Assignment 3:

```
-- Create Students table
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(100),
    student_major VARCHAR(100)
);
-- Create Courses table
CREATE TABLE Courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100),
    course_description VARCHAR(255)
);

-- Create Enrollments table
CREATE TABLE Enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);

-- Insert data into Students table
```

```
INSERT INTO Students (student_id, student_name, student_major) VALUES
(1, 'Alice', 'Computer Science'),
(2, 'Bob', 'Biology'),
(3, 'Charlie', 'History'),
(4, 'Diana', 'Mathematics');
```

-- Insert data into Courses table

```
INSERT INTO Courses (course_id, course_name, course_description) VALUES
(101, 'Introduction to CS', 'Basics of Computer Science'),
(102, 'Biology Basics', 'Fundamentals of Biology'),
(103, 'World History', 'Historical events and cultures'),
(104, 'Calculus I', 'Introduction to Calculus'),
(105, 'Data Structures', 'Advanced topics in CS');
```

-- Insert data into Enrollments table

```
INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date) VALUES
(1, 1, 101, '2023-01-15'),
(2, 2, 102, '2023-01-20'),
(3, 3, 103, '2023-02-01'),
(4, 1, 105, '2023-02-05'),
(5, 4, 104, '2023-02-10'),
(6, 2, 101, '2023-02-12'),
(7, 3, 105, '2023-02-15'),
(8, 4, 101, '2023-02-20'),
(9, 1, 104, '2023-03-01'),
(10, 2, 104, '2023-03-05');
```

1. Inner Join:

<u>Question:</u> Retrieve the list of students and their enrolled courses.

<u>Query</u>

```
SELECT student_name, course_name
        FROM Students s
JOIN Enrollments e
        ON s.student_id = e.student_id
JOIN Courses c
        ON c.course_id = e.course_id;
```

<u>Output</u>

Data Output Messages Notifications

	student_name character varying (100)	course_name character varying (100)
1	Alice	Introduction to CS
2	Bob	Biology Basics
3	Charlie	World History
4	Alice	Data Structures
5	Diana	Calculus I
6	Bob	Introduction to CS
7	Charlie	Data Structures
8	Diana	Introduction to CS
9	Alice	Calculus I
10	Bob	Calculus I

2. Left Join:

<u>Question:</u> List all students and their enrolled courses, including those who haven't enrolled in any course.

<u>Query</u>

```
SELECT student_name, course_name
        FROM Students s
LEFT JOIN Enrollments e
        ON s.student_id = e.student_id
LEFT JOIN Courses c
        ON c.course_id = e.course_id;
```

<u>Output</u>

Data Output Messages Notifications

	student_name character varying (100)	course_name character varying (100)
1	Alice	Introduction to CS
2	Bob	Biology Basics
3	Charlie	World History
4	Alice	Data Structures
5	Diana	Calculus I
6	Bob	Introduction to CS
7	Charlie	Data Structures
8	Diana	Introduction to CS
9	Alice	Calculus I
10	Bob	Calculus I
11	Ganesh	[null]

3. Right Join:

<u>Question:</u> Display all courses and the students enrolled in each course, including courses with no enrolled students.

Query

```
SELECT student_name, course_name
        FROM Students s
RIGHT JOIN Enrollments e
        ON s.student_id = e.student_id
RIGHT JOIN Courses c
        ON c.course_id = e.course_id;
```

Output

Data Output Messages Notifications

	student_name character varying (100)	course_name character varying (100)
1	Alice	Introduction to CS
2	Bob	Biology Basics
3	Charlie	World History
4	Alice	Data Structures
5	Diana	Calculus I
6	Bob	Introduction to CS
7	Charlie	Data Structures
8	Diana	Introduction to CS
9	Alice	Calculus I
10	Bob	Calculus I
11	[null]	Machine Learning

4. Self Join:

Question: Find pairs of students who are enrolled in at least one common course.

Query

```
SELECT DISTINCT
   e1.student_id AS student1_id,
```

```
    s1.student_name AS student1_name,
    e2.student_id AS student2_id,
    s2.student_name AS student2_name,
    e1.course_id
FROM
    Enrollments e1
    JOIN Enrollments e2 ON e1.course_id = e2.course_id AND e1.student_id < e2.student_id
    JOIN Students s1 ON e1.student_id = s1.student_id
    JOIN Students s2 ON e2.student_id = s2.student_id;
```

Output

Data Output Messages Notifications

	student1_id integer	student1_name character varying (100)	student2_id integer	student2_name character varying (100)	course_id integer
1	1	Alice	3	Charlie	105
2	1	Alice	2	Bob	104
3	1	Alice	4	Diana	101
4	2	Bob	4	Diana	104
5	1	Alice	4	Diana	104
6	2	Bob	4	Diana	101
7	1	Alice	2	Bob	101

5. Complex Join:

Question: Retrieve students who are enrolled in 'Introduction to CS' but not in 'Data Structures'.

Query

```
SELECT s.student_id, student_name, course_name
        FROM Students s
JOIN Enrollments e
```

```
        ON s.student_id = e.student_id
JOIN Courses c
        ON c.course_id = e.course_id
WHERE c.course_name = 'Introduction to CS' AND s.student_id NOT IN (
        SELECT en1.student_id
        FROM Enrollments en1
        JOIN Courses c2
        ON en1.course_id = c2.course_id
        WHERE c2.course_name = 'Data Structures'
);
```

Output

Data Output Messages Notifications

	student_id integer	student_name character varying (100)	course_name character varying (100)
1	2	Bob	Introduction to CS
2	4	Diana	Introduction to CS

Windows function:

1. Using ROW_NUMBER():

Question: List all students along with a row number based on their enrollment date in ascending order.

Query

```
SELECT s.student_id, student_name, enrollment_date,
ROW_NUMBER() OVER(
```

```
        ORDER BY e.enrollment_date
)
FROM Students s
JOIN Enrollments e
        ON s.student_id = e.student_id;
```

Output

Data Output Messages Notifications

	student_id integer	student_name character varying (100)	enrollment_date date	row_number bigint
1	1	Alice	2023-01-15	1
2	2	Bob	2023-01-20	2
3	3	Charlie	2023-02-01	3
4	1	Alice	2023-02-05	4
5	4	Diana	2023-02-10	5
6	2	Bob	2023-02-12	6
7	3	Charlie	2023-02-15	7
8	4	Diana	2023-02-20	8
9	1	Alice	2023-03-01	9
10	2	Bob	2023-03-05	10

2. Using RANK():

Question: Rank students based on the number of courses they are enrolled in, handling ties by assigning the same rank.

Query

```
SELECT e.student_id, COUNT(e.student_id) AS Num_of_courses,
RANK() OVER (
        ORDER BY COUNT(e.student_id) DESC
```

```
) AS Student_Rank
FROM Students s
JOIN Enrollments e
        ON s.student_id = e.student_id
        GROUP BY e.student_id;
```

Output

Data Output Messages Notifications

	student_id integer	num_of_courses bigint	student_rank bigint
1	2	3	1
2	1	3	1
3	3	2	3
4	4	2	3

3. Using DENSE_RANK():

Question: Determine the dense rank of courses based on their enrollment count across all students?

Query

```
SELECT e.course_id, COUNT(e.student_id) AS Num_of_students,
DENSE_RANK() OVER (
        ORDER BY COUNT(e.course_id) DESC
) AS Course_rank
FROM Students s
JOIN Enrollments e
        ON s.student_id = e.student_id
        GROUP BY e.course_id;
```

Output

Data Output Messages Notifications

	course_id integer	num_of_students bigint	course_rank bigint
1	101	3	1
2	104	3	1
3	105	2	2
4	103	1	3
5	102	1	3