-- A. Pizza Metrics

--1 How many pizzas were ordered?

SELECT

COUNT(*) AS total_pizzas_ordered

FROM customer_orders_clean;

RESULT:

	total_pizzas_ordered bigint
1	14

- Toatal 14 pizzas were ordered

-- 2 How many unique customer orders were made?

SELECT

COUNT(DISTINCT customer_id) AS unique_customers

FROM customer_orders_clean;

RESULT:

	unique_customers bigint
1	5

-- 5 unique customers' orders were made

--3 How many successful orders were delivered by each runner?

SELECT runner_id,COUNT(order_id) AS tot_order_delivered

FROM runner_orders_clean

WHERE distance>0

GROUP BY runner_id

ORDER BY runner_id;

RESULT:

	runner_id integer	tot_order_delivered bigint
1	1	4
2	2	3
3	3	1

-- 4 How many of each type of pizza was delivered?

```
SELECT pn.pizza_name,COUNT(c.pizza_id) AS pizza_delivered_Number

FROM customer_orders_clean c

JOIN runner_orders_clean r

ON c.order_id=r.order_id

JOIN pizza_names pn

ON c.pizza_id=pn.pizza_id

WHERE r.distance>0

GROUP BY pn.pizza_name

ORDER BY pn.pizza_name;
```

RESULT:

	pizza_name text	pizza_delivered_number bigint
1	Meatlovers	9
2	Vegetarian	3

-- 5 How many Vegetarian and Meatlovers were ordered by each customer?

```
SELECT c.customer_id,pn.pizza_name,

COUNT(pn.pizza_name) AS order_count

FROM customer_orders_clean c

JOIN pizza_names pn

ON c.pizza_id=pn.pizza_id

GROUP BY c.customer_id,pn.pizza_name

ORDER BY c.customer_id
```

RESULT:

	customer_id integer	pizza_name text	order_count bigint
1	101	Meatlovers	2
2	101	Vegetarian	1
3	102	Meatlovers	2
4	102	Vegetarian	1
5	103	Meatlovers	3
6	103	Vegetarian	1
7	104	Meatlovers	3
8	105	Vegetarian	1

-- 6 What was the maximum number of pizzas delivered in a single order?

```
WITH count_pizza
AS
(
SELECT c.order_id,COUNT(c.pizza_id) AS pizza_count
        FROM customer_orders_clean c
        JOIN runner_orders_clean r
        ON c.order_id=r.order_id
        WHERE r.distance>0
        GROUP BY c.order_id
)
SELECT MAX(pizza_count) AS pizza_count
FROM count_pizza;
```

RESULT:

	pizza_count bigint
1	3

-- 7 For each customer, how many delivered pizzas had at least 1 change and how many had no changes?

```
SELECT c.customer_id,
SUM(
CASE WHEN c.exclusions <> NULL OR c.extras <> NULL THEN 1
        ELSE 0
        END
)AS change,
SUM(
CASE
        WHEN c.exclusions=NULL or c.extras=NULL THEN 1
        ELSE 0
        END
) AS no_change
FROM customer_orders_clean c
JOIN runner_orders_clean r
ON c.order_id=r.order_id
WHERE r.distance > 0
GROUP BY c.customer_id
ORDER BY c.customer_id;
```

RESULT:

	customer_id integer	change bigint	no_change bigint
1	101	0	0
2	102	0	0
3	103	0	0
4	104	0	0
5	105	0	0

-- 8 How many pizzas were delivered that had both exclusions and extras?

```
SELECT COUNT(*) AS pizza_delivered_with_exclusions_and_extras

FROM customer_orders_clean c JOIN runner_orders_clean r

ON c.order_id=r.order_id

AND r.distance>0

WHERE c.exclusions IS NOT NULL

  AND c.exclusions <> ''

  AND c.extras IS NOT NULL

  AND c.extras <> '';
```

RESULT:

	pizza_delivered_with_exclusions_and_extras bigint
1	1

-- 9 What was the total volume of pizzas ordered for each hour of the day?

```
SELECT EXTRACT(HOUR FROM c.order_time) AS order_hour,

COUNT(c.pizza_id) AS pizza_order_count  -- CAN USE order_id both produce same result

FROM customer_orders_clean c

GROUP BY order_hour

ORDER BY order_hour;
```

RESULT:

	order_hour numeric	pizza_order_count bigint
1	11	1
2	13	3
3	18	3
4	19	1
5	21	3
6	23	3

-- 10 What was the volume of orders for each day of the week?

```
SELECT
        EXTRACT(DOW FROM c.order_time) AS day_number,
        TO_CHAR(c.order_time,'Day') AS day_of_week,
        COUNT(c.order_id) AS order_count
FROM customer_orders_clean c
GROUP BY day_number,day_of_week
ORDER BY day_number;
```

RESULT:

	day_number numeric	day_of_week text	order_count bigint
1	3	Wednesday	5
2	4	Thursday	3
3	5	Friday	1
4	6	Saturday	5

-- B. Runner and Customer Experience

-- 1 How many runners signed up for each 1 week period? (i.e. week starts 2021-01-01)

```
SELECT
        DATE_TRUNC('week',registration_date) AS signup_week,
        COUNT(runner_id) AS runner_count
        FROM runners
        WHERE registration_date>='2021-01-01'
        GROUP BY signup_week
        ORDER BY signup_week;
```

RESULT:

	signup_week timestamp with time zone	runner_count bigint
1	2020-12-28 00:00:00-08	2
2	2021-01-04 00:00:00-08	1
3	2021-01-11 00:00:00-08	1

-- 2 What was the average time in minutes it took for each runner to arrive at the Pizza Runner HQ to pickup the order?

```
WITH minutes_cte AS

(

        SELECT

        r.runner_id,

        ROUND(EXTRACT(EPOCH FROM(r.pickup_time-c.order_time))/60 )as avg_arrival_in_minutes

        --EXTRACT(EPOCH FROM ...) returns the difference between two timestamps in seconds.

        --It calculates the "epoch" or the number of seconds that have passed between two timestamps,

        -- which can then be converted to minutes by dividing by 60.

          FROM customer_orders_clean c

          JOIN runner_orders_clean r

          ON c.order_id=r.order_id

        WHERE r.pickup_time IS NOT NULL

          AND c.order_time IS NOT NULL

        GROUP BY r.runner_id,r.pickup_time,c.order_time

        ORDER BY avg_arrival_in_minutes


)


SELECT ROUND(avg(avg_arrival_in_minutes),2) AS avg_time_in_minutes

                    FROM minutes_cte;
```

RESULT:

	avg_time_in_minutes numeric
1	15.88

--3 Is there any relationship between the number of pizzas and how long the order takes to prepare?

```
--PIZZA & ORDER PREPARATION TIME

WITH order_preparation_time_cte

AS

(

SELECT

        r.order_id,

        COUNT(c.pizza_id) AS pizza_count,

        EXTRACT(EPOCH FROM(r.pickup_time-c.order_time))/60 AS preparation_time_in_minutes

        FROM runner_orders_clean r

        JOIN customer_orders_clean c

        ON r.order_id=c.order_id

        AND r.distance>0

        WHERE r.pickup_time IS NOT NULL

        AND c.order_time IS NOT NULL

        GROUP BY r.order_id,r.pickup_time,c.order_time

)

SELECT

        pizza_count,

        ROUND(AVG(preparation_time_in_minutes)) AVG_prep_time_minutes

        FROM order_preparation_time_cte

        GROUP BY pizza_count

        ORDER BY pizza_count;
```

RESULT:

	pizza_count bigint	avg_prep_time_minutes numeric
1	1	12
2	2	18
3	3	29

-- TO make 1 pizza=12min,3 pizza=18min,3pizza=29 min time

-- 4 What was the average distance travelled for each customer?

```
-- CUSTOMER AND DISTANCE

SELECT

c.customer_id,

ROUND(AVG(r.distance)) AS average_distance

FROM customer_orders_clean c

JOIN runner_orders_clean r

ON c.order_id=r.order_id

AND r.duration>0

GROUP BY c.customer_id

ORDER BY c.customer_id;
```

RESULT:

	customer_id integer	average_distance double precision
1	101	20
2	102	17
3	103	23
4	104	10
5	105	25

--Customer 104 stays the nearest to Pizza Runner HQ at average distance of 10km, whereas Customer 105 stays the furthest at 25km.

-- 5 What was the difference between the longest and shortest delivery times for all orders?

```
SELECT MAX(duration),MIN(duration),(MAX(duration)-MIN(duration)) delivery_time_difference

FROM runner_orders_clean

WHERE duration IS NOT NULL;
```

RESULT:

	max integer	min integer	delivery_time_difference integer
1	40	10	30

--Diff in largest and shortest delivry time is 30 min

-- 6 What was the average speed for each runner for each delivery and do you notice any trend for these values?

```
SELECT

  runner_id,

  order_id,

  ROUND(((distance::numeric(3, 1)) /(duration::numeric(3, 1)/60)),2) as speed_km_per_hour

FROM

  runner_orders_clean

WHERE

  duration IS NOT NULL

ORDER BY

  runner_id,

  order_id;
```

RESULT:

	runner_id integer	order_id integer	speed_km_per_hour numeric
1	1	1	37.50
2	1	2	44.44
3	1	3	40.20
4	1	10	60.00
5	2	4	35.10
6	2	7	60.00
7	2	8	93.60
8	3	5	40.00

/*(Average speed = Distance in km / Duration in hour)

- Runner 1's average speed runs from 37.5km/h to 60km/h.
- Runner 2's average speed runs from 35.1km/h to 93.6km/h. Danny should investigate Runner 2 as the average speed has a 300% fluctuation rate!
- Runner 3's average speed is 40km/h

*/

-- 7 What is the successful delivery percentage for each runner?

```
SELECT
        runner_id,
        COUNT(*) As tot_deliveries,
        COUNT(
        CASE WHEN duration IS NOT NULL AND pickup_time IS NOT NULL THEN 1 END
        ) AS successful_deliveries,
        ROUND(
                COUNT(
        CASE WHEN duration IS NOT NULL AND pickup_time IS NOT NULL THEN 1 END
        )::DECIMAL/COUNT(*)*100,2
        )AS success_percentage
        FROM runner_orders_clean
        GROUP BY runner_id
        ORDER BY success_percentage DESC;
```

RESULT:

	runner_id integer	tot_deliveries bigint	successful_deliveries bigint	success_percentage numeric
1	1	4	4	100.00
2	2	4	3	75.00
3	3	2	1	50.00

/*Runner 1 has 100% successful delivery.

Runner 2 has 75% successful delivery.

Runner 3 has 50% successful delivery*/

--C. Ingredient Optimisation

-- 1 What are the standard ingredients for each pizza?

```
SELECT

pn.pizza_name,

pt.topping_name

FROM

pizza_names pn JOIN pizza_recepies_clean pr

ON pn.pizza_id=pr.pizza_id

JOIN pizza_toppings pt

ON pr.topping_id=pt.topping_id

ORDER BY pn.pizza_name,pt.topping_name;
```

RESULT:

	pizza_name text	topping_name text
1	Meatlovers	Bacon
2	Meatlovers	BBQ Sauce
3	Meatlovers	Beef
4	Meatlovers	Cheese
5	Meatlovers	Chicken
6	Meatlovers	Mushrooms
7	Meatlovers	Pepperoni
8	Meatlovers	Salami
9	Vegetarian	Cheese
10	Vegetarian	Mushrooms
11	Vegetarian	Onions
12	Vegetarian	Peppers
13	Vegetarian	Tomato Sauce
14	Vegetarian	Tomatoes

-- 2 What was the most commonly added extra?

```
WITH extras_cte AS (
   SELECT
      UNNEST(STRING_TO_ARRAY(extras, ',')):INTEGER AS extras_id
   FROM customer_orders
            WHERE extras IS NOT NULL
   AND extras != ''
   AND extras != 'null'  -- Exclude any 'null' strings

)
SELECT
  e.extras_id,
   pt.topping_name AS extras_name,
   COUNT(e.extras_id) AS extras_count
FROM extras_cte e
JOIN pizza_toppings pt
ON e.extras_id = pt.topping_id
GROUP BY e.extras_id,pt.topping_name
ORDER BY extras_count DESC
--LIMIT 1;
```

RESULT:

	extras_id integer	extras_name text	extras_count bigint
1	1	Bacon	4
2	4	Cheese	1
3	5	Chicken	1

--most common added extra is bacon which count =4

-- 3 What was the most common exclusion?

```
WITH exclusion_cte AS (
   SELECT
      UNNEST(STRING_TO_ARRAY(exclusions, ',')):INTEGER AS exclusion_id
   FROM customer_orders
            WHERE exclusions IS NOT NULL
   AND exclusions != ''
   AND exclusions != 'null'  -- Exclude any 'null' strings

)
SELECT
  e.exclusion_id,
   pt.topping_name AS exclusion_name,
   COUNT(e.exclusion_id) AS exclusion_count
FROM exclusion_cte e
JOIN pizza_toppings pt
ON e.exclusion_id = pt.topping_id
GROUP BY e.exclusion_id,pt.topping_name
ORDER BY exclusion_count DESC
--LIMIT 1;
```

RESULT:

	exclusion_id integer	exclusion_name text	exclusion_count bigint
1	4	Cheese	4
2	6	Mushrooms	1
3	2	BBQ Sauce	1

--most common exclusion is chees.

-- 4 Generate an order item for each record in the customers_orders table

--in the format of one of the following:

--Meat Lovers

--Meat Lovers - Exclude Beef

--Meat Lovers - Extra Bacon

--Meat Lovers - Exclude Cheese, Bacon - Extra Mushroom, Peppers

```
WITH exclusions_cte AS
(
  SELECT
        c.order_id,
        c.pizza_id,
        pt.topping_id,
        pt.topping_name
        FROM customer_orders_clean c
        JOIN pizza_toppings pt
        ON topping_id=ANY(STRING_TO_ARRAY(c.exclusions,',')::INT[])
),
extras_cte AS
(
 SELECT
        c.order_id,
        c.pizza_id,
        pt.topping_id,
        pt.topping_name
        FROM customer_orders_clean c
        JOIN pizza_toppings pt
        ON topping_id=ANY(STRING_TO_ARRAY(c.extras,',')::INT[])

),
```

```
orders_cte AS (

        SELECT
        DISTINCT c.order_id,
        c.pizza_id,
        pr.topping_id
        FROM customer_orders_clean c
        INNER JOIN pizza_recepies_clean pr
        ON c.pizza_id=pr.pizza_id

),
orders_with_extras_and_exclusions_cte AS
(
        SELECT o.order_id,o.pizza_id,
        CASE WHEN o.pizza_id=1 THEN 'Meat Lovers'
            WHEN o.pizza_id=2 THEN pn.pizza_name
        END AS pizza_names,
        STRING_AGG(DISTINCT ext.topping_name,',') AS extras,
        STRING_AGG(DISTINCT excl.topping_name,',') AS exclusions
        FROM orders_cte o
        LEFT JOIN extras_cte ext
        ON ext.order_id=o.order_id AND ext.pizza_id=o.pizza_id
        LEFT JOIN exclusions_cte excl
        ON excl.order_id=o.order_id AND excl.pizza_id=o.pizza_id AND excl.topping_id=o.topping_id
    INNER JOIN pizza_names pn
         ON o.pizza_id=pn.pizza_id
        GROUP BY o.order_id,o.pizza_id,pizza_names

)
```

```
SELECT
        order_id,
        pizza_id,
        CONCAT(pizza_names,
                CASE WHEN exclusions='' THEN '' ELSE '-Exclude'|| exclusions END,
                CASE WHEN extras='' THEN '' ELSE '-Extras'||extras END) AS order_item
                FROM orders_with_extras_and_exclusions_cte
                ORDER BY order_id;
```

RESULT:

	order_id integer	pizza_id integer	order_item text
1	1	1	Meat Lovers
2	2	1	Meat Lovers
3	3	1	Meat Lovers
4	3	2	Vegetarian
5	4	1	Meat Lovers-ExcludeCheese
6	4	2	Vegetarian-ExcludeCheese
7	5	1	Meat Lovers-ExtrasBacon
8	6	2	Vegetarian
9	7	2	Vegetarian-ExtrasBacon
10	8	1	Meat Lovers
11	9	1	Meat Lovers-ExcludeCheese-ExtrasBacon,Chicken
12	10	1	Meat Lovers-ExcludeBBQ Sauce,Mushrooms-ExtrasBacon,Chee...

-- 5 Generate an alphabetically ordered comma separated ingredient list

--for each pizza order from the customer_orders table and add a 2x in front of any relevant ingredients

--For example: "Meat Lovers: 2xBacon, Beef, ... , Salami"

```
WITH exclusions_cte AS
(
  SELECT
        c.order_id,
        c.pizza_id,
        pt.topping_id,
        pt.topping_name
        FROM customer_orders_clean c
        JOIN pizza_toppings pt
        ON topping_id=ANY(STRING_TO_ARRAY(c.exclusions,',')::INT[])
),

extras_cte AS
(
         SELECT
        c.order_id,
        c.pizza_id,
        pt.topping_id,
        pt.topping_name
        FROM customer_orders_clean c
        JOIN pizza_toppings pt
        ON topping_id=ANY(STRING_TO_ARRAY(c.extras,',')::INT[])
),
```

```
orders_cte AS (
    SELECT
        DISTINCT c.order_id,
        c.pizza_id,
        pr.topping_id,
        pt.topping_name
        FROM customer_orders_clean c
        INNER JOIN pizza_recepies_clean pr
        ON c.pizza_id=pr.pizza_id
    LEFT JOIN pizza_toppings pt
        ON pr.topping_id=pt.topping_id

),

orders_with_extras_and_exclusions_cte AS
(
    SELECT
    O.order_id,
    O.pizza_id,
    O.topping_id,
    o.topping_name
    FROM orders_cte AS O
    LEFT JOIN exclusions_cte AS excl
        ON excl.order_id=o.order_id
        AND excl.pizza_id=o.pizza_id
        AND excl.topping_id=o.topping_id
        WHERE excl.topping_id IS NULL

    UNION ALL
```

```
    SELECT
    ext.order_id,
   ext.pizza_id,
   ext.topping_id,
    ext.topping_name
    FROM extras_cte ext
    WHERE ext.topping_id IS NOT NULL
),
count_topping_cte AS
(
SELECT o.order_id,
        o.pizza_id,
        o.topping_name,
        count(*) AS n
        FROM orders_with_extras_and_exclusions_cte AS o
        GROUP BY o.order_id,o.pizza_id,o.topping_name
)

SELECT
        order_id,
        pizza_id,
        STRING_AGG(
                CASE WHEN n>1 THEN n ||'x'||topping_name
                ELSE topping_name
                END,' , ') AS ingredient
                FROM count_topping_cte
                GROUP BY order_id,pizza_id;
```

RESULT:

	order_id integer	pizza_id integer	ingredient text
1	1	1	Bacon , BBQ Sauce , Beef , Cheese , Chicken , Mushrooms , Pepperoni , Salami
2	2	1	Bacon , BBQ Sauce , Beef , Cheese , Chicken , Mushrooms , Pepperoni , Salami
3	3	1	Bacon , BBQ Sauce , Beef , Cheese , Chicken , Mushrooms , Pepperoni , Salami
4	3	2	Cheese , Mushrooms , Onions , Peppers , Tomato Sauce , Tomatoes
5	4	1	Bacon , BBQ Sauce , Beef , Chicken , Mushrooms , Pepperoni , Salami
6	4	2	Mushrooms , Onions , Peppers , Tomato Sauce , Tomatoes
7	5	1	2xBacon , BBQ Sauce , Beef , Cheese , Chicken , Mushrooms , Pepperoni , Sala...
8	6	2	Cheese , Mushrooms , Onions , Peppers , Tomato Sauce , Tomatoes
9	7	2	Bacon , Cheese , Mushrooms , Onions , Peppers , Tomato Sauce , Tomatoes
10	8	1	Bacon , BBQ Sauce , Beef , Cheese , Chicken , Mushrooms , Pepperoni , Salami
11	9	1	2xBacon , BBQ Sauce , Beef , 2xChicken , Mushrooms , Pepperoni , Salami
12	10	1	2xBacon , Beef , 2xCheese , Chicken , Pepperoni , Salami

-- 6 What is the total quantity of each ingredient used in all delivered pizza sorted by most frequent first?

```
WITH exclusions_cte AS
(
  SELECT
        c.order_id,
        c.pizza_id,
        pt.topping_id,
        pt.topping_name
        FROM customer_orders_clean c
        JOIN pizza_toppings pt
        ON topping_id=ANY(STRING_TO_ARRAY(c.exclusions,',')::INT[])
),
```

```
extras_cte AS
(
        SELECT
        c.order_id,
        c.pizza_id,
        pt.topping_id,
        pt.topping_name
        FROM customer_orders_clean c
        JOIN pizza_toppings pt
        ON topping_id=ANY(STRING_TO_ARRAY(c.extras,','))::INT[])
),

orders_cte AS (

  SELECT
        DISTINCT c.order_id,
        c.pizza_id,
        pr.topping_id,
        pt.topping_name
        FROM customer_orders_clean c
        INNER JOIN pizza_recepies_clean pr
        ON c.pizza_id=pr.pizza_id
  LEFT JOIN pizza_toppings pt
        ON pr.topping_id=pt.topping_id

),
```

```
orders_with_extras_and_exclusions_cte AS
( SELECT
    O.order_id,
    O.pizza_id,
    O.topping_id,
    o.topping_name
    FROM orders_cte AS O
    LEFT JOIN exclusions_cte AS excl
            ON excl.order_id=o.order_id
            AND excl.pizza_id=o.pizza_id
            AND excl.topping_id=o.topping_id
            WHERE excl.topping_id IS NULL
    UNION ALL
    SELECT
    ext.order_id,
   ext.pizza_id,
   ext.topping_id,
    ext.topping_name
    FROM extras_cte ext
    WHERE ext.topping_id IS NOT NULL
)
SELECT
                                oc.topping_name,
                                 COUNT(oc.pizza_id) AS ingredient_count
                                 FROM orders_with_extras_and_exclusions_cte AS oc
                                 INNER JOIN runner_orders_clean AS r
                                 ON oc.order_id=r.order_id
                                 WHERE pickup_time IS NOT NULL
                                 GROUP BY oc.topping_name
                                 ORDER BY COUNT(oc.pizza_id) DESC;
```

RESULT:

	topping_name text	ingredient_count bigint
1	Bacon	10
2	Cheese	9
3	Mushrooms	9
4	Pepperoni	7
5	Salami	7
6	Chicken	7
7	Beef	7
8	BBQ Sauce	6
9	Tomatoes	3
10	Onions	3
11	Peppers	3
12	Tomato Sauce	3

--D. Pricing and Ratings

-- 1 If a Meat Lovers pizza costs $12 and Vegetarian costs $10 and there were no charges for changes how much money has Pizza Runner made so far if there are no delivery fees?

```
WITH pizza_prices AS
(
SELECT 'Meatlovers' AS pizza_name, 12 AS price
        UNION ALL
        SELECT 'vegetarian',10
)
SELECT SUM(p.price) AS total_price
FROM customer_orders_clean c
JOIN pizza_names pn
ON c.pizza_id=pn.pizza_id
JOIN pizza_prices p
ON pn.pizza_name=p.pizza_name
WHERE
        c.order_id NOT IN(
                                SELECT r.order_id
            FROM runner_orders_clean r
            WHERE r.cancellation IS NOT NULL
            );
```

RESULT:

	total_price bigint
1	108

-- 2 What if there was an additional $1 charge for any pizza extras? Add cheese is $1 extra

```
WITH pizza_prices AS
(
SELECT 'Meatlovers' AS pizza_name, 12 AS price
        UNION ALL
        SELECT 'vegetarian',10
),
extra_cost_cte AS
(
SELECT c.order_id,
        c.pizza_id,
        COALESCE(extra_count.extra_cost,0) AS extra_cost
        FROM customer_orders_clean c
        LEFT JOIN LATERAL
        (
        --split the extras and count num of extras
                SELECT COUNT(*) AS extra_cost
         FROM UNNEST(STRING_TO_ARRAY(c.extras,',')) AS extra
                WHERE c.extras IS NOT NULL AND c.extras <>''
        ) AS extra_count
        ON TRUE
)
SELECT SUM(p.price+ec.extra_cost) AS total_cost
FROM customer_orders_clean c
JOIN pizza_names pn
ON c.pizza_id=pn.pizza_id
JOIN pizza_prices p
ON pn.pizza_name=p.pizza_name
JOIN extra_cost_cte ec
ON c.order_id=ec.order_id
```

```
WHERE
        c.order_id NOT IN(
                SELECT r.order_id
                FROM runner_orders_clean r
                WHERE r.cancellation IS NOT NULL
                );
```

RESULT:

	total_cost numeric
1	197

-- 3 The Pizza Runner team now wants to add an additional ratings system that allows customers to rate their runner, how would you design an additional table for this new dataset generate a schema for this new table and

insert your own data for ratings for each successful customer order between 1 to 5.

```
CREATE TABLE runner_ratings (
   rating_id SERIAL PRIMARY KEY,
   order_id INTEGER NOT NULL,
   runner_id INTEGER NOT NULL CHECK(runner_id  BETWEEN 1 AND 4),
   rating INTEGER CHECK (rating >= 1 AND rating <= 5),
   comment TEXT,
   rating_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO runner_ratings (order_id, runner_id, rating, comment)
VALUES
   (1, 1, 5, 'The runner was very prompt and friendly!'),
   (2, 1, 4, 'Great service, but a bit delayed.'),
   (3, 2, 5, 'Perfect delivery, thank you!'),
   (4, 2, 3, 'The delivery was okay, but the runner was not very polite.'),
```

(5, 3, 4, 'Good service overall, will order again.'),

(6, 3, 5, 'Outstanding delivery experience!'),

(7, 2, 4, 'The runner was quick but forgot my drink.'),

(8, 2, 5, 'Excellent, very satisfied with the service.'),

(9, 1, 3, 'It was fine, but nothing special.'),

(10, 1, 4, 'The runner was nice and on time, thanks!');

```
SELECT * FROM runner_ratings;
```

RESULT:

	rating_id [PK] integer	order_id integer	runner_id integer	rating integer	comment text	rating_date timestamp without time zone
1	1	1	1	5	The runner was very prompt and friendly!	2024-09-29 04:33:09.870347
2	2	2	1	4	Great service, but a bit delayed.	2024-09-29 04:33:09.870347
3	3	3	2	5	Perfect delivery, thank you!	2024-09-29 04:33:09.870347
4	4	4	2	3	The delivery was okay, but the runner was not very polit...	2024-09-29 04:33:09.870347
5	5	5	3	4	Good service overall, will order again.	2024-09-29 04:33:09.870347
6	6	6	3	5	Outstanding delivery experience!	2024-09-29 04:33:09.870347
7	7	7	2	4	The runner was quick but forgot my drink.	2024-09-29 04:33:09.870347
8	8	8	2	5	Excellent, very satisfied with the service.	2024-09-29 04:33:09.870347
9	9	9	1	3	It was fine, but nothing special.	2024-09-29 04:33:09.870347
10	10	10	1	4	The runner was nice and on time, thanks!	2024-09-29 04:33:09.870347

-- 4 Using your newly generated table - can you join all of the information together to form a table

--which has the following information for successful deliveries?

--customer_id

--order_id

--runner_id

--rating

--order_time

--pickup_time

--Time between order and pickup

--Delivery duration

--Average speed

--Total number of pizzas

```
WITH delivery_info_cte AS

(

SELECT c.customer_id,

c.order_id,

r.runner_id,

rr.rating,

c.order_time,

r.pickup_time,

-- Calculate time between order and pickup

ROUND(EXTRACT(EPOCH FROM(r.pickup_time-c.order_time))/60,2)||' min' AS
time_between_order_and_pickup,

r.duration||' min' AS delivery_duration_in_minutes,

-- Calculate average speed assuming distance is in km

      CASE WHEN r.distance IS NOT NULL AND r.distance>0 THEN

            ROUND((r.distance::FLOAT/r.duration*60))||'km/hr'

END AS average_speed,

COUNT(c.pizza_id) AS total_pizzas

FROM

       customer_orders_clean c

   JOIN

       runner_orders_clean  r ON c.order_id = r.order_id

   LEFT JOIN

       runner_ratings rr ON c.order_id = rr.order_id

   WHERE

       r.cancellation IS NULL  -- Make sure the order is successful

   GROUP BY

       c.customer_id, c.order_id, r.runner_id, rr.rating,

                        c.order_time, r.pickup_time, r.duration, r.distance

)
```

```
SELECT *

FROM delivery_info_cte

ORDER BY order_id;  -- Order by order_id for better readability
```

RESULT:

	customer_id integer	order_id integer	runner_id integer	rating integer	order_time timestamp without time zone	pickup_time timestamp without time zone	time_between_order_and_pickup text
1	101	1	1	5	2020-01-01 18:05:02	2020-01-01 18:15:34	10.53 min
2	101	2	1	4	2020-01-01 19:00:52	2020-01-01 19:10:54	10.03 min
3	102	3	1	5	2020-01-02 23:51:23	2020-01-03 00:12:37	21.23 min
4	103	4	2	3	2020-01-04 13:23:46	2020-01-04 13:53:03	29.28 min
5	104	5	3	4	2020-01-08 21:00:29	2020-01-08 21:10:57	10.47 min
6	105	7	2	4	2020-01-08 21:20:29	2020-01-08 21:30:45	10.27 min
7	102	8	2	5	2020-01-09 23:54:33	2020-01-10 00:15:02	20.48 min
8	104	10	1	4	2020-01-11 18:34:49	2020-01-11 18:50:20	15.52 min

delivery_duration_in_minutes text	average_speed text	total_pizzas bigint
32 min	38km/hr	1
27 min	44km/hr	1
20 min	40km/hr	2
40 min	35km/hr	3
15 min	40km/hr	1
25 min	60km/hr	1
15 min	94km/hr	1
10 min	60km/hr	2

--5 If a Meat Lovers pizza was $12 and Vegetarian $10 fixed prices with no cost for extras and each runner is paid $0.30 per kilometre traveled - how much money does Pizza Runner have left over after these deliveries?

```
WITH pizza_sales AS (
  SELECT
    c.pizza_id,
    p.pizza_name,
    COUNT(*) AS pizza_count,
    CASE
      WHEN p.pizza_name = 'Meatlovers' THEN COUNT(*) * 12
      WHEN p.pizza_name = 'Vegetarian' THEN COUNT(*) * 10
    END AS total_revenue
  FROM customer_orders_clean c
  JOIN pizza_names p ON c.pizza_id = p.pizza_id
  GROUP BY c.pizza_id, p.pizza_name
),      runner_costs AS (
  SELECT
    order_id,
    runner_id,
    COALESCE( distance_in_km, 0)::NUMERIC AS distance_km
  FROM runner_orders_clean
  WHERE cancellation IS NULL  -- Only include non-canceled orders
)
SELECT
  (SELECT SUM(total_revenue) FROM pizza_sales) --revenue calculation
  -
  (SELECT SUM(distance_km) * 0.30 FROM runner_costs)--runner cost calculation
  AS profit_left_over;
```

RESULT:

	profit_left_over numeric
1	116.440

--E. Bonus Questions

--If Danny wants to expand his range of pizzas - how would this impact the existing data design?

--Write an INSERT statement to demonstrate what would happen if a new Supreme pizza

--with all the toppings was added to the Pizza Runner menu?

```
-- Insert into pizza_names
INSERT INTO pizza_names (pizza_id, pizza_name)
VALUES (3, 'Supreme');

-- Insert into pizza_recipes with all toppings
INSERT INTO pizza_recipes (pizza_id, toppings)
VALUES (3, '1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12');

SELECT * FROM pizza_names;
```

RESULT:

	pizza_id integer	pizza_name text
1	1	Meatlovers
2	2	Vegetarian
3	3	Supreme

```
SELECT * FROM pizza_recipes;
```

RESULT:

	pizza_id integer	toppings text
1	1	1, 2, 3, 4, 5, 6, 8, 10
2	2	4, 6, 7, 9, 11, 12
3	3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12