

Coding challenge

SQL

Name: Rohan Vinayak Chaudhari

Batch: Data Engineering 1

Question2: Execute all the join with examples.

1)Creating Extra table to implement join:

Created Extra table to connect with the order table and perform joins operation.



The screenshot shows the MySQL Workbench interface. The SQL editor contains the following code:

```
37  
38 /*Create additional tables To implement joins*/  
39 CREATE TABLE customers (  
40     customer_id INT PRIMARY KEY,  
41     customer_name VARCHAR(50)  
42 );  
43  
44
```

The Output pane at the bottom shows the execution results:

Time	Action	Message	Duration
15:42:56	CREATE TABLE customers	customer_id INT PRIMARY KEY, customer_name VARCHAR(50)	0 rows affected

2)Inserting data in Customers table:

Inserted data in the customers table



The screenshot shows the MySQL Workbench interface. The SQL editor contains the following code:

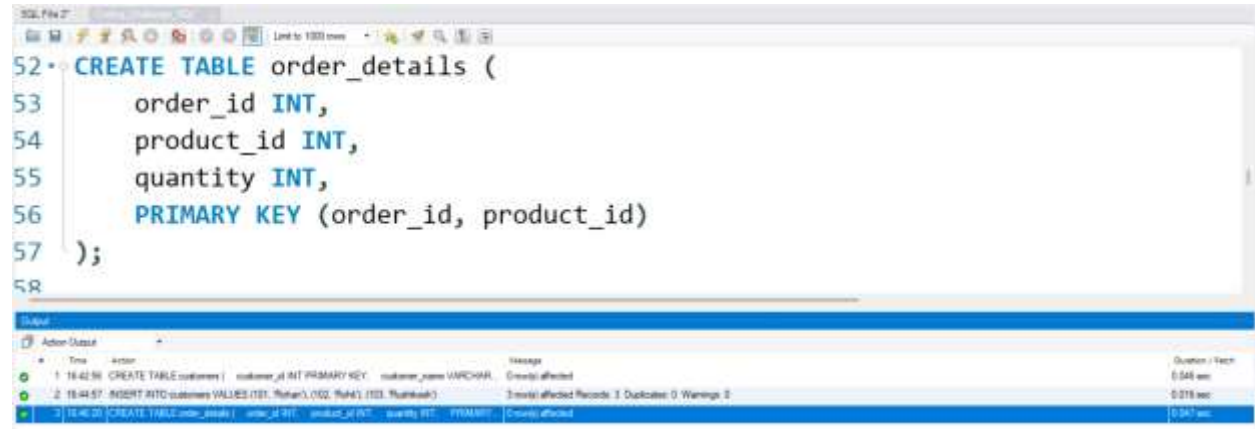
```
44 INSERT INTO customers VALUES  
45 (101, 'Rohan'),  
46 (102, 'Rohit'),  
47 (103, 'Rushikesh');  
48
```

The Output pane at the bottom shows the execution results:

Time	Action	Message	Duration
15:42:56	CREATE TABLE customers	customer_id INT PRIMARY KEY, customer_name VARCHAR(50)	0 rows affected
15:49:57	INSERT INTO customers VALUES	(101, 'Rohan'), (102, 'Rohit'), (103, 'Rushikesh')	3 rows affected

3)Creating 3rd table

Created Extra table to connect with the order & customer table and perform joins operation.



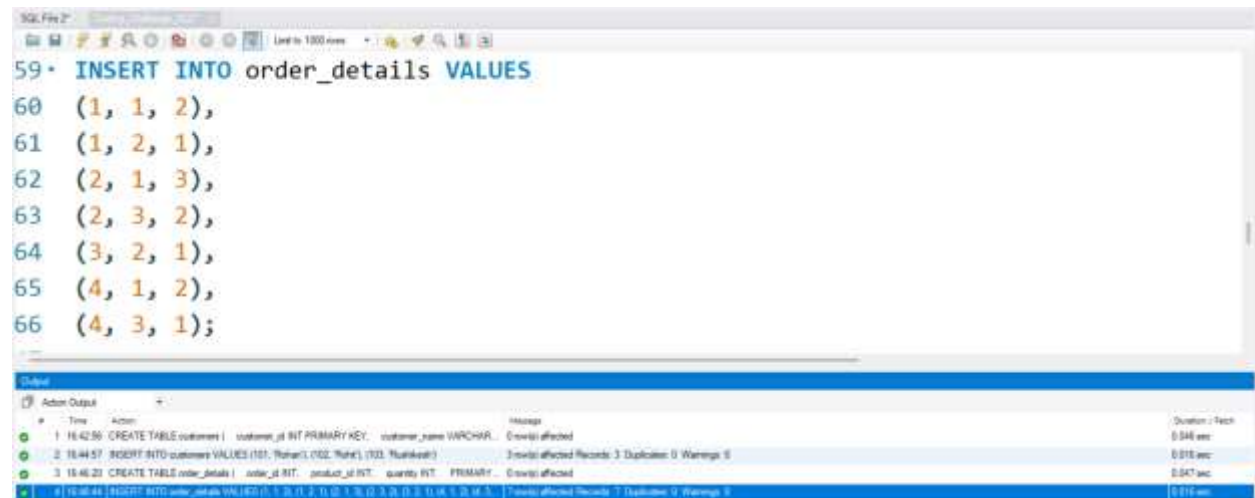
```
52 CREATE TABLE order_details (
53     order_id INT,
54     product_id INT,
55     quantity INT,
56     PRIMARY KEY (order_id, product_id)
57 );
58
```

Output:

#	Time	Action	Message	Duration / Tech
1	16:42:56	CREATE TABLE customers (customer_id INT PRIMARY KEY, customer_name VARCHAR(100));	0 records affected	0.048 sec
2	16:44:57	INSERT INTO customers VALUES (101, 'Rohan'), (102, 'Rishi'), (103, 'Nishu');	3 records affected Records: 3 Duplicates: 0 Warnings: 0	0.018 sec
3	16:46:30	CREATE TABLE order_details (order_id INT, product_id INT, quantity INT, PRIMARY KEY (order_id, product_id));	0 records affected	0.047 sec

4)Inserting data in order_details:

Inserted data in the order details table.




```
59 INSERT INTO order_details VALUES
60 (1, 1, 2),
61 (1, 2, 1),
62 (2, 1, 3),
63 (2, 3, 2),
64 (3, 2, 1),
65 (4, 1, 2),
66 (4, 3, 1);
```

Output:

#	Time	Action	Message	Duration / Tech
1	16:42:56	CREATE TABLE customers (customer_id INT PRIMARY KEY, customer_name VARCHAR(100));	0 records affected	0.048 sec
2	16:44:57	INSERT INTO customers VALUES (101, 'Rohan'), (102, 'Rishi'), (103, 'Nishu');	3 records affected Records: 3 Duplicates: 0 Warnings: 0	0.018 sec
3	16:46:30	CREATE TABLE order_details (order_id INT, product_id INT, quantity INT, PRIMARY KEY (order_id, product_id));	0 records affected	0.047 sec
4	16:48:44	INSERT INTO order_details VALUES (1, 1, 2), (1, 2, 1), (2, 1, 3), (2, 3, 2), (3, 2, 1), (4, 1, 2), (4, 3, 1);	7 records affected Records: 7 Duplicates: 0 Warnings: 0	0.016 sec

Returns only the rows where there is a match in both tables based on the specified condition (orders.customer_id = customers.customer_id).



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

```

73  /*LEFT JOIN*/
74  SELECT *
75  FROM orders
76  LEFT JOIN customers ON orders.customer_id = customers.customer_id;

```

The results pane displays a table with 6 columns: order_id, customer_id, order_date, order_amount, customer_id, and customer_name. The table contains 4 rows of data:

order_id	customer_id	order_date	order_amount	customer_id	customer_name
1	101	2024-01-01	150.00	101	Rohan
2	102	2024-01-02	150.00	102	Rishi
3	101	2024-01-03	200.00	101	Rohan
4	103	2024-01-04	120.00	103	Ramkishan

The status bar at the bottom indicates that the query was executed successfully, showing 4 rows returned.

7)Performing Right join:

Returns all rows from the right table (customers) and the matched rows from the left table (orders). If there is no match, NULL values are returned for columns from the left table.

The screenshot shows a SQL query window with the following code:

```
78 /*RIGHT JOIN*/
79 SELECT *
30 FROM orders
31 RIGHT JOIN customers ON orders.customer_id = customers.customer_id;
```

The results grid displays the following data:

order_id	customer_id	order_date	order_amount	customer_id	customer_name
1	101	2024-01-01	200.00	101	Ruhan
2	101	2024-01-01	100.00	101	Ruhan
3	102	2024-01-02	150.00	102	Rishi
4	103	2024-01-04	120.00	103	Rushabh

The output pane shows the execution message: "1 | 10.52 sec | SQL Server: FROM orders RIGHT JOIN customers ON orders.customer_id = customers.customer_id | 4 rows returned | 0.100 sec / 10.500 sec".

8)Performing Cross join:

Returns the Cartesian product of rows from both tables, resulting in all possible combinations of rows from orders and customers.

The screenshot shows a SQL query window with the following code:

```
32 /*CROSS JOIN*/
33 SELECT *
34 FROM orders
35 CROSS JOIN customers;
36
```

The results grid displays the following data:

order_id	customer_id	order_date	order_amount	customer_id	customer_name
1	101	2024-01-01	200.00	101	Rushabh
1	101	2024-01-01	200.00	102	Rishi
1	101	2024-01-01	200.00	103	Ruhan
2	102	2024-01-02	150.00	102	Rushabh
2	102	2024-01-02	150.00	103	Rishi
3	101	2024-01-03	200.00	103	Rushabh
3	101	2024-01-03	200.00	102	Rishi
3	101	2024-01-03	200.00	101	Ruhan

The output pane shows the execution message: "1 | 07.00 sec | SQL Server: FROM orders CROSS JOIN customers | 10617 rows returned | 0.300 sec / 0.000 sec".

9)Performing simple JOIN:

Gives all the result where data gets match with the given condition.

```
91
92 /*Full join*/
93 SELECT *
94 FROM orders
95 JOIN customers ON orders.customer_id = customers.customer_id;
96
```

order_id	customer_id	order_date	order_amount	customer_id	customer_name
1	101	2024-01-01	100.00	101	Rohan
2	102	2024-01-02	150.00	102	Rishi
3	101	2024-01-03	200.00	101	Rohan
4	103	2024-01-04	120.00	103	Rishabh

Result 24

Output

Action Output

1 | 1794.00 | SELECT * FROM orders JOIN customers ON orders.customer_id = customers.customer_id | 2 rows returned | 0.000 sec / 0.000 sec

10)Performing Self join:

Joins a table with itself. Here it retrieves rows from orders where the customer_id is the same, but the order_id is different, essentially finding pairs of orders for the same customer.

```
86
87 -- SELF JOIN
88 SELECT *
89 FROM orders o1
90 JOIN orders o2 ON o1.customer_id = o2.customer_id AND o1.order_id <> o2.order_id;
91
```

order_id	customer_id	order_date	order_amount	order_id	customer_id	order_date	order_amount
3	101	2024-01-03	200.00	1	101	2024-01-01	100.00
1	101	2024-01-01	100.00	3	101	2024-01-03	200.00

Result 22

Output

Action Output

1 | 1792.00 | SELECT * FROM orders o1 JOIN orders o2 ON o1.customer_id = o2.customer_id AND o1.order_id <> o2.order_id | 2 rows returned | 0.000 sec / 0.000 sec