**-- A. Pizza Metrics**

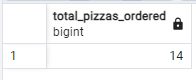
***--1 How many pizzas were ordered?***

SELECT

COUNT(\*) AS total\_pizzas\_ordered

FROM customer\_orders\_clean;

**RESULT:**



* 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:**

****

-- 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:**



***-- 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:**



***-- 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

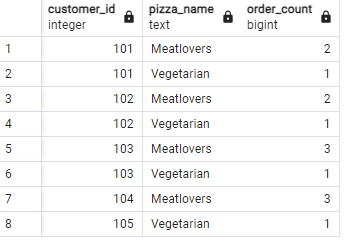
JOIN pizza\_names pn

ON c.pizza\_id=pn.pizza\_id

GROUP BY c.customer\_id,pn.pizza\_name

ORDER BY c.customer\_id

**RESULT:**



***-- 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:**

****

***-- 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

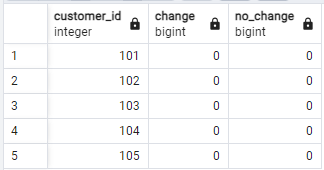
ON c.order\_id=r.order\_id

WHERE r.distance > 0

GROUP BY c.customer\_id

ORDER BY c.customer\_id;

**RESULT:**

****

***-- 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

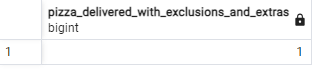
WHERE c.exclusions IS NOT NULL

AND c.exclusions <> ''

AND c.extras IS NOT NULL

AND c.extras <> '';

**RESULT:**



***-- 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,

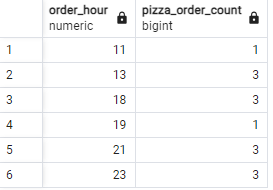
COUNT(c.pizza\_id) AS pizza\_order\_count -- CAN USE order\_id both produce same result

FROM customer\_orders\_clean c

GROUP BY order\_hour

ORDER BY order\_hour;

**RESULT:**



***-- 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,

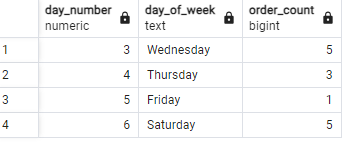
COUNT(c.order\_id) AS order\_count

FROM customer\_orders\_clean c

GROUP BY day\_number,day\_of\_week

ORDER BY day\_number;

**RESULT:**



**-- 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

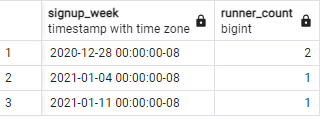
FROM runners

WHERE registration\_date>='2021-01-01'

GROUP BY signup\_week

ORDER BY signup\_week;

**RESULT:**



***-- 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

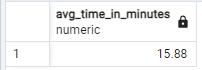
ORDER BY avg\_arrival\_in\_minutes

)

SELECT ROUND(avg(avg\_arrival\_in\_minutes),2) AS avg\_time\_in\_minutes

FROM minutes\_cte;

**RESULT:**



***--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,

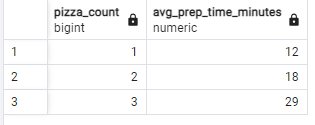
ROUND(AVG(preparation\_time\_in\_minutes)) AVG\_prep\_time\_minutes

FROM order\_preparation\_time\_cte

GROUP BY pizza\_count

ORDER BY pizza\_count;

**RESULT:**



-- 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

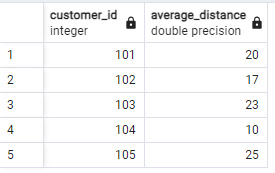
ON c.order\_id=r.order\_id

AND r.duration>0

GROUP BY c.customer\_id

ORDER BY c.customer\_id;

**RESULT:**



--Customer 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:**



--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

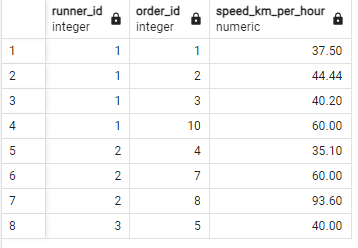
duration IS NOT NULL

ORDER BY

runner\_id,

order\_id;

**RESULT:**



*/\**(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

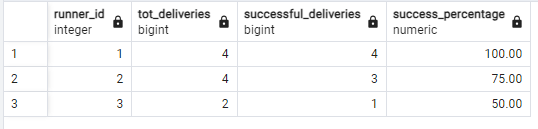
)AS success\_percentage

FROM runner\_orders\_clean

GROUP BY runner\_id

ORDER BY success\_percentage DESC;

**RESULT:**



/\*Runner 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:**



***-- 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

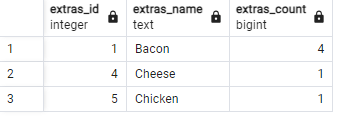
ON e.extras\_id = pt.topping\_id

GROUP BY e.extras\_id,pt.topping\_name

ORDER BY extras\_count DESC

--LIMIT 1;

**RESULT:**



--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

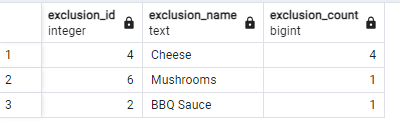
ON e.exclusion\_id = pt.topping\_id

GROUP BY e.exclusion\_id,pt.topping\_name

ORDER BY exclusion\_count DESC

--LIMIT 1;

**RESULT:**



--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:**



***-- 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

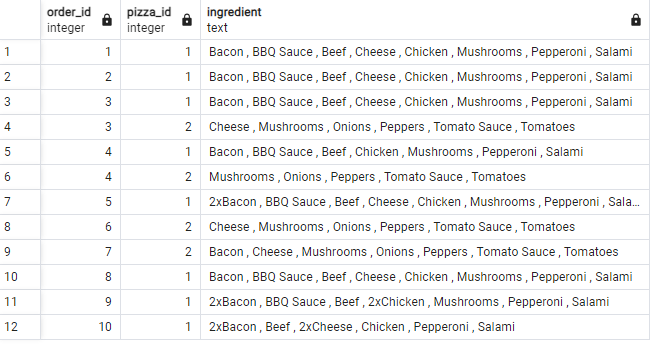
ELSE topping\_name

END,' , ') AS ingredient

FROM count\_topping\_cte

GROUP BY order\_id,pizza\_id;

**RESULT:**



***-- 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:**



**--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(

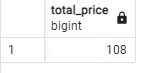
SELECT r.order\_id

FROM runner\_orders\_clean r

WHERE r.cancellation IS NOT NULL

);

**RESULT:**



***-- 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(

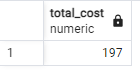
SELECT r.order\_id

FROM runner\_orders\_clean r

WHERE r.cancellation IS NOT NULL

);

**RESULT:**



***-- 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:**



***-- 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

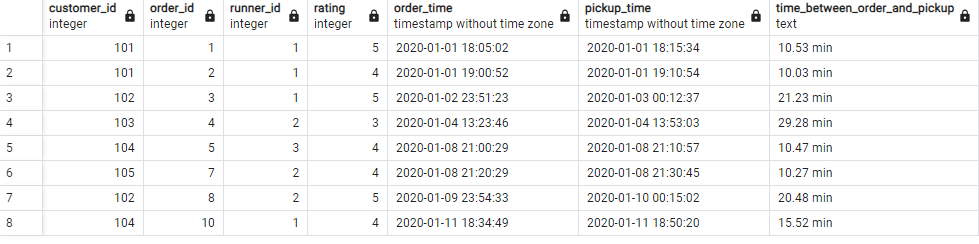
)

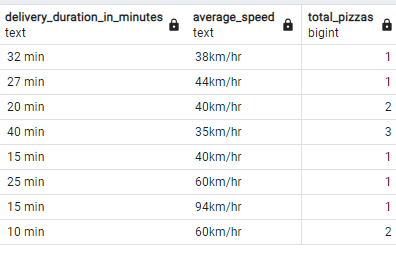
SELECT \*

FROM delivery\_info\_cte

ORDER BY order\_id; -- Order by order\_id for better readability

**RESULT:**





***--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:**



**--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');

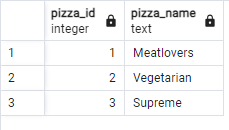
-- Insert into pizza\_recipes with all toppings

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:**

****

SELECT \* FROM pizza\_recipes;

**RESULT:**

