Assignment solution

Subject: Database Management System

1) Given relation schema as below
Employee(emp_id, name, address, telephone, salary, age)
Works_on (emp_id, project_id, join_date)
Project(project_id, project_name, city,duration, budget)
Write the SQL commands for the following.

1. **Insert new record in project relation.**

```
INSERT INTO Project
VALUES (102, 'DBMS project', 'kathmandu', 2, 55000);
```

2. **Find the name of employees with the name of project they work on.**

```
SELECT Employee.name, Project.project_name
FROM Employee, Works_on, Project
WHERE Employee.emp_id = Works_on.emp_id
 AND Works_on.project_id = Project.project_id;
```

3. **Find the name and city of that project on which salary of employee is greater than or equal to 20000.**

```
SELECT Project.project_name, Project.city
FROM Employee, Works_on, Project
WHERE Employee.emp_id = Works_on.emp_id
AND Works_on.project_id = Project.project_id
AND Employee.salary >= 20000;
```

4. **List the name of employees whose name starts with “m” and ends with “a”**

```
SELECT name
FROM Employee
WHERE name LIKE 'm%a';
```

5. **Find the employee name and project name of those employees who living in address Pokhara.**

```
SELECT Employee.name, Project.project_name
FROM Employee, Works_on, Project
WHERE Employee.emp_id = Works_on.emp_id
 AND Works_on.project_id = Project.project_id
 AND Employee.address = 'Pokhara';
```

6. **List name of employee whose age is greater than average age of all employees.**
```
SELECT name
FROM Employee
WHERE age > (SELECT AVG(age) FROM Employee);
```

7. **List employee id of all employees whose age is greater than minimum budget of all projects.**
```
SELECT emp_id
FROM Employee
WHERE age > (SELECT MIN(budget) FROM Project);
```

8. **List employee id of all employees who joint project on "05/01/2015"**
```
SELECT emp_id
FROM Works_on
WHERE join_date = '2015-05-01';
```

9. **List the name of employees whose name starts with N or with K**
```
SELECT name
FROM Employee
WHERE name LIKE 'N%' OR name LIKE 'K%';
```

10. **Display the project id with maximum budget.**
```
SELECT project_id
FROM Project
WHERE budget = (SELECT MAX(budget) FROM Project);
```

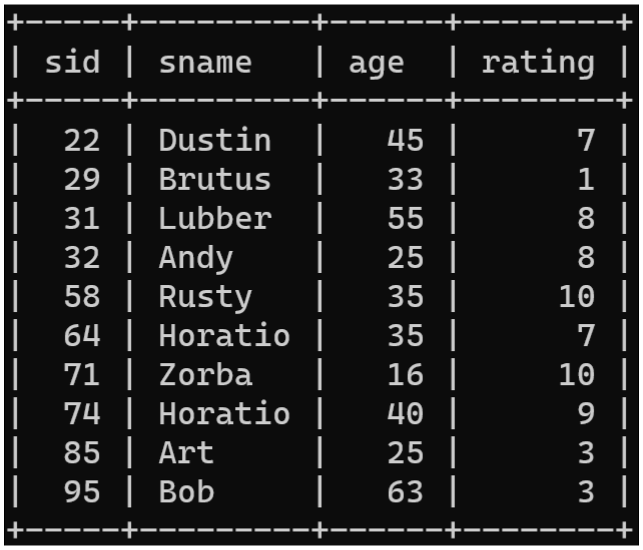
2) Given Relational Schema as below.

Sailors (sid, sname, age,rating)
Boats(bid, bname,color)
Reserves(sid,bid,day)
Write the SQL for the following.

Note: For your convenience, rows in the above relation are inserted as needed, facilitating easier demonstration of the output. However, during exams, it is not mandatory to insert all rows unless prompted by specific questions.

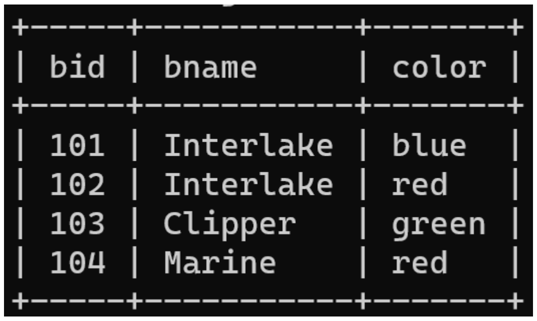
The output is generated to verify whether the query conforms to the question's requirements, ensuring its relevance and accuracy. But it is not required in exam.

Sailors



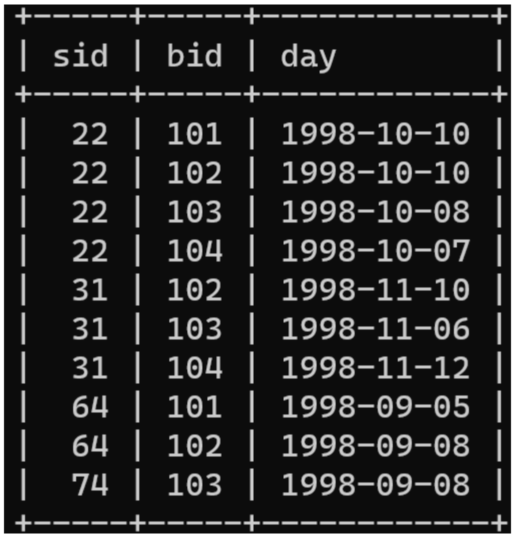
```
+-----+---------+------+--------+
| sid | sname   | age  | rating |
+-----+---------+------+--------+
|  22 | Dustin  |   45 |      7 |
|  29 | Brutus  |   33 |      1 |
|  31 | Lubber  |   55 |      8 |
|  32 | Andy    |   25 |      8 |
|  58 | Rusty   |   35 |     10 |
|  64 | Horatio |   35 |      7 |
|  71 | Zorba   |   16 |     10 |
|  74 | Horatio |   40 |      9 |
|  85 | Art     |   25 |      3 |
|  95 | Bob     |   63 |      3 |
+-----+---------+------+--------+
```

Boats



```
+-----+-----------+-------+
| bid | bname     | color |
+-----+-----------+-------+
| 101 | Interlake | blue  |
| 102 | Interlake | red   |
| 103 | Clipper   | green |
| 104 | Marine    | red   |
+-----+-----------+-------+
```

Reserves

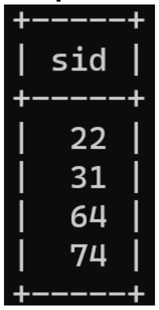


```
+-----+-----+------------+
| sid | bid | day        |
+-----+-----+------------+
|  22 | 101 | 1998-10-10 |
|  22 | 102 | 1998-10-10 |
|  22 | 103 | 1998-10-08 |
|  22 | 104 | 1998-10-07 |
|  31 | 102 | 1998-11-10 |
|  31 | 103 | 1998-11-06 |
|  31 | 104 | 1998-11-12 |
|  64 | 101 | 1998-09-05 |
|  64 | 102 | 1998-09-08 |
|  74 | 103 | 1998-09-08 |
+-----+-----+------------+
```

1. **Find the sid of the sailor who have reserved red or green boat.**

```
SELECT distinct Sailors.sid
FROM Sailors, Boats, Reserves
WHERE Sailors.sid = Reserves.sid
AND Reserves.bid = Boats.bid
AND (Boats.color = 'red' OR Boats.color = 'green');
```

Output

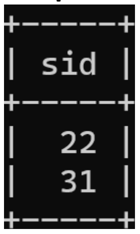


```
+-----+
| sid |
+-----+
|  22 |
|  31 |
|  64 |
|  74 |
+-----+
```

2. **Find sid's of sailors who've reserved a red and a green boat.**

```
SELECT Sailors.sid
FROM Sailors, Boats, Reserves
WHERE Sailors.sid = Reserves.sid
AND Reserves.bid = Boats.bid
AND Boats.color = 'red'
INTERSECT
SELECT Sailors.sid
FROM Sailors, Boats, Reserves
WHERE Sailors.sid = Reserves.sid
AND Reserves.bid = Boats.bid
AND Boats.color = 'green';
```

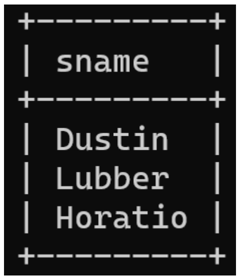
Output



```
+-----+
| sid |
+-----+
|  22 |
|  31 |
+-----+
```

3. **Find names of sailors who have reserved boat id 103:**

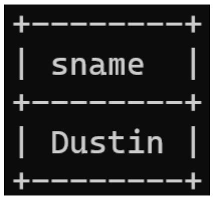
```
SELECT Sailors.sname
FROM Sailors, Reserves
WHERE Sailors.sid = Reserves.sid
AND Reserves.bid = 103;
```



```
+---------+
| sname   |
+---------+
| Dustin  |
| Lubber  |
| Horatio |
+---------+
```

4. **Find sailors who've reserved all boats**

```
SELECT Sailors. sname
FROM Sailors
WHERE NOT EXISTS
(SELECT Boats.bid FROM Boats
 WHERE NOT EXISTS
( SELECT Reserves.bid  FROM Reserves  WHERE Reserves.bid=Boats.bid AND
Reserves.sid=Sailors.sid));
```



```
+--------+
| sname  |
+--------+
| Dustin |
+--------+
```

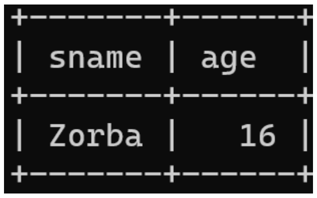
This SQL query retrieves the names of sailors who have reserved all boats. Here's an explanation of each part of the query:

- ✓ **SELECT Sailors.sname:** This is the main query that selects the sailor names from the Sailors table.
- ✓ **FROM Sailors:** Specifies that the data is being selected from the Sailors table.
- ✓ **WHERE NOT EXISTS (SELECT Boats.bid FROM Boats WHERE NOT EXISTS (SELECT Reserves.bid FROM Reserves WHERE Reserves.bid=Boats.bid AND Reserves.sid=Sailors.sid)):** This is a subquery that filters the sailors who have reserved all boats.
- ✓ The innermost subquery **(SELECT Reserves.bid FROM Reserves WHERE Reserves.bid=Boats.bid AND Reserves.sid=Sailors.sid)** checks for each sailor (Sailors.sid) whether there exists a reservation for a boat (Reserves.bid) that matches both the sailor and the boat.
- ✓ The middle subquery **(SELECT Boats.bid FROM Boats WHERE NOT EXISTS (SELECT Reserves.bid FROM Reserves WHERE Reserves.bid=Boats.bid AND Reserves.sid=Sailors.sid))** checks for each boat (Boats.bid) whether there does not exist a reservation by the current sailor for that boat.
- ✓ The outer query **WHERE NOT EXISTS (SELECT Boats.bid FROM Boats WHERE NOT EXISTS ...)** checks if there are no boats for which a reservation does not exist for the current sailor.
- ✓ Finally, the outermost WHERE clause ensures that only sailors who do not meet the condition of not having reserved all boats are selected.

Overall, the query ensures that it selects sailors who have made reservations for all available boats.

5. **Find the name and age of youngest sailor**

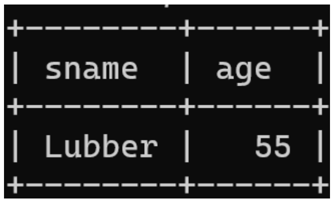
SELECT Sailors.sname, Sailors.age
FROM Sailors
WHERE Sailors.age = (SELECT MIN(Sailors.age)
 FROM Sailors Sailors);



```
+-------+------+
| sname | age  |
+-------+------+
| Zorba |   16 |
+-------+------+
```

6. **Find name and age of the oldest sailor(s) with rating >7**

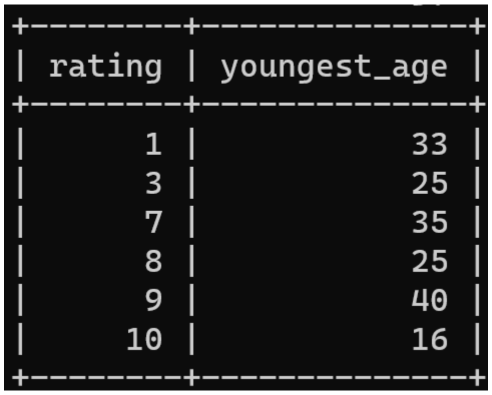
```
SELECT Sailors.sname, Sailors.age
FROM Sailors
WHERE Sailors.rating > 7
AND Sailors.age = (
   SELECT MAX(Sailors.age)
   FROM Sailors
   WHERE Sailors.rating > 7
);
```



```
+--------+------+
| sname  | age  |
+--------+------+
| Lubber |   55 |
+--------+------+
```

7. **Find the age of the youngest sailor for each rating level**

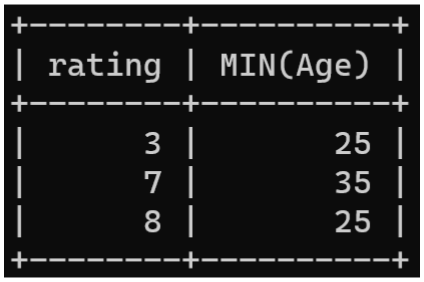
SELECT rating, MIN(Age) as youngest_age
FROM Sailors
GROUP BY rating;



```
+--------+--------------+
| rating | youngest_age |
+--------+--------------+
|      1 |           33 |
|      3 |           25 |
|      7 |           35 |
|      8 |           25 |
|      9 |           40 |
|     10 |           16 |
+--------+--------------+
```

8. **Find the age of the youngest sailor with age >= 18 for each rating level with at least 2 such sailors**

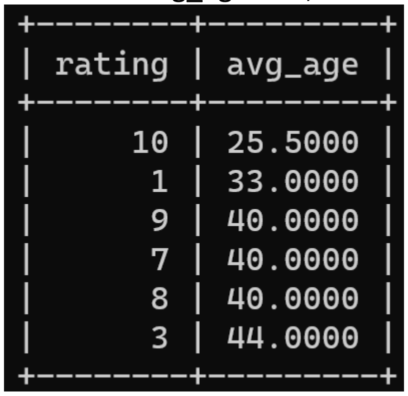
```
SELECT rating, MIN(Age)
FROM Sailors
WHERE age>= 18
GROUP BY rating
HAVING COUNT(*)>= 2;
```



```
+--------+----------+
| rating | MIN(Age) |
+--------+----------+
|      3 |       25 |
|      7 |       35 |
|      8 |       25 |
+--------+----------+
```

9. **Find the average age for each rating, and order results in ascending order on avg.**

```
SELECT rating, AVG(age) AS avg_age
FROM Sailors
GROUP BY rating
ORDER BY avg_age ASC;
```



```
+--------+---------+
| rating | avg_age |
+--------+---------+
|     10 | 25.5000 |
|      1 | 33.0000 |
|      9 | 40.0000 |
|      7 | 40.0000 |
|      8 | 40.0000 |
|      3 | 44.0000 |
+--------+---------+
```

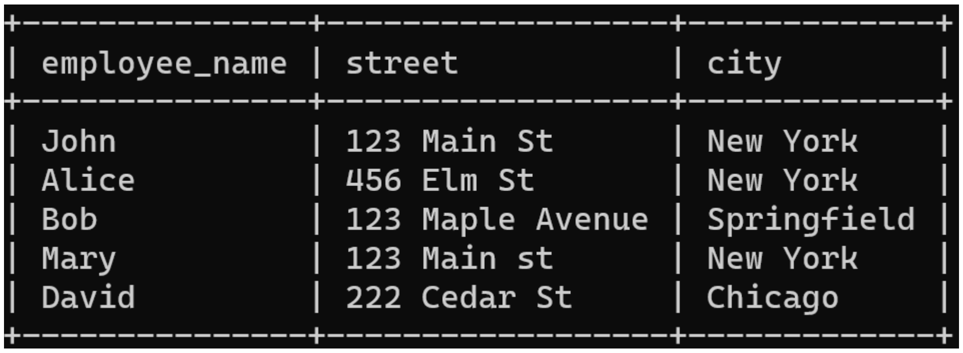
3) Consider the relational database of figure given below, where primay keys are underlined. Given an expression in SQL for each of the following queries.

Employee(employee_name,street,city)
Works(employee_name,company_name,salary)
Company(company_name,ciy)
Manages(employee_name,manager_name)

Note: For your convenience, rows in the above relation are inserted as needed, facilitating easier demonstration of the output. However, during exams, it is not mandatory to insert all rows unless prompted by specific questions.

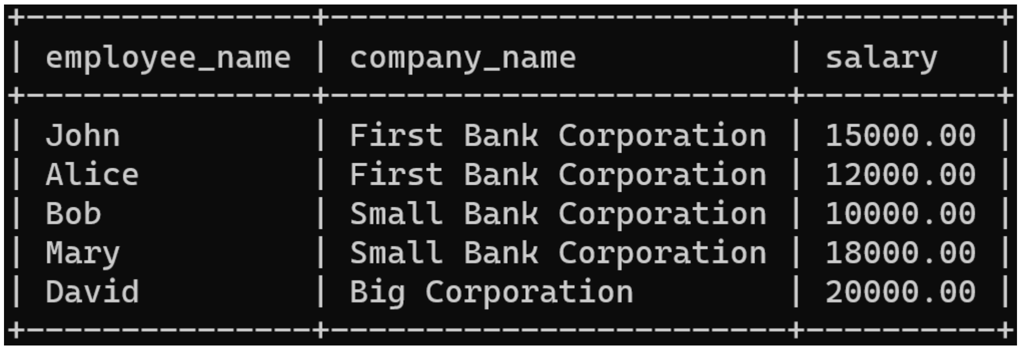
The output is generated to verify whether the query conforms to the question's requirements, ensuring its relevance and accuracy. But it is not required in exam.

Employee



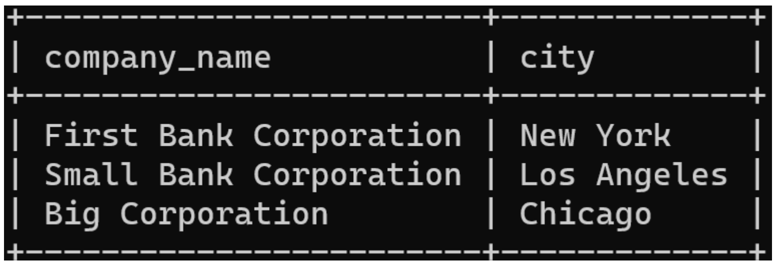
```
+---------------+------------------+-------------+
| employee_name | street           | city        |
+---------------+------------------+-------------+
| John          | 123 Main St      | New York    |
| Alice         | 456 Elm St       | New York    |
| Bob           | 123 Maple Avenue | Springfield |
| Mary          | 123 Main st      | New York    |
| David         | 222 Cedar St     | Chicago     |
+---------------+------------------+-------------+
```

Works



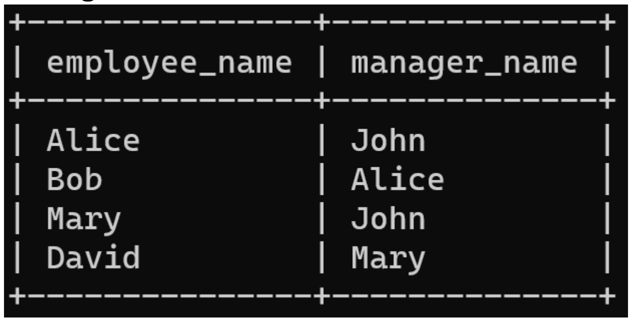
```
+---------------+------------------------+----------+
| employee_name | company_name           | salary   |
+---------------+------------------------+----------+
| John          | First Bank Corporation | 15000.00 |
| Alice         | First Bank Corporation | 12000.00 |
| Bob           | Small Bank Corporation | 10000.00 |
| Mary          | Small Bank Corporation | 18000.00 |
| David         | Big Corporation        | 20000.00 |
+---------------+------------------------+----------+
```

company



```
+------------------------+-------------+
| company_name           | city        |
+------------------------+-------------+
| First Bank Corporation | New York    |
| Small Bank Corporation | Los Angeles |
| Big Corporation        | Chicago     |
+------------------------+-------------+
```

Manages

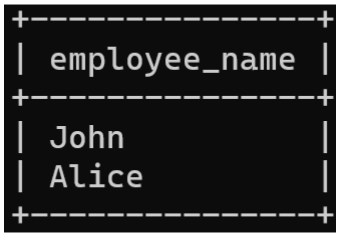


```
+---------------+--------------+
| employee_name | manager_name |
+---------------+--------------+
| Alice         | John         |
| Bob           | Alice        |
| Mary          | John         |
| David         | Mary         |
+---------------+--------------+
```

i. **Find the names of all employees who work for the First Bank Corporation.**

```
SELECT employee_name
FROM Works
WHERE company_name = 'First Bank Corporation';
```

Output

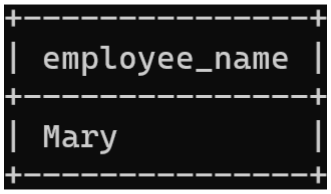


```
+---------------+
| employee_name |
+---------------+
| John          |
| Alice         |
+---------------+
```

ii. **Find the names of all employees who live in the same city and on the same street as do their managers.**

```
SELECT E1.employee_name
FROM Employee AS E1, Employee AS E2, Manages AS M
WHERE E1.employee_name = M.employee_name
  AND E2.employee_name = M.manager_name
  AND E1.street = E2.street
  AND E1.city = E2.city;
```

Output



```
+---------------+
| employee_name |
+---------------+
| Mary          |
+---------------+
```

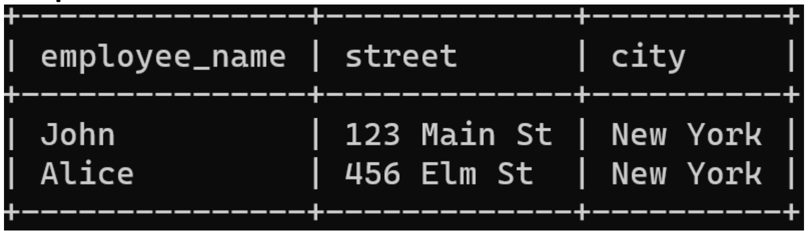
- ✓ **SELECT clause**: Finally, the SELECT clause specifies what data to retrieve. In this case, it selects the employee_name from E1, which represents the name of the employees who meet all the conditions specified in the WHERE clause.
- ✓ **FROM clause:** The query starts by selecting from three tables: Employee (twice, aliased as E1 and E2) and Manages (aliased as M). This is the source of data for the query.
- ✓ **WHERE clause:** The conditions specified in the WHERE clause filter the results. Here's what each condition does:
  - ➢ **E1.employee_name = M.employee_name:** This condition ensures that the employee in E1 is the same as the employee in the Manages table, meaning that E1 is an employee who is managed by someone.
  - ➢ **E2.employee_name = M.manager_name:** This condition ensures that E2 represents the manager of the employee in E1.
  - ➢ **E1.street = E2.street:** This condition checks if both the employee (E1) and their manager (E2) live on the same street.
  - ➢ **E1.city = E2.city:** This condition checks if both the employee (E1) and their manager (E2) live in the same city.

By combining these elements, the query retrieves the names of employees who live in the same city and on the same street as their managers. This is achieved by joining the Employee table with itself (via the E1 and E2 aliases) based on the relationships defined in the Manages table and filtering the results based on the specified conditions.

iii. **Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000 per annum**

```
SELECT Employee.employee_name, Employee.street, Employee.city
FROM Employee, Works
WHERE Employee.employee_name = Works.employee_name
AND Works.company_name = 'First Bank Corporation'
AND Works.salary > 10000;
```

Output

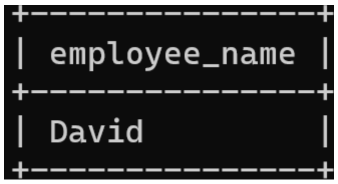


```
+---------------+-------------+----------+
| employee_name | street      | city     |
+---------------+-------------+----------+
| John          | 123 Main St | New York |
| Alice         | 456 Elm St  | New York |
+---------------+-------------+----------+
```

iv. **Find the names of all employees who earn more than every employee of Small Bank Corporation**

```
SELECT employee_name
FROM Works
WHERE salary > ALL
(
   SELECT salary
   FROM Works
   WHERE company_name = 'Small Bank Corporation'
);
```

Output

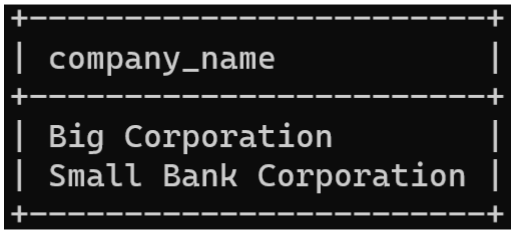


```
+---------------+
| employee_name |
+---------------+
| David         |
+---------------+
```

v. **Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.**

```
SELECT company_name
FROM Works
GROUP BY company_name
HAVING AVG(salary) > (
   SELECT AVG(salary)
   FROM Works
   WHERE company_name = 'First Bank Corporation'
);
```

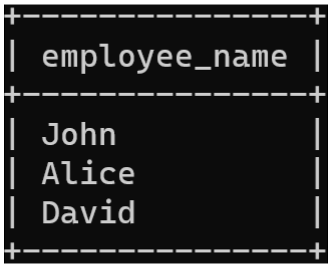
Output



```
+------------------------+
| company_name           |
+------------------------+
| Big Corporation        |
| Small Bank Corporation |
+------------------------+
```

vi. **Find the names of all employees in this database who live in the same city as the company for which they work**

```
SELECT Employee.employee_name
FROM Employee, Works, Company
WHERE Employee.employee_name = Works.employee_name
AND Works.company_name = Company.company_name
AND Employee.city = Company.city;
```

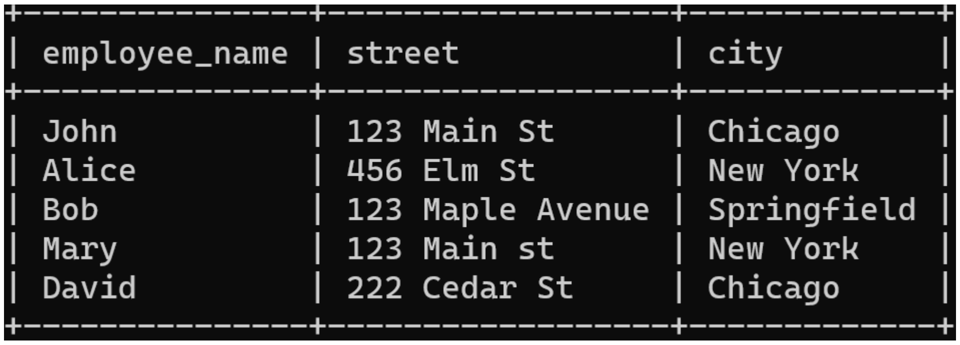


```
+---------------+
| employee_name |
+---------------+
| John          |
| Alice         |
| David         |
+---------------+
```

vii. **modify the databases so that John now lives in Chicago**

UPDATE Employee
SET city = 'Chicago'
WHERE employee_name = 'John';

Now ,Employee table becomes

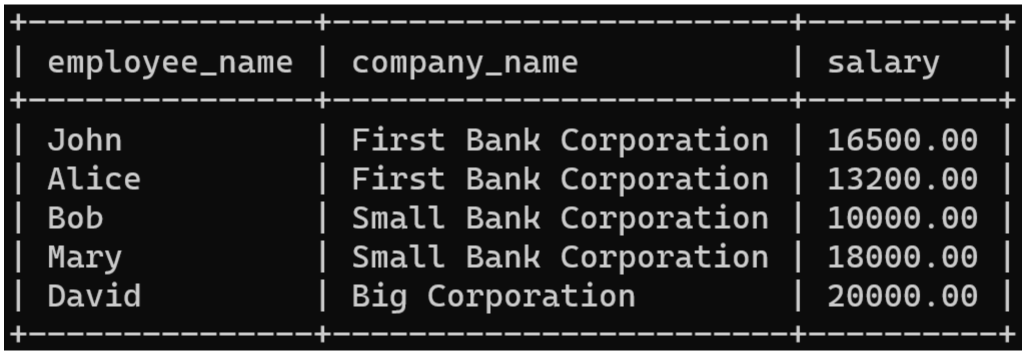


```
+---------------+------------------+-------------+
| employee_name | street           | city        |
+---------------+------------------+-------------+
| John          | 123 Main St      | Chicago     |
| Alice         | 456 Elm St       | New York    |
| Bob           | 123 Maple Avenue | Springfield |
| Mary          | 123 Main st      | New York    |
| David         | 222 Cedar St     | Chicago     |
+---------------+------------------+-------------+
```

viii. **Give all employees of First Bank Corporation a 10 percent raise**

UPDATE Works
SET salary = salary * 1.1
WHERE company_name = 'First Bank Corporation';

Now works table looks like,

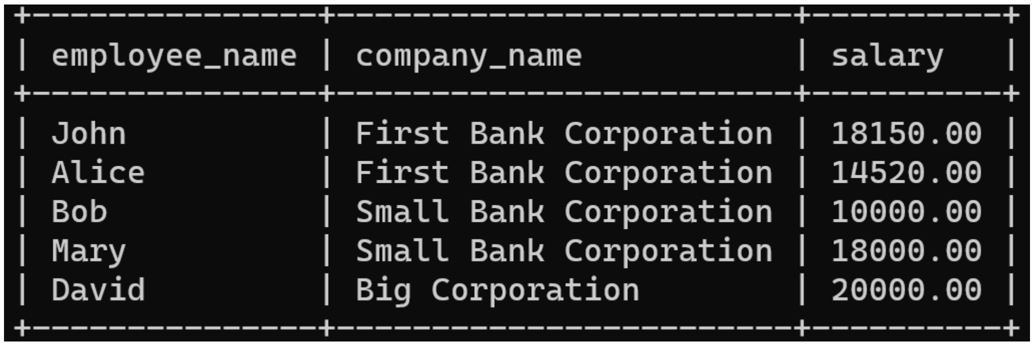


```
+---------------+------------------------+----------+
| employee_name | company_name           | salary   |
+---------------+------------------------+----------+
| John          | First Bank Corporation | 16500.00 |
| Alice         | First Bank Corporation | 13200.00 |
| Bob           | Small Bank Corporation | 10000.00 |
| Mary          | Small Bank Corporation | 18000.00 |
| David         | Big Corporation        | 20000.00 |
+---------------+------------------------+----------+
```

ix. **Give salary of all managers of First Bank Corporation a 10 percent raise.**

UPDATE Works
SET salary = salary * 1.1
WHERE employee_name IN (SELECT manager_name FROM Manages)
AND company_name = 'First Bank Corporation'**;**

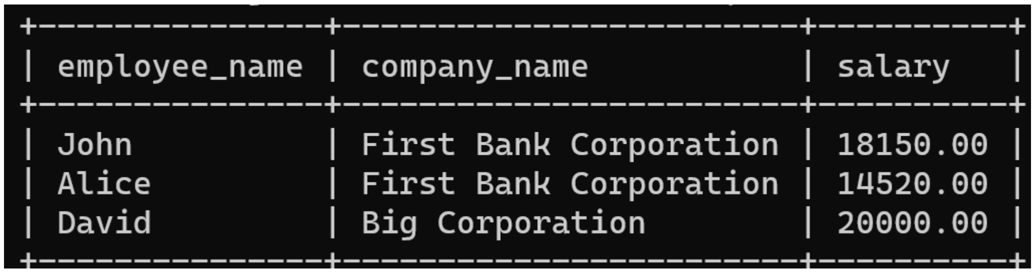
Note that John and Alice both are managers of First Bank corporation


```
+---------------+------------------------+----------+
| employee_name | company_name           | salary   |
+---------------+------------------------+----------+
| John          | First Bank Corporation | 18150.00 |
| Alice         | First Bank Corporation | 14520.00 |
| Bob           | Small Bank Corporation | 10000.00 |
| Mary          | Small Bank Corporation | 18000.00 |
| David         | Big Corporation        | 20000.00 |
+---------------+------------------------+----------+
```

x. **Delete all in the work relation for employees of small bank corporation.**

DELETE FROM Works
WHERE company_name = 'small bank corporation';

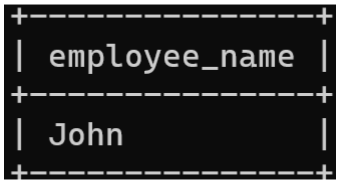
Now ,works relation becomes


```
+---------------+------------------------+----------+
| employee_name | company_name           | salary   |
+---------------+------------------------+----------+
| John          | First Bank Corporation | 18150.00 |
| Alice         | First Bank Corporation | 14520.00 |
| David         | Big Corporation        | 20000.00 |
+---------------+------------------------+----------+
```

xi. **Find all employees who earn more than the average salary of all employees of their company**

```
SELECT employee_name
FROM works a
WHERE salary > (SELECT AVG(salary)
FROM works b
WHERE a.company_name = b.company_name);
```

Output


```
+---------------+
| employee_name |
+---------------+
| John          |
+---------------+
```

4) Consider the following schema

```
Branch(bname, bcity, assets)
Customer(cname, street, ccity)
Depositor (cname, account#)
Account(bname, account#, balance)
Loan(bname, loan#, amount)
Borrower(cname, loan#)
```

Write SQL statement for the following

1. **Find the name of all branch in account relation.**

```
SELECT DISTINCT bname
FROM Account;
```

2. **Finds the names of all branches that have assets greater than at least one branch located in Burnaby.**

```
SELECT bname
FROM Branch
WHERE assets > some (SELECT assets FROM Branch WHERE bcity = 'Burnaby');
```

3. **Find all customers whose street includes the substring "Main".**

```
SELECT cname
FROM Customer
WHERE street LIKE '%Main%';
```

4. **Finds all customers who have a loan and an account at the SFU branch.**

```
SELECT DISTINCT Borrower.cname
FROM Borrower, Loan
WHERE Loan.`loan#` = Borrower.`loan#`
AND Loan.bname = 'SFU'
INTERSECT
SELECT DISTINCT Depositor.cname
FROM Depositor, Account
WHERE Depositor.`account#` = Account.`account#`
AND Account.bname = 'SFU';
```

5. **Find the name and loan number of all customers who have a loan at SFU branch.**

```
SELECT DISTINCT Borrower.cname, Borrower.`loan#`
FROM Borrower, Loan
WHERE Borrower.`loan#` = Loan.`loan#`
AND Loan.bname = 'SFU';
```

6. **Finding all customers who have a loan but not an account at the SFU branch**

```
SELECT DISTINCT Borrower.cname
FROM Borrower, Loan
WHERE Loan.`loan#` = Borrower.`loan#`
AND Loan.bname = 'SFU'
EXCEPT
SELECT DISTINCT Depositor.cname
FROM Depositor, Account
WHERE Depositor.`account#` = Account.`account#`
AND Account.bname = 'SFU';
```

7. **Find the branches whose assets are greater than some branch in Pokhara.**

```
SELECT bname
FROM Branch
WHERE assets > SOME(
   SELECT assets
   FROM branch
   WHERE bcity='Pokhara'
);
```

5) Consider the following relational schema

branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)
Write relational algebraic expressions for the following

1. **Find all loans of over $1200**
   $(\sigma_{amount>1200}(loan))$

2. **Find the loan number for each loan of an amount greater than $1200**
   $\prod_{loan\text{-}number} (\sigma_{amount > 1200} (loan))$

3. **Find the names of all customers who have a loan, an account, or both, from the bank.**
   $\prod_{customer\text{-}name} (borrower) \cup \prod_{customer\text{-}name} (depositor)$

4. **Find the names of all customers who have a loan and an account at bank.**
   $\prod_{customer\text{-}name} (borrower) \cap \prod_{customer\text{-}name} (depositor)$

5. **Find the names of all customers who have a loan at the Perryridge branch.**
$\Pi_{\text{customer-name}}$ ($\sigma_{\text{branch-name="Perryridge"}}$ (borrower ⋈ loan))

6. **Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.**

$\Pi_{\text{customer-name}}$ ($\sigma_{\text{branch-name="Perryridge"}}$ (borrower ⋈ loan)) -$\Pi_{\textit{customer-name}}$ (depositor)

7. **Find the names of all customers who have a loan at the Perryridge branch.**
$\Pi_{\text{customer-name}}$($\sigma_{\text{branch-name = "Perryridge"}}$ (borrower ⋈ loan))

8. **Find the largest account balance.**

$\mathcal{G}_{\text{max(balance)}}$ (account)

9. **Find all customers who have an account from at least the "Downtown" and the Uptown" branches.**

$\Pi_{\text{customer-name}}$ ($\sigma_{\text{branch-name ="Downtown"}}$(depositor ⋈ account)) $\cap$
$\Pi_{\text{customer-name}}$ ($\sigma_{\text{branch-name ="Uptown"}}$(depositor⋈ account))

10. **Find all customers who have an account at all branches located in Brooklyn city.**
$\Pi_{\textit{customer-name, branch-name}}$ (*depositor* ⋈ *account*)
$\div$ $\Pi_{\textit{branch-name}}$ ($\sigma_{\textit{branch-city} = \text{"Brooklyn"}}$ (*branch*))

11. **Delete all account records in the Perryridge branch.**

account $\leftarrow$ *account* – $\sigma_{\textit{branch-name = "Perryridge"}}$ *(account)*

12. **Delete all loan records with amount in the range of 0 to 50.**
loan $\leftarrow$ *loan* – $\sigma_{\textit{amount} \geq 0 \textit{ and amount} \leq 50}$ (*loan*)

13. **Delete all accounts at branches located in Needham.**

r$_1$$\leftarrow$ $\Pi_{\textit{branch-name, account-number, balance}}$ ($\sigma_{\textit{branch-city = "Needham"}}$ *(account* ⋈ *branch)*)
r2 $\leftarrow$ $\Pi_{\textit{customer-name, account-number}}$ (r_1⋈depositor)
account $\leftarrow$ account – r_1
depositor $\leftarrow$ depositor – r_2

14. **Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.**

account $\leftarrow$ *account* $\cup$ {("Perryridge", A-973, 1200)}
depositor $\leftarrow$ *depositor* $\cup$ {("Smith", A-973)}

15. Update interest payments by increasing all balances by 5 percent.

$account \leftarrow \Pi_{account\text{-}number,\ branch\text{-}name,\ balance * 1.05}(account)$

16. Update all accounts with balances over $10,000 6 percent interest and pay all other 5 percent.

$account \leftarrow \Pi_{account\text{-}number,\ branch\text{-}name,\ balance * 1.06}(\sigma_{balance > 10000}(account))$
$\cup\ \Pi_{account\text{-}number,\ branch\text{-}name,\ balance * 1.05}(\sigma_{balance \leq 10000}(account))$

17. Create the view (named all-customer) consisting of branches and their customers.

create view all-customer **as**
$\Pi_{branch\text{-}name,\ customer\text{-}name}(depositor \bowtie account)$
$\cup\ \Pi_{branch\text{-}name,\ customer\text{-}name}(borrower \bowtie loan)$

6) Consider the following relational database, where primary keys are underlined.

employee(<u>person_name</u>,street,city)

works(<u>person_name</u>,company_name,salary)

company(company_name,city)

manages(<u>person_name</u>,manager_name)

Given an expression in the relational algebra to express each of the following queries.

i) Find the name of all employees who work for first bank corporation.

$\Pi_{person\_name}(\sigma_{company\_name=\text{"first bank corporation"}}(works))$

ii) Find the name and cities of residence of all employees who work for first bank corporation.

$\Pi_{person\_name,city}(\sigma_{company\_name=\text{"first bank corporation"}}(employee \bowtie works))$

iii) Find the name, street address and city of residence of all employees who work for first bank corporation and earn more than $10000 per annum.

$\Pi_{person\_name,street,city}(\sigma_{company\_name=\text{"first bank corporation"} \wedge salary>10000}(employee \bowtie works))$
(Here we assume attribute named salary represent Annual Salary)

iv) Find the name of all employees in this database who lives in same city as the company for which they work.
$\Pi_{person\_name}(employee \bowtie works \bowtie company)$

v) Find the name of all employees who live in the same city and on the same street as do their managers.

$\Pi_{\text{person\_name}}$ ((employee $\bowtie$ manages) $\bowtie_{\text{(manages.manager\_name=employee2.person\_name} \wedge \text{employee.street=employee2.street} \wedge \text{employee.city=employee2.city)}}$ ($\rho_{\text{employee2}}$ (employee)))

vi) Find the name of all employees in this database who do not work for first bank corporation.
$\Pi_{\text{person\_name}}$ ($\sigma_{\text{company\_name} \neq \text{"first bank corporation"}}$(works)

vii) Display the name of employee whose name begin from S.

viii) Find the average salary of employee.
$\mathcal{G}_{\text{avg(salary)}}$ (works)

ix) Find the average salary of employee company wise.
$_{\text{company\_name}}\mathcal{G}_{\text{avg(salary)}}$ (works)

x) Find the name of all employees who earn more than their managers.

$\Pi_{\text{person\_name}}$ ((works $\bowtie$ manages) $\bowtie_{\text{manages.manager\_name=e2.person\_name} \wedge \text{works.salary>e2.salary}}$ (ρ_{e2} (works)))

In SQL this will works
SELECT works.person_name
FROM works NATURAL JOIN manages
JOIN works AS e2 ON manages.manager_name = e2.person_name
AND works.salary > e2.salary;

xi) Find the name of employees who earn more than top earner at "NBL company" in the database.

topearner$\leftarrow\mathcal{G}_{\text{max(salary)}}$ ($\sigma_{\text{company\_name="NBL company"}}$(works))
result$\leftarrow\Pi_{\text{personname}}$($\sigma_{\text{salary>topearner}}$ (works))

xii) Find the name of all employee who live in "Lalitpur" and salary is less than 50000

$\Pi_{\text{person\_name}}$($\sigma_{\text{city="Lalitpur"} \wedge \text{salary<50000}}$ (Employee $\bowtie$ Works))

xiii) **Assume that companies may located in several cities. Find all companies located in every city in which small bank corporation is located**.

$\Pi_{\text{company\_name,city}}$(company) $\div$ Π_{city} ($\sigma_{\text{company\_name ="small bank corporation"}}$(company))

xiv)List the name and city of employee who work in “Pokhara” and have salary greater than Rs.50000

$\prod_{\text{Employee.person\_name,Employee.city}} (\sigma_{\text{Company.city="Pokhara"}\wedge\text{salary>50000}} (\text{Employee} \bowtie_{\text{Employee.person\_name=works.person\_name}} (\text{works} \bowtie \text{company})))$

xv) Delete all employee who come from “Chitwan”.

$\text{Employee} \leftarrow \text{Employee} - \sigma_{\text{city="Chitwan"}}(\text{Employee})$

xvi)Increase salary of all employee by 15%.

$\text{Works} \leftarrow \prod_{\text{person\_name,company\_name,salary*1.15}}(\text{Works})$

xvii) Insert new records in employee relation.

$\text{employee} \leftarrow \text{employee} \cup \{\text{“Ram”,”patan”, ”Lalitpur”}\}$

xviii) Create view for which employee earns 20000$ or more

Create view high_earning_view

$\sigma_{\text{salary>=20000}} (\prod_{\text{person\_name,street,city,company\_name,salary}}(\text{employee} \bowtie \text{works}))$

xix)Modify the database so that Ramesh now lives in Kathmandu

$\text{employee} \leftarrow \prod_{\text{person\_name,street,"Kathmandu"}} (\sigma_{\text{person\_name="Ramesh"}}(\text{employee}))$
$\cup (\text{employee} - \sigma_{\text{person\_name="Ramesh"}}(\text{Employee}))$

xx) Give all salary of employee of “ABC company” 18.5% rise

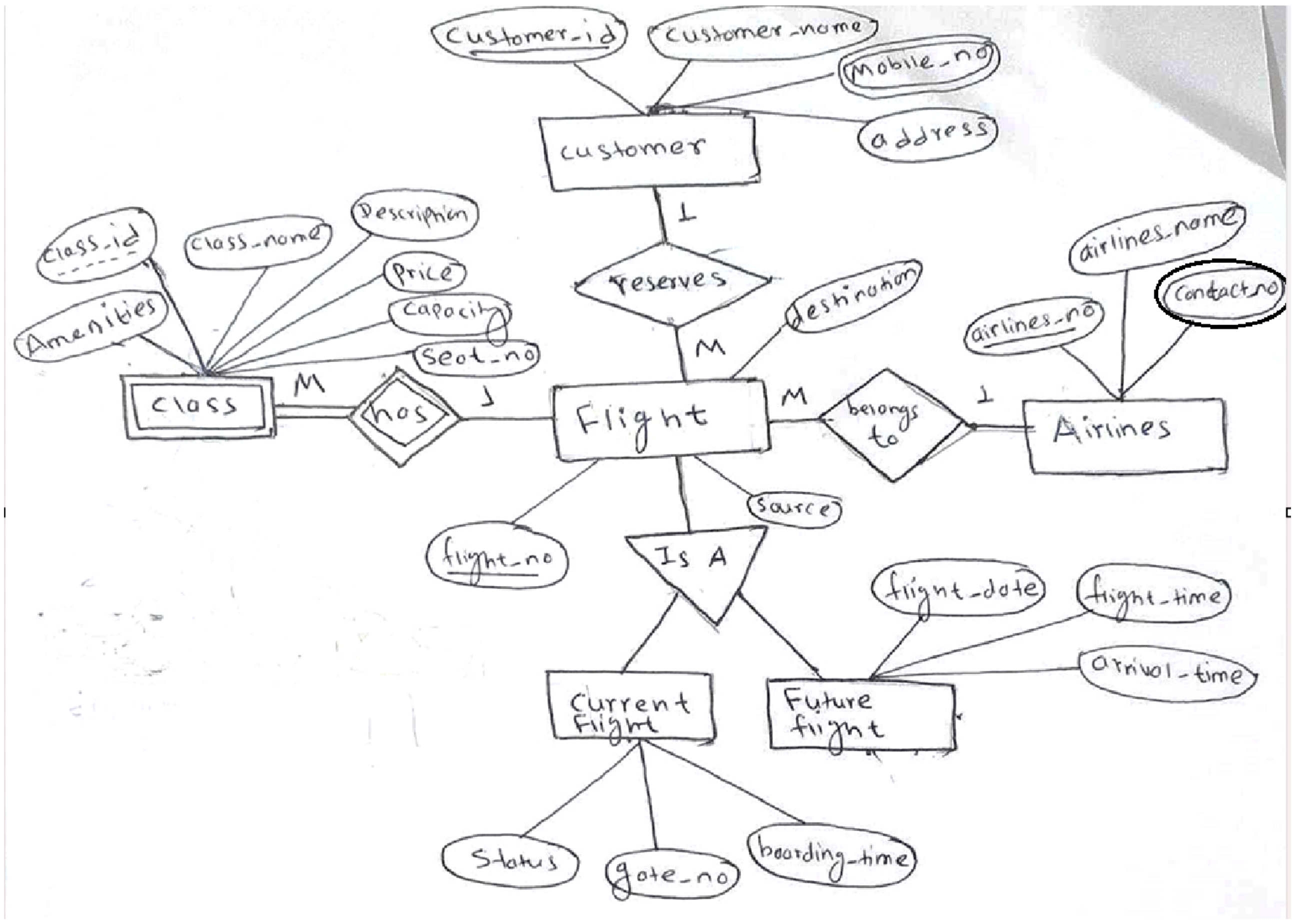
$\text{works} \leftarrow \prod_{\text{person\_name,company\_name,salary*1.185}} (\sigma_{\text{company\_name="ABC company"}}(\text{works}))$
$\cup (\text{works} - \sigma_{\text{company\_name="ABC company"}}(\text{works}))$

ER -Diagram

Tips to draw ER diagram

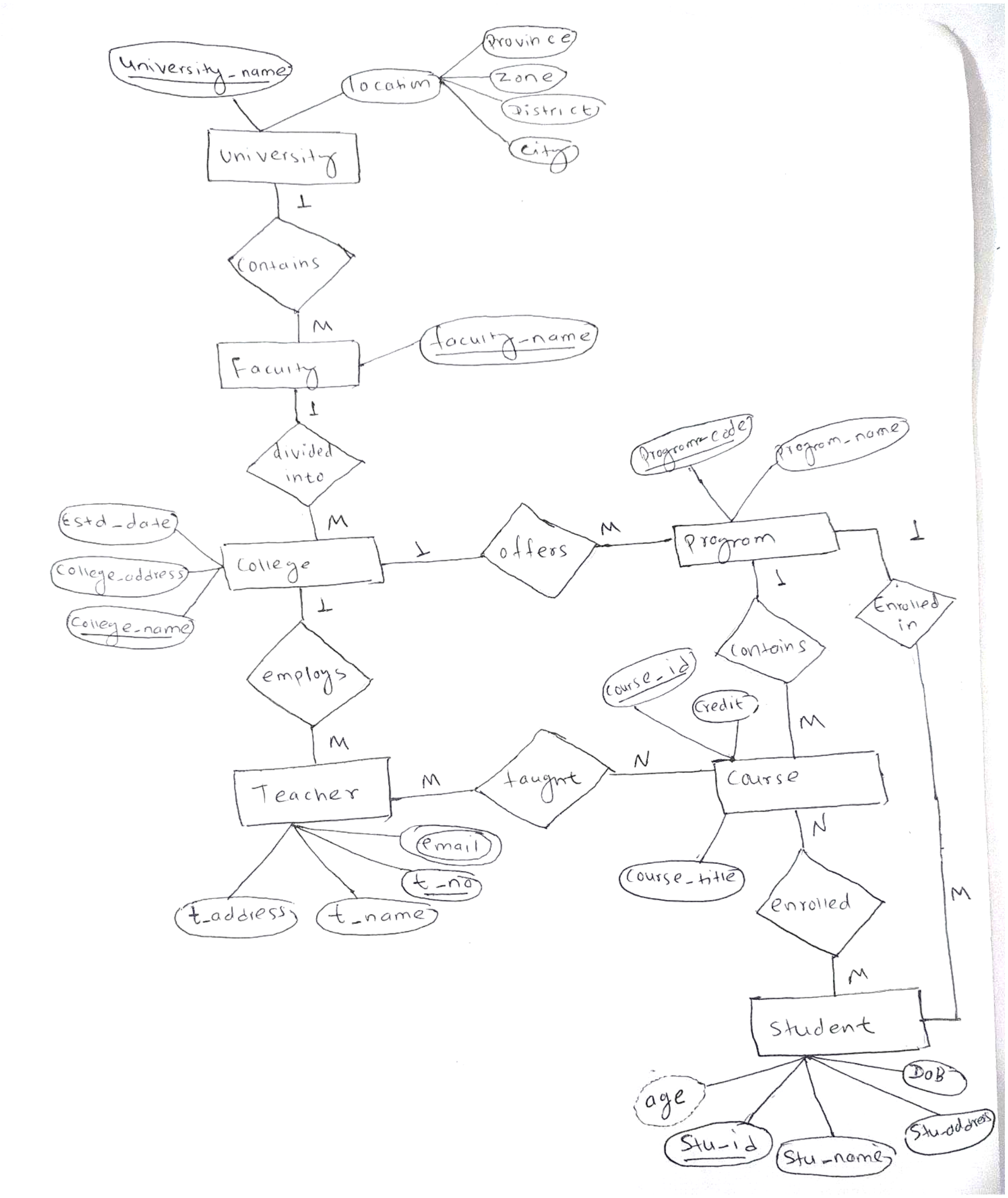
1. Identify Entities
2. Find the relationship between them
3. Identify attributes
4. Identify cardinality constraints
5. Create ER diagram

7) i) Draw an E-R diagram for airline reservation system. The system must keep track of customers and their reservations, flights and their status, seat assignments on individual flights etc.

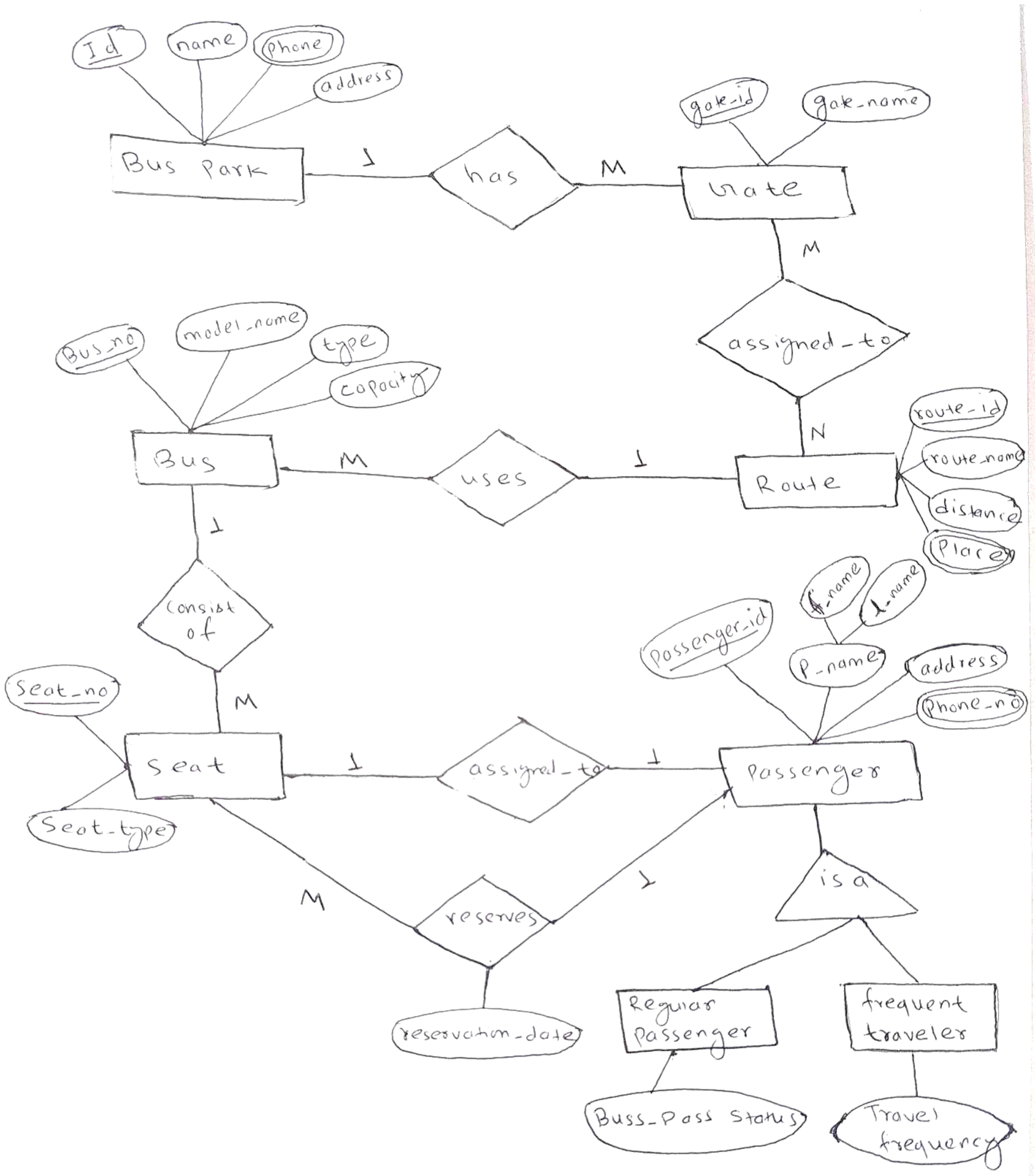


ii) Draw an ER diagram for the following scenario.

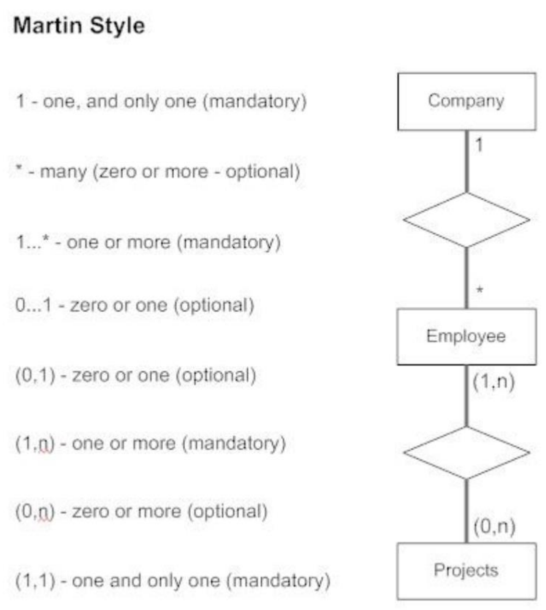
A university contains many faculties. The faculties in turn are divided into several colleges. Each college offers numerous programs, and each program contains many courses. Teachers can teach many different courses and even the same course numerous times. Courses can also be taught by many teachers. A student is enrolled in only one program, but a program can contain many students. Students can be enrolled in many courses at the same time and the courses have many students enrolled.



iii) Construct an ER diagram for a Metropolitian Bus Park. There are many gates for entering bus park. Different gates are assigned to different routes. A route uses different buses. Bus consists of different seats which are assigned to different passengers. Frequent travelers are also in passenger. Associate a log of reservation date while reserving seats. The passenger's name must have two attributes first_name and last_name. Each of entities must have primary key attribute as far as possible. The cardinality mapping should be explained properly.



Note:


Martin Style

1 - one, and only one (mandatory)

* - many (zero or more - optional)

1...* - one or more (mandatory)

0...1 - zero or one (optional)

(0,1) - zero or one (optional)

(1,n) - one or more (mandatory)

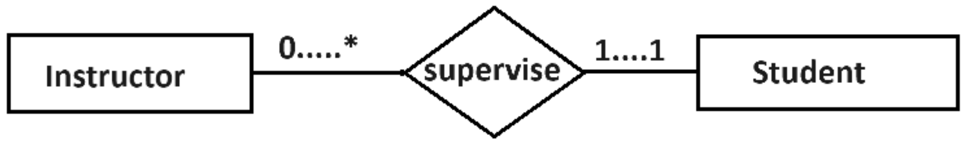
(0,n) - zero or more (optional)

(1,1) - one and only one (mandatory)

Notation for expressing more complex constraints.

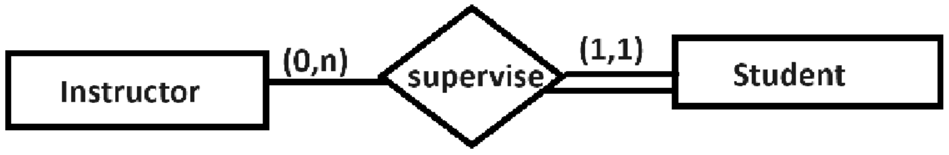
l…..h

- minimum value of 1 indicates total participation
- maximum value of 1 indicates that entity participates in at most one relationship.
- A maximum value of * indicates no limit



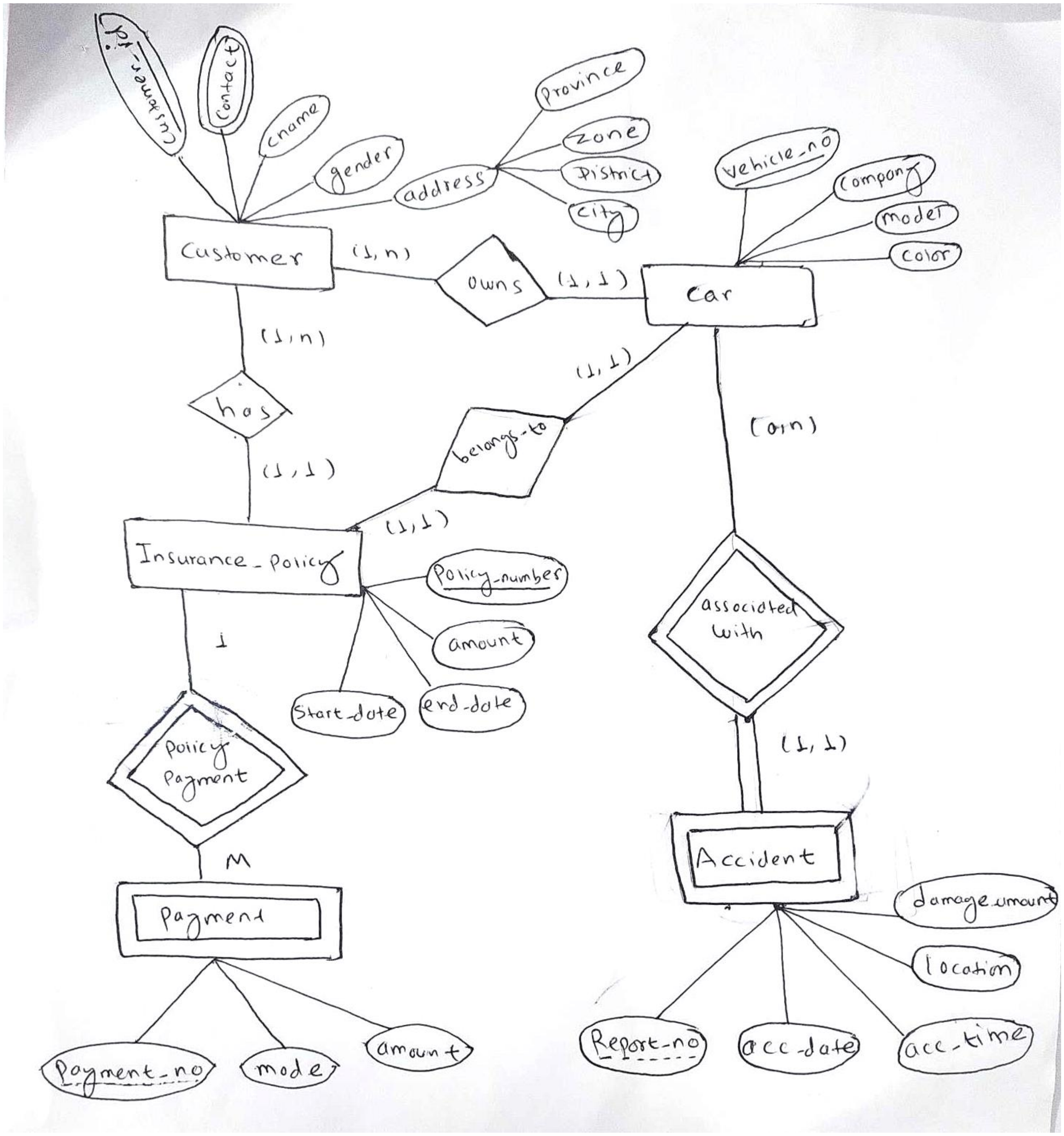
Alternatively

l…..h can be represented as (l, h)



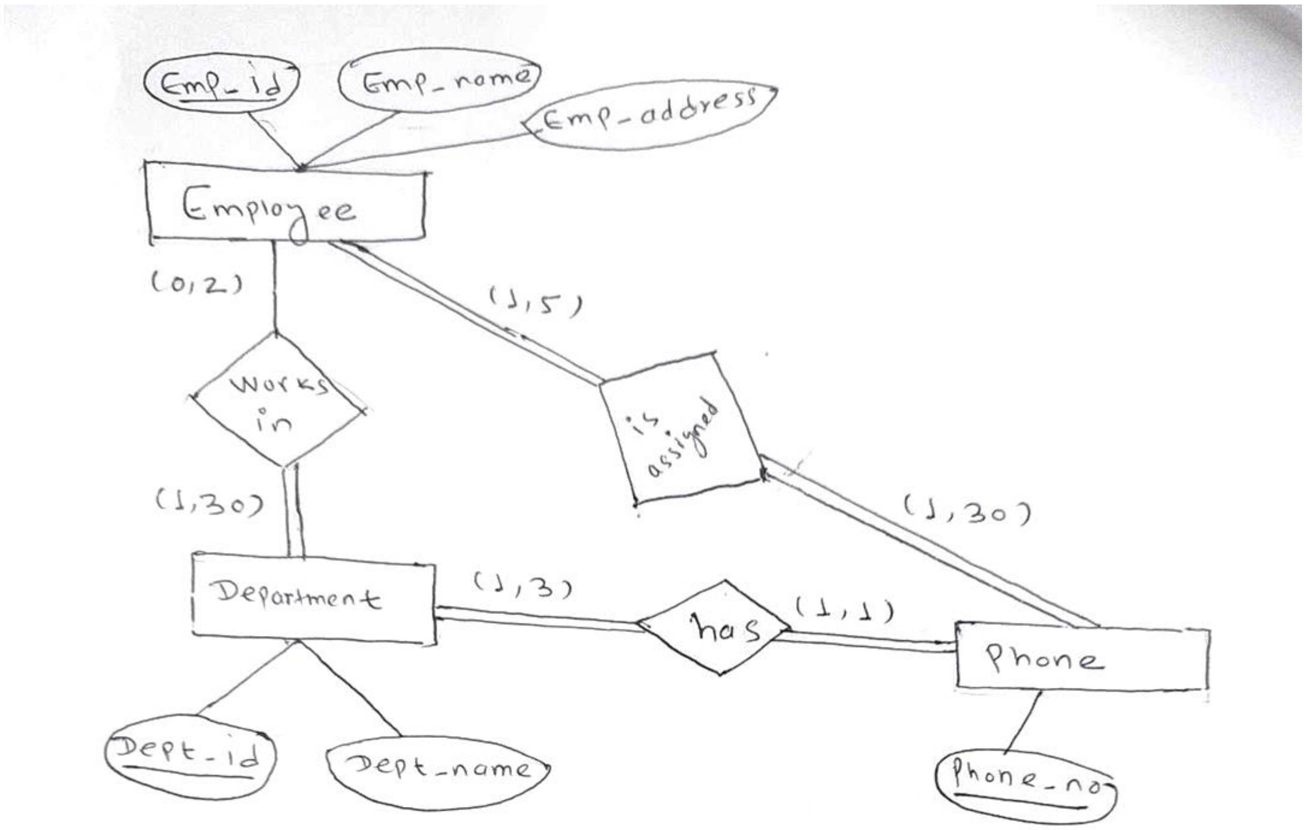
- Instructor can supervise 0 or more students.
- A student is supervised by only one instructor

iv) Construct ER model of a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents. Also design a relational database corresponding to the ER diagram.



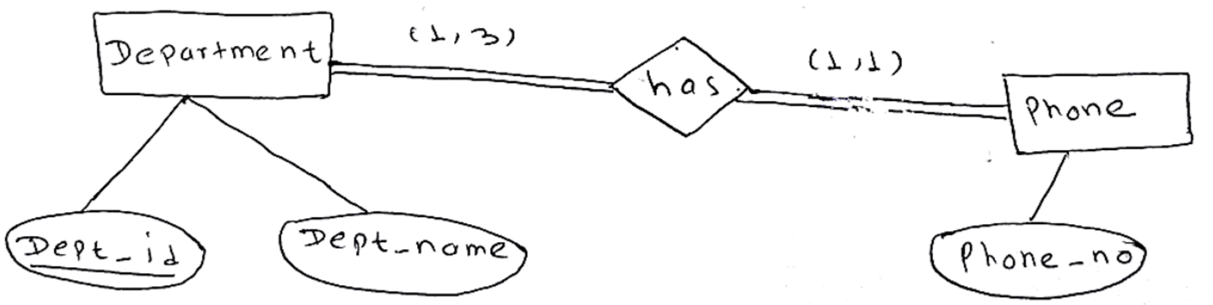
v) Suppose you are given the following requirements for a simple database for the employee management system.

a) An employee may work in upto two departments or may not be assigned to any department.
b) Each department must have one and may have upto three phone numbers.
c) Each department can have anywhere between 1 and 30 employees.
d) Each phone is used by one, and only one, department.
e) Each phone is assigned to at least one and may be assigned to upto 30 employees.
f) Each employee is assigned at least one, but no more than 5 phones.



Things to be understand

Considering two entities and their relationship as follows:

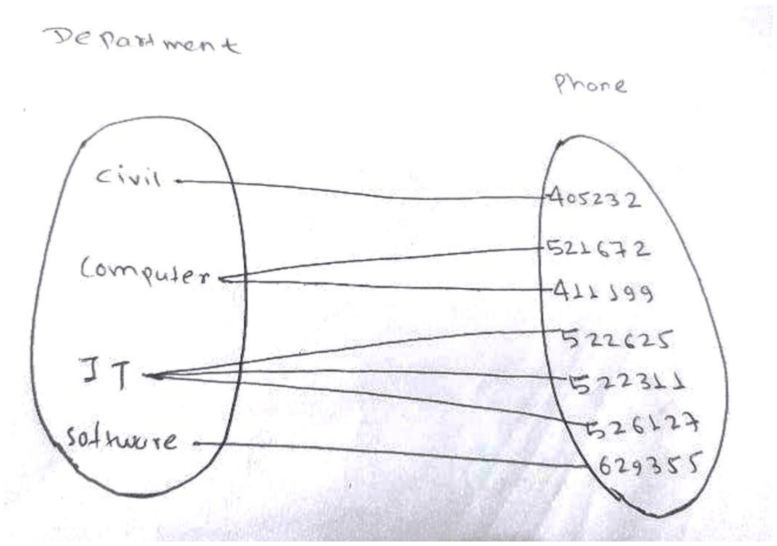


Department and phone are two entities .**“has”** is relationship between them. Which means Department has Phone.

Here, (1,3) means ,Each department have at least 1 phone and at most 3 three phones.

Similarly (1,1) means Each phone is used by one department; one(same) phone cannot be used by multiple department.

For an illustration



(1,3) means ,Each department have at least 1 phone and at most 3 three phones.

Here ,we can see that,

- ✓ Civil department has one phone.(i.e 405232)
- ✓ Computer department has two phones. (i.e 521672,411199)
  Similarly,
- ✓ IT department has three phones.
- ✓ Software department has one phone.

(1,1) means Each phone is used by one department; one phone cannot be used by multiple department.

- ✓ 405232 is used only by civil department.
- ✓ 521672 is used only by computer department.
- ✓ 411199 is used only by computer department.
- ✓ 522625 is used only by IT department.
  ….So on.

8) Differentiate between relational algebra and relational calculus? Define TRC and DRC?

Relational algebra	Relational calculus
It is a procedural language.	Relational Calculus is a Declarative(non-procedural) language.
Relational algebra means how to obtain the result.	Relational calculus means what result we have to obtain.
In relational algebra, the order is specified in which the operations have to be performed.	In relational calculus, the order is not specified.
Relational algebra is independent of domain.	Relational calculus can be domain dependent because of domain relational calculus.
The SQL includes only some features from the relational algebra.	SQL is based to a greater extent on the tuple relational calculus.
The evaluation of the query relies on the order specification in which the operations must be performed.	The order of operations does not matter in relational calculus for the evaluation of queries.
The basic operation included in relational algebra are: 1. Selection (σ) 2. Projection (Π) 3. Union (U) 4. Set Difference (-) 5. Cartesian product (X) 6. Rename (ρ)	Relational Calculus is denoted as: { t | P(t) } Where, t: the set of tuples p: is the condition which is true for the given set of tuples.

Tuple Relational Calculus (TRC)

- ✓ It is a nonprocedural query language, where each query is of the form $\{t \mid P(t)\}$
- ✓ where t is a tuple variable and P(t) is a formula(similar to predicate calculus) that describes the conditions that the tuples in the result must satisfy .
- ✓ *t* is a *tuple variable*, $t[A]$ denotes the value of tuple *t* on attribute *A*
- ✓ $t \in r$ denotes that tuple *t* is in relation

Example:

Let us consider the following schema

Loan(loan_number,branch_name,amount)

Find the loan-number, branch-name, and amount for loans of over $1200

$\{t \mid t \in loan \land t[amount] > 1200\}$

Domain Relational Calculus(DRC)

Domain Relational Calculus is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it.

Each query is an expression of the form:

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

where, $< x_1, x_2, x_3, \ldots, x_n >$ represents resulting domains variables and $P(x_1, x_2, x_3, \ldots, x_n)$ represents the condition or formula equivalent to the Predicate calculus.

Example:

Let us consider the following schema

Loan(loan_number,branch_name,amount)

Find the loan-number, branch-name, and amount for loans of over $1200

$\{< l, b, a > \mid < l, b, a > \in loan \land a > 1200\}$

9)

i) Employee(person_name,street,city)
Works(person_name,company_name,salary)
Company(company_name,city)

Write the relational algebra expression for the query **"Find the names of all employees whose company is located in pokhara".** Construct the initial operator tree and final efficient operator tree after applying transformation rules.

Solution :

To Find the names of all employees whose company is located in pokhara from above schemas the relational algebraic expression can be written as

$\prod_{person\_name}(\sigma_{city="Pokhara"}(Works \bowtie Company))$

In the above expression, the size of intermediate relation is large. Here, the tuples that are not required for final results are also participated in join operation. So we can apply some transformation rules to reduce the size of intermediate relation.

As we know, Natural joins are also commutative.

$E1 \bowtie E2 \equiv E2 \bowtie E1$

Now, above relational algebraic expression can be written as,

$\prod_{person\_name}(\sigma_{city="Pokhara"}(Company \bowtie Works))$

By using distribution rule of selection ,

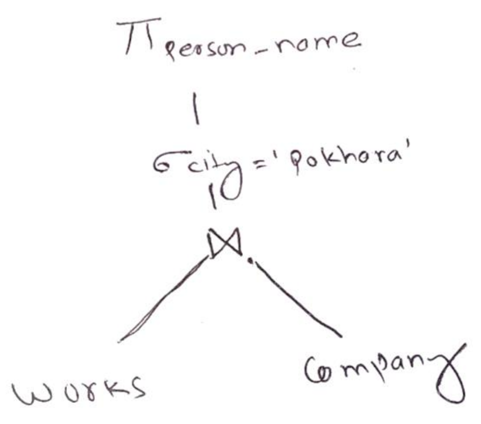
$\sigma_{\theta 0}(E_1 \bowtie E_2) \equiv (\sigma_{\theta 0}(E_1)) \bowtie E_2$

When all the attributes in θ_0 involve only the attributes of one of the expressions (**E**$_1$) being joined.

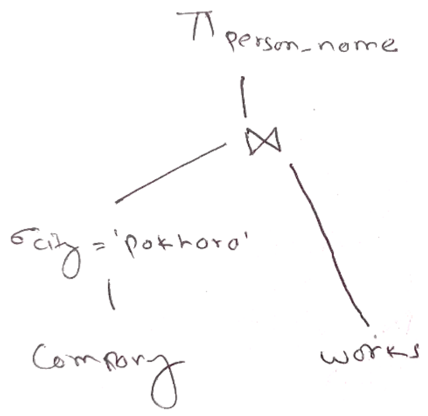
$\prod_{person\_name}((\sigma_{city="Pokhara"}(Company)) \bowtie Works)$

Which is efficient form than the previous expression because performing early selection reduces the size of relation to be joined.

Initial Operator tree



Final operator tree after multiple transformations



ii) consider the following relational schema

Sailors (sid,sname,rating,age)

Boats(bid,bname,color)

Reserves(sid,bid,day)

Write a relational algebra expression for the query "find the name of sailors who have reserved red or green boat". Construct the initial operator tree and final efficient operator tree after applying transformation rules.

Solution :

To find the name of sailors who have reserved a red or green boat, the relational algebraic expression can be written as:

$$\Pi_{sname} (\sigma_{color="red" \lor color="green"} (Sailors \bowtie (Reserves \bowtie Boats)))$$

In the above expression, the size of intermediate relation is large. Here, the tuples that are not required for final results are also participated in join operation. So we can apply some transformation rules to reduce the size of intermediate relation.

As we know, Natural joins are associative,

$$(E1 \bowtie E2) \bowtie E3 \equiv E1 \bowtie (E2 \bowtie E3)$$

Now, above relational algebraic expression can be written as,

$$\Pi_{sname} (\sigma_{color=" red" \lor color="green"} ((Sailors \bowtie Reserves) \bowtie Boats))$$

As we know, Natural joins are also commutative.

$$E1 \bowtie E2 \equiv E2 \bowtie E1$$

Now, above relational algebraic expression can be written as,

$$\Pi_{sname} (\sigma_{color=" red" \lor color="green"} (Boats \bowtie (Sailors \bowtie Reserves))$$

By using distribution rule of selection ,

$\sigma_{\theta 0}$ (E$_1 \bowtie$ E$_2$) ≡ ($\sigma_{\theta 0}$(E$_1$)) $\bowtie$ E$_2$

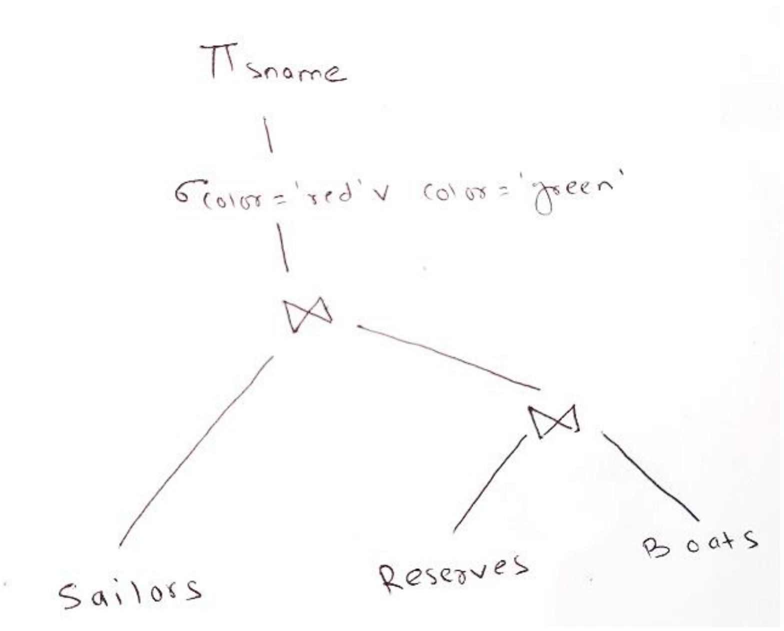
When all the attributes in θ_0 involve only the attributes of one of the expressions (**E**$_1$) being joined.

Now, above expression can be written as:

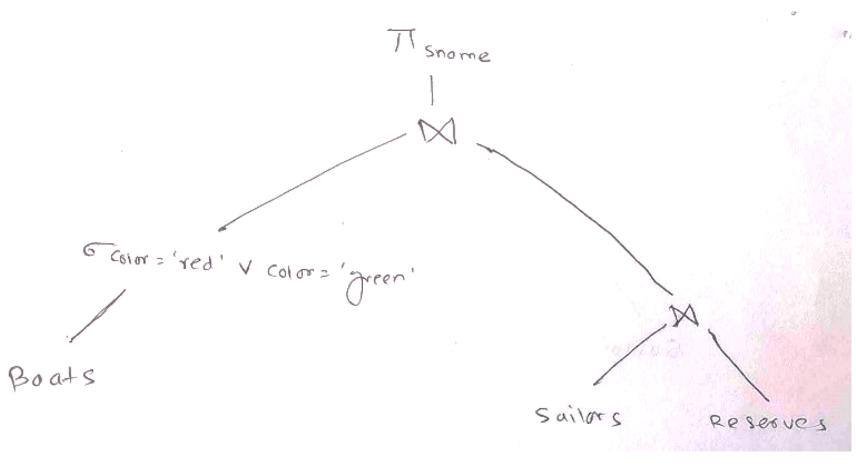
$$\Pi_{sname}\ ((\sigma_{color=\text{"red"} \vee color=\text{"green"}} (Boats)) \bowtie (Sailors \bowtie Reserves))$$

Which is efficient form than the previous expression because performing early selection reduces the size of relation to be joined.

Initial Operator tree



Final Efficient operator tree after applying multiple transformation.



iii) Consider the relational schema

Employee(person_name,street,city)

Works(person_name,company_name,salary)

Company(company_name,city)

Write the relational algebra expression for the query **"Find the names of all employees who lives in Pokhara".** Construct the initial operator tree and final efficient operator tree after applying transformation rules.

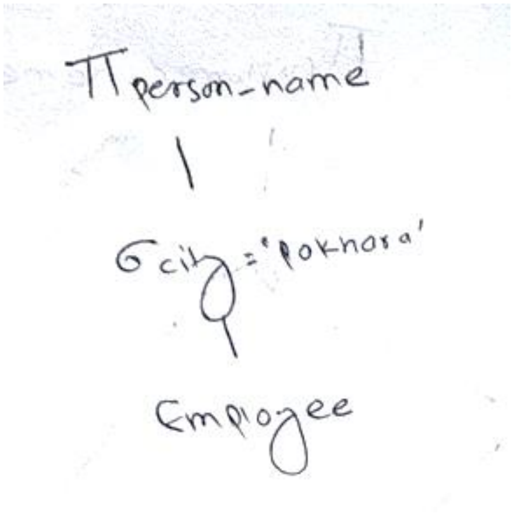
Solution :

To Find the names of all employees who lives in pokhara from above schemas the relational algebraic expression can be written as

$\Pi_{person\_name}(\sigma_{city="Pokhara"}(Employee))$

Here, the above expression is already in its most efficient form. So the initial operator tree and final efficient operator tree are the same. Hence, we have

Initial /final operator tree



iv) Make an operator tree for the following SQL expression.

```
Select customer_name
From branch,account,depositor
Where branch_city='btl' AND balance>2000;
```

For the above SQL expression let us consider the following schemas

Branch(<u>branch_name</u>,branch_city,assets)
Account(<u>account_number</u>,branch_name,balance)
Depositor (<u>customer_id</u>,account_number)

Now, relational algebraic expression for the above SQL expression can be written as:

$\Pi_{customer\_name} (\sigma_{branch\_city='btl' \wedge balance>2000} (branch \bowtie (account \bowtie depositor)))$

In the above expression, the size of intermediate relation is large. Here, the tuples that are not required for final results are also participated in join operation. So we can apply some transformation rules to reduce the size of intermediate relation.

As we know, Natural joins are associative,

$(E1 \bowtie E2) \bowtie E3 \equiv E1 \bowtie (E2 \bowtie E3)$

Now, above relational algebraic expression can be written as,

$$\Pi_{customer\_name} (\sigma_{branch\_city='btl' \wedge balance>2000}((branch \bowtie account) \bowtie depositor))$$

By using distributive rule of selection,

$$\sigma_{\theta 1 \wedge \theta 2} (E_1 \bowtie E_2) \equiv (\sigma_{\theta 1}(E_1)) \bowtie (\sigma_{\theta 2}(E_2))$$

When θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2.

Here, above subexpression becomes,

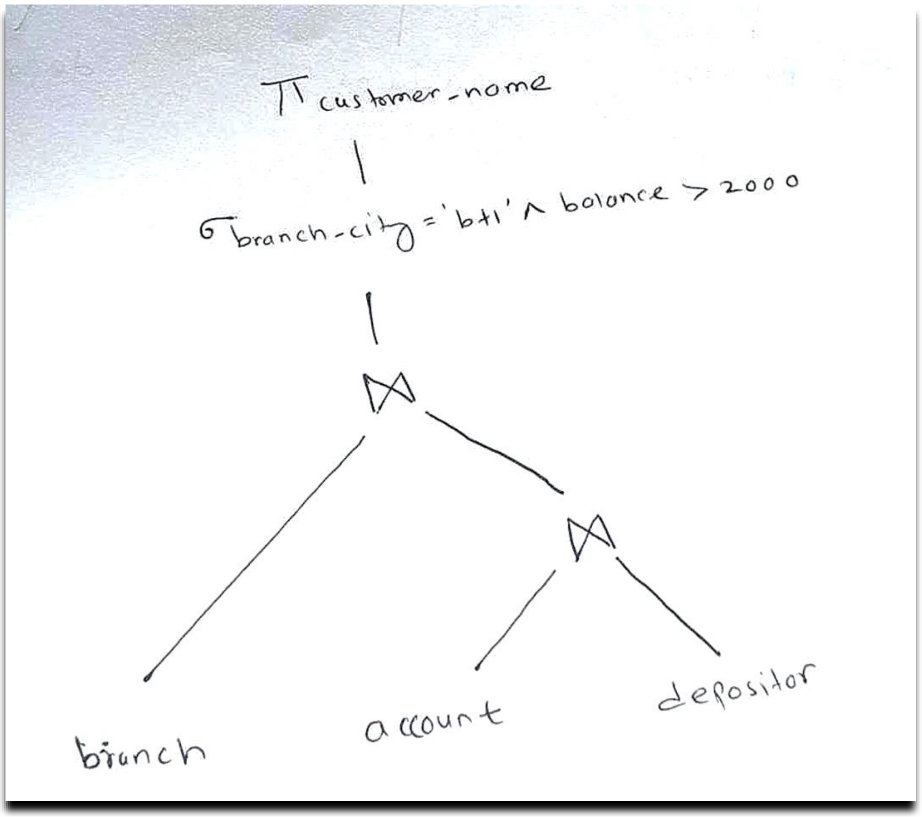
$(\sigma_{branch\_city='btl'} (branch)) \bowtie (\sigma_{balance>2000}(account))$

Now, finally relational algebraic expression becomes,

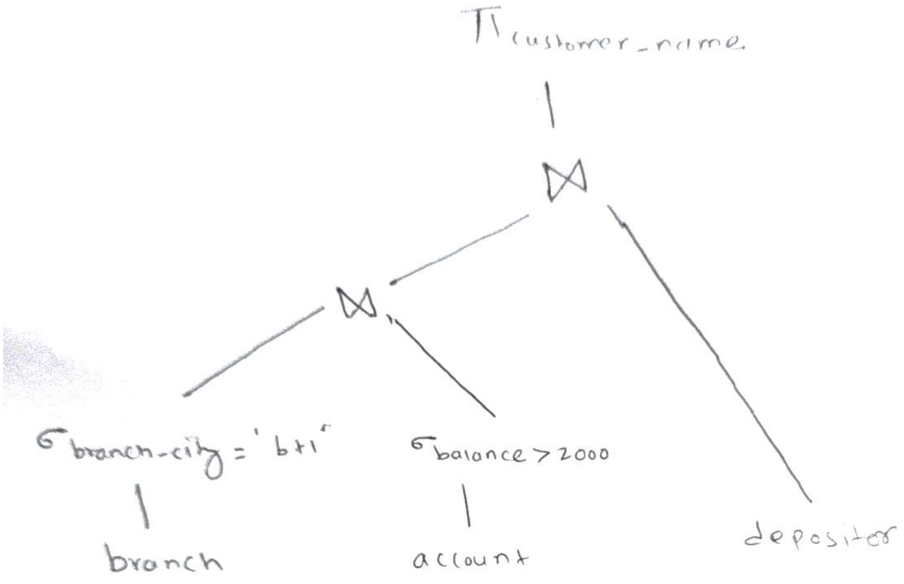
$\Pi_{customer\_name}(((\sigma_{branch\_city='btl'} (branch)) \bowtie (\sigma_{balance>2000}(account))) \bowtie depositor)$

Which is efficient form than the previous expression because performing early selection reduces the size of relation to be joined.

Initial Operator tree



Final Efficient operator tree after applying multiple transformation



10.

i) **Find the closure set of attributes and functional dependency from following**
A→BC, B→C ,AB→C, AC→D

The set of all those attributes which can be functionally determined from an attribute set is called as closure of that attribute set.

As we know,
X^+ → contains the set of attributes determined by X
Now,
A^+ ={ABCD}
(Here, Attribute A can determine A itself. A can determine BC from above functional dependency, again AC can determine D)
Similarly,
B^+ ={BC}
C^+ ={C}
D^+ ={D}

Closure of a functional dependency in database design refers to the set of functional dependencies that can be derived from a given set of functional dependencies.

Here, Given functional dependency,
F={A→BC, B→C ,AB→C, AC→D }

Now for F^+

A→B, A→C

[∵ A→BC and applying Decomposition rule]

A→C
[∵ A→B, B→C and applying Transitivity rule]

AB→D
[∵ B→C and AC →D and applying pseudo transitivity rule]

AB→CD

[∵ AB→C and AB →D and applying union rule]

Now,
F^+ ={A→BC, B→C ,AB→C, AC→D, A→B,A→C,AB→D, AB→CD}

ii) Let R={A,B,C,D,E,F} F= {A → BC, E → CF, B→ E, CD → EF} Now compute $(AB)^+$

Solution:

<u>1st Iteration</u>

Let result=AB then,

For A→BC

A ⊆ {AB } is true.

∴result ={ABC}

For E →CF

E ⊆ {ABC} is false .

∴result ={ABC}

For B →E

B⊆ {ABC} is true.

∴result ={ABCE}

For CD →EF

CD⊆{ABCE} is false .

∴result ={ABCE}

<u>2nd Iteration</u>

For A→BC

A ⊆ {ABCE} is true.

∴result ={ABCE}

For E →CF

E ⊆ {ABCE} is True.

∴result ={ABCEF}

For B →E

B⊆ {ABCEF} is true.

∴result ={ABCE}

For CD →EF

CD⊆{ABCEF} is false .

∴result ={ABCEF}

3rd Iteration

For A→BC

A ⊆ {ABCEF} is true.

∴result ={ABCEF}

For E →CF

E ⊆ {ABCEF} is True.

∴result ={ABCEF}

For B →E

B⊆ {ABCEF} is true.

∴result ={ABCEF}

For CD →EF

CD⊆{ABCEF} is false .

∴result ={ABCEF}

Here ,result does not change further.

$(AB)^+$ ={ABCEF}

Procedure for determining the candidate key

Candidate Key:

An attribute or the combination of attributes is called candidate key if and only if:

- They derive all the attributes of the relation.
- They are the minimal subset of the super key.

Note:

- ✓ Candidate keys can be either simple or composite.
- ✓ Minimal subset is not with respect to the no of attributes however it always refer to the minimal level of subset which does not have any proper subsets that derives all the attributes of the relation.
- ✓ A relation can have more than one candidate key.

Determining candidate key

- ✓ Compute closure set of each attributes.
- ✓ Any attribute is called candidate key of any if attribute closure is equal to the relation.
  - ➢ Attribute that are part of candidate key are called **prime attribute.**
  - ➢ Attribute that are not part of candidate key are called **non-prime attribute.**

<u>Example:</u>

R=(A,B,C,D)

F={AB→C, C→D, D→A}

List all the candidate keys, prime and non-prime attributes.

Solution:

For A

A^+ ={A}; A is not candidate key

For B

B^+={B}; B is not candidate key

For C

C^+={CDA}; C is not candidate key

For D

D^+={DA}; D is not candidate key

For AB

$(AB)^+$ ={ABCD}; AB is candidate key

For AC

$(AC)^+$ ={ACD}; AC is not candidate key

For AD

$(AD)^+$ ={AD}; AD is not candidate key

For BC

$(BC)^+$ ={BCDA}; BD is a candidate key

For BD

$(BD)^+$ ={BDAC}; BD is a candidate key

For CD

$(CD)^+$ ={CDA}; CD is not a candidate key

∴ Candidate key={AB,BC,BD}

∴ prime attribute ={A,B,C,D}

Here is no any non-prime attribute.

11. Create a un-normalized table and normalize it up to 3NF.

Let us consider the following table.

SID	Name	CID	Course	Mobile	Marks	Salutation
1	Rajesh	027	DBMS	9851155666, 9802466774	89	Mr.
2	Rita	032	CN	9847025512, 9818065256	54	Miss.
3	Harsh	027	DBMS	9851167343	95	Mr.
4	Jyoti	091	OS	9841476423	68	Miss.

For 1 NF

- ✓ A relation is in 1NF if all the contained values should be atomic values.

But the above table is not in 1NF because there are multiple values in column named mobile, which violates the rule of 1NF.

Now above table can be converted into 1NF as follows.

Student

SID	Name	CID	Course	Marks	Salutation
1	Rajesh	027	DBMS	89	Mr.
2	Rita	032	CN	54	Miss.
3	Harsh	027	DBMS	95	Mr.
4	Jyoti	091	OS	68	Miss.

Student_contact

SID	Mobile
1	9851155666
1	9802466774
2	9847025512
2	9818065256
3	9851167343
4	9841476423

<u>For 2 NF</u>

A relation will be in 2NF if

- ✓ It is in 1NF and
- ✓ All non-prime attributes are fully functional dependent on the key attribute or primary key (There must not be any partial dependency)

In the above table named **Student** {SID +CID} can be considered as composite primary key. Due to this combination, we can uniquely identify all the records of the table.

But here partial dependency exists. All attributes are not fully functional dependent on primary key.

For example:

SID→ Name

CID → Course

Now, we can remove this partial dependency as follows.

Student_info

SID	Name	Salutation
1	Rajesh	Mr.
2	Rita	Miss.
3	Harsh	Mr.
4	Jyoti	Miss.

Course_info

CID	Course
027	DBMS
032	CN
091	OS

Marks_info

SID	CID	Marks
1	027	89
2	032	54
3	027	95
4	091	68

Student_contact

SID	Mobile
1	9851155666
1	9802466774
2	9847025512
2	9818065256
3	9851167343
4	9841476423

For 3NF

For any relation to be in 3NF it must satisfied following properties.

- ✓ It is in 2NF.
- ✓ Every non-prime attribute is non-transitively dependent on the primary key. Which means there should not be case that non-prime attribute is functionally dependent on another non-prime attribute.

In above table named **student_info**

Finding functional dependencies

SID →Name
Name →Salutation.

Here we can find transitive dependencies. Salutation is transitively dependent on primary key SID. Now, we can remove this dependency as follows.

Studentname_info

SID	Name
1	Rajesh
2	Rita
3	Harsh
4	Jyoti

Salutation_info

Name	Salutation
Rajesh	Mr.
Rita	Miss.
Harsh	Mr.
Jyoti	Miss.

Course_info

CID	Course
027	DBMS
032	CN
091	OS

Marks_info

SID	CID	Marks
1	027	89
2	032	54
3	027	95
4	091	68

Student_contact

SID	Mobile
1	9851155666
1	9802466774
2	9847025512
2	9818065256
3	9851167343
4	9841476423

Now ,we can see that above tables satisfy the condition upto 3NF.

12.

i) Consider the following table. Identify all the functional dependencies in the table and convert it into 3NF.

Employee

EmpNo	EName	City	Post	Salary
E001	Ram	Pokhara,Dharan	Manager	30000
E002	Sita	Kathmandu	Analyst	25000
E003	Bharat	Bharatpur,Pokhara	Programmer	22000
E004	Laxman	Butwal	Manager	30000
E005	Dashrath	Lalitpur,Kathmandu	Analyst	25000

From the above table we can find the following functional dependencies.

EmpNo→ {EName, City, Post,Salary}

Post →Salary

To convert the given table into 3NF, Lets check for 1NF

For table to be in 1NF, Each cell must be single valued i.e no repeating groups.

But, the above table violates the rule of 1NF. Now converting in 1NF we get

Employee_info

EmpNo	**EName**	**Post**	**Salary**
E001	Ram	Manager	30000
E002	Sita	Analyst	25000
E003	Bharat	Programmer	22000
E004	Laxman	Butwal	30000
E005	Dashrath	Analyst	25000

City_info

EmpNo	City
E001	Pokhara
E001	Dharan
E002	Kathmandu
E003	Bhaktapur
E003	Pokhara
E004	Manager
E005	Lalitpur
E005	Kathmandu

For a relation to be in 2NF,

1. It must already be in 1NF
2. Every non-prime attribute is fully dependent on primary key (has no partial dependency)

Here ,In above relation Employee_info EmpNo can be considered as primary key. Here EmpNo determine all other attributes. So there is no partial dependency. So it is in 2NF.

For 3NF,

1. It must already be in 2NF
2. There should not be transitive dependency. (i.e. there should not be the case that non-prime attribute is functionally dependent on another non-prime attribute)

Here, In the table Employee_info Salary is determined by Post which is non-prime attribute. We can convert above relation Employee_info into 3NF as follows and finally relation becomes.

Emp_post

EmpNo	Ename	Post
E001	Ram	Manager
E002	Sita	Analyst
E003	Bharat	Programmer
E004	Laxman	Butwal
E005	Dashrath	Analyst

Emp_salary

Post	Salary
Manager	30000
Analyst	25000
Programmer	22000

City_info

EmpNo	City
E001	Pokhara
E001	Dharan
E002	Kathmandu
E003	Bhaktapur
E003	Pokhara
E004	Manager
E005	Lalitpur
E005	Kathmandu

ii) Explain second Normal form? Whether the following table is in 2NF?

Sector	PlotNo	City
A	1	KTM
A	2	KTM
A	3	KTM
B	1	Pokhara
B	2	Pokhara

Explain the different anomalies in the database table.

Solution:

In the following table we can find following anomalies.

Insertion Anomaly: Insertion anomalies occur when it is not possible to add new data to table without including additional or unrelated information(i.e. data cannot be added without providing information of all attributes.

In the above table if we want to insert new row for different sector or city ,we must also provide valid plot number.

Update Anomaly: If the table if we want to update the city for particular sector, we must have to ensure that updation must be done in all rows in which particular sector is exist, otherwise it leads to inconsistency of database.

Deletion Anomaly: If we want to delete a row corresponding to specific sector and plot number, we might unintentionally delete information about city.

A relation is in 2NF if and only if

1. It is in 1NF and
2. Every non-prime attribute is fully dependent on primary key.(has no partial dependency)

Here ,considering (sector +plot No) as composite primary key as per question, there exist a partial dependency ,because city is functionally dependent on sector.

sectory→city

so it is not in 2NF.

By removing partial dependencies ,this table can be converted to 2NF as follows:

Sector_info

Sector	City
A	KTM
B	Pokhara

Plot_info

Sector	Plot_no
A	1
A	2
A	3
B	1
B	2

iii) What do you mean by functional dependency, multivalued dependency and transitive dependency in Normalization process of Database? Why Normalization is needed? Assume the un-normalized relation as given below and find the final normalized logical ER diagram normalizing the un-normalized relation upto 3NF explaining what you going to check at each normal step.

Roll.No	Name	SubID	SubName	FeePaid
1	Hari Man Dangol	DBMS	Database Management System	20000
2	Mohan Prasad Shah	DBMS	Database Management System	20000
3	Indira Rimal	DBMS	Database Management System	30000
1	Hari Man Dangol	CPROG	C programming	20000
2	Mohan Prasad Shah	CPROG	C programming	15000
3	Indira Rimal	MATH	Mathematics	30000

(For functional dependency ,multivalued dependency and transitive dependency refer note)

As we know, Database Normalization is a technique of organizing the data in the database. It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Normalization is needed due to the following reasons:

- ✓ Normalization helps in reducing redundancy by organizing data into separate tables and ensuring that each piece of data is stored only once.
- ✓ It optimizes data storage, reducing disk space usage and storage costs.
- ✓ Normalization helps maintain data integrity by reducing the risk of anomalies such as insertion, update, and deletion anomalies.
- ✓ Well-normalized databases typically perform better in terms of query execution and data retrieval.
- ✓ Normalized databases are generally more scalable and adaptable to changing requirements.

To convert the given Un-normalized relation into 3NF,

Let's check for 1NF.

- ✓ For relation to be in 1NF, each cell must be single valued i.e. no repeating groups.

But in the above given relation there is no repeating groups. So the given relation is already in 1NF.

For a relation to be in 2NF,

1. It must already be in 1NF
2. Every non-prime attribute is fully dependent on primary key (has no partial dependency)

Here, in above relation (Roll.No +SubID) is composite primary key.

But there exists a partial dependency such as

Roll.No →Name

SubID→SubName

By removing partial dependencies, this table can be converted to 2NF as follows:

Student_info

Roll.No	Name	SubID
1	Hari Man Dangol	DBMS
2	Mohan Prasad Shah	DBMS
3	Indira Rimal	DBMS

Subject_info

SubID	SubName
DBMS	Database Management System
CPROG	C programming
MATH	Mathematics

Fee_info

Roll.No	SubID	FeePaid
1	DBMS	20000
2	DBMS	20000
3	DBMS	30000
1	CPROG	20000
2	CPROG	15000
3	MATH	30000

For 3NF,

1. It must already be in 2NF
2. There should not be transitive dependency. (i.e. there should not be the case that non-prime attribute is functionally dependent on another non-prime attribute)

In above 2NF relations there is no any transitive dependency. So its is already in 3NF.