🍕 Case Study #2D: Pizza Pricing

D. Pricing and Ratings

1. How much money has Pizza Runner made so far if there are no delivery fees?

Steps:

- Use the `customer_orders` table to count the number of each type of pizza.
- Multiply the counts by their respective prices:
  - Meat Lovers: $12
  - Vegetarian: $10
- Sum the total revenue from all orders.

```
SELECT
  SUM(
    CASE
      WHEN p.pizza_name = 'Meat Lovers' THEN 12
      WHEN p.pizza_name = 'Vegetarian' THEN 10
      ELSE 0
    END
  ) AS total_revenue
FROM customer_orders_temp AS c
JOIN pizza_runner.pizza_names AS p
  ON c.pizza_id = p.pizza_id;
```

Answer: Pizza Runner made $X in total revenue so far.

total_revenue
40

2. How much money has Pizza Runner made with a $1 charge for each extra topping?

Steps:

- Use the `customer_orders` table to calculate the total base revenue as in the previous query.
- Count the number of extras for each order using `REGEXP_SPLIT_TO_TABLE` and sum their charges.
- Add the extra topping revenue to the base revenue.

```
WITH base_revenue_cte AS (
  SELECT
    SUM(
      CASE
        WHEN p.pizza_name = 'Meat Lovers' THEN 12
        WHEN p.pizza_name = 'Vegetarian' THEN 10
        ELSE 0
      END
    ) AS base_revenue
  FROM customer_orders_temp AS c
  JOIN pizza_runner.pizza_names AS p
    ON c.pizza_id = p.pizza_id
),
extras_revenue_cte AS (
  SELECT
    SUM(1) AS extras_revenue
  FROM customer_orders_temp AS c,
```

```
    LATERAL REGEXP_SPLIT_TO_TABLE(c.extras, '[,\s]+') AS t(to
pping_id)
  WHERE TRIM(c.extras) <> ''
)
SELECT
  br.base_revenue + COALESCE(er.extras_revenue, 0) AS total_r
evenue_with_extras
FROM base_revenue_cte AS br
CROSS JOIN extras_revenue_cte AS er;
```

Answer: Pizza Runner made $46 including charges for extra toppings.

total_revenue_with_extras
46

---

3. How would you design a ratings system for runners?

Steps:

- Create a new table named `runner_ratings` to store customer feedback on runners.
- Include the following fields:
  - `rating_id`: Primary key for unique ratings.
  - `order_id`: Foreign key linking to `customer_orders`.
  - `runner_id`: Foreign key linking to `runner_orders`.
  - `customer_id`: Foreign key linking to `customer_orders`.
  - `rating`: Integer (1 to 5).
  - `rating_date`: Timestamp of when the rating was given.

Schema:

```
CREATE TABLE runner_ratings (
  rating_id SERIAL PRIMARY KEY,
  order_id INT NOT NULL, -- No foreign key constraint
  runner_id INT NOT NULL, -- No foreign key constraint
  customer_id INT NOT NULL, -- No foreign key constraint
  rating INT NOT NULL CHECK (rating BETWEEN 1 AND 5),
  rating_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Answer:

4. Join all data to create a comprehensive delivery table

Steps:

- Join `customer_orders`, `runner_orders`, and `runner_ratings` tables.
- Calculate additional metrics:
  - Time between `order_time` and `pickup_time`.
  - Average speed as `distance / duration`.
  - Total pizzas per order.

```
SELECT
  c.customer_id,
  c.order_id,
  r.runner_id,
  rr.rating,
  c.order_time,
  r.pickup_time,
  EXTRACT(EPOCH FROM (r.pickup_time - c.order_time)) / 60 AS
time_to_pickup, -- Time in minutes
  r.duration AS delivery_duration,
  ROUND(r.distance / (r.duration / 60), 2) AS avg_speed, -- A
```

```
verage speed in km/h
  COUNT(c.pizza_id) AS total_pizzas
FROM customer_orders_temp AS c
JOIN runner_orders_temp AS r
  ON c.order_id = r.order_id
LEFT JOIN runner_ratings AS rr
  ON c.order_id = rr.order_id
WHERE r.distance > 0 -- Only include successful deliveries
GROUP BY
  c.customer_id, c.order_id, r.runner_id, rr.rating,
  c.order_time, r.pickup_time, r.duration, r.distance;
```

Answer:

customer_id	order_id	runner_id	rating	order_time	pickup_time	time_to_pickup	delivery_duration	avg_speed	total_pizzas
104	10	1	null	2020-01-11 18:34:49	2020-01-11 18:50:20	15.516666666666667	10	60.00	2
104	5	3	null	2020-01-08 21:00:29	2020-01-08 21:10:57	10.466666666666667	15	40.00	1
101	2	1	null	2020-01-01 19:00:52	2020-01-01 19:10:54	10.033333333333333	27	44.44	1
102	8	2	null	2020-01-09 23:54:33	2020-01-10 00:15:02	20.483333333333334	15	93.60	1
101	1	1	null	2020-01-01 18:05:02	2020-01-01 18:15:34	10.533333333333333	32	37.50	1
105	7	2	null	2020-01-08 21:20:29	2020-01-08 21:30:45	10.266666666666667	25	60.00	1
103	4	2	null	2020-01-04 13:23:46	2020-01-04 13:53:03	29.283333333333335	40	35.10	3
102	3	1	null	2020-01-02 23:51:23	2020-01-03 00:12:37	21.233333333333334	20	40.20	2

5. How much money does Pizza Runner have left after paying runners?

Steps:

- Calculate total revenue as in question 2.
- Calculate the total cost of runner payments:
  - Each runner is paid $0.30 per kilometer traveled.
- Subtract the runner payments from the total revenue to get the profit.

```
WITH runner_payment_cte AS (
  SELECT
    SUM(r.distance * 0.30) AS total_runner_payments
  FROM runner_orders_temp AS r
```

```
  WHERE r.distance > 0
),
total_revenue_cte AS (
  SELECT
    SUM(
      CASE
        WHEN p.pizza_name = 'Meat Lovers' THEN 12
        WHEN p.pizza_name = 'Vegetarian' THEN 10
        ELSE 0
      END
    ) AS total_revenue
  FROM customer_orders_temp AS c
  JOIN pizza_runner.pizza_names AS p
    ON c.pizza_id = p.pizza_id
)
SELECT
  COALESCE(tr.total_revenue, 0) - COALESCE(rp.total_runner_pa
yments, 0) AS profit
FROM total_revenue_cte AS tr, runner_payment_cte AS rp;
```

Answer: Pizza Runner's profit after paying runners is -$3.5.

profit
-3.560