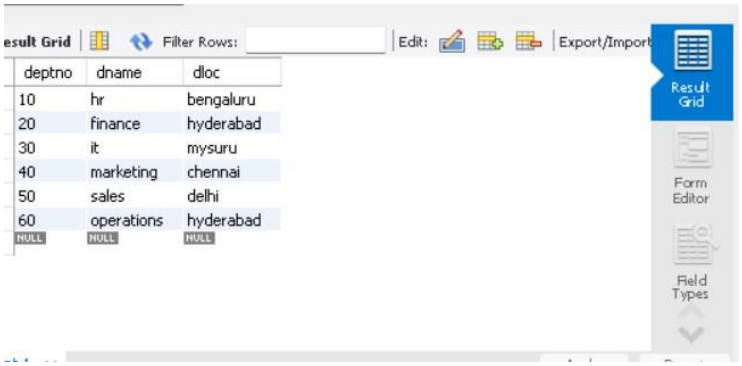


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

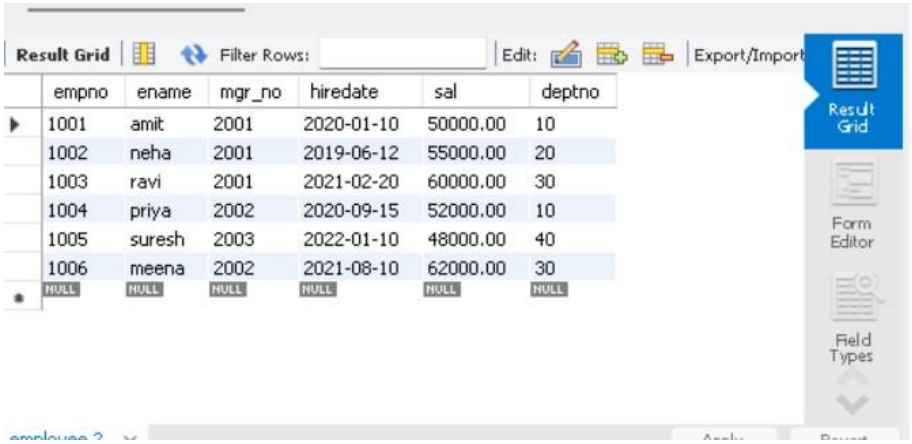
Field	Type	Null	Key	Default	Extra
emp_id	int(11)	YES	MUL	NULL	
incentive_amount	int(11)	YES		NULL	
incentive_date	date	YES		NULL	

-insert 5 values into each tuple

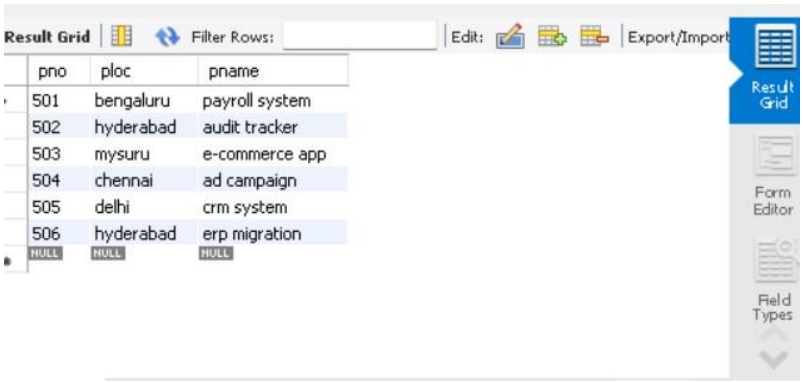
```
insert into dept values
(10, 'hr', 'bengaluru'),
(20, 'finance', 'hyderabad'),
(30, 'it', 'mysuru'),
(40, 'marketing', 'chennai'),
(50, 'sales', 'delhi'),
(60, 'operations', 'hyderabad');
Select * from dept;
```



insert into employee values
(1001, 'amit', 2001, '2020-01-10', 50000, 10),
(1002, 'neha', 2001, '2019-06-12', 55000, 20),
(1003, 'ravi', 2001, '2021-02-20', 60000, 30),
(1004, 'priya', 2002, '2020-09-15', 52000, 10),
(1005, 'suresh', 2003, '2022-01-10', 48000,
40), (1006, 'meena', 2002, '2021-08-10',
62000, 30); select * from employee;



insert into project values
(501, 'bengaluru', 'payroll
system'),
(502, 'hyderabad', 'audit tracker'),
(503, 'mysuru', 'e-commerce app'),
(504, 'chennai', 'ad campaign'),
(505, 'delhi', 'crm system'),
(506, 'hyderabad', 'erp migration');
Select * from project;

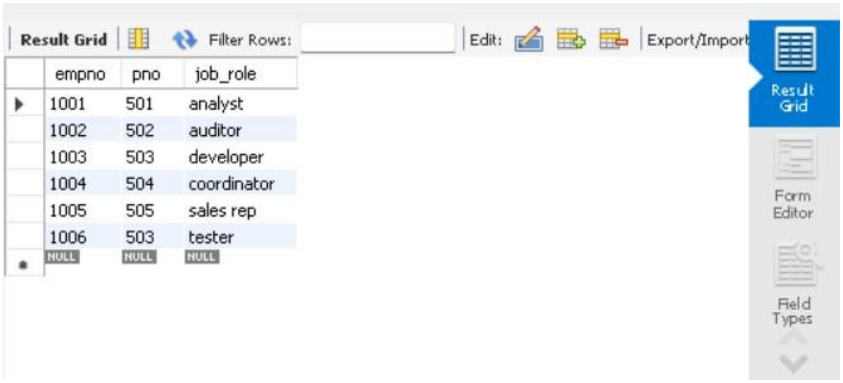


pno	ploc	pname
501	bengaluru	payroll system
502	hyderabad	audit tracker
503	mysuru	e-commerce app
504	chennai	ad campaign
505	delhi	crm system
506	hyderabad	erp migration
NULL	NULL	NULL

Insert into assigned_to values

(1001, 501, 'analyst'),(1002, 502, 'auditor'),
(1003, 503, 'developer'),
(1004, 504, 'coordinator'),
(1005, 505, 'sales rep'),
(1006, 503, 'tester');

Select * from assigned_to;

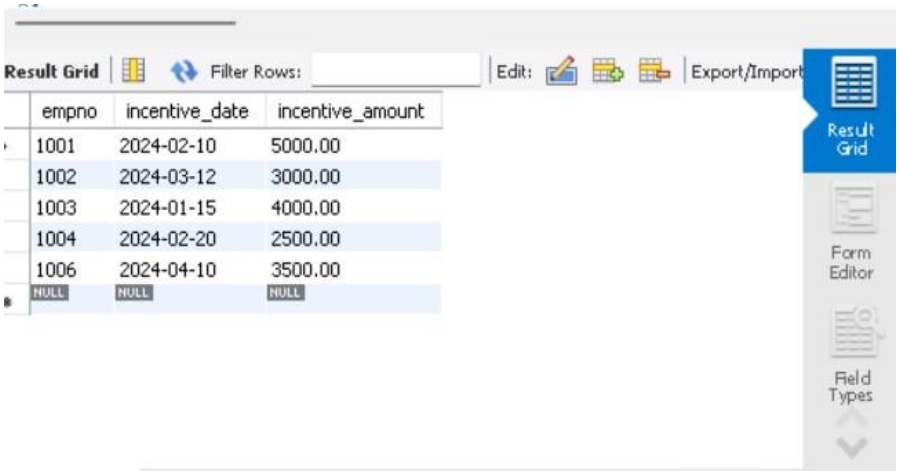


empno	pno	job_role
1001	501	analyst
1002	502	auditor
1003	503	developer
1004	504	coordinator
1005	505	sales rep
1006	503	tester
NULL	NULL	NULL

Insert into incentives values

(1001, '2024-02-10', 5000),
(1002, '2024-03-12', 3000),
(1003, '2024-01-15', 4000),
(1004, '2024-02-20', 2500),
(1006, '2024-04-10', 3500);

Select * from incentives;



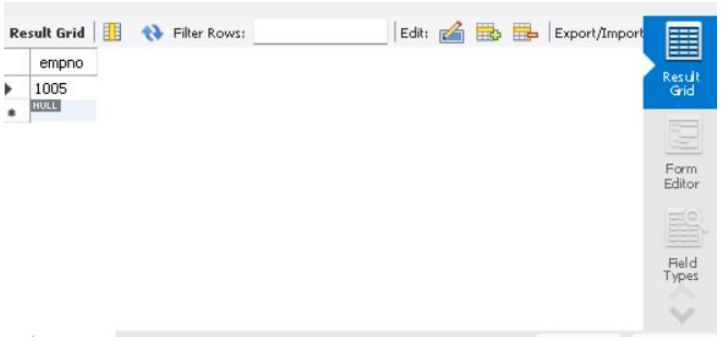
empno	incentive_date	incentive_amount
1001	2024-02-10	5000.00
1002	2024-03-12	3000.00
1003	2024-01-15	4000.00
1004	2024-02-20	2500.00
1006	2024-04-10	3500.00
NULL	NULL	NULL

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

```
select distinct a.empno
from assigned_to a join
project p on a.pno =
p.pno
where p.ploc in ('bengaluru', 'hyderabad', 'mysuru');
```

empno
1001
1002
1003
1006

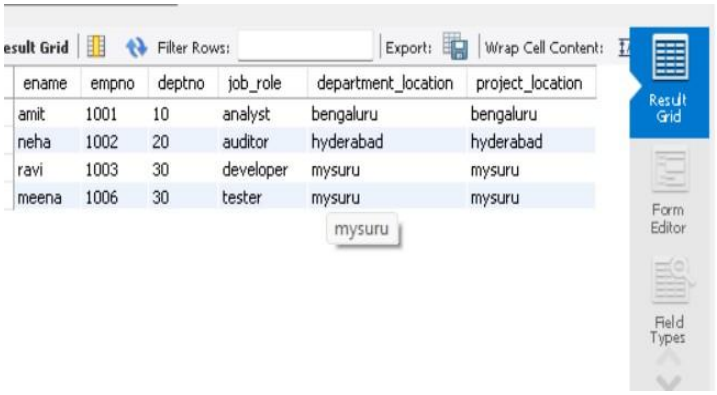
-Get Employee ID's of those employees who didn't receive incentives select e.empno from employee e
where e.empno not in (select empno from incentives);



empno
1005
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.

```
select e.ename,
e.empno,
        e.deptno,
        a.job_role,
        d.dloc as department_location,
        p.ploc as project_location
from employee e join dept d on
e.deptno = d.deptno join
assigned_to a on e.empno =
a.empno
join project p on a.pno = p.pno
where d.dloc = p.ploc;
```

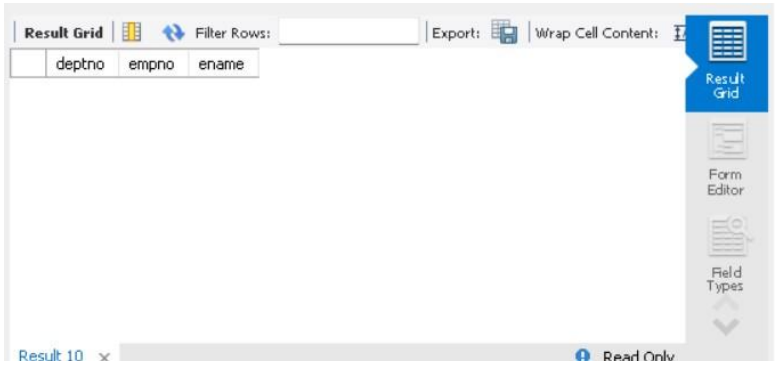


EXPERIMENT 6: MORE QUERIES ON EMPLOYEE DATABASE

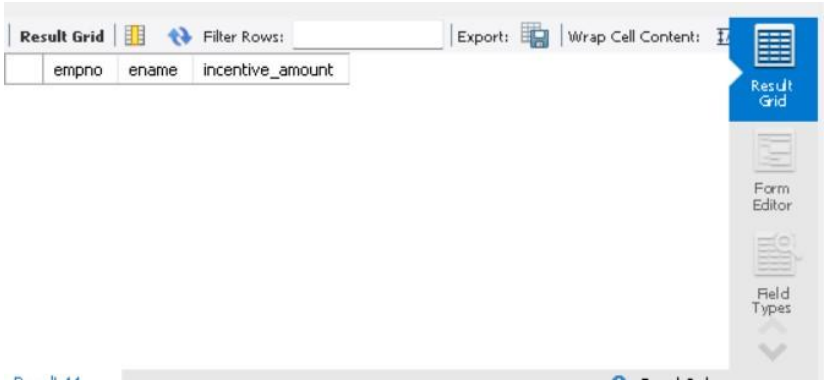
List the names of managers with the maximum number of employees select m.empno, m.ename from employee m join employee e on m.empno = e.mgr_no group by m.empno, m.ename having count(*) = (select max(cnt)
from (select
count(*) as cnt
from employee
group by mgr_no
)
);

Find the name of the second top level managers of each department select deptno, empno, ename
from (select
m.deptno,
m.empno,
m.ename,
dense_rank() over (
partition by m.deptno
order by count(e.empno) desc
) as rnk
from employee m join
employee e on m.empno = e.mgr_no
group by m.deptno, m.empno,
m.ename
) t

where rnk = 2;

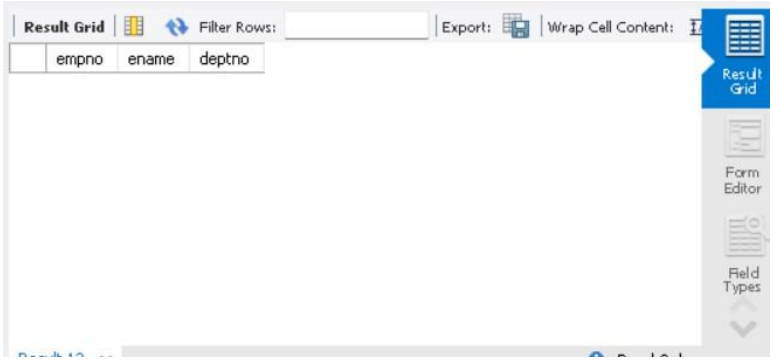


Find employee details who got the second maximum incentive in january 2019 select e.empno, e.ename, i.incentive_amount from employee e
join incentives i on e.empno = i.empno
where extract(year from i.incentive_date)
= 2019 and extract(month from
i.incentive_date) = 1 and
i.incentive_amount = (select
max(incentive_amount) from
incentives where incentive_amount < (
select max(incentive_amount)
from incentives where extract(year
from incentive_date) = 2019
and extract(month from incentive_date) = 1
)
);



Display employees who are working in the same department as their manager select e.empno, e.ename, e.deptno
from employee e

join employee m on e.mgr_no =
m.empno where e.deptno = m.deptno;



Experiment 7: Supplier Database

Specification of Supplier Database Application

The supplier–parts catalog database must maintain information about suppliers, the parts they supply, and the prices charged by each supplier for specific parts. Each supplier in the system is uniquely identified by a supplier ID (sid), along with additional details such as supplier name and address. Each part in the system is uniquely identified by a part ID (pid) and includes descriptive attributes such as part name and color.

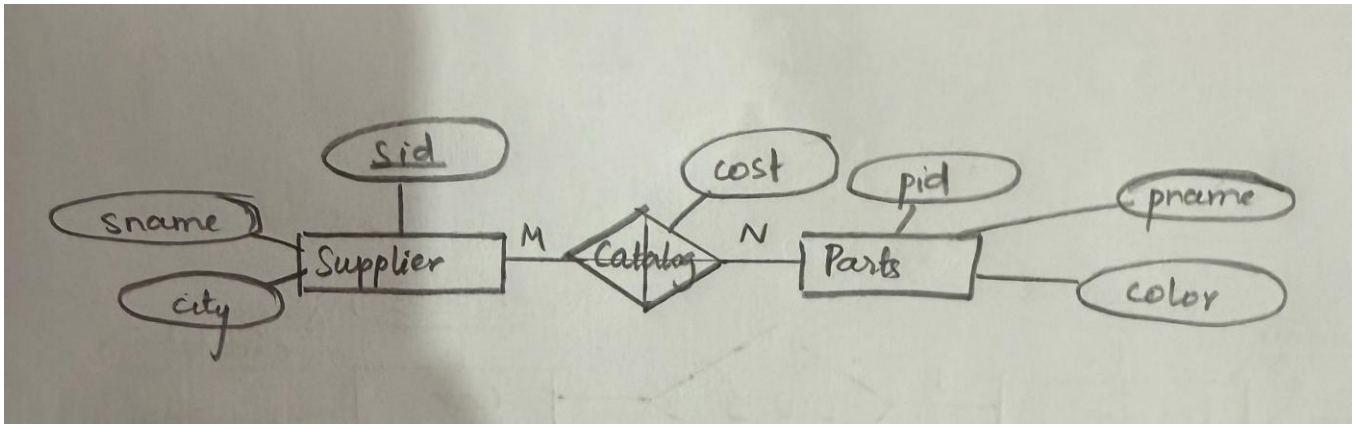
The database must record pricing information through a catalog relationship that links suppliers and parts. This relationship captures the cost at which a particular supplier supplies a specific part. A supplier may supply multiple parts, and a part may be supplied by multiple suppliers, resulting in a many-to-many relationship between suppliers and parts. This relationship is resolved using the CATALOG relation.

Each catalog entry must reference an existing supplier and an existing part. The combination of supplier ID and part ID must be unique in the catalog, ensuring that no duplicate price entries exist for the same supplier–part combination. The cost attribute must be a nonnegative real value representing the price charged by the supplier for that part.

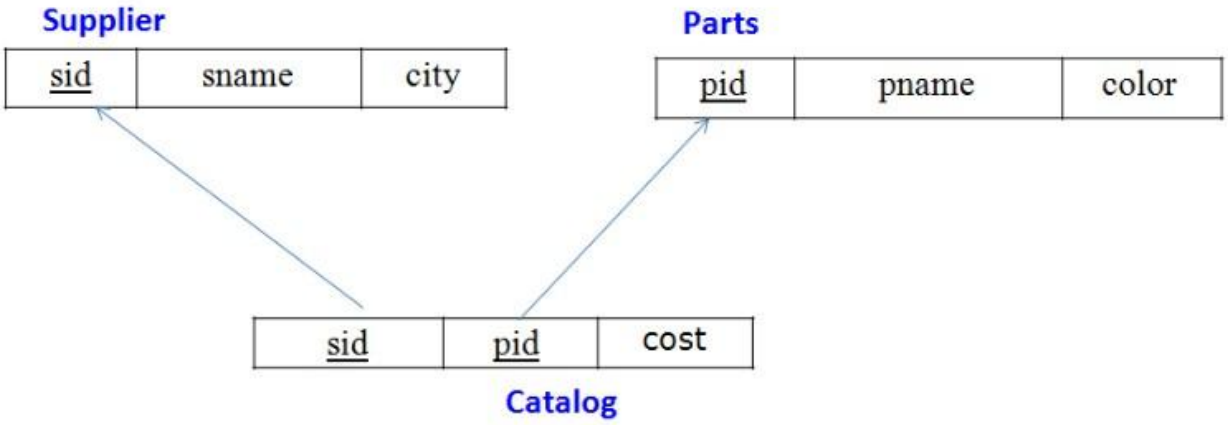
The database must enforce referential integrity such that no catalog record can exist unless the corresponding supplier and part records are already present in the system. Deletion of supplier or part records that are referenced in the catalog must be restricted or handled using controlled cascading or archival mechanisms to preserve historical pricing consistency.

Overall, the system must maintain accurate and consistent relationships between suppliers, parts, and catalog entries, enabling reliable retrieval of supplier details, part information, and pricing data for procurement, inventory management, and administrative purposes.

ENTITY RELATONSHIP DIAGRAM:



SCHEMA DIAGRAM:



SUPPLIERS(sid: integer, sname: string, address: string)
PARTS(pid: integer, pname: string, color: string)
CATALOG(sid: integer, pid: integer, cost: real)
The Catalog relation lists the prices charged for parts by Suppliers.

Create database

create database

supplierdb; use

supplierdb; **Create**

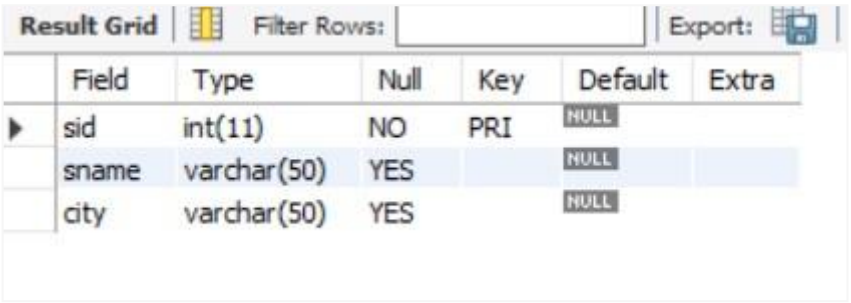
Tables:

create table suppliers
(sid int primary

```
key,     sname
varchar(50),
   city varchar(50)
);

create table parts (
pid int primary
key,     pname
varchar(50),
   color varchar(20)
);

create table
catalog (     sid
int,     pid int,
   cost int,
   foreign key (sid) references suppliers(sid),
   foreign key (pid) references parts(pid)
);
```

Structure of the table

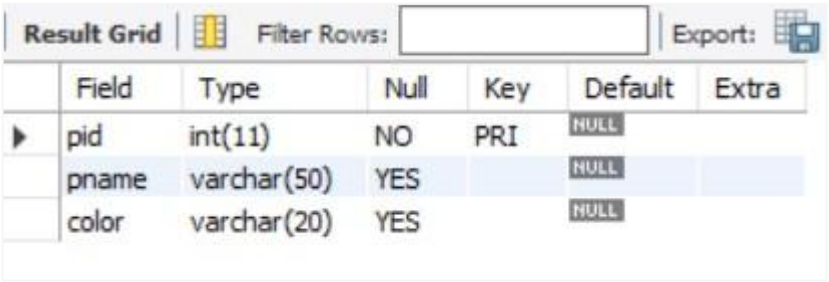
desc suppliers;



Result Grid | Filter Rows: | Export:

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

desc parts;



Result Grid | Filter Rows: | Export:

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

desc catalog;

Result Grid | Filter Rows: | Export:

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

Inserting Values to the table insert
into suppliers (sid, sname, city)
values
(10001, 'Acme Widget', 'Bangalore'),
(10002, 'Johns', 'Kolkata'),
(10003, 'Vimal', 'Mumbai'),
(10004, 'Reliance', 'Delhi');
select * from suppliers;

Result Grid | Filter Rows:

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

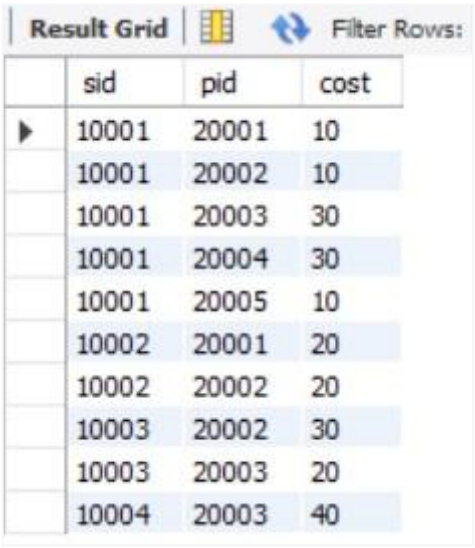
insert into parts (pid, pname, color) values
(20001, 'Book', 'Red'),
(20002, 'Pen', 'Red'),
(20003, 'Pencil', 'Green'),
(20004, 'Mobile', 'Green'),
(20005, 'Charger', 'Black');
select * from parts;

Result Grid | Filter Rows:

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

insert into catalog (sid, pid, cost) values

(10001, 20001, 10),
(10001, 20002, 10),
(10001, 20003, 30),
(10001, 20004, 30),
(10001, 20005, 10),
(10002, 20001, 20),
(10002, 20002, 20),
(10003, 20002, 30),
(10003, 20003, 20),
(10004, 20003, 40);

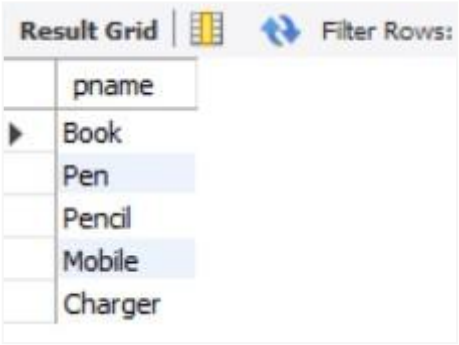


Result Grid | Filter Rows:

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

Query 1:

Find the pnames of parts for which there is some supplier. select distinct p.pname from parts p join catalog c on p.pid = c.pid;



Result Grid | Filter Rows:

pname
Book
Pen
Pencil
Mobile
Charger

Query 2:

Find the snames of suppliers who supply every part. select s.sname from suppliers s join catalog c on s.sid = c.sid group by s.sid, s.sname having count(distinct c.pid) = (select count(*) from parts);

Query 3:

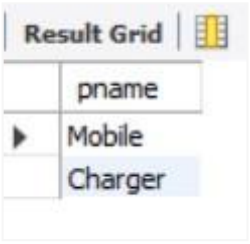
Find the snames of suppliers who supply every red part.

select s.sname from suppliers s join catalog c on s.sid = c.sid where c.pid in (select pid from parts where color = 'Red') group by s.sid, s.sname having count(distinct c.pid) = (select count(*) from parts where color = 'Red');

Query 4:

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

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



Query 5:

Find the sids of suppliers who charge more for some part than the average cost of that part (averaged over all the suppliers who supply that part).

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

Result Grid

sid
10002
10003
10004

Query 6:

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

```
        select p.pname, s.sname from parts p join catalog c on p.pid
= c.pid join suppliers s on c.sid = s.sid where (c.pid, c.cost) in
(select pid, max(cost)      from catalog group by pid);
```

Result Grid | Filter Rows:

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

Experiment 8: More Queries on Supplier Database

Queries:

Query 1:

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

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

Result Grid | Filter Rows:

sname	pname	cost
Reliance	Pencil	40

Query 2:

Find suppliers who do NOT supply any red parts.

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

Result Grid | Filter Rows:

sid	sname	city
10004	Reliance	Delhi

Query 3:

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

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

Result Grid | Filter Rows:

sname	totalvalue
Acme Widget	90
Johns	40
Vimal	50
Reliance	40

Query 4:

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

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

Result Grid | Filter Rows:

sid	sname
10001	Acme Widget

Query 5:

List suppliers who offer the cheapest cost for each part.

select s.sname, p.pname, c.cost from catalog c join suppliers s on c.sid = s.sid join parts p on c.pid = p.pid where c.cost = (select min(c2.cost) from catalog c2 where c2.pid = c.pid);

Result Grid | Filter Rows:

sname	pname	cost
Acme Widget	Book	10
Acme Widget	Pen	10
Acme Widget	Mobile	30
Acme Widget	Charger	10
Vimal	Pencil	20

Query 6:

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

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

Result Grid | Filter Rows:

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

Query 7:

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

```
create view most_expensive_supplier as select s.sname, p.pname, c.cost from catalog c join suppliers s on c.sid = s.sid join parts p on c.pid = p.pid where c.cost = (select max(c2.cost) from catalog c2 where c2.pid = c.pid);
```

Result Grid | Filter Rows:

sname	pname	cost
Acme Widget	Mobile	30
Acme Widget	Charger	10
Johns	Book	20
Vimal	Pen	30
Reliance	Pencil	40

Query 8:

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

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

Query 9:

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

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


//
DELIMITER ;
```

Experiment 9: NO SQL Student Database

Create Database

```
test> db.createCollection("student");
{ ok: 1 }
```

Inserting values

```
test> db.student.insert({RollNo:1,Age:21,Cont:9876,email:"antara.de9@gmail.com"});
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6936655093569dc27263b112') }
}
test> db.student.insert({RollNo:2,Age:22,Cont:9976,email:"anushka.de9@gmail.com"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6936656893569dc27263b113') }
}
test> db.student.insert({RollNo:3,Age:21,Cont:5576,email:"anubhav.de9@gmail.com"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6936658293569dc27263b114') }
}
test> db.student.insert({RollNo:4,Age:20,Cont:4476,email:"pani.de9@gmail.com"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6936659a93569dc27263b115') }
}
test> db.student.insert({RollNo:10,Age:23,Cont:2276,email:"rekha.de9@gmail.com"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693665af93569dc27263b116') }
}
```

View values

```
test> db.student.find()
[
  {
    _id: ObjectId('6936655093569dc27263b112'),
    RollNo: 1,
    Age: 21,
    Cont: 9876,
    email: 'antara.de9@gmail.com'
  },
  {
    _id: ObjectId('6936656893569dc27263b113'),
    RollNo: 2,
    Age: 22,
    Cont: 9976,
    email: 'anushka.de9@gmail.com'
  },
  {
    _id: ObjectId('6936658293569dc27263b114'),
    RollNo: 3,
    Age: 21,
    Cont: 5576,
    email: 'anubhav.de9@gmail.com'
  },
  {
    _id: ObjectId('6936659a93569dc27263b115'),
    RollNo: 4,
    Age: 20,
    Cont: 4476,
    email: 'pani.de9@gmail.com'
  },
  {
    _id: ObjectId('693665af93569dc27263b116'),
    RollNo: 10,
    Age: 23,
    Cont: 2276,
    email: 'rekha.de9@gmail.com'
  }
]
```

<u>Queries:</u>

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

```
test> db.student.update({ RollNo: 10 }, { $set: { email: "Abhinav@gmail.com" } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

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

```
test> db.student.insert({RollNo:11,Age:22,Name:"ABC",Cont:2276,email:"rea.de9@gmail.com"});
{ncaught:
  acknowledged: true,ed token (4:0)
  insertedIds: { '0': ObjectId('6936690093569dc27263b118') }
} 2 |
```

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

```
  {
    _id: ObjectId('6936690093569dc27263b118'),
    RollNo: 11,
    Age: 22,
    Name: 'FEM',
    Cont: 2276,
    email: 'rea.de9@gmail.com'
  }
]
```

3. Drop the table

```
test> db.student.drop();
true
test> db.student.find();
```

Experiment 10: NO SQL Customer Database

Create Database

```
test> db.createCollection("customer");
{ ok: 1 }
```

Create table

```
test> db.customer.insert({"Cust_id":1,"Acc_Bal":5000,"Acc_Type":"Savings"});
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693faa69c75a2a4b111e2626') }
}
test> db.customer.insert({"Cust_id":2,"Acc_Bal":7500,"Acc_Type":"Savings"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693faa75c75a2a4b111e2627') }
}
test> db.customer.insert({"Cust_id":3,"Acc_Bal":200,"Acc_Type":"Transactions"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693faaa4c75a2a4b111e2628') }
}
test> db.customer.insert({"Cust_id":4,"Acc_Bal":10000,"Acc_Type":"Savings"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693faabac75a2a4b111e2629') }
}
test> db.customer.insert({"Cust_id":5,"Acc_Bal":1000,"Acc_Type":"Transactions"});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('693faacbc75a2a4b111e262a') }
}
```

Queries

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

```
Atlas atlas-3acvm6-shard-0 [primary] DBME_DEMO> db.Customers.find({ Acc_Bal: { $gt: 1200 }, Acc_Type:
"Z" })
[
  {
    _id: ObjectId('693f8376701108f7231e262c'),
    Cust_id: 1,
    Acc_Bal: 1500,
    Acc_Type: 'Z'
  },
  {
    _id: ObjectId('693f8376701108f7231e262e'),
    Cust_id: 3,
    Acc_Bal: 2000,
    Acc_Type: 'Z'
  },
  {
    _id: ObjectId('693f8376701108f7231e2630'),
    Cust_id: 5,
    Acc_Bal: 1800,
    Acc_Type: 'Z'
  }
]
```

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

```
Atlas atlas-3acvm6-shard-0 [primary] DBME_DEMO> db.Customers.aggregate([
...   {
...     $group: {
...       _id: "$Cust_id",
...       Min_Bal: { $min: "$Acc_Bal" },
...       Max_Bal: { $max: "$Acc_Bal" }
...     }
...   }
... ])
...
[
  { _id: 5, Min_Bal: 1800, Max_Bal: 1800 },
  { _id: 2, Min_Bal: 800, Max_Bal: 800 },
  { _id: 4, Min_Bal: 500, Max_Bal: 500 },
  { _id: 3, Min_Bal: 2000, Max_Bal: 2000 },
  { _id: 1, Min_Bal: 1500, Max_Bal: 1500 }
]
```

3. Export the created collection into local file system

```
C:\Users\BMSCECSE-L3-27>mongoexport --uri="mongodb+srv://annapurna_cs24_user:anusm060706@cluster0.eavk
tnp.mongodb.net/DBME_DEMO" --collection=New_Customer --type=csv --fields=Cust_id,Acc_Bal,Acc_Type --ou
t="C:\Users\BMSCECSE-L3-27\Downloads\New_Customer_Export.csv"
2025-12-15T09:35:30.774+0530    connected to: mongodb+srv://[**REDACTED**]@cluster0.eavktnp.mongodb.ne
t/DBME_DEMO
2025-12-15T09:35:30.856+0530    exported 5 records
```

4. Drop the table

```
Atlas atlas-3acvm6-shard-0 [primary] DBME_DEMO> db.Customers.drop()
true
```

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

```
C:\Users\BMSCECSE-L3-27>mongoimport --uri="mongodb+srv://annapurna_cs24_user:anusm060706@cluster0.eavk
tnp.mongodb.net/DBME_DEMO" --collection=New_Customer --type=csv --headerline --file="C:\Users\BMSCECSE
-L3-27\Downloads\New_Customer_Export.csv"
2025-12-15T09:35:58.160+0530    connected to: mongodb+srv://[**REDACTED**]@cluster0.eavktnp.mongodb.ne
t/DBME_DEMO
2025-12-15T09:35:58.234+0530    5 document(s) imported successfully. 0 document(s) failed to import.
```

EXPERIMENT 11: NO SQL RESTAURANT DATABASE

Create Database

```
test> db.createCollection( "restaurants")
{ ok: 1 }
```

Inserting Values

```
test> db.restaurants.insertMany([
... { name: "Meghna Foods", town: "Jayanagar", cuisine: "Indian", score: 8, address: { zipcode: "10001", street: "Jayanagar"
... } },
... { name: "Empire", town: "MG Road", cuisine: "Indian", score: 7, address: { zipcode: "10100", street: "MG Road" } },
... { name: "Chinese WOK", town: "Indiranagar", cuisine: "Chinese", score: 12, address: { zipcode: "20000", street: "Indiranagar" } },
... { name: "Kyotos", town: "Majestic", cuisine: "Japanese", score: 9, address: { zipcode: "10300", street: "Majestic" } },
... { name: "WOW Momos", town: "Malleshwaram", cuisine: "Indian", score: 5, address: { zipcode: "10400", street: "Malleshwaram" }
... } ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('693fa0ac01c1a3091e1e2621'),
    '1': ObjectId('693fa0ac01c1a3091e1e2622'),
    '2': ObjectId('693fa0ac01c1a3091e1e2623'),
    '3': ObjectId('693fa0ac01c1a3091e1e2624'),
    '4': ObjectId('693fa0ac01c1a3091e1e2625')
  }
}
```

Queries:

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

```
test> db.restaurants.find({})
[
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2621'),
    name: 'Meghna Foods',
    town: 'Jayanagar',
    cuisine: 'Indian',
    score: 8,
    address: { zipcode: '10001', street: 'Jayanagar' }
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2622'),
    name: 'Empire',
    town: 'MG Road',
    cuisine: 'Indian',
    score: 7,
    address: { zipcode: '10100', street: 'MG Road' }
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2623'),
    name: 'Chinese WOK',
    town: 'Indiranagar',
    cuisine: 'Chinese',
    score: 12,
    address: { zipcode: '20000', street: 'Indiranagar' }
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2624'),
    name: 'Kyotos',
    town: 'Majestic',
    cuisine: 'Japanese',
    score: 9,
    address: { zipcode: '10300', street: 'Majestic' }
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2625'),
    name: 'WOW Momos',
    town: 'Malleshwaram',
    cuisine: 'Indian',
    score: 5,
    address: { zipcode: '10400', street: 'Malleshwaram' }
  }
]
```

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

```
test> db.restaurants.find({}).sort({ name: -1 })
[
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2625'),
    name: 'WOW Momos',
    town: 'Malleshwaram',
    cuisine: 'Indian',
    score: 5,
    address: { zipcode: '10400', street: 'Malleshwaram' }
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2621'),
    name: 'Meghna Foods',
    town: 'Jayanagar',
    cuisine: 'Indian',
    score: 8,
    address: { zipcode: '10001', street: 'Jayanagar' }
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2624'),
    name: 'Kyotos',
    town: 'Majestic',
    cuisine: 'Japanese',
    score: 9,
    address: { zipcode: '10300', street: 'Majestic' }
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2622'),
    name: 'Empire',
    town: 'MG Road',
    cuisine: 'Indian',
    score: 7,
    address: { zipcode: '10100', street: 'MG Road' }
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2623'),
    name: 'Chinese WOK',
    town: 'Indiranagar',
    cuisine: 'Chinese',
    score: 12,
    address: { zipcode: '20000', street: 'Indiranagar' }
  }
]
```

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

```
test> db.restaurants.find({ "score": { $lte: 10 } }, { _id: 1, name: 1, town: 1, cuisine: 1 })
[
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2621'),
    name: 'Meghna Foods',
    town: 'Jayanagar',
    cuisine: 'Indian'
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2622'),
    name: 'Empire',
    town: 'MG Road',
    cuisine: 'Indian'
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2624'),
    name: 'Kyotos',
    town: 'Majestic',
    cuisine: 'Japanese'
  },
  {
    _id: ObjectId('693fa0ac01c1a3091e1e2625'),
    name: 'WOW Momos',
    town: 'Malleshwaram',
    cuisine: 'Indian'
  }
]
test>
```

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

```
test> db.restaurants.aggregate([ { $group: { _id: "$name", average_score: { $avg: "$score" } } }
... ])
[
  { _id: 'Kyotos', average_score: 9 },
  { _id: 'Meghna Foods', average_score: 8 },
  { _id: 'Chinese WOK', average_score: 12 },
  { _id: 'Empire', average_score: 7 },
  { _id: 'WOW Momos', average_score: 5 }
]
test>
```

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

```
test> db.restaurants.find({ "address.zipcode": /^10/ }, { name: 1, "address.street": 1, _id: 0 })
[
  { name: 'Meghna Foods', address: { street: 'Jayanagar' } },
  { name: 'Empire', address: { street: 'MG Road' } },
  { name: 'Kyotos', address: { street: 'Majestic' } },
  { name: 'WOW Momos', address: { street: 'Malleshwaram' } }
]
test>
```

LEETCODE PRACTICE QUESTION-1

Table: `Products`

Code

MySQL Auto

```mysql
# Write your MySQL query statement below
SELECT sell_date,
       COUNT(DISTINCT product) AS num_sold,
       GROUP_CONCAT(DISTINCT product ORDER BY product ASC) AS products
FROM Activities
GROUP BY sell_date
ORDER BY sell_date;
```

```
| product_id    | int     |
| order_date    | date    |
| unit          | int     |
+---------------+---------+
This table may have duplicate rows.
product_id is a foreign key (reference column) to the Products table.
unit is the number of products ordered in order_date.
```

Write a solution to get the names of products that have at least `100` units ordered in **February 2020** and their amount.

Return the result table in **any order**.

OUTPUT

Code

MySQL Auto

```mysql
# Write your MySQL query statement below
SELECT sell_date,
       COUNT(DISTINCT product) AS num_sold,
       GROUP_CONCAT(DISTINCT product ORDER BY product ASC) AS products
FROM Activities
GROUP BY sell_date
ORDER BY sell_date;
```

LEETCODE PRACTICE QUESTION -2

Table `Activities`:

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| sell_date   | date    |
| product     | varchar |
+-------------+---------+
```

There is no primary key (column with unique values) for this table. It may contain duplicates.
Each row of this table contains the product name and the date it was sold in a market.

Write a solution to find for each date the number of different products sold and their names.

The sold products names for each date should be sorted lexicographically.

Return the result table ordered by `sell_date`.

OUTPUT

</> Code

MySQL Auto

```mysql
# Write your MySQL query statement below
SELECT p.product_name, SUM(o.unit) AS unit
FROM Orders o
JOIN Products p ON o.product_id = p.product_id
WHERE o.order_date BETWEEN '2020-02-01' AND '2020-02-29'
GROUP BY o.product_id, p.product_name
HAVING SUM(o.unit) >= 100;
```