MS SQL Coding Challenge

```
CREATE DATABASE Case_Study;
USE Case_Study;
--TABLE CREATION

CREATE TABLE burger_names(
  burger_id   INT PRIMARY KEY ,      burger_name VARCHAR(10) NOT NULL
);

CREATE TABLE burger_runner(
  runner_id   INT PRIMARY KEY,        registration_date date NOT NULL
);

CREATE TABLE customer_orders(
order_id    INT  NOT NULL,     customer_id INT NOT NULL,   burger_id    INT  NOT NULL,
exclusions  VARCHAR(4),        extras  VARCHAR(4),   order_time  timestamp NOT NULL
);

ALTER TABLE customer_orders
ADD FOREIGN KEY (burger_id) REFERENCES burger_names(burger_id);

CREATE TABLE runner_orders(
order_id     INT PRIMARY KEY,          runner_id    INT  NOT NULL,   pickup_time  timestamp,
distance     VARCHAR(7),        duration     VARCHAR(10),       cancellation VARCHAR(23)
);

ALTER TABLE runner_orders
ADD FOREIGN KEY (runner_id) REFERENCES burger_runner(runner_id);

--ADD DATA INTO DATABASE

INSERT INTO burger_names(burger_id,burger_name) VALUES (1,'Meatlovers');
INSERT INTO burger_names(burger_id,burger_name) VALUES (2,'Vegetarian');

INSERT INTO burger_runner VALUES (1,'2021-01-01');
INSERT INTO burger_runner VALUES (2,'2021-01-03');
INSERT INTO burger_runner VALUES (3,'2021-01-08');
INSERT INTO burger_runner VALUES (4,'2021-01-15');

INSERT INTO customer_orders VALUES (1,101,1,NULL,NULL,'2021-01-01 18:05:02');
INSERT INTO customer_orders VALUES (2,101,1,NULL,NULL,'2021-01-01 19:00:52');
INSERT INTO customer_orders VALUES (3,102,1,NULL,NULL,'2021-01-02 23:51:23');
INSERT INTO customer_orders VALUES (3,102,2,NULL,NULL,'2021-01-02 23:51:23');
INSERT INTO customer_orders VALUES (4,103,1,'4',NULL,'2021-01-04 13:23:46');
INSERT INTO customer_orders VALUES (4,103,1,'4',NULL,'2021-01-04 13:23:46');
```

```
INSERT INTO customer_orders VALUES (4,103,2,'4',NULL,'2021-01-04 13:23:46');
INSERT INTO customer_orders VALUES (5,104,1,NULL,'1','2021-01-08 21:00:29');
INSERT INTO customer_orders VALUES (6,101,2,NULL,NULL,'2021-01-08 21:03:13');
INSERT INTO customer_orders VALUES (7,105,2,NULL,'1','2021-01-08 21:20:29');
INSERT INTO customer_orders VALUES (8,102,1,NULL,NULL,'2021-01-09 23:54:33');
INSERT INTO customer_orders VALUES (9,103,1,'4','1, 5','2021-01-10 11:22:59');
INSERT INTO customer_orders VALUES (10,104,1,NULL,NULL,'2021-01-11 18:34:49');
INSERT INTO customer_orders VALUES (10,104,1,'2, 6','1, 4','2021-01-11 18:34:49');

INSERT INTO runner_orders VALUES (1,1,'2021-01-01 18:15:34','20km','32 minutes',NULL);
INSERT INTO runner_orders VALUES (2,1,'2021-01-01 19:10:54','20km','27 minutes',NULL);
INSERT INTO runner_orders VALUES (3,1,'2021-01-03 00:12:37','13.4km','20 mins',NULL);
INSERT INTO runner_orders VALUES (4,2,'2021-01-04 13:53:03','23.4','40',NULL);
INSERT INTO runner_orders VALUES (5,3,'2021-01-08 21:10:57','10','15',NULL);
INSERT INTO runner_orders VALUES (6,3,NULL,NULL,NULL,'Restaurant Cancellation');
INSERT INTO runner_orders VALUES (7,2,'2021-01-08 21:30:45','25km','25mins',NULL);
INSERT INTO runner_orders VALUES (8,2,'2021-01-10 00:15:02','23.4 km','15 minute',NULL);
INSERT INTO runner_orders VALUES (9,2,NULL,NULL,NULL,'Customer Cancellation');
INSERT INTO runner_orders VALUES (10,1,'2021-01-11 18:50:20','10km','10minutes',NULL);
```

Question & Answer

1.Querying Data by Using Joins and Subqueries & subtotal

JOIN:

JOIN is used to combine rows from two or more tables based on a related column. Joins allow retrieving data spread across multiple tables in a single query.

Types of Joins:

1. **INNER JOIN**: Returns only matching rows from both tables.
2. **LEFT JOIN**: Returns all rows from the left table and matching rows from the right table. Non-matching rows from the right table are filled with NULL.
3. **RIGHT JOIN:** Returns all rows from the right table and matching rows from the left table. Non-matching rows from the left table are filled with NULL.
4. **FULL JOIN:** Returns all rows when there is a match in either table, filling NULL for non-matching rows in both tables.
5. **CROSS JOIN**: Returns the Cartesian product of both tables, combining every row from the first table with every row from the second.
6. **SELF JOIN**: Joins a table with itself to compare rows within the same table.\

Subquery:

A subquery is a query nested inside another query, often in the SELECT, FROM, or WHERE clauses. It retrieves data used by the main query to refine or filter results.

Subtotal:

To calculate subtotal ROLLUP is used with the GROUP BY clause to calculate subtotals and grand totals. It adds summary rows for each group and a final total row.

QUERIES

1. --Find the Most Popular Burger [using subquery, group by, order by]

```
SELECT burger_name FROM burger_names WHERE burger_id = (
   SELECT TOP 1 burger_id  FROM customer_orders  GROUP BY burger_id
   ORDER BY COUNT(order_id) DESC
);
```

Results | Messages

	burger_name
1	Meatlovers

2. -- Find Customers Who Ordered Every Type of Burger [using subquery, group by, having, count, distinct]

```
SELECT customer_id  FROM customer_orders GROUP BY customer_id
HAVING COUNT(DISTINCT burger_id) = (  SELECT COUNT(burger_id)
FROM burger_names
);
```

Results | Messages

	customer_id
1	101
2	102
3	103

3. -- Find All Orders with Burger Names and Runner Details [using inner join, left join]

```
SELECT  co.order_id, co.customer_id, bn.burger_name, br.runner_id, ro.pickup_time
FROM  customer_orders co JOIN burger_names bn ON co.burger_id = bn.burger_id
LEFT JOIN runner_orders ro ON co.order_id = ro.order_id
LEFT JOIN burger_runner br ON ro.runner_id = br.runner_id;
```

89 %

Results | Messages

	order_id	customer_id	burger_name	runner_id	pickup_time
1	1	101	Meatlovers	1	2021-01-01 18:15:00.000
2	2	101	Meatlovers	1	2021-01-01 19:10:00.000
3	3	102	Meatlovers	1	2021-01-03 00:12:00.000
4	3	102	Vegetarian	1	2021-01-03 00:12:00.000
5	4	103	Meatlovers	2	2021-01-04 13:53:00.000
6	4	103	Meatlovers	2	2021-01-04 13:53:00.000
7	4	103	Vegetarian	2	2021-01-04 13:53:00.000
8	5	104	Meatlovers	3	2021-01-08 21:10:00.000
9	6	101	Vegetarian	3	NULL
10	7	105	Vegetarian	2	2021-01-08 21:30:00.000
11	8	102	Meatlovers	2	2021-01-10 00:15:00.000
12	9	103	Meatlovers	2	NULL
13	10	104	Meatlovers	1	2021-01-11 18:50:00.000
14	10	104	Meatlovers	1	2021-01-11 18:50:00.000

4. --Calculate Subtotals for Total Orders by Burger [using subtotal, count, group by rollup]

```
SELECT bn.burger_name, COUNT(co.order_id) AS total_orders FROM
   customer_orders co JOIN burger_names bn ON co.burger_id = bn.burger_id
GROUP BY  ROLLUP(bn.burger_name);
```

Results | Messages

	burger_name	total_orders
1	Meatlovers	10
2	Vegetarian	4
3	NULL	14

5. -- all runners and their corresponding orders[using right join]

```
SELECT    br.runner_id,    ro.order_id,    ro.pickup_time FROM
   runner_orders ro RIGHT JOIN  burger_runner br ON ro.runner_id = br.runner_id;
```

Results | Messages

	runner_id	order_id	pickup_time
1	1	1	2021-01-01 18:15:00.000
2	1	2	2021-01-01 19:10:00.000
3	1	3	2021-01-03 00:12:00.000
4	1	10	2021-01-11 18:50:00.000
5	2	4	2021-01-04 13:53:00.000
6	2	7	2021-01-08 21:30:00.000
7	2	8	2021-01-10 00:15:00.000
8	2	9	NULL
9	3	5	2021-01-08 21:10:00.000
10	3	6	NULL
11	4	NULL	NULL

6. -- all rows from both burger_runner and runner_orders, including rows where there is no match between the two tables [using full outer join]

```
SELECT   br.runner_id,    ro.order_id,    ro.pickup_time FROM
runner_orders ro FULL OUTER JOIN    burger_runner br ON ro.runner_id = br.runner_id;
```

Results | Messages

	runner_id	order_id	pickup_time
1	1	1	2021-01-01 18:15:00.000
2	1	2	2021-01-01 19:10:00.000
3	1	3	2021-01-03 00:12:00.000
4	2	4	2021-01-04 13:53:00.000
5	3	5	2021-01-08 21:10:00.000
6	3	6	NULL
7	2	7	2021-01-08 21:30:00.000
8	2	8	2021-01-10 00:15:00.000
9	2	9	NULL
10	1	10	2021-01-11 18:50:00.000
11	4	NULL	NULL

2.Manipulate data by using sql commands using groupby and having clause.

GROUP BY: Groups rows that have the same values into summary rows, like summing or counting data. It is used with aggregate functions (e.g., SUM(), COUNT(), AVG()) to perform operations on each group.

HAVING: Filters the results of a GROUP BY query based on a condition, similar to WHERE, but used for aggregated data. HAVING is used after grouping to filter grouped data, while WHERE filters data before grouping.

QUERIES

1. -- Total Orders per Runner with a Minimum Order Requirement [using count, group by, having]

```
SELECT ro.runner_id, COUNT(ro.order_id) AS total_orders FROM runner_orders ro

GROUP BY ro.runner_id HAVING COUNT(ro.order_id) >= 3;
```

Results | Messages

	runner_id	total_orders
1	1	4
2	2	4

2. -- Total Orders per Day with a Minimum Order Requirement [using cast, count, group by, having]

```
SELECT CAST(co.order_time AS DATE) AS order_date, COUNT(co.order_id) AS total_orders

FROM  customer_orders co GROUP BY CAST(co.order_time AS DATE)

HAVING COUNT(co.order_id) > 2;
```

Results | Messages

	order_date	total_orders
1	2021-01-04	3
2	2021-01-08	3

3. -- Count Total Orders per Burger Type [using count, inner join, group by]

```
SELECT bn.burger_name, COUNT(co.order_id) AS total_orders

FROM customer_orders co JOIN burger_names bn  ON co.burger_id = bn.burger_id

GROUP BY bn.burger_name;
```

Results | Messages

	burger_name	total_orders
1	Meatlovers	10
2	Vegetarian	4