Department of Artificial Intelligence & Multimedia Gamming

CSC-207: Database Systems

Lab # 06: To Work with SQL Joins

Objectives

Introduction of JOIN

1) INNER JON
2) LEFT JOIN
3) LEFT EXCLUDING JOIN
4) RIGHT JOIN
5) RIGHR EXCLUDING JOIN
6) OUTER JOIN
7) OUTER EXCLUDING JOIN
8) CROSS JOIN
9) STRAIGHT JOIN
10) NATURAL JOIN
11) EQUI JOIN
12) NON EQUI JOIN
13) SELF JOIN

Introduction of JOIN

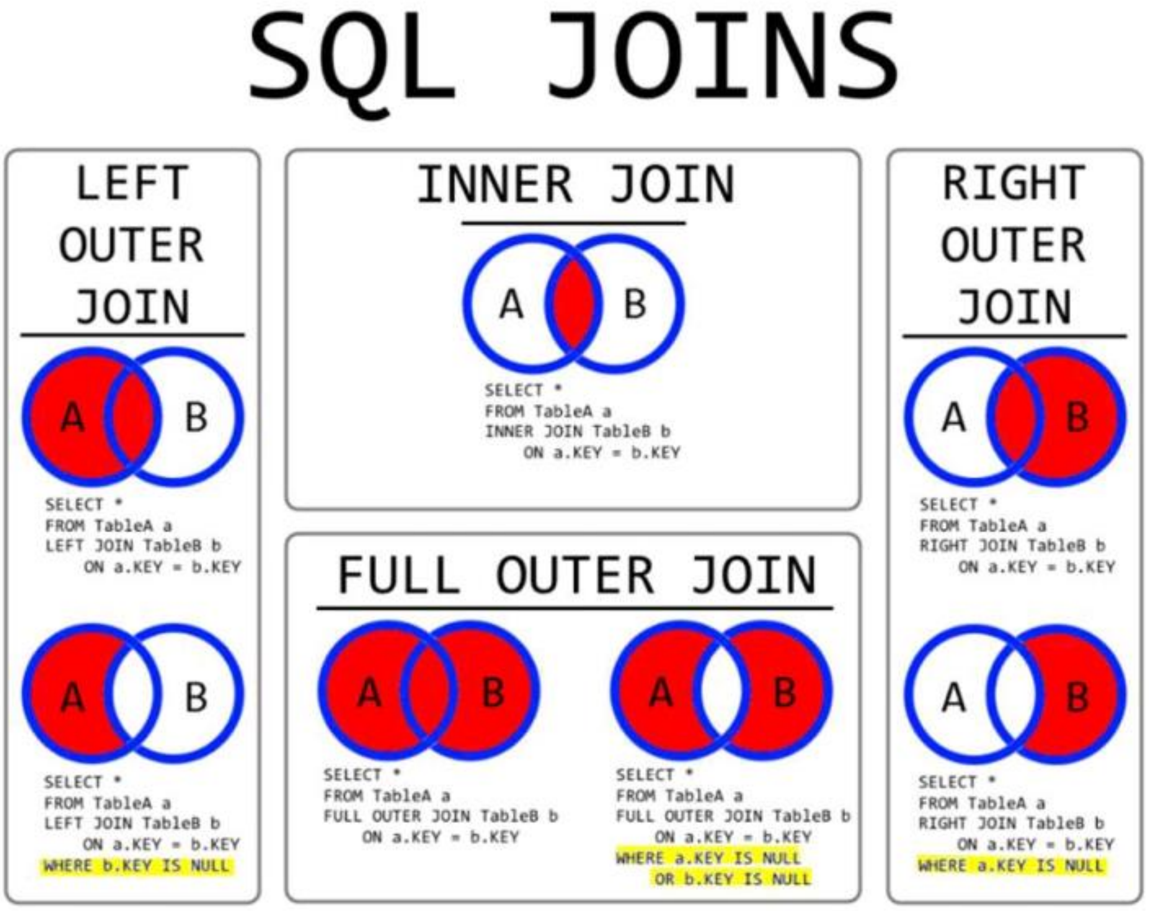
A relational database consists of multiple related tables linking together using common columns which are known as foreign key columns. Because of this, data in each table is incomplete from the business perspective.

By using joins, you can retrieve data from two or more tables based on logical relationships between the tables. Joins indicate how SQL should use data from one table to select the rows in another table.

A join condition defines the way two tables are related in a query by:

- Specifying the column from each table to be used for the join. A typical join condition specifies a foreign key from one table and its associated key in the other table.
- Specifying a logical operator (for example, = or <>,) to be used in comparing values from the columns.
- Here is visualization for basic joins.

  [Visual JOIN (spathon.com)]



```
SELECT *
FROM TableA a
LEFT JOIN TableB b
  ON a.KEY = b.KEY
```

```
SELECT *
FROM TableA a
LEFT JOIN TableB b
  ON a.KEY = b.KEY
WHERE b.KEY IS NULL
```

```
SELECT *
FROM TableA a
INNER JOIN TableB b
  ON a.KEY = b.KEY
```

```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
  ON a.KEY = b.KEY
```

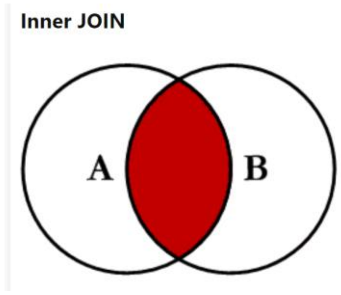
```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
  ON a.KEY = b.KEY
WHERE a.KEY IS NULL
  OR b.KEY IS NULL
```

```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
  ON a.KEY = b.KEY
```

```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
  ON a.KEY = b.KEY
WHERE a.KEY IS NULL
```

1. INNER JOIN

The INNER JOIN keyword selects records that have matching values in both tables.



Syntax

SELECT column_name(s)

FROM table1

INNER JOIN table2

ON table1.column_name = table2.column_name;

If the join condition uses the equal operator (=) and the column names in both tables used for matching are the same, you can use the USING clause instead:

Syntax

SELECT column_name(s)

FROM table1

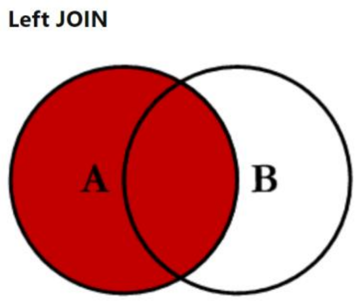
INNER JOIN table2

USING(column_name);

select * from customers inner join orders using(customer_id);

2. LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.



Syntax

SELECT column_name(s)

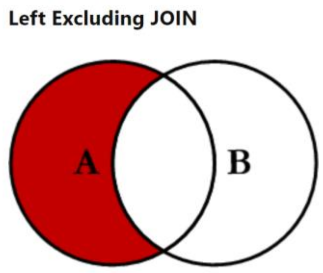
FROM table1

LEFT JOIN table2

ON table1.column_name = table2.column_name;

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.

3. LEFT EXCLUDING JOIN



This query will return all of the records in the left table (table A) that do not match any records in the right table (table B).

Syntax

SELECT <select_list>

FROM Table_A A

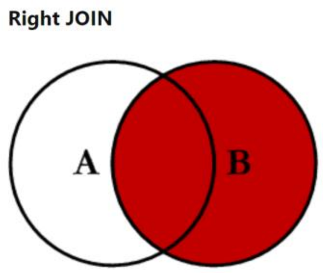
LEFT JOIN Table_B B

ON A.Key = B.Key

WHERE B.Key IS NULL

4. RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.



Syntax

SELECT column_name(s)

FROM table1

RIGHT JOIN table2

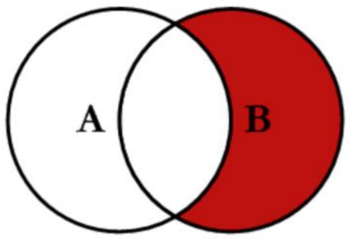
ON table1.column_name = table2.column_name;

Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

5. RIGHT EXCLUDING JOIN

This query will return all of the records in the right table (table B) that do not match any records in the left table (table A).

Right Excluding JOIN



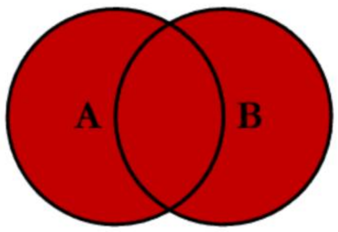
Syntax:

```
SELECT <select_list>
FROM Table_A A
RIGHT JOIN Table_B B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

6. Outer Join

The FULL OUTER JOIN or OUTER JOIN/ FULL JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Outer JOIN

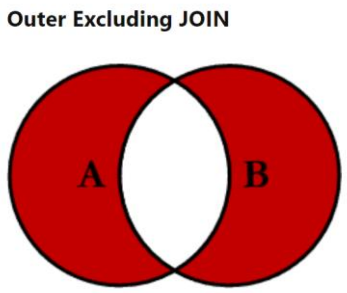


Syntax:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

7. OUTER EXCLUDING JOIN

This query will return all of the records in the left table (table A) and all of the records in the right table (table B) that do not match.



Syntax:

SELECT <select_list>

FROM Table_A A

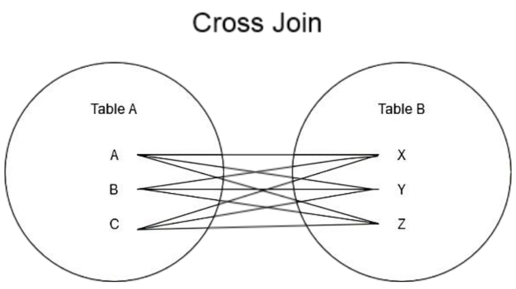
FULL OUTER JOIN Table_B B

ON A.Key = B.Key

WHERE A.Key IS NULL OR B.Key IS NULL

8. CROSS JOIN

A cross join, also known as a Cartesian join, is a type of join operation in SQL that returns the Cartesian product of two tables. In other words, it generates a result set in which each row from the first table is combined with every row from the second table, resulting in a combination of all possible pairs of rows.

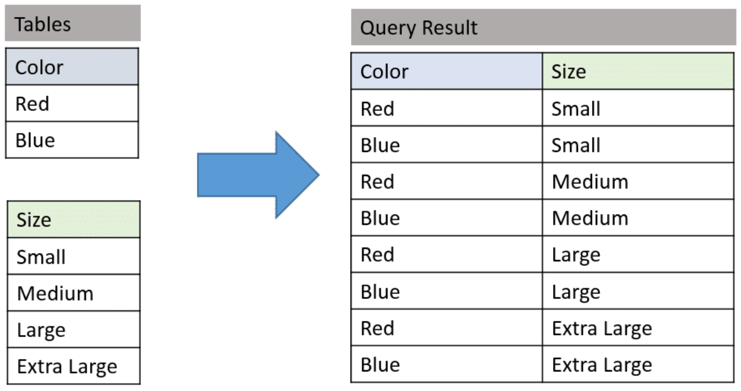


Syntax:

SELECT *

FROM table1

CROSS JOIN table2;



Tables

Color
Red
Blue

Size
Small
Medium
Large
Extra Large

Query Result	
Color	Size
Red	Small
Blue	Small
Red	Medium
Blue	Medium
Red	Large
Blue	Large
Red	Extra Large
Blue	Extra Large

9. STRAIGHT JOIN

In SQL, a "straight join" is not a standard join type like INNER JOIN, LEFT JOIN, or CROSS JOIN. Instead, it's a MySQL-specific keyword that instructs the MySQL query optimizer to process tables in the order specified in the query.

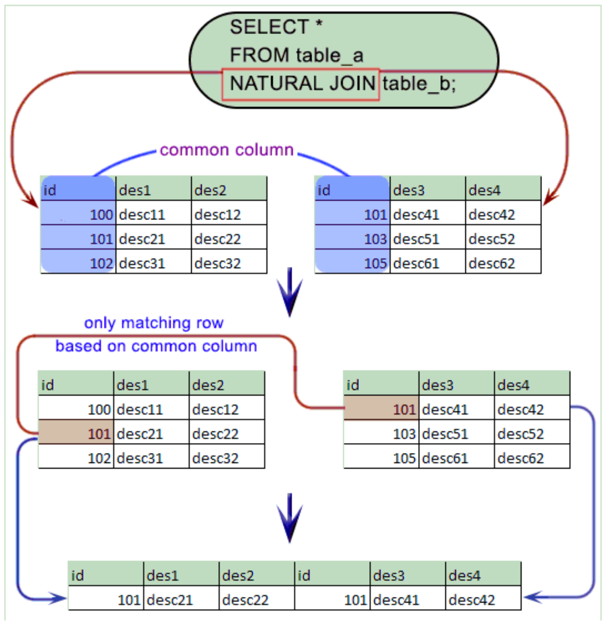
SELECT *

FROM table1

STRAIGHT_JOIN table2 ON conditions;

10. NATURAL JOIN

A natural join is a type of join operation in SQL that automatically joins tables based on columns with the same name and data type. In other words, it performs a join by implicitly matching columns with identical names in the two tables being joined.



```
SELECT *
FROM table_a
NATURAL JOIN table_b;
```

id	des1	des2
100	desc11	desc12
101	desc21	desc22
102	desc31	desc32

id	des3	des4
101	desc41	desc42
103	desc51	desc52
105	desc61	desc62

id	des1	des2
100	desc11	desc12
101	desc21	desc22
102	desc31	desc32

id	des3	des4
101	desc41	desc42
103	desc51	desc52
105	desc61	desc62

id	des1	des2	id	des3	des4
101	desc21	desc22	101	desc41	desc42

11. EQUI JOIN

An equi join is a type of join operation in SQL that matches rows between two tables based on a specified equality condition involving columns with the same name or related values. In other words, it joins tables by comparing values in corresponding columns and selecting rows where the values are equal.

The term "equi" comes from "equality," emphasizing that the join condition involves equality between values in the columns being compared.

Syntax:

```
SELECT column_list
FROM table1
JOIN table2 ON join_conditions;
OR
SELECT column_list
FROM table1, table2
WHERE table1.col_name = table2.col_name;
```

12. NON EQUI JOIN

A non-equijoin, or non-equi join, is a type of join operation in SQL where the join condition is not based on equality between columns in the joined tables. Instead, it uses comparison operators other than equality, such as <, >, <=, >=, or <>, to compare values between columns.

The non-equijoin is less common than the equijoin but can be useful in certain scenarios where you need to join tables based on relationships that are not strictly equal.

13. SELF JOIN

A self join is a type of join operation in SQL where a table is joined with itself. In other words, you use the same table twice in the join operation, with each instance of the table being given a different alias to distinguish between them.

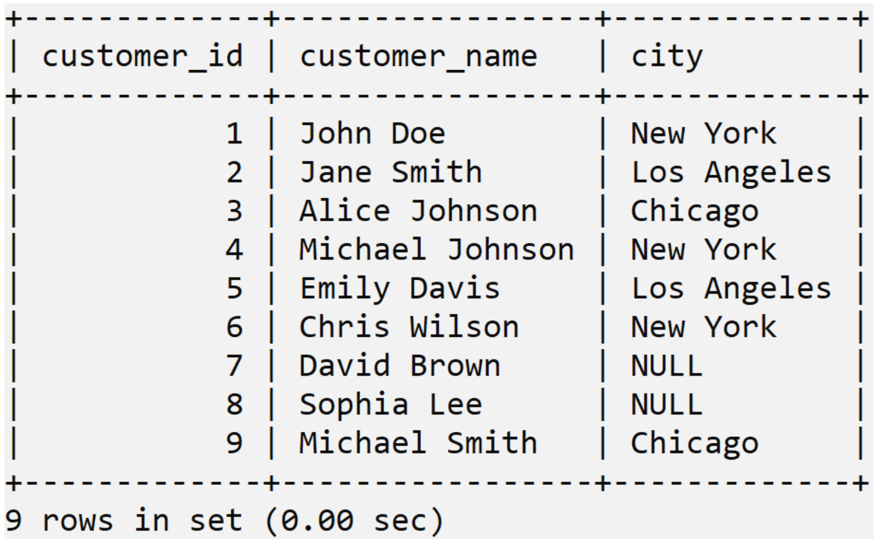
Self joins are used when you need to compare rows within the same table or retrieve hierarchical data from a table. They are commonly used in scenarios where you have a hierarchical relationship within a table, such as with parent-child relationships.

Syntax:

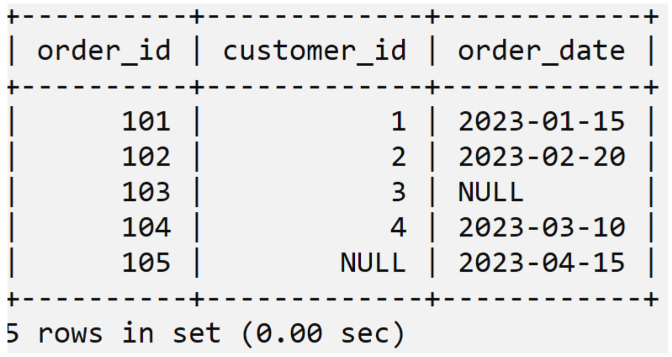
```
SELECT a.col_name, b.col_name, …
FROM table1 a, table2 b
WHERE a.common_field = b.common_field;
```

Exercises (Class)

```
CREATE TABLE customers (
    customer_id INT,
    customer_name VARCHAR(50),
    city VARCHAR(50)
);

CREATE TABLE orders (
    order_id INT,
    customer_id INT,
    order_date DATE
);

INSERT INTO customers (customer_id, customer_name, city) VALUES
(1, 'John Doe', 'New York'),
(2, 'Jane Smith', 'Los Angeles'),
(3, 'Alice Johnson', 'Chicago');

INSERT INTO orders (order_id, customer_id, order_date) VALUES
(101, 1, '2023-01-15'),
(102, 2, '2023-02-20'),
(103, 3, NULL),
(104, 4, '2023-03-10'),
(105, NULL, '2023-04-15');

INSERT INTO customers (customer_id, customer_name, city) VALUES
(4, 'Michael Johnson', 'New York'),
(5, 'Emily Davis', 'Los Angeles'),
(6, 'Chris Wilson', 'New York'),
(7, 'David Brown', NULL),
(8, 'Sophia Lee', NULL),
(9, 'Michael Smith', 'Chicago');
```



```
+-------------+-----------------+-------------+
| customer_id | customer_name   | city        |
+-------------+-----------------+-------------+
|           1 | John Doe        | New York    |
|           2 | Jane Smith      | Los Angeles |
|           3 | Alice Johnson   | Chicago     |
|           4 | Michael Johnson | New York    |
|           5 | Emily Davis     | Los Angeles |
|           6 | Chris Wilson    | New York    |
|           7 | David Brown     | NULL        |
|           8 | Sophia Lee      | NULL        |
|           9 | Michael Smith   | Chicago     |
+-------------+-----------------+-------------+
9 rows in set (0.00 sec)
```



```
+----------+-------------+------------+
| order_id | customer_id | order_date |
+----------+-------------+------------+
|      101 |           1 | 2023-01-15 |
|      102 |           2 | 2023-02-20 |
|      103 |           3 | NULL       |
|      104 |           4 | 2023-03-10 |
|      105 |        NULL | 2023-04-15 |
+----------+-------------+------------+
5 rows in set (0.00 sec)
```

INNER JOIN: Inner join returns rows when there is a match in both tables.

SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date

FROM customers

INNER JOIN orders ON customers.customer_id = orders.customer_id;

Or

SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date

FROM customers

JOIN orders ON customers.customer_id = orders.customer_id

WHERE customers.city = 'New York';

Or

SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date

FROM customers

JOIN orders ON customers.customer_id = orders.customer_id;

LEFT JOIN:

Left join returns all rows from the left table and the matched rows from the right table.

SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date

FROM customers

LEFT JOIN orders ON customers.customer_id = orders.customer_id;

LEFT EXCLUDING JOIN:

This is not a standard SQL join type, but let's assume it returns all rows from the left table that do not have a match in the right table.

SELECT customers.customer_id, customers.customer_name

FROM customers

LEFT JOIN orders ON customers.customer_id = orders.customer_id

WHERE orders.order_id IS NULL;

RIGHT JOIN:

Right join returns all rows from the right table and the matched rows from the left table.

SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date

FROM customers

RIGHT JOIN orders ON customers.customer_id = orders.customer_id;

RIGHT EXCLUDING JOIN:

This is not a standard SQL join type, but let's assume it returns all rows from the right table that do not have a match in the left table.

SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date

FROM customers

RIGHT JOIN orders ON customers.customer_id = orders.customer_id

```
WHERE customers.customer_id IS NULL;
```

OUTER JOIN:

Outer join returns all rows when there is a match in one of the tables.

```
SELECT c.customer_id, c.customer_name, o.order_id, o.order_date
FROM customers c
RIGHT JOIN orders o ON c.customer_id = o.customer_id
UNION
SELECT c.customer_id, c.customer_name, o.order_id, o.order_date
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id;
```

Or

```
SELECT c.customer_id, c.customer_name, o.order_id, o.order_date
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
UNION
SELECT c.customer_id, c.customer_name, o.order_id, o.order_date
FROM customers c
RIGHT JOIN orders o ON c.customer_id = o.customer_id;
```

OUTER EXCLUDING JOIN:

This is not a standard SQL join type, but let's assume it returns all rows from both tables that do not have a match in the other table.

```
SELECT c.customer_id, c.customer_name, o.order_id, o.order_date
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
WHERE o.customer_id IS NULL
UNION
SELECT customers.customer_id, customers.customer_name, orders.order_id,
orders.order_date
FROM customers
RIGHT JOIN orders ON customers.customer_id = orders.customer_id
WHERE customers.customer_id IS NULL;
```

CROSS JOIN:

Cross join returns the Cartesian product of the two tables.

```
SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date
FROM customers
CROSS JOIN orders;
```

STRAIGHT JOIN:

Straight join specifies the order in which tables are joined.

```
SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date
FROM customers
STRAIGHT_JOIN orders ON customers.customer_id = orders.customer_id;
```

NATURAL JOIN:

Natural join matches columns with the same name in both tables.

```
SELECT *
FROM customers
NATURAL JOIN orders;
```

EQUI JOIN:

Equi join matches rows based on equality between columns in the joined tables.

```
SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date
FROM customers
JOIN orders ON customers.customer_id = orders.customer_id;
```

NON EQUI JOIN:

Non-equi join matches rows based on comparison operators other than equality.

```
SELECT *
FROM customers c
INNER JOIN orders o
ON c.customer_id = o.customer_id AND o.order_date > '2022-01-01';
```

SELF JOIN:

Self join joins a table with itself.

```
SELECT DISTINCT c1.customer_name, c1.city
FROM customers c1
JOIN customers c2 ON c1.city = c2.city;
```

OR

```
SELECT c1.customer_name AS customer1_name, c2.customer_name AS customer2_name, c1.city
FROM customers c1
JOIN customers c2 ON c1.city = c2.city
          AND c1.customer_id < c2.customer_id;
```

Add here all the tasks performed in lab.

```
SELECT e.first_name, e.last_name, e.department_id, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id;


SELECT e.first_name, e.last_name, e.job_id, e.department_id, d.department_name, l.city
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
WHERE l.city = 'London';
```

Exercises (Weekly)

1. Write a query in SQL to display the first name, last name, department number, and department name for each employee. **(Sample tables: employees & departments)**
2. Write a query to find the name (first_name, last_name), job, department ID and name of the department who works in London**. (Sample tables: employees , locations & departments)**
3. Write a query in SQL to display the first and last name, department, city, and state province for each employee. **(Sample tables: employees , locations & departments)**
4. Write a query to find the employee id, name (last_name) along with their manager_id and name (last_name). **(Sample tables: employees)**
5. Write a query to find the name (first_name, last_name) and hire date of the employees who was hired after 'Jones'**. (Sample tables: employees)**
6. Write a query to get the department name and number of employees in the department.

(Sample tables: employees & departments)

7. Write a query to display the department ID and name and first name of manager. **(Sample tables: employees & departments)**

8. Write a query to display the department name, manager name, and city. **(Sample tables: employees , locations & departments)**

9. Write a query to display the job history that were done by any employee who is currently drawing more than 10000 of salary. **(Sample tables: employees & job_history)**

10. Write a query to display the first name, last name, hire date, salary of the manager for all managers whose experience is more than 15 years. **(Sample tables: employees & departments)**

11. Write a query in SQL to display the name of the department, average salary and number of employees working in that department who got commission. **(Sample tables: employees & departments)**

12. Write a query in SQL to display the name of the country, city, and the departments which are running there. **(Sample tables: countries , locations & departments)**

13. Write a query in SQL to display department name and the full name (first and last name) of the manager. **(Sample tables: employees & departments)**

14. Write a query in SQL to display the details of jobs which was done by any of the employees who is presently earning a salary on and above 12000. **(Sample tables: employees & job_history)**

15. Write a query in SQL to display the full name (first and last name), and salary of those employees who working in any department located in London. **(Sample tables: employees , locations & departments)**

16. Write a query to display job title, employee name, and the difference between salary of the employee and minimum salary for the job. **(Sample tables: employees & jobs)**

17. Write a query to display the job title and average salary of employees. **(Sample tables: employees & jobs)**

18. Write a query to find the employee ID, job title, number of days between ending date and starting date for all jobs in department 90 from job history. **(Sample tables: jobs & job_history)**