Chap-3 SQL and Relational Algebra

Features of SQL

-> Uses simple English-like commands.

-> Works across various database systems like MySQL, PostgreSQL, SQL Server, and Oracle.

-> Optimized for quick data retrieval and manipulation.

-> Handles large datasets efficiently.

-> Supports various operations like CRUD (Create, Read, Update, Delete)

SQL Queries

-> SQL queries are commands used to interact with a database.

1. Data Query Language (DQL): Used for retrieving data from the database.

all

```
SELECT * FROM students;
SELECT name, age FROM students WHERE age > 18;
```

2. Data Definition Language (DDL) : Used to define and modify database structures.

```
CREATE TABLE students (id INT PRIMARY KEY, name VARCHAR(50),
 age INT);
ALTER TABLE students ADD COLUMN email VARCHAR(100);
DROP TABLE students;
```

3. Data Manipulation Language (DML) : Used to modify data in tables.

```
INSERT INTO students (id, name, age) VALUES (1, 'John', 20);
UPDATE students SET age = 21 WHERE id = 1;
DELETE FROM students WHERE id = 1;
```

4. Data Control Language (DCL) : Used for granting and revoking user permissions

GRANT SELECT ON students TO user1;
REVOKE SELECT ON students FROM user1;

SQL Subqueries

A subquery is a query within another SQL query. It is used to retrieve data that will be used in the main query.

1. SELECT name FROM students WHERE age = (SELECT MAX(age) FROM students); (Output (Student with Maximum Age))

2. SELECT name FROM students WHERE age IN (SELECT age FROM students WHERE age > 20); (Output (Students Older Than 20))

3. SELECT name FROM students s1 WHERE age > (SELECT AVG(age) FROM students s2 WHERE s1.id != s2.id); Output (Students Older Than Average Age):

4. SELECT name FROM students WHERE age = (SELECT MIN(age) FROM (SELECT age FROM students WHERE age > 18) AS temp);
Output (Youngest Student Over 18):

Set Operations in SQL

Set operations are used to combine the results of two or more SELECT queries. These operations require that both queries have the same number of columns and compatible data types.

Let's consider two tables:

Table 1: Students

id	name
1	Alex
2	Bob
3	Charlie
4	David

Table 2: Employees

id	name
1	Bob
2	Charlie
3	Eve
4	Frank

1. UNION (Removes Duplicates)

-> The UNION operator combines results from multiple queries but removes duplicate rows.

```
SELECT name FROM Students
UNION
SELECT name FROM Employees;
```

Result Table

name
Alex
Bob
Charlie
David
Eve
Frank

2. UNION ALL (Keeps Duplicates)

-> The UNION ALL operator works like UNION but keeps duplicates.

```
SELECT name FROM Students
UNION ALL
SELECT name FROM Employees;
```

3. INTERSECT (Finds Common Rows)

-> The INTERSECT operator returns only the rows that appear in both queries.

```
SELECT name FROM Students
INTERSECT
SELECT name FROM Employees;
```

Result Table

name

Bob

Charlie

4. EXCEPT (Finds Unique Rows in First Query)

-> The EXCEPT operator (or MINUS in some databases) returns rows from the first query that do not exist in the second query.

```
SELECT name FROM Students
EXCEPT / MINUS / DIFFERENCE
SELECT name FROM Employees;
```

name

Bob

Charlie

Joins

Joins in SQL allow us to combine rows from two or more tables based on a related column. This is useful when data is spread across multiple tables.

Types of Joins

Join Type	Description
INNER JOIN	Returns rows that have matching values in both tables.(no nukk)
LEFT JOIN (LEFT OUTER JOIN)	Returns all rows from the left table and matching rows from the right table. (null can in right)
RIGHT JOIN (RIGHT OUTER JOIN)	Returns all rows from the right table and matching rows from the left table. (null can on left table)
FULL JOIN (FULL OUTER JOIN	Returns all rows from both tables and NULL for missing matches.

Example Tables

Table: Students

student_id	name	course_id
1	Alex	101
2	Bob	102
3	Charlie	NULL
4	David	104

Table: Courses

course_id	course_name
101	Math
102	Science
103	History
104	English

```
1. SELECT Students.name, Courses.course_name
FROM Students
INNER JOIN Courses
ON Students.course_id = Courses.course_id;
```

Result Table

name	course_name
Alex	Math
Bob	Science
David	English

✕ Charlie is missing because he has NULL in course_id, and History is missing because no student is enrolled in course 103.

2. SELECT Students.name, Courses.course_name
FROM Students
LEFT JOIN Courses
ON Students.course_id = Courses.course_id;

Result Table
name course_name
Alex Math
Bob Science
Charlie NULL
David English

Charlie appears with NULL because he has no matching course.

3. SELECT Students.name, Courses.course_name
FROM Students
RIGHT JOIN Courses
ON Students.course_id = Courses.course_id;

Result Table
name course_name
Alex Math
Bob Science
David English
NULL History

History appears with NULL because no student is enrolled in it.

4. SELECT Students.name, Courses.course_name
FROM Students
FULL JOIN Courses
ON Students.course_id = Courses.course_id;

Result Table
name course_name
Alex Math
Bob Science
Charlie NULL
David English
NULL History

Both Charlie and History appear with NULL since they don't have a match.

SQL Commands

a. DDL

1. Create Database

```
CREATE DATABASE database_name;
CREATE DATABASE Nepal;
```

2. Create Table

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    ...
);
```

```
CREATE TABLE student (
    s_id INT,
    s_name VARCHAR(20),
    s_address VARCHAR(20)
);
```

3. Add a Column

```
ALTER TABLE table_name
ADD column_name datatype;
```

```
ALTER TABLE student
ADD date_of_birth DATE;
```

4. Delete a Column

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

```
ALTER TABLE student
DROP COLUMN date_of_birth;
```

5. Modify Column Data Type

```
ALTER TABLE table_name
ALTER COLUMN column_name new_datatype;
```

```
ALTER TABLE student
ALTER COLUMN s_id VARCHAR(20);
```

6. Add Constraint

```
ALTER TABLE student
ADD CONSTRAINT PK_student PRIMARY KEY (s_id);
```

7. Drop Constraint

```
ALTER TABLE student
DROP CONSTRAINT PK_student;
```

8. Drop Database, Table

```
DROP DATABASE database_name;
DROP TABLE table_name;
```

9. Deletes all data from a table but retains its structure.

```
TRUNCATE TABLE table_name;
```

10. Rename Database, Table

```
ALTER DATABASE Nepal MODIFY NAME = Nepal_college;
RENAME TABLE emp TO employee;
```

b. Data Manipulation Language (DML)

1. INSERT Operation :

-> insert single row

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

```
INSERT INTO student (s_id, s_name, s_address)
VALUES (1, 'Binary', 'Pokhara');
```

-> Insert Multiple Rows

```
INSERT INTO student (s_id, s_name, s_address)
VALUES (2, 'Bibek', 'Dharan'),
       (3, 'Ajay', 'Baglung'),
       (4, 'Meera', 'Pokhara');
```

2. Select Operation

-> Select All Rows

```
SELECT * FROM table_name;
SELECT * FROM emp;
```

-> Select Specific Columns

```
SELECT column1, column2 FROM table_name;
SELECT s_id, s_name FROM emp;
```

-> Filter with WHERE Clause

```
SELECT column1, column2
FROM table_name
WHERE condition;
```

```
SELECT empno, empname
FROM emp
WHERE sal > 40000;
```

-> Sort with ORDER BY

```
SELECT column1, column2
FROM table_name
ORDER BY column_name [ASC|DESC];  -- ASC is default

SELECT name FROM instructor ORDER BY name;  (ascending
alphabetical order)
SELECT name FROM instructor ORDER BY name DESC;
(descending order)
```

-> Insert Records via SELECT

```
INSERT INTO target_table (SELECT columns FROM source_table);
INSERT INTO emp1 (SELECT * FROM emp);
```

-> Column Aliases

```
SELECT column_name AS alias_name FROM table_name;
SELECT emp_id AS Id, ename AS Name FROM emp;
```

-> Table Aliases

```
SELECT e.column_name
FROM emp AS e;
```

Pattern Matching with LIKE

Wildcards:

Wildcard in SQL server

Symbol	Description	Example
%	Represents zero or more characters	ma% finds ma, man, manager, mango
_	Represents a single character	c_t finds cat, cup and cut
[]	Represents any single character within the brackets	p[ou]t finds pot and put but not pit
^	Represents any character not in the brackets	h[^oa]t finds hit but not hot and hat
-	Represents a range of characters	c[a-b]t finds cat and cbt

All the wildcards can be used in combination.

SQL Query Patterns with LIKE Operator

i. Ends with 'engineer':

```
SELECT * FROM emp WHERE position LIKE '%engineer';
```

ii. Starts with 'Associate':

```
SELECT * FROM emp WHERE position LIKE 'Associate%';
```

iii. Contains 'engineer':

```
SELECT * FROM emp WHERE position LIKE '%engineer%';
```

iv. Starts with 'A' and ends with 'r':

```
SELECT * FROM emp WHERE position LIKE 'A%r';
```

v. Second letter is 's':

```
SELECT * FROM emp WHERE position LIKE '_s%';
```

(Underscore _ matches a single character.)

vi. 'e' as the third-last letter:

```
SELECT * FROM emp WHERE position LIKE '%e_ _';
```

(Two underscores after e to match the last two characters.)

vii. Department head name starts with 'r' and has ≥3 letters:

```
SELECT * FROM dept WHERE d.head LIKE 'r_ _%';
```

Update Operations

a. Update Single Column:

```
UPDATE table_name SET column_name = new_value WHERE condition;
UPDATE emp SET salary = 100000 WHERE emp_id = 5;
```

b. Update Multiple Columns:

```
UPDATE table_name SET column1 = value1, column2 = value2, ...
WHERE condition;
UPDATE emp SET ename = 'Bijay', address = 'Bhaktapur' WHERE
 emp_id = 5;
```

3. Delete Operation

```
DELETE FROM table_name WHERE condition;
DELETE FROM emp WHERE eid = 5;
```

-> Warning: Omitting the WHERE clause deletes all records in the table.

4. Data Control Language (DCL)

```
GRANT privilege_list ON table_name TO user [WITH GRANT OPTION];

GRANT SELECT, UPDATE ON employees TO rabin;
-- Allows viewing/modifying records

GRANT SELECT, UPDATE ON employees TO rabin WITH GRANT OPTION;
-- -- Allows rabin to grant these privileges to others

REVOKE privilege_list ON table_name FROM user;
REVOKE INSERT, SELECT ON employees FROM rabin;

CREATE ROLE testing;
GRANT CREATE TABLE TO testing;
GRANT testing TO rabbin;
REVOKE CREATE TABLE FROM testing;
```

Aggregate Functions

Return a single value from a set of values.

Function	Description	Syntax
`AVG()`	Average of numeric values	`SELECT AVG(column) FROM table;`
`COUNT()`	Number of rows	`SELECT COUNT(column) FROM table;`
`MAX()`	Maximum value	`SELECT MAX(column) FROM table;`
`MIN()`	Minimum value	`SELECT MIN(column) FROM table;`
`SUM()`	Sum of numeric values	`SELECT SUM(column) FROM table;`

- SELECT AVG(salary) FROM emp;
  SELECT COUNT(*) FROM emp; -> total employess

SQL Aggregation and Grouping

1. GROUP BY Clause

=> Groups rows with identical values in specified columns. Used with aggregate functions (e.g., COUNT, SUM).

```
SELECT column_name(s), aggregate_function(column)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name;
```

-> (Count customers per country)

```
SELECT COUNT(Cust_ID), Country
FROM Customers
GROUP BY Country;
```

COUNT(Cust_ID)	Country
1	Bangladesh
2	India
1	Nepal
1	Pakistan

-> Sorting Results (High to Low):

```
SELECT COUNT(Cust_ID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(Cust_ID) DESC;
```

2. GROUP BY with JOINs

-> Groups rows with identical values in specified columns. Used with aggregate functions (e.g., COUNT, SUM).

-> Example (Count orders per shipper using LEFT JOIN):

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrder
FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

ShipperName	NumberOfOrder
Federal Shipping	1
Speedy Express	1
United Package	2

HAVING Clause

-> Filters groups created by GROUP BY (unlike WHERE, which filters rows).
Used with aggregate functions.

```
SELECT column_name(s), aggregate_function(column)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition;
```

Example: (Countries with >1 customer):

```
SELECT COUNT(Cust_ID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(Cust_ID) > 1;
```

```
COUNT(Cust_ID) | Country
2              | India
```

Key Takeaways:

Order of Clauses:

SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY.

Best Practices:

->Always alias aggregate results for readability (e.g., AS NumberOfOrder).
->Use LEFT JOIN to include all rows from the left table, even if there are no matches.

Views

A view in SQL is a virtual table that provides a customized representation of data from one or more tables. Unlike actual tables, views do not store data; instead, they store queries that dynamically fetch data from the underlying tables.

Creating a View

The CREATE VIEW statement is used to define a view.

Example 1: Creating a Simple View

Let's assume we have a table named Employees:

emp_id	name	department	salary
1	Alice	HR	50000
2	Bob	IT	60000
3	Charlie	IT	55000
4	David	Finance	70000

Now, we create a view that shows only IT department employees:

```
CREATE VIEW IT_Employees AS
SELECT emp_id, name, salary
FROM Employees
WHERE department = 'IT';

SELECT * FROM IT_Employees;
```

emp_id	name	salary
2	Bob	60000
3	Charlie	55000

Updating Views

Views can be updated if they meet certain conditions (like being based on a single table without aggregations).

```
UPDATE IT_Employees
SET salary = 62000
WHERE emp_id = 2;
```

Modifying a View

```
CREATE OR REPLACE VIEW IT_Employees AS
SELECT emp_id, name, department, salary
FROM Employees
WHERE department = 'IT';
```

Deleting a View

```
DROP VIEW IT_Employees;
```

Aggregated View Example

If we want to see the average salary per department:

```
CREATE VIEW Avg_Salary AS
SELECT department, AVG(salary) AS avg_salary
FROM Employees
GROUP BY department;

SELECT * FROM Avg_Salary;
```

SQL OLD Questions

Consider the following relational database model:
Hotel (Hotel_No, Name, Address)
Room (Room_No, Hotel_No, Type, Price)
Booking (Hotel_No, Guest_No, Date_From, Date_To, Room_No)
Guest (Guest_No, Name, Address) [2079 Ashwin]
Write SQL statement for the following:

a. List all the guests who have booked rooms at the Everest Hotel.

Inner join

```
SELECT G.Guest_No, G.Name, G.Address
FROM Guest G
JOIN Booking B ON G.Guest_No = B.Guest_No
JOIN Hotel H ON B.Hotel_No = H.Hotel_No
WHERE H.Name = 'Everest Hotel';
```

(Guest ko cha, Guest ra hotel ko direct link not possible so joining Guest with booking and again with hotel)

b. Create a view to expose only the Hotel_No, Guest_No, Room_No. and price of the room of all booked rooms.

```
CREATE VIEW BookedRoomDetails AS
SELECT B.Hotel_No, B.Guest_No, B.Room_No, R.Price
FROM Booking B
JOIN Room R ON B.Hotel_No = R.Hotel_No
AND B.Room_No = R.Room_No;
```

ON
r.Room_No = b.Room_No;

c. **Find total cost of all the deluxe room of Everest Hotel after offering 5% discount.**

= SUM(r.Price * 0.95) AS TotalCost

```
SELECT SUM(r.Price * 0.95) AS TotalCost
FROM Room r
JOIN Hotel h ON r.Hotel_No = h.Hotel_No
WHERE h.Name = 'Everest' AND r.Type = 'Deluxe';
```

d. Identify the Hotel name which has the highest total guests.

```
SELECT H.Name, COUNT(B.Guest_No) AS TotalGuests
FROM Hotel H
JOIN Booking B ON H.Hotel_No = B.Hotel_No
GROUP BY H.Name
ORDER BY TotalGuests DESC
LIMIT 1;
```

2. Consider the relational database as follows:
Worker(worker_id, first_name, last_name, salary, joining_date, department)
Bonus(worker_id, bonus_amount, bonus_date)
Title(worker_id, worker_title, affected_from) [2079 JES.]

i. Write an expression in SQL to fetch unique values of department from worker table.

```
SELECT DISTINCT department FROM Worker;
```

-> DISTINCT: Ensures only unique values are returned for the specified column.

ii. Write an expression in SQL to print details of Workers with DEPARTMENT name as "Admin".

```
SELECT * FROM Worker WHERE department = 'Admin';
```

iii. Write an expression in SQL to print details of Workers who are also Managers.

```
SELECT * FROM Worker w
JOIN Title t ON w.worker_id = t.worker_id
WHERE t.worker_title = 'Manager';
```

iv. Write an expression in SQL to show the second highest salary from a table.

```
SELECT DISTINCT salary
FROM Worker
ORDER BY salary DESC  - Sorts salaries in descending order.
LIMIT 1 OFFSET 1;  Skips the highest salary (OFFSET 1) and
selects the second highest.
```

3. Consider the following relational database model:

Passenger (pid, pname, pgender, pbirthplace)

Agency (aid, aname, acity)

Flight (fid, fdate, time, source, destination)

Booking (pid, aid, fid, bookdate, amount) [2078 Chaitra]

a. Find all the passenger details who are travelling from "Kathmandu" to "Pokhara".

```
SELECT DISTINCT *
FROM Passenger P
JOIN Booking B ON P.pid = B.pid
JOIN Flight F ON B.fid = F.fid
WHERE F.source = 'Kathmandu' AND F.destination = 'Pokhara';
```

b. Update the booking amount with 10% discount if the flight destination is same as the passenger's birth city.

```
UPDATE Booking B
SET amount = amount * 0.9
FROM Booking B
JOIN Passenger P ON B.pid = P.pid
JOIN Flight F ON B.fid = F.fid
WHERE F.destination = P.pbirthplace;
```

c. Create a VIEW named "EsewaReport" in which calculate the total amount of booking made in the current date through the agency name "Esewa".

```
CREATE VIEW EsewaReport AS
SELECT SUM(B.amount) AS total_amount
FROM Booking B
JOIN Agency A ON B.aid = A.aid
WHERE A.aname = 'Esewa' AND B.bookdate = CURRENT_DATE;
```

built in

d. List Flight wise total number of bookings for current date.

```
SELECT B.fid, COUNT(B.pid) AS total_bookings
FROM Booking B
WHERE B.bookdate = CURRENT_DATE
GROUP BY B.fid;
```

---

3. Consider the relational model.

Employee (empid, empname, address, title)

Project (pid, pname, budget, location)

Assignment (empid, pid, responsibility, duration)

Payment (title, salary) [2077 Chaitra]

a. Write SQL to count the number of projects with duration more than 2 years.

```
SELECT COUNT(DISTINCT pid) AS NumberOfProjects
FROM Assignment
WHERE duration > 2;
```

b. Write SQL query to find the name of engineers working in ICTC project and earning salary more than 20K

```
SELECT DISTINCT e.empname
FROM Employee e
JOIN Assignment a ON e.empid = a.empid
JOIN Project p ON a.pid = p.pid
JOIN Payment pay ON e.title = pay.title
WHERE p.pname = 'ICTC project'
  AND e.title = 'engineer'
  AND pay.salary > 20000;
```

c. Write SQL to update salary of employees by 5% if salary less than 10K, by 7% if salary between 10k and 20K and, by 9% if salary greater than 20K.

```
UPDATE Payment
SET salary = CASE
    WHEN salary < 10000 THEN salary * 1.05
    WHEN salary BETWEEN 10000 AND 20000 THEN salary * 1.07
    WHEN salary > 20000 THEN salary * 1.09
END;
```

4. Consider the following relational data model:

Student (crn, name, address, phone, dob)

Course (courseid, crn, duration, fee)

Enroll (enrolled, cname, courseid, enrolldata, completedata) [2075 Bhadra]

i. Write the SQL statements required to create the above relations, including appropriate versions of all primary key integrity constraints.

```
CREATE TABLE Student (
    crn INT PRIMARY KEY,
    name VARCHAR(50),
    address VARCHAR(100),
    phone VARCHAR(15),
    dob DATE
);

CREATE TABLE Course (
    courseid INT PRIMARY KEY,
    crn INT,
    duration INT,
    fee DECIMAL(10,2),
    FOREIGN KEY (crn) REFERENCES Student(crn)
);

CREATE TABLE Enroll (
    enrolled INT PRIMARY KEY,
    cname VARCHAR(50),
    courseid INT,
    enrolldata DATE,
    completedata DATE,
    FOREIGN KEY (courseid) REFERENCES Course(courseid)
);
```

ii. Write an expression in SQL to find crn, names and enroll data of all students who have taken the course 'java'(cname)

```
SELECT s.crn, s.name, e.enrolldata
FROM Student s
JOIN Course c ON s.crn = c.crn
JOIN Enroll e ON c.courseid = e.courseid
WHERE e.cname = 'java';
```

iii. Write SQL to find the names and address of all students who have taken both course java and linux.

```
SELECT s.name, s.address
FROM Student s
JOIN Course c1 ON s.crn = c1.crn
JOIN Enroll e1 ON c1.courseid = e1.courseid
JOIN Course c2 ON s.crn = c2.crn
JOIN Enroll e2 ON c2.courseid = e2.courseid
WHERE c1.cname = 'Java'
  AND c2.cname = 'Linux'
  AND e1.enrolled = e2.enrolled;
```

iv. Write an expression in SQL to create a view 'student_course' having the attributes crn, name, phone, coursename, enrolldata.

```
CREATE VIEW student_course AS
SELECT s.crn, s.name, s.phone, c.cname AS coursename, e.enrolldata
FROM Student s
JOIN Course c ON s.crn = c.crn;
JOIN Enroll e ON c.courseid = e.courseid
```

5. 12. Consider the following relational schema:
tblSalesman (s_id, name, city, commission)
tblOrders (ord_no, prch_amt, ord_date, c_id, s_id)
tblCustomer (c_id, name, city, grade, s_id)
[2075 Baisakh]

a. Find those salesmen with all information whose name contains the 1st character as 'N' and the 4th character as 'R', while the rest may be any character.

```
SELECT *
FROM tblSalesman
WHERE name LIKE 'N_ _R%';
```

b. Find the highest purchase amount on the date '2017-07-17' for each salesman with their ID.

```
SELECT MAX(prch_amt), s_id
FROM tblOrders
WHERE ord_date = '2017-07-17'
GROUP BY s_id;
```

c. Count the customers with grades above Kathmandu's average.

```
SELECT COUNT(c_id) AS num_customers_above_avg
FROM tblCustomer
WHERE grade > (
    SELECT AVG(grade)
    FROM tblCustomer
    WHERE city = 'Kathmandu'
);
```

d. Increase the commission of salesmen by 2% if they are from Humla.

```
UPDATE tblSalesman
SET commission = 1.02 * commission
WHERE city = 'Humla';
```

duration is more than 5 years.

b) Consider the following insurance database. [4×2]

PERSON(licenseNo, name, address)
CAR(modelNo, brand, year)
ACCIDENT(reportNo, date, location)
OWNS(licenseNo,modelNo)
PARTICIPATED(licenseNo, reportNo, damageAmount)

Write SQL expression for the given queries:

i) Display all the detail of a Person whose name ends with 'ta' and is involved in some accident.
ii) Display the license numbers and location where the accident took place on Jan 20, 2020.
iii) Update the brand name "BMW" to "BMW-X" for car manufactured in year 2020.
iv) Create a view named PERSON_REPORT which contains license No, Name and report No as its members where the damage amount is less than or equal to 100000.

i) Display all the detail of a Person whose name ends with 'ta' and is involved in some accident.

```
SELECT *
FROM PERSON P
INNER JOIN PARTICIPATED PT ON P.licenseNo = PT.licenseNo
WHERE P.name LIKE '%ta';
```

ii) Display the license numbers and location where the accident took place on Jan 20, 2020.

```
SELECT PT.licenseNo, A.location
FROM ACCIDENT A
INNER JOIN PARTICIPATED PT ON A.reportNo = PT.reportNo
WHERE A.date = '2020-01-20';
```

iii) Update the brand name "BMW" to "BMW-X" for car manufactured in year 2020.

```
UPDATE CAR
SET brand = 'BMW-X'
WHERE brand = 'BMW' AND year = 2020;
```

iv) Create a view named PERSON_REPORT which contains license No, Name, and report No where the damage amount is ≤ 100000.

```
CREATE VIEW PERSON_REPORT AS
SELECT P.licenseNo, P.name, PT.reportNo
FROM PERSON P
INNER JOIN PARTICIPATED PT ON P.licenseNo = PT.licenseNo
WHERE PT.damageAmount <= 100000;
```

. a) Consider the relational database as follows. Write SQL for each of the following. [8]

Doctor (name, age, address)
Works (name, dept no)
Department (dept no, floor, room)

i) To display the records of doctor with their department information.
ii) To find total number of rooms assigned in each floor.
iii) To display the name of doctor with maximum age.
iv) To delete the records of doctors whose name start with 'M' and works in 10^{th} floor.

i) Display the records of doctors with their department information.

```
SELECT D.*, Dept.floor, Dept.room
FROM Doctor D
INNER JOIN Works W ON D.name = W.name
INNER JOIN Department Dept ON W.dept_no = Dept.dept_no;
```

ii) Find the total number of rooms assigned in each floor.

```
SELECT floor, COUNT(room) AS total_rooms
FROM Department
GROUP BY floor;
```

iii) Display the name of the doctor with maximum age.

```
SELECT name
FROM Doctor
WHERE age = (SELECT MAX(age) FROM Doctor);
```

iv) Delete the records of doctors whose name starts with 'M' and works in 10th floor.

```
DELETE FROM Doctor
WHERE name LIKE 'M%' AND name IN (
    SELECT W.name
    FROM Works W
    INNER JOIN Department D ON W.dept_no = D.dept_no
    WHERE D.floor = 10
);
```

supervisor can change over the lifetime of the contract.

3. Write SQL query. [Consider following relations]
   Product(Pid, Pname, Price, description)
   Customer(Cid, Cname, Address)
   Sells(Pid, Cid, quantity)
   a) Retrieve the record of product who were sold to customer id 12.
   b) Create above table product as indicated.
   c) Find the product whose sells quantity is maximum.
   d) Find the total number of customer whose name start with S.

a) Retrieve the record of product who were sold to customer id 12.

```
SELECT P.*
FROM Product P
INNER JOIN Sells S ON P.Pid = S.Pid
WHERE S.Cid = 12;
```

b) Create the Product table as indicated.

```
CREATE TABLE Product (
    Pid INT PRIMARY KEY,
    Pname VARCHAR(255),
    Price DECIMAL(10, 2),
    description VARCHAR(255)
);
```

c) Find the product whose sells quantity is maximum.

```
SELECT P.*
FROM Product P
JOIN Sells S ON P.Pid = S.Pid
GROUP BY P.Pid
ORDER BY SUM(S.quantity) DESC
LIMIT 1;
```

d) Find the total number of customers whose name starts with S.

```
SELECT COUNT(cid) AS total_customers
FROM Customer
WHERE Cname LIKE 'S%';
```

3. a) Consider the following relational data model [2×4

Employee (empid, empname, address, title)
Project (pid, proj_name, budget, location)
Assignment (empid, pid, responsibility, duration)
Payment (title, salary)

(i) Write an SQL query to find the name and salary of Engineers.
(ii) Write an SQL query to find the name of employee working in projects in their own city.
(iii) Write a query to create a view named empdetails with empname, address, proj_name and salary.
(iv) Write an SQL to find the names of employees who works in "CAD/CAM" project

(i) Find the name and salary of Engineers:

```
SELECT e.empname, p.salary
FROM Employee e
JOIN Payment p ON e.title = p.title
WHERE e.title = 'Engineer';
```

(ii) Find the name of employees working in projects in their own city:

```
SELECT e.empname
FROM Employee e
JOIN Assignment a ON e.empid = a.empid
JOIN Project p ON a.pid = p.pid
WHERE e.address = p.location;
```

(iii) Create a view named empdetails with empname, address, proj_name, and salary:

```
CREATE VIEW empdetails AS
SELECT e.empname, e.address, p.proj_name, py.salary
FROM Employee e
JOIN Assignment a ON e.empid = a.empid
JOIN Project p ON a.pid = p.pid
JOIN Payment py ON e.title = py.title;
```

(iv) Find names of employees who work in the "CAD/CAM" project:

```
SELECT e.empname
FROM Employee e
JOIN Assignment a ON e.empid = a.empid
JOIN Project p ON a.pid = p.pid
WHERE p.proj_name = 'CAD/CAM';
```

3. Consider the following relational data model

   Employee (empid, ename, age, salary)
   Department (deptid, dname, budget, managerid)
   Works (empid, deptid, hours)

   (i) Write the SQL statements required to create the above relations, including appropriate versions of all primary and foreign key integrity constraints.
   (ii) Write an expression in SQL to find the name of department whose employee earns the maximum salary.
   (iii) Write SQL to find the name of the employee, department name and the number of hours they work
   (iv) Write an expression in SQL to give every employee a 20% raise in salary whose age is in between 45 to 50 years.

(i) SQL Create Table Statements

```
CREATE TABLE Employee (
    empid INT PRIMARY KEY,
    ename VARCHAR(255) NOT NULL,
    age INT,
    salary DECIMAL(10,2)
);

CREATE TABLE Department (
    deptid INT PRIMARY KEY,
    dname VARCHAR(255) NOT NULL,
    budget DECIMAL(15,2),
    managerial INT,
    FOREIGN KEY (managerial) REFERENCES Employee(empid)
);

CREATE TABLE Works (
    empid INT,
    deptid INT,
    hours INT,
    PRIMARY KEY (empid, deptid),
    FOREIGN KEY (empid) REFERENCES Employee(empid),
    FOREIGN KEY (deptid) REFERENCES Department(deptid)
);
```

(ii) Department with Employee Earning Maximum Salary

```
SELECT d.dname
FROM Department d
JOIN Works w ON d.deptid = w.deptid
JOIN Employee e ON w.empid = e.empid
WHERE e.salary = (SELECT MAX(salary) FROM Employee);
```

(iii) Employee Name, Department Name, and Hours Worked

```
SELECT e.ename, d.dname, w.hours
FROM Employee e
JOIN Works w ON e.empid = w.empid
JOIN Department d ON w.deptid = d.deptid;
```

(iv) 20% Salary Raise for Employees Aged 45–50

```
UPDATE Employee
SET salary = salary * 1.20
WHERE age BETWEEN 45 AND 50;
```

3. Consider the following relational scheme: [2×6

Account (account number, branch_name, balance)

Branch (branch name, branch_city, assets)

Customer (Customer name, customer_street, customers_city)

Loan (loan number, branch_name, amount)

Depositor (customer name, account number)

Borrower (customer name, loan number)

a) Write SQL Query expressions to list all the customers details, branch details and account details according to account number.
b) Write SQL Query expressions to list the branch name where the average account balance is more than 50,000.
c) Write SQL Query expressions to increase all accounts with balances over $10,000 by 5% and other accounts receive 6%
d) Write a query in SQL to list the branch _cities and total assets where the total assets are more than $10,00,000 in the city.

a) List all customers, branch, and account details by account number

```
SELECT c.*, b.*, a.*
FROM Customer c
JOIN Depositor d ON c.customer_name = d.customer_name
JOIN Account a ON d.account_number = a.account_number
JOIN Branch b ON a.branch_name = b.branch_name
ORDER BY a.account_number;
```

b) Branch names with average account balance > $50,000

```
SELECT branch_name
FROM Account
GROUP BY branch_name
HAVING AVG(balance) > 50000;
```

c) Increase account balances by 5% or 6%

```
UPDATE Account
SET balance = CASE
    WHEN balance > 10000 THEN balance * 1.05
    ELSE balance * 1.06
END;
```

d) Branch cities with total assets > $10,000,000

```
SELECT branch_city, SUM(assets) AS total_assets
FROM Branch
GROUP BY branch_city
HAVING SUM(assets) > 10000000;
```

4. Consider the relational schema given below. [2 X 4 = 8]

Product (pid, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, quantity, pid)
Company (cid, name, stock price, country)
Person(ssn, name, phone number, city)

a) Write an SQL query to find the names of all Japanese companies which sell products of "Computer" category.
b) Write an SQL query to create a view to expose only the product id, name, category and maker country.
c) Write a query in SQL to decrease the stock price of all makers of "LCD" category products by 1%.

a) Names of Japanese companies selling "Computer" category products

```
SELECT c.name
FROM Company c
JOIN Product p ON c.cid = p.`maker-cid`
WHERE c.country = 'Japan' AND p.category = 'Computer';
```

b. b) Create view for product details and maker country

```
CREATE VIEW ProductDetails AS
SELECT p.pid, p.name AS product_name, p.category, c.country
 AS maker_country
FROM Product p
JOIN Company c ON p.`maker-cid` = c.cid;
```

c) Decrease stock price of "LCD" category makers by 1%

```
UPDATE Company
JOIN Product ON Company.cid = Product.`maker-cid`
SET stock_price = stock_price * 0.99
WHERE Product.category = 'LCD';
```

3. Consider the following relational database model:

employee (employee-name, street, city)
works (employee-name, company-name, salary)
company (company-name, city)
manages (employee-name, manager-name)

a) Write SQL queries for the following needs. [2×4]

i) Modify the database so that Jones now lives in city Pokhara.
ii) Give all employees of 'NABIL Bank' a 10 percent raise.
iii) Give all managers of 'NABIL Bank' a 30 percent raise unless the salary becomes greater than 100,000.
iv) Delete employee who has maximum amount of salary.

i) Update Jones's city to Pokhara

```
UPDATE employee
SET city = 'Pokhara'
WHERE employee-name = 'Jones';
```

ii) 10% raise for all 'NABIL Bank' employees

```
UPDATE works
SET salary = salary * 1.10
WHERE company-name = 'NABIL Bank';
```

iii) 30% raise for 'NABIL Bank' managers (if new salary ≤ 100,000)

```
UPDATE works
SET salary = CASE
    WHEN salary <= 100000 THEN salary = salary * 1.30
    ELSE salary
END
WHERE company-name = 'NABIL Bank'
AND employee-name IN (SELECT manager-name FROM manages);
```

iv) Delete employee with the maximum salary

```
DELETE FROM employee
WHERE employee-name IN (
    SELECT employee-name
    FROM works
    WHERE salary = (SELECT MAX(salary) FROM works)
);
```

employee(empname, street, city)
works(empname, companyname, salary)
company(companyname, city)
manages(empname, managername)
For the case of above database schema:

I. Write an expression in SQL to create the table employee.
II. Write an expression in SQL to inset a row into the table works.
III. Write an expression in SQL to find the name and cities of resident of all the employees who do not work for XYZ Pvt. Ltd.

I. Create the employee table

```
CREATE TABLE employee (
    empname VARCHAR(255) PRIMARY KEY,
    street VARCHAR(255),
    city VARCHAR(255)
);
```

II. Insert a row into the works table

```
INSERT INTO works (empname, companyname, salary)
VALUES ('John Doe', 'ABC Corp', 75000);
```

III. Names and cities of employees not working for "XYZ-Pvt Ltd"

```
SELECT e.empname, e.city
FROM employee e
WHERE e.empname NOT IN (
    SELECT w.empname
    FROM works w
    WHERE w.companyname = 'XYZ-Pvt Ltd'
);
```

4. Consider the following relational database. [2 X 5 = 10]

account (account-number, branch-name, balance)
branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

a) Write an SQL query to list the names of all depositors along with their account number, street and city address.
b) Write a query in SQL to list the branch-cities and total assets where the total assets are more than $1,000,000 in the city.
c) Write an SQL query to find the names and loan-numbers of all customers who have a loan of over $15,000.
d) Write a query in SQL to increase all accounts with balances over $10,000 by 6%.

a) Depositors with account number, street, and city

```
SELECT c.customer_name, d.account_number, c.customer_street,
 c.customer_city
FROM customer c
JOIN depositor d ON c.customer_name = d.customer_name;
```

b) Branch cities with total assets > $1,000,000

```
SELECT branch_city, SUM(assets) AS total_assets
FROM branch
GROUP BY branch_city
HAVING SUM(assets) > 1000000;
```

c) Customers with loans over $15,000

```
SELECT b.customer_name, l.loan_number
FROM borrower b
JOIN loan l ON b.loan_number = l.loan_number
WHERE l.amount > 15000;
```

d) Increase balances over $10,000 by 6%

```
UPDATE account
SET balance = balance * 1.06
WHERE balance > 10000;
```

4. Consider the following relational database. [2 X 5 = 10]

account (account-number, branch-name, balance)
branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

a) Write an SQL query to list the names of all depositors along with their account number and balance.
b) Write an SQL query to find the names of all customers who have a loan of over $12,000.
c) Write a query in SQL to increase all accounts with balances over $10,000 by 6%, and all other accounts by 5%.
d) Write a query in SQL to list the branch-names where the average account balance is more than $10,000.

a) Depositors with account number and balance

```
SELECT d.`customer-name`, d.`account-number`, a.balance
FROM depositor d
JOIN account a ON d.`account-number` = a.`account-number`;
```

b) Customers with loans over $12,000

```
SELECT b.`customer-name`
FROM borrower b
JOIN loan l ON b.`loan-number` = l.`loan-number`
WHERE l.amount > 12000;
```

c) Conditional balance increase (6% or 5%)

```
UPDATE account
SET balance = CASE
    WHEN balance > 10000 THEN balance * 1.06
    ELSE balance * 1.05
END;
```

d) Branches with average balance > $10,000

```
SELECT `branch-name`
FROM account
GROUP BY `branch-name`
HAVING AVG(balance) > 10000;
```

4. Consider the relational schema given below. [2 X 4 = 8]

Product (pid, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, quantity, pid)
Company (cid, name, stock price, country)
Person(ssn, name, phone number, city)

a) Write an SQL query to find the name and price of all products of "camera" category made in "Japan".
b) Write an SQL query to create a view to expose only the Buyer name, Seller name and product name from all transactions.
c) Write a query in SQL to increase the price of all products from DELL company by 5 %.

a) Name and price of "camera" category products made in Japan

```
SELECT p.name, p.price
FROM Product p
JOIN Company c ON p.`maker-cid` = c.cid
WHERE p.category = 'camera' AND c.country = 'Japan';
```

b) View for Buyer, Seller, and Product names

```
CREATE VIEW TransactionDetails AS
SELECT buyer.name AS buyer_name, seller.name AS seller_name, pr.name
AS product_name
FROM Purchase pu
JOIN Person buyer ON pu.`buyer-ssn` = buyer.ssn
JOIN Person seller ON pu.`seller-ssn` = seller.ssn
JOIN Product pr ON pu.pid = pr.pid;
```

c) Increase price of DELL products by 5%

```
UPDATE Product
JOIN Company ON Product.`maker-cid` = Company.cid
SET price = price * 1.05
WHERE Company.name = 'DELL';
```