```sql
-- Create Database
CREATE DATABASE iiuc_4dm;


-- Use Database
USE iiuc_4dm;


-- Create table
CREATE TABLE C241135 (
      s_id INT PRIMARY KEY,
    s_name VARCHAR(50)NOT NULL,
    s_phone VARCHAR(50),
    s_address VARCHAR(50)
);

CREATE TABLE teacher1 (
      t_id INT PRIMARY KEY,
    t_name VARCHAR(50) NOT NULL
);


CREATE TABLE teacher2 (
      T_id INT PRIMARY KEY,
    T_name VARCHAR(50) NOT NULL,
    T_dept VARCHAR(50),
    T_salary INT
);


CREATE TABLE students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    class VARCHAR(10)
);


CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(50),
    student_id INT
);







-- Insert in table
INSERT INTO C241135 (s_id, s_name, s_phone, s_address) VALUES
      (101, 'Hasan', '123456789', 'DHK'),
    (102, 'Kamal', '987654321', 'CTG'),
    (103, 'Rony', NULL, 'DHK'),
    (104, 'Rakib', '234681369', 'COM'),
```

```
    (105, 'Hasan', '234681369', 'CTG');


INSERT INTO teacher1 (t_id, t_name) VALUES
      (11, 'A'),
    (12, 'B'),
    (13, 'C');



INSERT INTO teacher2 (T_id, T_name, T_dept, T_salary) VALUES
      (201, 'Azad', "CSE", 1000),
    (202, 'Kamal', "EEE", 2000),
    (203, 'Hasan', "CE", 1000),
    (204, 'Kamal', "EEE", 1000),
    (205, 'Rony', "CSE", 2000),
    (206, 'Rakib', "CSE", 4000);



INSERT INTO students (student_id, student_name, class) VALUES
      (1, 'Alice', '4DM'),
      (2, 'Bob', '4DM'),
      (3, 'Charlie', '4DM'),
      (4, 'David', '4DM');


INSERT INTO courses (course_id, course_name, student_id) VALUES
      (101, 'Database', 1),
      (102, 'Math', 1),
      (103, 'Physics', 2),
      (104, 'Chemistry', NULL);






-- SELECT ( Basic SELECT )
SELECT * FROM C241135;

SELECT * FROM teacher1;

SELECT * FROM teacher2;

SELECT * FROM students;

SELECT * FROM courses;
```

```sql
-- Select ( Column Projection - specific columns only )
SELECT s_id FROM C241135;

SELECT s_name FROM C241135;

SELECT s_phone FROM C241135;

SELECT s_address FROM C241135;



-- Select ( Conditional Selection - using WHERE )
SELECT * FROM C241135 WHERE s_id = 102;

SELECT * FROM C241135 WHERE s_address = 'DHK';

SELECT s_id FROM C241135 WHERE s_name = 'Rakib';

SELECT s_name FROM C241135 WHERE s_id = 102;



-- Select ( Distinct Selection )
SELECT DISTINCT  s_address FROM C241135;



-- Select ( Column Manipulation )
SELECT s_id + 5 FROM C241135;

SELECT s_id + 5 AS new_s_id FROM C241135;



-- Select ( Range and Set Conditions )
SELECT * FROM C241135 WHERE s_id > 102;

SELECT * FROM C241135 WHERE s_id BETWEEN 102 AND 104;

SELECT * FROM C241135 WHERE s_id IN(101,104);

SELECT * FROM C241135 WHERE s_id % 2 = 0;



-- Select ( Pattern Matching with LIKE )
```

```
SELECT * FROM teacher2 WHERE UPPER(T_name) LIKE 'K%';
      -- upper function ...shuru hobe K diye

SELECT * FROM teacher2 WHERE UPPER(T_name) LIKE '%L';
      -- shesh hobe L diye

SELECT * FROM teacher2 WHERE UPPER(T_name) LIKE 'K%L';
      -- shuru hbe K diye shesh hbe L diye

SELECT * FROM teacher2 WHERE UPPER(T_name) LIKE '%a%';
      -- a ase kina dekhte name

SELECT * FROM teacher2 WHERE UPPER(T_name) LIKE '%a%d%';
      -- a ta d er age hbe

SELECT * FROM teacher2 WHERE UPPER(T_name) LIKE '_____';
      -- j  name e 5 ta char ase..mane 5 ta _



-- ========================================
-- 1. Starts with a letter
-- Names that start with 'K' (case-insensitive using UPPER)
SELECT *
FROM teacher2
WHERE UPPER(T_name) LIKE 'K%';
-- % matches any number of characters after 'K'


-- ========================================
-- 2. Ends with a letter
-- Names that end with 'L' (case-insensitive)
SELECT *
FROM teacher2
WHERE UPPER(T_name) LIKE '%L';
-- % matches any number of characters before 'L'


-- ========================================
-- 3. Starts and ends with specific letters
-- Names that start with 'K' and end with 'L'
SELECT *
FROM teacher2
WHERE UPPER(T_name) LIKE 'K%L';
-- % matches any characters in between


-- ========================================
-- 4. Contains a specific letter
-- Names that contain 'A' anywhere
SELECT *
FROM teacher2
WHERE UPPER(T_name) LIKE '%A%';
-- % before and after allows any characters around 'A'
```

```
-- ========================================
-- 5. Contains letters in specific order
-- Names that contain 'A' followed by 'D' anywhere
SELECT *
FROM teacher2
WHERE UPPER(T_name) LIKE '%A%D%';
-- % matches any characters between or around 'A' and 'D'


-- ========================================
-- 6. Exact length
-- Names with exactly 5 characters
SELECT *
FROM teacher2
WHERE UPPER(T_name) LIKE '_____';
-- Each _ represents one character (5 underscores = 5 characters)


-- ========================================
-- 7. Specific characters at specific positions
-- Names where first letter is 'A', 2nd letter can be anything, 3rd is
'D'
SELECT *
FROM teacher2
WHERE UPPER(T_name) LIKE 'A_D%';
-- _ matches a single character, % matches remaining characters


-- ========================================
-- 8. Match one of several characters at a position
-- Names where the first letter is A, H, or K
SELECT *
FROM teacher2
WHERE UPPER(T_name) LIKE '[AHK]%';
-- Note: In MySQL, [] character set is not supported in LIKE, use REGEXP
for this


-- ========================================
-- 9. Escape a wildcard character
-- Names containing an actual % symbol
SELECT *
FROM teacher2
WHERE T_name LIKE '%\%%' ESCAPE '\';
-- \% treats % as literal


-- ========================================
-- 10. Case-insensitive search
-- Names containing 'kamal' ignoring case
SELECT *
FROM teacher2
```

```
WHERE T_name LIKE '%kamal%';
-- MySQL default collation is case-insensitive; otherwise use UPPER() as
before



-- Select ( GROUP BY Queries )
SELECT T_dept FROM teacher2 WHERE T_id > 201 GROUP BY T_dept;
      -- sum mane value + korar jnno...count mane kotojon ase

SELECT T_dept, COUNT(*) FROM teacher2 WHERE T_id > 201 GROUP BY T_dept;
           -- kon dept e koijon ase tar count kore

SELECT T_dept, SUM(T_salary) AS Total_salary FROM teacher2 WHERE T_id >
201 GROUP BY T_dept;              -- Total salary ber kortesi SUM hishebe

SELECT T_dept, MIN(T_salary) AS Min_salary FROM teacher2 WHERE T_id > 201
GROUP BY T_dept;             -- prottek dept e min salary koto

SELECT T_dept, MAX(T_salary) AS Max_salary FROM teacher2 WHERE T_id > 201
GROUP BY T_dept;             -- proottek dept e max salary koto

SELECT T_dept, AVG(T_salary) AS AVG_salary FROM teacher2 WHERE T_id > 201
GROUP BY T_dept;             -- prottek dept e avg

SELECT T_dept, MAX(T_salary) AS Max_salary_CSE FROM teacher2 WHERE T_dept
= "CSE" GROUP BY T_dept;          -- cse er dept e max kar




-- ========================================
-- 1. Basic GROUP BY
-- Group rows by department, just return department names (distinct
groups)
SELECT T_dept
FROM teacher2
GROUP BY T_dept;


-- ========================================
-- 2. GROUP BY with COUNT
-- Count how many teachers are in each department
SELECT T_dept, COUNT(*) AS total_teachers
FROM teacher2
GROUP BY T_dept;


-- ========================================
-- 3. GROUP BY with SUM
-- Calculate total salary for each department
```

```
SELECT T_dept, SUM(T_salary) AS total_salary
FROM teacher2
GROUP BY T_dept;


-- ========================================
-- 4. GROUP BY with MIN
-- Find the minimum salary in each department
SELECT T_dept, MIN(T_salary) AS min_salary
FROM teacher2
GROUP BY T_dept;


-- ========================================
-- 5. GROUP BY with MAX
-- Find the maximum salary in each department
SELECT T_dept, MAX(T_salary) AS max_salary
FROM teacher2
GROUP BY T_dept;


-- ========================================
-- 6. GROUP BY with AVG
-- Find the average salary in each department
SELECT T_dept, AVG(T_salary) AS avg_salary
FROM teacher2
GROUP BY T_dept;


-- ========================================
-- 7. GROUP BY with WHERE
-- Only include teachers with T_id > 201 before grouping
SELECT T_dept, COUNT(*) AS total_teachers
FROM teacher2
WHERE T_id > 201
GROUP BY T_dept;


-- ========================================
-- 8. GROUP BY with HAVING
-- Filter groups based on aggregate condition (e.g., departments with
total salary > 2000)
SELECT T_dept, SUM(T_salary) AS total_salary
FROM teacher2
GROUP BY T_dept
HAVING SUM(T_salary) > 2000;


-- ========================================
-- 9. GROUP BY multiple columns
-- Group by department and salary to see how many teachers earn same
salary in a department
SELECT T_dept, T_salary, COUNT(*) AS count_teachers
FROM teacher2
```

```
GROUP BY T_dept, T_salary;


-- ========================================
-- 10. GROUP BY with ORDER BY
-- Group by department and order by total salary descending
SELECT T_dept, SUM(T_salary) AS total_salary
FROM teacher2
GROUP BY T_dept
ORDER BY total_salary DESC;


-- ========================================
-- 11. GROUP BY specific department
-- Find max salary for CSE department only
SELECT T_dept, MAX(T_salary) AS max_salary_CSE
FROM teacher2
WHERE T_dept = "CSE"
GROUP BY T_dept;









-- Select ( ORDER BY Queries )
SELECT * FROM teacher2 WHERE T_dept = "CSE" ORDER BY T_id ASC;
                      -- ascending order e shajaisi order by diye

SELECT * FROM teacher2 WHERE T_dept = "CSE" ORDER BY T_id DESC;


SELECT * FROM teacher2  ORDER BY T_dept DESC;

SELECT * FROM teacher2  ORDER BY T_salary DESC, T_dept ASC;



-- ========================================
-- 1. ORDER BY ASC (Ascending Order)
-- Sort results from smallest to largest (default is ASC)
SELECT *
FROM teacher2
ORDER BY T_salary ASC;  -- Salary from lowest to highest


-- ========================================
-- 2. ORDER BY DESC (Descending Order)
-- Sort results from largest to smallest
SELECT *
FROM teacher2
```

```
ORDER BY T_salary DESC;  -- Salary from highest to lowest


-- ========================================
-- 3. ORDER BY multiple columns
-- First sort by T_dept ascending, then by T_salary descending
SELECT *
FROM teacher2
ORDER BY T_dept ASC, T_salary DESC;


-- ========================================
-- 4. ORDER BY with WHERE condition
-- Sort only CSE department teachers by salary descending
SELECT *
FROM teacher2
WHERE T_dept = "CSE"
ORDER BY T_salary DESC;


-- ========================================
-- 5. ORDER BY with column alias
-- Create a calculated column and sort by it
SELECT T_name, T_salary * 2 AS double_salary
FROM teacher2
ORDER BY double_salary DESC;


-- ========================================
-- 6. ORDER BY using expressions
-- Sort by length of the teacher name
SELECT *, LENGTH(T_name) AS name_length
FROM teacher2
ORDER BY LENGTH(T_name) ASC;  -- shortest name first


-- ========================================
-- 7. ORDER BY with NULLs first/last (MySQL default: NULLS first for ASC)
-- Show teachers in descending order of salary, NULL salaries (if any)
appear last
SELECT *
FROM teacher2
ORDER BY T_salary DESC;


-- ========================================
-- 8. ORDER BY RAND()
-- Random order of rows
SELECT *
FROM teacher2
ORDER BY RAND();


-- ========================================
```

```sql
-- 9. ORDER BY with CASE
-- Custom ordering: prioritize "CSE" first, then others
SELECT *
FROM teacher2
ORDER BY CASE
           WHEN T_dept = 'CSE' THEN 1
           ELSE 2
         END ASC, T_salary DESC;


-- ========================================
-- 10. ORDER BY in subquery
-- Sort inside subquery and pick top 3 highest salary teachers
SELECT *
FROM (SELECT * FROM teacher2 ORDER BY T_salary DESC) AS t
LIMIT 3;






-- Select ( HAVING Query )
SELECT T_dept, AVG(T_salary) AS AVG_salary FROM teacher2  GROUP BY T_dept
HAVING COUNT(*);


 -- Alter table
ALTER TABLE C241135 ADD COLUMN s_cgpa VARCHAR(50);

ALTER TABLE C241135 DROP COLUMN s_cgpa;



-- Update data
UPDATE C241135 SET s_name = 'Abid' WHERE s_id = 105;



-- Delete data
DELETE FROM C241135 WHERE s_id = 105;



-- Truncate table
TRUNCATE TABLE teacher1;




-- Join Table
```

```
SELECT s.student_name, c.course_name FROM students s INNER JOIN courses c
ON s.student_id = c.student_id;

SELECT s.student_name, c.course_name FROM students s LEFT JOIN courses c
ON s.student_id = c.student_id;

SELECT s.student_name, c.course_name FROM students s RIGHT JOIN courses c
ON s.student_id = c.student_id;


SELECT s.student_name, c.course_name
FROM students s
LEFT JOIN courses c ON s.student_id = c.student_id
UNION
SELECT s.student_name, c.course_name
FROM students s
RIGHT JOIN courses c ON s.student_id = c.student_id;


SELECT s.student_name, c.course_name FROM students s CROSS JOIN courses
c;

SELECT s1.student_name AS student1, s2.student_name AS student2 FROM
students s1 INNER JOIN students s2 ON s1.student_id < s2.student_id;

SELECT * FROM students NATURAL JOIN courses;

SELECT * FROM students s INNER JOIN courses c USING(student_id);

SELECT s.student_name FROM students s LEFT JOIN courses c ON s.student_id
= c.student_id WHERE c.course_id IS NULL;

SELECT s.student_name FROM students s WHERE s.student_id IN (SELECT
student_id FROM courses);


-- ========================================
-- 1. INNER JOIN
-- Returns only the rows that match in both tables
-- Students with courses (only those who have courses)

SELECT s.student_name, c.course_name
FROM students s
INNER JOIN courses c
ON s.student_id = c.student_id;


-- ========================================
-- 2. LEFT JOIN (LEFT OUTER JOIN)
-- Returns all rows from the left table; unmatched right table rows will
be NULL
-- All students, even if they don't have courses

SELECT s.student_name, c.course_name
```

```
FROM students s
LEFT JOIN courses c
ON s.student_id = c.student_id;


-- ========================================
-- 3. RIGHT JOIN (RIGHT OUTER JOIN)
-- Returns all rows from the right table; unmatched left table rows will
be NULL
-- All courses, even if no student assigned

SELECT s.student_name, c.course_name
FROM students s
RIGHT JOIN courses c
ON s.student_id = c.student_id;


-- ========================================
-- 4. FULL OUTER JOIN (simulated with UNION in MySQL)
-- Returns all rows from both tables; NULL if no match

SELECT s.student_name, c.course_name
FROM students s
LEFT JOIN courses c
ON s.student_id = c.student_id
UNION
SELECT s.student_name, c.course_name
FROM students s
RIGHT JOIN courses c
ON s.student_id = c.student_id;


-- ========================================
-- 5. CROSS JOIN
-- Returns the cartesian product of both tables (every combination)

SELECT s.student_name, c.course_name
FROM students s
CROSS JOIN courses c;


-- ========================================
-- 6. SELF JOIN
-- Join a table with itself; useful for comparing rows within the same
table
-- Compare students with each other (all pairs)

SELECT s1.student_name AS student1, s2.student_name AS student2
FROM students s1
INNER JOIN students s2
ON s1.student_id < s2.student_id;


-- ========================================
```

```sql
-- 7. NATURAL JOIN
-- Automatically joins on columns with the same name in both tables
SELECT *
FROM students
NATURAL JOIN courses;


-- ========================================
-- 8. JOIN USING
-- Join tables explicitly using a shared column

SELECT *
FROM students s
INNER JOIN courses c
USING(student_id);


-- ========================================
-- 9. ANTI JOIN (Find rows in left table with no match in right table)
-- Students without courses

SELECT s.student_name
FROM students s
LEFT JOIN courses c
ON s.student_id = c.student_id
WHERE c.course_id IS NULL;


-- ========================================
-- 10. SEMI JOIN (Find rows in left table that exist in right table)
-- Students with at least one course
SELECT s.student_name
FROM students s
WHERE s.student_id IN (SELECT student_id FROM courses);




-- Drop table
DROP TABLE c241135;

DROP TABLE teacher1;

DROP TABLE teacher2;

DROP TABLE students;

DROP TABLE courses;
```

```
-- SQL Format
-- SELECT -> FROM -> WHERE -> GROPUP BY -> HAVING -> ORDER BY
-- <> not equal sign
```