Lab 10: Cartesian Product and JOIN Operations

Introduction:

In this lab, we explore the concept of Cartesian Product in database systems. A Cartesian Product combines every row from one table with every row from another table. We also learn how to filter this product to create meaningful relationships between students and their courses based on their academic stream.

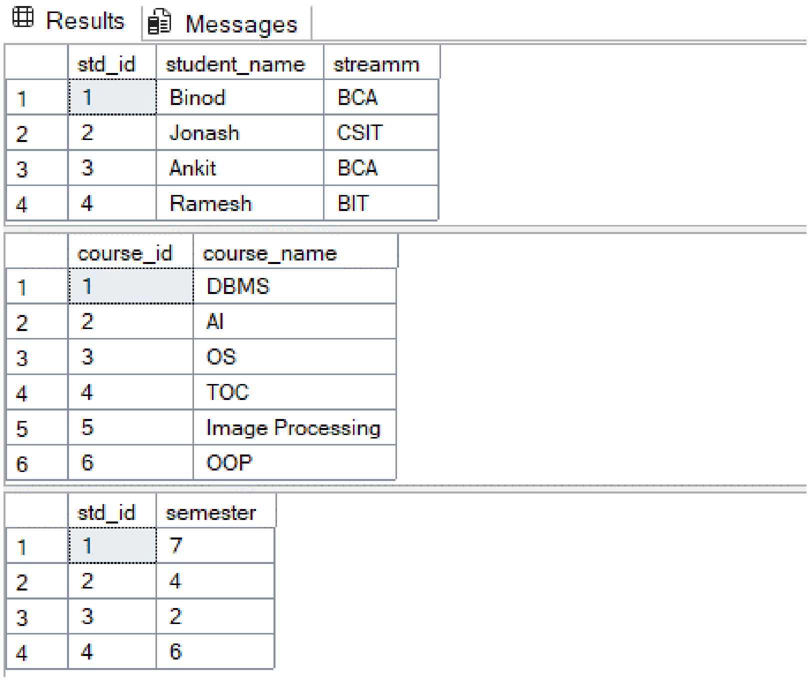
The lab covers:

- **Cartesian Product** – All possible combinations between tables
- **Filtered Cartesian Product** – Using conditions to get relevant data
- **JOIN Operations** – Combining tables efficiently
- **Views** – Creating simplified data representations

Step 1: Creating Tables and Inserting Data

We create four tables to manage student course enrollment and populate them with sample data.

```
-- Create tables
CREATE TABLE Students (
    std_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    streamm VARCHAR(50)
);

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

CREATE TABLE StudentCourses (
    std_id INT,
    course_id INT,
    semester INT,
    PRIMARY KEY (std_id, course_id, semester),
    FOREIGN KEY (std_id) REFERENCES Students(std_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);

CREATE TABLE StudentSemester (
    std_id INT,
    semester INT,
    PRIMARY KEY (std_id, semester),
    FOREIGN KEY (std_id) REFERENCES Students(std_id)
);

-- Insert sample data
INSERT INTO Students (std_id, student_name, streamm)
VALUES
(1, 'Binod', 'BCA'),
(2, 'Jonash', 'CSIT'),
(3, 'Ankit', 'BCA'),
(4, 'Ramesh', 'BIT');

INSERT INTO Courses (course_id, course_name)
VALUES
(1, 'DBMS'),
(2, 'AI'),
(3, 'OS'),
(4, 'TOC'),
(5, 'Image Processing'),
(6, 'OOP');

INSERT INTO StudentSemester (std_id, semester)
VALUES
(1, 7),
(2, 4),
(3, 2),
(4, 6);

-- Verify data
SELECT * FROM Students;
SELECT * FROM Courses;
SELECT * FROM StudentSemester;
```



	std_id	student_name	streamm
1	1	Binod	BCA
2	2	Jonash	CSIT
3	3	Ankit	BCA
4	4	Ramesh	BIT

	course_id	course_name
1	1	DBMS
2	2	AI
3	3	OS
4	4	TOC
5	5	Image Processing
6	6	OOP

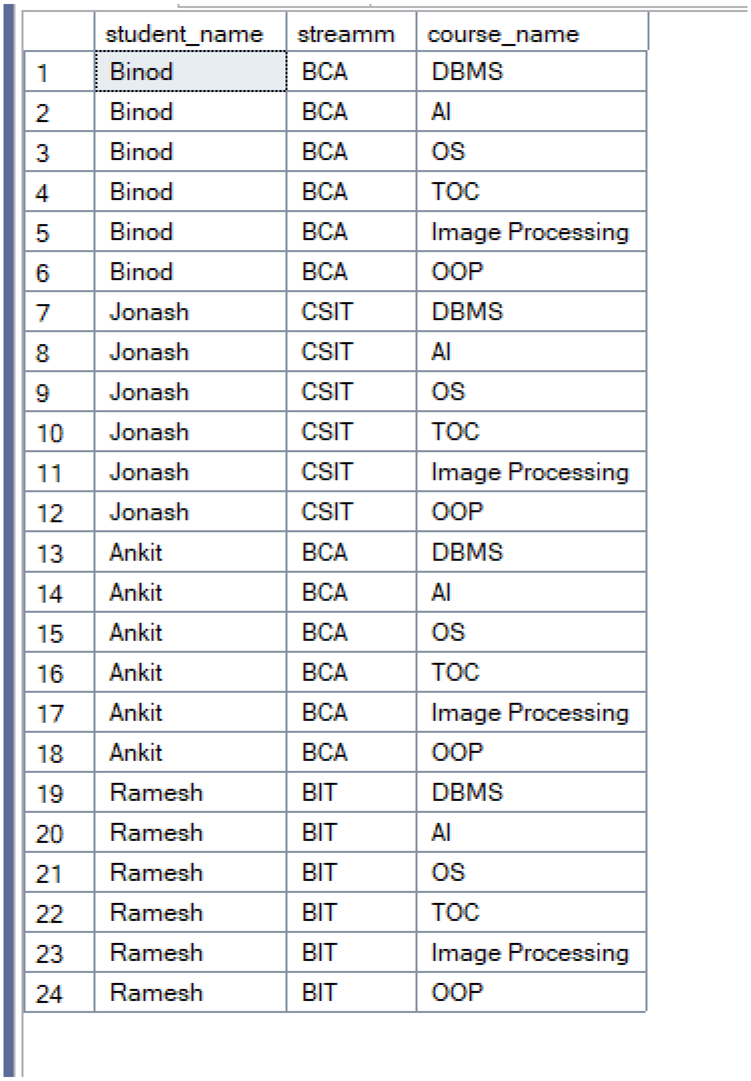
	std_id	semester
1	1	7
2	2	4
3	3	2
4	4	6

Figure 1 The tables are created

Step 2: Understanding Cartesian Product

A Cartesian Product creates all possible combinations between two tables.

```
-- Cartesian Product: All possible student-course combinations
SELECT
    s.student_name,
    s.streamm,
    c.course_name
FROM Students s, Courses c

ORDER BY s.std_id, c.course_id;
```

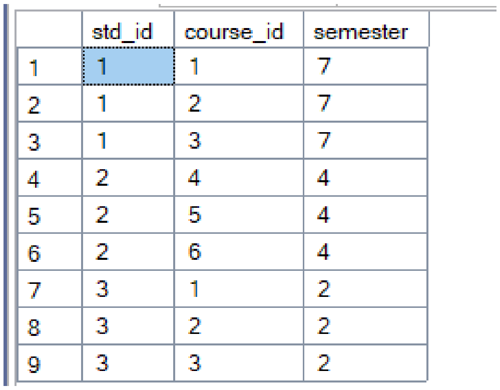


	student_name	streamm	course_name
1	Binod	BCA	DBMS
2	Binod	BCA	AI
3	Binod	BCA	OS
4	Binod	BCA	TOC
5	Binod	BCA	Image Processing
6	Binod	BCA	OOP
7	Jonash	CSIT	DBMS
8	Jonash	CSIT	AI
9	Jonash	CSIT	OS
10	Jonash	CSIT	TOC
11	Jonash	CSIT	Image Processing
12	Jonash	CSIT	OOP
13	Ankit	BCA	DBMS
14	Ankit	BCA	AI
15	Ankit	BCA	OS
16	Ankit	BCA	TOC
17	Ankit	BCA	Image Processing
18	Ankit	BCA	OOP
19	Ramesh	BIT	DBMS
20	Ramesh	BIT	AI
21	Ramesh	BIT	OS
22	Ramesh	BIT	TOC
23	Ramesh	BIT	Image Processing
24	Ramesh	BIT	OOP

Figure 2 The Cartesian Product produces 24 rows

Step 3: Creating Filtered Course Enrollment

```
-- Insert filtered enrollments based on stream
INSERT INTO StudentCourses (std_id, course_id, semester)
SELECT s.std_id, c.course_id, ss.semester

FROM Students s
JOIN StudentSemester ss ON s.std_id = ss.std_id
JOIN Courses c ON (
    (s.streamm = 'CSIT' AND c.course_name IN ('OOP', 'Image Processing', 'TOC')) OR
    (s.streamm = 'BCA' AND c.course_name IN ('OS', 'DBMS', 'AI'))
);
-- Verify enrollments
SELECT * FROM StudentCourses ORDER BY std_id, course_id;
```



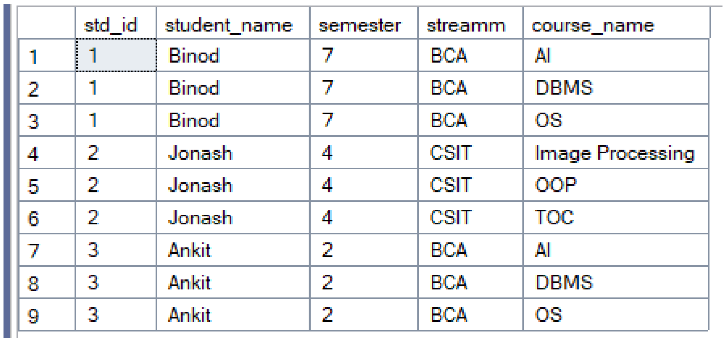
	std_id	course_id	semester
1	1	1	7
2	1	2	7
3	1	3	7
4	2	4	4
5	2	5	4
6	2	6	4
7	3	1	2
8	3	2	2
9	3	3	2

Figure 3 : The filtered result shown

Step 4: Creating a View for Easy Access

We create a view that combines all information for easy querying.

```
-- Create view
CREATE VIEW StudentCourseDetails AS
SELECT
    s.std_id,
    s.student_name,
    sc.semester,
    s.streamm,
    c.course_name
FROM
    Students s
JOIN
    StudentCourses sc ON s.std_id = sc.std_id
JOIN
    Courses c ON sc.course_id = c.course_id;
GO

-- Query the view
SELECT * FROM StudentCourseDetails ORDER BY std_id, course_name;
```



	std_id	student_name	semester	streamm	course_name
1	1	Binod	7	BCA	AI
2	1	Binod	7	BCA	DBMS
3	1	Binod	7	BCA	OS
4	2	Jonash	4	CSIT	Image Processing
5	2	Jonash	4	CSIT	OOP
6	2	Jonash	4	CSIT	TOC
7	3	Ankit	2	BCA	AI
8	3	Ankit	2	BCA	DBMS
9	3	Ankit	2	BCA	OS

Conclusion:

This lab demonstrated how Cartesian Product works and how to filter it for practical use. We learned that:

- Cartesian Product generates all possible combinations (24 rows from 4 students × 6 courses)
- Filtering reduces results to meaningful data (12 valid enrollments)
- Views simplify data access by combining multiple tables
- Business rules ensure data quality (students only get courses from their stream)

Understanding Cartesian Products helps us write better queries and understand how database joins work efficiently.