Assignment Solution SQL and Relational Algebra

- Subject: Database Management System

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	i 1 i
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

+	+	+
	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red
+	+	+

Reserves

+	bid	day
+	101 102 103 104 102 103 104 101	1998-10-10 1998-10-10 1998-10-08 1998-10-07 1998-11-10 1998-11-06 1998-09-05 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



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



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;



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));



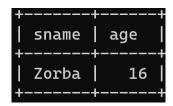
This SQL query retrieves the names of sailors who have reserved all boats. Here's an explanation of each part of the query:

- ✓ **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.

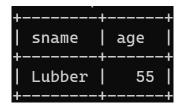
5. Find the name and age of youngest sailor

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



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
);
```



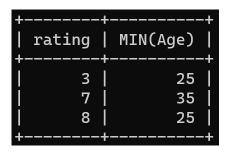
7. Find the age of the youngest sailor for each rating level

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

+	+	
rating	İ	youngest_age
1	Ī	33
3	Τ	25
7	Ĺ	35
8	Ĺ	25
9	ĺ	40
10	ĺ	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;



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;

<u> </u>	ONDER DI avg_age 7.5c,				
+		++			
1	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(<u>employee name</u>,street,city)
Works(<u>employee name</u>,company_name,salary)
Company(<u>company_name</u>,ciy)
Manages(<u>employee name</u>,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

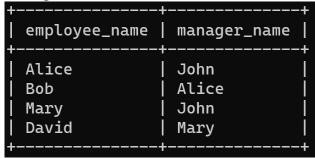
Works

+	 ++ salary
John Alice Bob Mary David	15000.00 12000.00 10000.00 18000.00 20000.00

company

company_name	
First Bank Corporation Small Bank Corporation Big Corporation	.

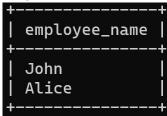
Manages



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';
```

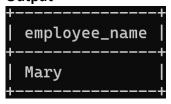
Output



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



- ✓ **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;

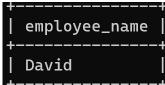
Output

employee_name	+ street 	+ city
John Alice +	123 Main St 456 Elm St +	

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'
);
```

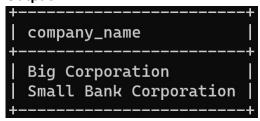
Output



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

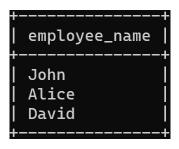
```
SELECT company_name
FROM Works
GROUP BY company_name
HAVING AVG(salary) > (
    SELECT AVG(salary)
    FROM Works
    WHERE company_name = 'First Bank Corporation'
);
```

Output



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;
```



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 Alice Bob Mary David	123 Main St 456 Elm St 123 Maple Avenue 123 Main st 222 Cedar St	Chicago New York Springfield New York 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
Alice Bob	First Bank Corporation First Bank Corporation Small Bank Corporation Small Bank Corporation Big Corporation	13200.00 10000.00

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
	First Bank Corporation First Bank Corporation Small Bank Corporation Small Bank Corporation Big Corporation	14520.00 10000.00

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

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

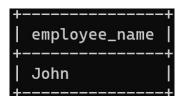
Now ,works relation becomes

+	company_name	+ salary
John Alice David	First Bank Corporation First Bank Corporation Big Corporation	

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



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

 $\prod_{loan-number} (\sigma_{amount > 1200} (loan))$

3. Find the names of all customers who have a loan, an account, or both, from the bank.

 $\prod_{customer-name}$ (borrower) $\cup \prod_{customer-name}$ (depositor)

4. Find the names of all customers who have a loan and an account at bank.

```
\prod_{customer-name} (borrower) \cap \prod_{customer-name} (depositor)
```

5. Find the names of all customers who have a loan at the Perryridge branch.

```
\prod_{\text{customer-name}} (\sigma_{\text{branch-name}} = \text{"Perryridge"} (\text{borrower} \bowtie \text{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.

```
\prod_{\text{customer-name}} (\sigma_{\text{branch-name}=\text{"Perryridge"}} (\text{borrower} \bowtie \text{loan})) - \prod_{\text{customer-name}} (\text{depositor})
```

7. Find the names of all customers who have a loan at the Perryridge branch.

```
\prod_{customer-name} (\sigma_{branch-name = "Perryridge"} (borrower \bowtie loan))
```

8. Find the largest account balance.

$${\cal G}_{_{\sf max(balance)}}$$
 (account

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

```
\prod customer-name (\sigmabranch-name = "Downtown" (depositor \bowtie account)) \cap \prod customer-name (\sigmabranch-name = "Uptown" (depositor \bowtie account))
```

10. Find all customers who have an account at all branches located in Brooklyn city.

```
\prod_{customer-name, branch-name} (depositor \bowtie account)
\div \prod_{branch-name} (\sigma_{branch-city} = \text{``Brooklyn''} (branch))
```

11. Delete all account records in the Perryridge branch.

```
account \leftarrow account - \sigma_{branch-name = "Perryridge"} (account)
```

12. Delete all loan records with amount in the range of 0 to 50.

```
loan \leftarrow loan - \sigma_{amount \ge 0 \ and \ amount \le 50} \ (loan)
```

13. Delete all accounts at branches located in Needham.

```
r_1 \leftarrow \prod_{branch-name, account-number, balance} (\sigma_{branch-city} = "Needham" (account \bowtie branch))
r_2 \leftarrow \prod_{customer-name, account-number} (r_1 \bowtie 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 p	payments by	y increasing	g all balances b	y 5	percent.
------------	---------------	------------	-------------	--------------	------------------	-----	----------

```
account \leftarrow \prod account-number, branch-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 \prod_{account-number, branch-name, balance * 1.06} (\sigma_{balance > 10000} (account))
\cup \prod_{account-number, branch-name, balance * 1.05} (\sigma_{balance \le 10000} (account))
```

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

```
create view all-customer as
```

```
\prod_{\text{branch-name, customer-name}} (\text{depositor} \bowtie \text{account}) \\ \cup \prod_{\text{branch-name, customer-name}} (\text{borrower} \bowtie \text{loan})
```

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

employee(person name, street, city)

```
works(person_name,company_name,salary)

company(company_name,city)

manages(person_name,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="first bank corporation"}(works))
```

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

```
∏person_name,city (σcompany_name="first bank corporation" (employee⋈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="first bank corporation"^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.

```
\prod_{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.

```
\prod_{\text{person\_name}} ((\text{employee} \bowtie \text{manages}) \bowtie_{\text{(manages.manager\_name=employee2.person\_name }^{\land}} \\ \text{employee2.street=employee2.street }^{\land} (\rho_{\text{employee2}}, \rho_{\text{employee2}})))
```

- 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.

$$_{
m company_name} \mathcal{G}_{
m avg(salary)}$$
 (works)

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

```
\prod_{\text{person\_name}} ((works \bowtie manages) \bowtie_{\text{manages.manager\_name}=e2.person\_name ^ works.salary>e2.salary} (\rho_{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 \in \mathcal{G}_{\text{max(salary)}} (\sigma_{\text{company\_name}="NBL company"}(works)) result \in \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

```
\prod_{person\_name} (\sigma_{city="Lalitpur"^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.

```
\prod company_name,city(company) \div \prodcity (\sigmacompany_name = "small bank corporation"(company))
```

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

```
\Pi_{\text{Employee.person\_name,Employee.city}} (\sigma_{\text{Company.city="Pokhara"^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".

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

xvi)Increase salary of all employee by 15%.

```
Works←∏ person_name,company_name,salary*1.15(Works)
```

xvii) Insert new records in employee relation.

```
employee←employee U {"Ram","patan", "Lalitpur"}
```

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

```
Create view high_earning_view

σ<sub>salary>=20000</sub> (∏ person name,street,city,company name,salary(employee ⋈ works))
```

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

```
employee \leftarrow \prod_{person\_name, street, "Kathmandu"} (\sigma_{person\_name="Ramesh"} (employee))
U (employee-\sigma_{person\_name="Ramesh"} (Employee))
```

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

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