

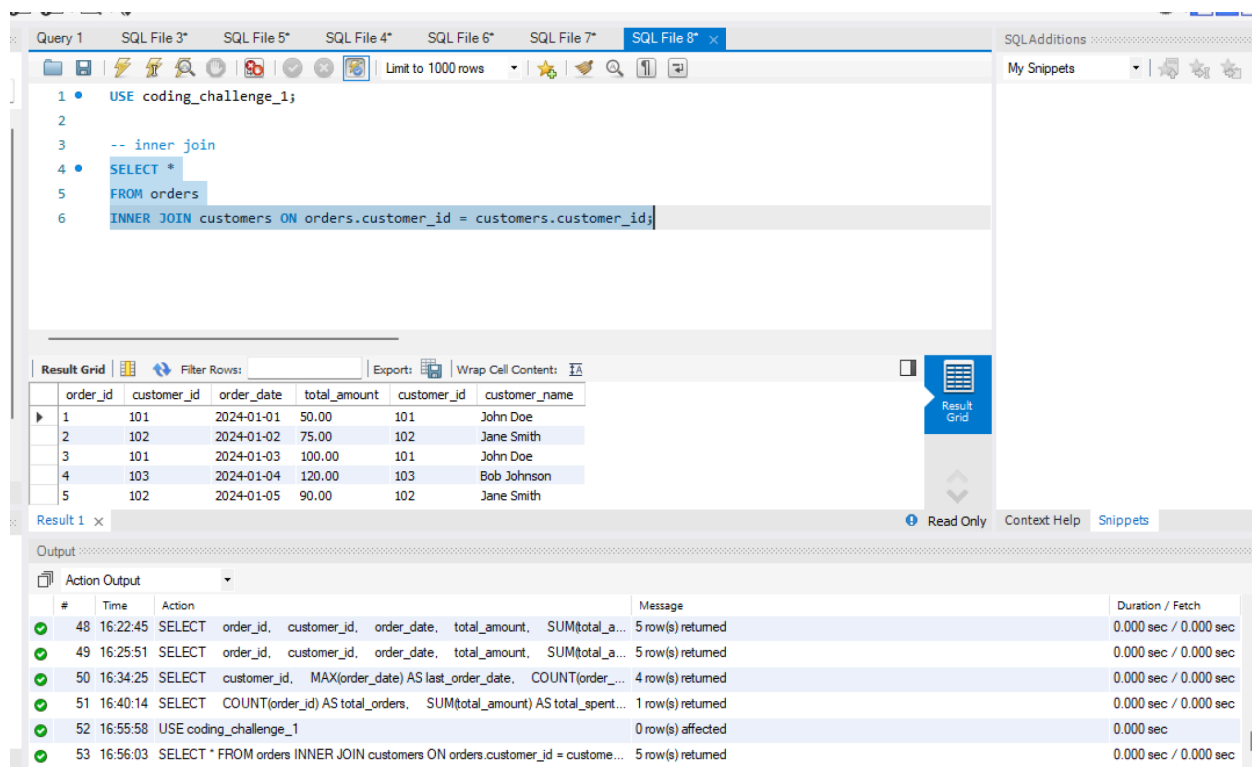
Asmita Porwal  
Data Engineering  
Batch-1  
25/1/24

## Coding Challenge -1 Question-2

**Execute all the joins with examples.**

### 1.Inner Join

An inner join returns only the rows where there is a match in both tables.



The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
1 • USE coding_challenge_1;
2
3 -- inner join
4 • SELECT *
5 FROM orders
6 INNER JOIN customers ON orders.customer_id = customers.customer_id;
```

The results pane displays a table with 6 columns: order\_id, customer\_id, order\_date, total\_amount, customer\_id, and customer\_name. The table contains 5 rows of data:

	order_id	customer_id	order_date	total_amount	customer_id	customer_name
1	101	101	2024-01-01	50.00	101	John Doe
2	102	102	2024-01-02	75.00	102	Jane Smith
3	101	101	2024-01-03	100.00	101	John Doe
4	103	103	2024-01-04	120.00	103	Bob Johnson
5	102	102	2024-01-05	90.00	102	Jane Smith

The bottom pane shows the output of the query, including the time taken and the number of rows returned for each step.

#	Time	Action	Message	Duration / Fetch
48	16:22:45	SELECT order_id, customer_id, order_date, total_amount, SUM(total_a...	5 row(s) returned	0.000 sec / 0.000 sec
49	16:25:51	SELECT order_id, customer_id, order_date, total_amount, SUM(total_a...	5 row(s) returned	0.000 sec / 0.000 sec
50	16:34:25	SELECT customer_id, MAX(order_date) AS last_order_date, COUNT(order_...	4 row(s) returned	0.000 sec / 0.000 sec
51	16:40:14	SELECT COUNT(order_id) AS total_orders, SUM(total_amount) AS total_spent...	1 row(s) returned	0.000 sec / 0.000 sec
52	16:55:58	USE coding_challenge_1	0 row(s) affected	0.000 sec
53	16:56:03	SELECT * FROM orders INNER JOIN customers ON orders.customer_id = custome...	5 row(s) returned	0.000 sec / 0.000 sec

### 2.Left Join (or Left Outer Join):

A left join returns all rows from the left table and the matched rows from the right table. If there is no match, NULL values are returned for columns from the right table.

The screenshot displays a SQL IDE interface. The top pane shows a query window with the following SQL code:

```

2
3 -- inner join
4 • SELECT *
5 FROM orders
6 INNER JOIN customers ON orders.customer_id = customers.customer_id;
7
8 -- left outer join
9 • SELECT *
10 FROM customers
11 LEFT JOIN orders ON customers.customer_id = orders.customer_id;

```

The middle pane shows a 'Result Grid' with the following data:

customer_id	customer_name	order_id	customer_id	order_date	total_amount
101	John Doe	3	101	2024-01-03	100.00
101	John Doe	1	101	2024-01-01	50.00
102	Jane Smith	5	102	2024-01-05	90.00
102	Jane Smith	2	102	2024-01-02	75.00
103	Bob Johnson	4	103	2024-01-04	120.00

The bottom pane shows the 'Output' window with the following execution logs:

#	Time	Action	Message	Duration / Fetch
49	16:25:51	SELECT	order_id, customer_id, order_date, total_amount, SUM(total_a...	5 row(s) returned 0.000 sec / 0.000 sec
50	16:34:25	SELECT	customer_id, MAX(order_date) AS last_order_date, COUNT(order_...	4 row(s) returned 0.000 sec / 0.000 sec
51	16:40:14	SELECT	COUNT(order_id) AS total_orders, SUM(total_amount) AS total_spent...	1 row(s) returned 0.000 sec / 0.000 sec
52	16:55:58	USE	coding_challenge_1	0 row(s) affected 0.000 sec
53	16:56:03	SELECT	* FROM orders INNER JOIN customers ON orders.customer_id = custome...	5 row(s) returned 0.000 sec / 0.000 sec
54	16:58:07	SELECT	* FROM customers LEFT JOIN orders ON customers.customer_id = orders...	5 row(s) returned 0.000 sec / 0.000 sec

### 3.Right Join (or Right Outer Join):

A right join returns all rows from the right table and the matched rows from the left table. If there is no match, NULL values are returned for columns from the left table.

This query is giving the total amount spent by each individual customer.

Query 1   SQL File 3\*   SQL File 5\*   SQL File 4\*   SQL File 6\*   SQL File 7\*   SQL File 8\* x

Limit to 1000 rows

```

13
14 -- Right join
15 • SELECT
16     customers.customer_id,
17     customers.customer_name,
18     COALESCE(SUM(orders.total_amount), 0) AS total_spent
19 FROM customers
20 RIGHT JOIN orders ON customers.customer_id = orders.customer_id
21 GROUP BY customers.customer_id, customers.customer_name;
22

```

Result Grid

customer_id	customer_name	total_spent
101	John Doe	150.00
102	Jane Smith	165.00
103	Bob Johnson	120.00

Result 14 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 62	17:03:42	SELECT * FROM orders RIGHT JOIN customers ON orders.customer_id = custome...	5 row(s) returned	0.000 sec / 0.000 sec
✓ 63	17:04:02	SELECT * FROM orders RIGHT JOIN customers ON orders.customer_id = custome...	5 row(s) returned	0.000 sec / 0.000 sec
✗ 64	17:05:47	SELECT orders.order_id, orders.customer_id, orders.order_date, orders.total_am...	Error Code: 1054. Unknown column 'customers.customers_name' in field list'	0.000 sec
✓ 65	17:06:02	SELECT * FROM customers LEFT JOIN orders ON customers.customer_id = order...	5 row(s) returned	0.000 sec / 0.000 sec
✓ 66	17:06:36	SELECT orders.order_id, orders.customer_id, orders.order_date, orders.total_am...	5 row(s) returned	0.000 sec / 0.000 sec
✓ 67	17:08:32	SELECT customers.customer_id, customers.customer_name, COALESCE(S...	3 row(s) returned	0.000 sec / 0.000 sec

## 4. Cross Join

CROSS JOIN returns the Cartesian product of rows from both tables.

The screenshot displays a SQL IDE interface with the following components:

- Query Window:** Contains a SQL query using a cross join.
 

```

      23 -- Cross join
      24 SELECT DISTINCT
      25     customers.customer_id,
      26     customers.customer_name,
      27     orders.order_id,
      28     orders.order_date,
      29     orders.total_amount,
      30     orders.total_amount * 1.2 AS total_amount_with_tax
      31 FROM customers
      32 CROSS JOIN orders;
      
```
- Result Grid:** Displays the results of the query.
 

customer_id	customer_name	order_id	order_date	total_amount	total_amount_with_tax
103	Bob Johnson	1	2024-01-01	50.00	60.000
102	Jane Smith	1	2024-01-01	50.00	60.000
101	John Doe	1	2024-01-01	50.00	60.000
103	Bob Johnson	2	2024-01-02	75.00	90.000
102	Jane Smith	2	2024-01-02	75.00	90.000
- Output Window:** Shows the execution logs.
 

#	Time	Action	Message	Duration / Fetch
66	17:06:36	SELECT	orders.order_id, orders.customer_id, orders.order_date, orders.total_am...	5 row(s) returned 0.000 sec / 0.000 sec
67	17:08:32	SELECT	customers.customer_id, customers.customer_name, COALESCE(S...	3 row(s) returned 0.000 sec / 0.000 sec
68	17:10:41	SELECT	customers.customer_id, customers.customer_name, orders.order_j...	15 row(s) returned 0.016 sec / 0.000 sec
69	17:11:23	SELECT	customers.customer_id, customers.customer_name, orders.order_j...	15 row(s) returned 0.000 sec / 0.000 sec
70	17:13:55	SELECT DISTINCT	customers.customer_id, customers.customer_name, ord...	15 row(s) returned 0.000 sec / 0.000 sec
71	17:14:39	SELECT DISTINCT	customers.customer_id, customers.customer_name, ord...	15 row(s) returned 0.000 sec / 0.000 sec

## 5. Self Join:

A self join is a regular join, but the table is joined with itself.

The purpose of this query is to identify pairs of orders made by the same customer where the order with id1 was placed before the order with id2

The screenshot shows a SQL IDE with multiple tabs. The active tab, 'SQL File 8\*', contains the following SQL query:

```

29 orders.total_amount,
30 orders.total_amount * 1.2 AS total_amount_with_tax
31 FROM customers
32 CROSS JOIN orders;
33
34 -- self join
35 • SELECT o1.order_id AS id1, o1.customer_id AS customer,
36       o2.order_id AS id2
37 FROM orders o1
38 JOIN orders o2 ON o1.customer_id = o2.customer_id AND o1.order_id < o2.order_id;

```

Below the query editor, the 'Result Grid' shows the output of the last query (line 38):

	id1	customer	id2
1	101	3	
2	102	5	

The 'Output' pane at the bottom shows the execution log for the last query (line 72):

#	Time	Action	Message	Duration / Fetch
67	17:08:32	SELECT	customers.customer_id, customers.customer_name, COALESCE(S...	3 row(s) returned 0.000 sec / 0.000 sec
68	17:10:41	SELECT	customers.customer_id, customers.customer_name, orders.order_j...	15 row(s) returned 0.016 sec / 0.000 sec
69	17:11:23	SELECT	customers.customer_id, customers.customer_name, orders.order_j...	15 row(s) returned 0.000 sec / 0.000 sec
70	17:13:55	SELECT DISTINCT	customers.customer_id, customers.customer_name, ord...	15 row(s) returned 0.000 sec / 0.000 sec
71	17:14:39	SELECT DISTINCT	customers.customer_id, customers.customer_name, ord...	15 row(s) returned 0.000 sec / 0.000 sec
72	17:19:34	SELECT o1.order_id AS id1, o1.customer_id AS customer, o2.order_id AS id2 ...	2 row(s) returned	0.015 sec / 0.000 sec

## 6.Equi Join:

An Equi join is a join using the equality operator (=) in the join condition.

This query is commonly used when you want to retrieve comprehensive information that involves columns from multiple tables based on a shared key, in this case, the **customer\_id**. It provides a consolidated view of orders along with the details of the customers who placed those orders.

The screenshot shows a SQL IDE interface. The query window contains the following SQL code:

```

39 JOIN orders o2 ON o1.customer_id = o2.customer_id AND o1.order_id < o2.order_id;
40
41 -- Equi join
42
43 SELECT *
44 FROM orders
45 JOIN customers ON orders.customer_id = customers.customer_id;
46
47
48

```

Below the query window is the 'Result Grid' showing 5 rows of data:

order_id	customer_id	order_date	total_amount	customer_id	customer_name
1	101	2024-01-01	50.00	101	John Doe
2	102	2024-01-02	75.00	102	Jane Smith
3	101	2024-01-03	100.00	101	John Doe
4	103	2024-01-04	120.00	103	Bob Johnson
5	102	2024-01-05	90.00	102	Jane Smith

At the bottom, the 'Output' window shows the execution log:

#	Time	Action	Message	Duration / Fetch
70	17:13:55	SELECT DISTINCT	customers.customer_id, customers.customer_name, ord...	15 row(s) returned
71	17:14:39	SELECT DISTINCT	customers.customer_id, customers.customer_name, ord...	15 row(s) returned
72	17:19:34	SELECT o1.order_id AS id1, o1.customer_id AS customer,	o2.order_id AS id2 ...	2 row(s) returned
73	17:21:37	select * from orders LIMIT 0, 1000		5 row(s) returned
74	17:21:49	SELECT o1.order_id AS id1, o1.customer_id AS customer,	o2.order_id AS id2 ...	2 row(s) returned
75	17:23:40	SELECT * FROM orders JOIN customers ON orders.customer_id = customers.custo...		5 row(s) returned

## 7.Non-Equi Join:

A non-equi join is a join using operators other than equality, such as <, >, <=, >=.

The query will return pairs of rows from the "orders" and "customers" tables where the customer\_id in the "orders" table is greater than the customer\_id in the "customers" table.

In other words, it will give combinations of orders and customers where the customer who placed the order has a higher customer\_id than the customer being joined.

The screenshot shows a SQL IDE interface. The query editor contains two SQL queries. The first query is a standard equi-join, and the second is a non-equijoin. The result grid displays the output of the first query, showing 5 rows of order data. The execution log at the bottom shows the sequence of SQL statements executed and the number of rows returned for each.

```
43 • SELECT *
44 FROM orders
45 JOIN customers ON orders.customer_id = customers.customer_id;
46
47 -- Non Equi join
48
49 • SELECT *
50 FROM orders
51 JOIN customers ON orders.customer_id > customers.customer_id;
52
```

order_id	customer_id	order_date	total_amount	customer_id	customer_name
2	102	2024-01-02	75.00	101	John Doe
4	103	2024-01-04	120.00	102	Jane Smith
4	103	2024-01-04	120.00	101	John Doe
5	102	2024-01-05	90.00	101	John Doe

#	Time	Action	Message	Duration / Fetch
72	17:19:34	SELECT o1.order_id AS id1, o1.customer_id AS customer, o2.order_id AS id2 ...	2 row(s) returned	0.015 sec / 0.000 sec
73	17:21:37	select * from orders LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
74	17:21:49	SELECT o1.order_id AS id1, o1.customer_id AS customer, o2.order_id AS id2 ...	2 row(s) returned	0.000 sec / 0.000 sec
75	17:23:40	SELECT * FROM orders JOIN customers ON orders.customer_id = customers.custo...	5 row(s) returned	0.000 sec / 0.000 sec
76	17:27:05	SELECT * FROM orders JOIN customers ON orders.customer_id < customers.custo...	6 row(s) returned	0.000 sec / 0.000 sec
77	17:28:29	SELECT * FROM orders JOIN customers ON orders.customer_id > customers custom...	4 row(s) returned	0.000 sec / 0.000 sec

## 8.Natural join

A NATURAL JOIN is a type of join in SQL that automatically matches columns with the same name in both tables, eliminating the need to explicitly specify the join condition.

Query 1SQL File 3\*SQL File 5\*SQL File 4\*SQL File 6\*SQL File 7\*SQL File 8\* x

SQLAdditionsMy Snippets

53-- natural join

54

55SELECT \*

56FROM orders

57NATURAL JOIN customers;

58

59

60

61

62

Result Grid

Filter Rows:

Export:

Wrap Cell Content: IA

Result Grid

customer_id	order_id	order_date	total_amount	customer_name
101	1	2024-01-01	50.00	John Doe
102	2	2024-01-02	75.00	Jane Smith
101	3	2024-01-03	100.00	John Doe
103	4	2024-01-04	120.00	Bob Johnson
102	5	2024-01-05	90.00	Jane Smith

Result 25 x

Read OnlyContext HelpSnippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
73	17:21:37	select *from orders LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
74	17:21:49	SELECT o1.order_id AS id1, o1.customer_id AS customer, o2.order_id AS id2 ...	2 row(s) returned	0.000 sec / 0.000 sec
75	17:23:40	SELECT * FROM orders JOIN customers ON orders.customer_id = customers.custo...	5 row(s) returned	0.000 sec / 0.000 sec
76	17:27:05	SELECT * FROM orders JOIN customers ON orders.customer_id < customers.custo...	6 row(s) returned	0.000 sec / 0.000 sec
77	17:28:29	SELECT * FROM orders JOIN customers ON orders.customer_id>customers.custo...	4 row(s) returned	0.000 sec / 0.000 sec
78	17:34:39	SELECT * FROM orders NATURAL JOIN customers LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec