Dannys Diner Case Study Week 2 Analysis

Schema (PostgreSQL v13)

```
CREATE SCHEMA pizza_runner;
SET search_path = pizza_runner;

DROP TABLE IF EXISTS runners;
CREATE TABLE runners (
  "runner_id" INTEGER,
  "registration_date" DATE
);
INSERT INTO runners
  ("runner_id", "registration_date")
VALUES
  (1, '2021-01-01'),
  (2, '2021-01-03'),
  (3, '2021-01-08'),
  (4, '2021-01-15');



DROP TABLE IF EXISTS customer_orders;
CREATE TABLE customer_orders (
  "order_id" INTEGER,
  "customer_id" INTEGER,
  "pizza_id" INTEGER,
  "exclusions" VARCHAR(4),
  "extras" VARCHAR(4),
  "order_time" TIMESTAMP
);

INSERT INTO customer_orders
  ("order_id", "customer_id", "pizza_id", "exclusions", "extras", "order_time")
VALUES
  ('1', '101', '1', '', '', '2020-01-01 18:05:02'),
  ('2', '101', '1', '', '', '2020-01-01 19:00:52'),
  ('3', '102', '1', '', '', '2020-01-02 23:51:23'),
  ('3', '102', '2', '', NULL, '2020-01-02 23:51:23'),
```

```sql
  ('4', '103', '1', '4', '', '2020-01-04 13:23:46'),
  ('4', '103', '1', '4', '', '2020-01-04 13:23:46'),
  ('4', '103', '2', '4', '', '2020-01-04 13:23:46'),
  ('5', '104', '1', 'null', '1', '2020-01-08 21:00:29'),
  ('6', '101', '2', 'null', 'null', '2020-01-08 21:03:13'),
  ('7', '105', '2', 'null', '1', '2020-01-08 21:20:29'),
  ('8', '102', '1', 'null', 'null', '2020-01-09 23:54:33'),
  ('9', '103', '1', '4', '1, 5', '2020-01-10 11:22:59'),
  ('10', '104', '1', 'null', 'null', '2020-01-11 18:34:49'),
  ('10', '104', '1', '2, 6', '1, 4', '2020-01-11 18:34:49');


DROP TABLE IF EXISTS runner_orders;
CREATE TABLE runner_orders (
  "order_id" INTEGER,
  "runner_id" INTEGER,
  "pickup_time" VARCHAR(19),
  "distance" VARCHAR(7),
  "duration" VARCHAR(10),
  "cancellation" VARCHAR(23)
);

INSERT INTO runner_orders
  ("order_id", "runner_id", "pickup_time", "distance", "duration", "cancellation")
VALUES
  ('1', '1', '2020-01-01 18:15:34', '20km', '32 minutes', ''),
  ('2', '1', '2020-01-01 19:10:54', '20km', '27 minutes', ''),
  ('3', '1', '2020-01-03 00:12:37', '13.4km', '20 mins', NULL),
  ('4', '2', '2020-01-04 13:53:03', '23.4', '40', NULL),
  ('5', '3', '2020-01-08 21:10:57', '10', '15', NULL),
  ('6', '3', 'null', 'null', 'null', 'Restaurant Cancellation'),
  ('7', '2', '2020-01-08 21:30:45', '25km', '25mins', 'null'),
  ('8', '2', '2020-01-10 00:15:02', '23.4 km', '15 minute', 'null'),
  ('9', '2', 'null', 'null', 'null', 'Customer Cancellation'),
  ('10', '1', '2020-01-11 18:50:20', '10km', '10minutes', 'null');


DROP TABLE IF EXISTS pizza_names;
CREATE TABLE pizza_names (
  "pizza_id" INTEGER,
  "pizza_name" TEXT
);
INSERT INTO pizza_names
  ("pizza_id", "pizza_name")
VALUES
  (1, 'Meatlovers'),
  (2, 'Vegetarian');


DROP TABLE IF EXISTS pizza_recipes;
CREATE TABLE pizza_recipes (
  "pizza_id" INTEGER,
  "toppings" TEXT
);
INSERT INTO pizza_recipes
  ("pizza_id", "toppings")
VALUES
  (1, '1, 2, 3, 4, 5, 6, 8, 10'),
  (2, '4, 6, 7, 9, 11, 12');


DROP TABLE IF EXISTS pizza_toppings;
CREATE TABLE pizza_toppings (
  "topping_id" INTEGER,
  "topping_name" TEXT
);
INSERT INTO pizza_toppings
  ("topping_id", "topping_name")
VALUES
  (1, 'Bacon'),
  (2, 'BBQ Sauce'),
  (3, 'Beef'),
  (4, 'Cheese'),
  (5, 'Chicken'),
  (6, 'Mushrooms'),
  (7, 'Onions'),
  (8, 'Pepperoni'),
  (9, 'Peppers'),
  (10, 'Salami'),
  (11, 'Tomatoes'),
  (12, 'Tomato Sauce');
```

CLEANING

Query #2

```
SET search_path = pizza_runner;
```

Purpose: Sets the schema search path to "pizza_runner" to ensure that subsequent queries operate within this schema.

Query #3

```
UPDATE customer_orders
SET
    exclusions = NULLIF(NULLIF(exclusions, ''), 'NaN'),
    extras = NULLIF(NULLIF(extras, ''), 'NaN');
```

Purpose: Replaces empty strings and 'NaN' values in the `exclusions` and `extras` columns with NULL.

Query #4

```
UPDATE runner_orders
SET
    pickup_time = NULLIF(pickup_time, 'NaN'),
    distance = NULLIF(distance, 'NaN'),
    duration = NULLIF(duration, 'NaN'),
    cancellation = NULLIF(NULLIF(cancellation, ''), 'NaN');
UPDATE runner_orders
SET
    pickup_time = NULLIF(pickup_time, 'null'),
    distance = NULLIF(regexp_replace(distance, '[^0-9.]', '', 'g'), '')::NUMERIC,
    duration = NULLIF(regexp_replace(duration, '[^0-9]', '', 'g'), '')::INTEGER,
    cancellation = NULLIF(NULLIF(cancellation, ''), 'null')::VARCHAR;
```

Purpose: Handles various cases of NULL, 'NaN', and 'null' values in the `pickup_time`, `distance`, `duration`, and `cancellation` columns of the `runner_orders` table.

Query #5

```
UPDATE runner_orders
SET pickup_time = NULLIF(pickup_time, 'null')
ALTER TABLE runner_orders
ALTER COLUMN pickup_time TYPE TIMESTAMP USING to_timestamp(pickup_time, 'YYYY-MM-DD
HH24:MI:SS');
```

Purpose: Replaces 'null' values in the `pickup_time` column with NULL and updates the format of `pickup_time` to a TIMESTAMP.

Query #6

```
UPDATE runner_orders
SET distance = NULLIF(regexp_replace(distance, '[^0-9.]', '', 'g'), '')::NUMERIC;
UPDATE runner_orders
SET duration = NULLIF(regexp_replace(duration, '[^0-9]', '', 'g'), '')::INTEGER;
ALTER TABLE runner_orders
RENAME COLUMN distance TO distance_kms;
ALTER TABLE runner_orders
RENAME COLUMN duration TO duration_mins;
```

Purpose: Cleans and transforms the `distance` and `duration` columns in the `runner_orders` table. Renames the columns for clarity.

Query #7

```
SELECT * FROM runner_orders;
```

order_id	runner_id	pickup_time	distance_kms	duration_mins	cancellation
1	1	2020-01-01T18:15:34.000Z	20	32	
2	1	2020-01-01T19:10:54.000Z	20	27	
3	1	2020-01-03T00:12:37.000Z	13.4	20	
4	2	2020-01-04T13:53:03.000Z	23.4	40	
5	3	2020-01-08T21:10:57.000Z	10	15	
6	3				Restaurant Cancellation
7	2	2020-01-08T21:30:45.000Z	25	25	
8	2	2020-01-10T00:15:02.000Z	23.4	15	
9	2				Customer Cancellation
10	1	2020-01-11T18:50:20.000Z	10	10	

Query #8

```
SELECT * FROM customer_orders;
```

order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1			2020-01-01T18:05:02.000Z
2	101	1			2020-01-01T19:00:52.000Z
3	102	1			2020-01-02T23:51:23.000Z
3	102	2			2020-01-02T23:51:23.000Z
4	103	1	4		2020-01-04T13:23:46.000Z
4	103	1	4		2020-01-04T13:23:46.000Z
4	103	2	4		2020-01-04T13:23:46.000Z
5	104	1	null	1	2020-01-08T21:00:29.000Z

order_id	customer_id	pizza_id	exclusions	extras	order_time
6	101	2	null	null	2020-01-08T21:03:13.000Z
7	105	2	null	1	2020-01-08T21:20:29.000Z
8	102	1	null	null	2020-01-09T23:54:33.000Z
9	103	1	4	1, 5	2020-01-10T11:22:59.000Z
10	104	1	null	null	2020-01-11T18:34:49.000Z
10	104	1	2, 6	1, 4	2020-01-11T18:34:49.000Z

CASE STUDY QUESTIONS

Query #9

```
SELECT COUNT(co.pizza_id) AS non_distinct_pizza_ids
    FROM customer_orders co
    WHERE co.order_id IN (
        SELECT ro.order_id
        FROM runner_orders ro
        WHERE COALESCE(ro.cancellation, '') = ''
    );
```

non_distinct_pizza_ids
12

- **Q1: Count of Non-Distinct Pizza IDs in Successful Orders**
    - **Question:** How many non-distinct pizza IDs were delivered successfully?
    - **Logic:** Counts the pizza IDs in the customer_orders table where the associated runner_orders have no cancellations.

Query #10

```
SELECT COUNT(DISTINCT order_id) AS unique_customer_orders
FROM customer_orders;
```

unique_customer_orders
10

- **Q2: Count of Unique Customer Orders**
  - **Question:** How many unique customer orders were made?
  - **Logic:** Counts the distinct order IDs in the customer_orders table.

Query #11

```
SELECT
    ro.runner_id,
    COUNT(DISTINCT co.order_id) AS successful_orders_count
FROM
    runner_orders ro
JOIN
    customer_orders co ON ro.order_id = co.order_id
WHERE
    COALESCE(ro.cancellation, '') = ''
GROUP BY
    ro.runner_id;
```

runner_id	successful_orders_count
1	4
2	3
3	1

- **Q3: Successful Orders Count for Each Runner**
  - **Question:** How many successful orders were made by each runner?
  - **Logic:** Counts the distinct customer order IDs for each runner where the associated runner_orders have no cancellations.

Query #12

```
SELECT
    pn.pizza_name,
    COUNT(*) AS pizza_count
FROM
    customer_orders co
JOIN
    pizza_names pn ON co.pizza_id = pn.pizza_id
JOIN
    runner_orders ro ON co.order_id = ro.order_id
WHERE
    COALESCE(ro.cancellation, '') = ''
GROUP BY
    pn.pizza_name;
```

pizza_name	pizza_count
Meatlovers	9
Vegetarian	3

- **Q4 and Q5: Pizza Count for Each Pizza Name**
    - **Question:** How many of each type of pizza was delivered?
    - **Logic:** Counts the occurrences of each pizza name in the customer_orders table, considering only successful orders.

Query #13

```
SELECT
    co.order_id,
    COUNT(co.pizza_id) AS num_pizzas_delivered
FROM
    customer_orders co
JOIN
    runner_orders ro ON co.order_id = ro.order_id
WHERE
    COALESCE(ro.cancellation, '') = ''
GROUP BY
    co.order_id
ORDER BY
    num_pizzas_delivered DESC
LIMIT 1;
```

order_id	num_pizzas_delivered
4	3

- **Q6: Number of Pizzas Delivered in a Single Order**
  - **Question:** What was the maximum number of pizzas delivered in a single order?
  - **Logic:** Counts the number of pizzas delivered for each order and retrieves the order with the maximum count.

Query #14

```
SELECT
    co.customer_id,
    co.order_id,
    co.pizza_id,
    MAX(CASE WHEN co.exclusions IS NOT NULL OR co.extras IS NOT NULL THEN 1 ELSE 0
END) AS pizzas_with_changes,
    MAX(CASE WHEN co.exclusions IS NULL AND co.extras IS NULL THEN 1 ELSE 0 END) AS
pizzas_no_changes
FROM
    customer_orders co
JOIN
    runner_orders ro ON co.order_id = ro.order_id
WHERE
    COALESCE(ro.cancellation, '') = ''
GROUP BY
    co.customer_id, co.order_id, co.pizza_id
ORDER BY
    co.customer_id, co.order_id, co.pizza_id;
```

customer_id	order_id	pizza_id	pizzas_with_changes	pizzas_no_changes
101	1	1	0	1
101	2	1	0	1
102	3	1	0	1
102	3	2	0	1
102	8	1	1	0
103	4	1	1	0

customer_id	order_id	pizza_id	pizzas_with_changes	pizzas_no_changes
103	4	2	1	0
104	5	1	1	0
104	10	1	1	0
105	7	2	1	0

- **Q7: Pizzas with Changes and Pizzas with No Changes for Each Customer**
    - **Question:** For each customer, how many delivered pizzas had at least one change and how many had no changes?
    - **Logic:** Identifies pizzas with changes (non-null exclusions or extras) and pizzas with no changes for each customer.

Query #15

```
SELECT
    COUNT(DISTINCT co.pizza_id) AS pizzas_with_exclusions_and_extras
FROM
    customer_orders co
JOIN
    runner_orders ro ON co.order_id = ro.order_id
WHERE
    COALESCE(ro.cancellation, '') = ''
    AND co.exclusions IS NOT NULL
    AND co.extras IS NOT NULL;
```

pizzas_with_exclusions_and_extras
2

- **Q8: Pizzas with Both Exclusions and Extras**
    - **Question:** How many pizzas were delivered that had both exclusions and extras?
    - **Logic:** Counts the pizzas in customer_orders where both exclusions and extras are not null.

Query #16

```
SELECT
    EXTRACT(HOUR FROM co.order_time) AS order_hour,
    COUNT(*) AS total_pizzas_ordered
FROM
    customer_orders co
JOIN
    runner_orders ro ON co.order_id = ro.order_id
WHERE
    COALESCE(ro.cancellation, '') = ''
GROUP BY
    order_hour
ORDER BY
    order_hour;
```

order_hour	total_pizzas_ordered
13	3
18	3
19	1
21	2
23	3

- **Q9: Total Pizzas Ordered for Each Hour of the Day**
    - **Question:** What was the total volume of pizzas ordered for each hour of the day?
    - **Logic:** Groups orders by the hour of the day and counts the total pizzas ordered.

Query #17

```
SELECT
    EXTRACT(DOW FROM co.order_time) AS day_of_week,
    COUNT(DISTINCT co.order_id) AS orders_count
FROM
    customer_orders co
JOIN
    runner_orders ro ON co.order_id = ro.order_id
WHERE
    COALESCE(ro.cancellation, '') = ''
```

```
GROUP BY
    day_of_week
ORDER BY
    day_of_week;
```

day_of_week	orders_count
3	4
4	2
6	2

- **Q10: Orders Count for Each Day of the Week**
  - **Question:** What was the volume of orders for each day of the week?
  - **Logic:** Groups orders by the day of the week and counts the distinct order IDs.

Query #18

```
SELECT
    date_trunc('week', registration_date) AS week_start,
    COUNT(*) AS new_runners_count
FROM
    pizza_runner.runners
GROUP BY
    week_start
ORDER BY
    week_start;
```

week_start	new_runners_count
2020-12-28T00:00:00.000Z	2
2021-01-04T00:00:00.000Z	1
2021-01-11T00:00:00.000Z	1

- **Q11: New Runners Count for Each 1-Week Period**
  - **Question:** How many runners signed up for each 1-week period?
  - **Logic:** Groups runners by week of registration and counts the new runners for each period.

Query #19

```
SELECT
    runner_id,
    AVG(duration_mins::numeric) AS average_pickup_time
FROM
    runner_orders
WHERE
    pickup_time IS NOT NULL
    AND COALESCE(cancellation, '') = ''
GROUP BY
    runner_id
ORDER BY
    runner_id;
```

runner_id	average_pickup_time
1	22.2500000000000000
2	26.6666666666666667
3	15.0000000000000000

- **Q12: Average Pickup Time for Each Runner**
  - **Question:** What was the average time in minutes it took for each runner to arrive at the Pizza Runner HQ to pick up the order?
  - **Logic:** Calculates the average pickup time for each runner, excluding null pickup times and canceled orders.

Query #20

```
SELECT
    COUNT(co.pizza_id) AS number_of_pizzas,
    ro.duration_mins AS preparation_time
FROM
    customer_orders co
JOIN
    runner_orders ro ON co.order_id = ro.order_id
WHERE
    COALESCE(ro.cancellation, '') = ''
```

```
GROUP BY
    ro.duration_mins;
```

number_of_pizzas	preparation_time
2	10
1	27
1	32
2	15
3	40
2	20
1	25

- **Q13: Number of Pizzas vs. Preparation Time**
  - **Question:** Is there any relationship between the number of pizzas and how long the order takes to prepare?
  - **Logic:** Counts the number of pizzas for each preparation time.

Query #21

```
SELECT
    co.customer_id,
    AVG(ro.distance_kms::numeric) AS average_distance
FROM
    customer_orders co
JOIN
    runner_orders ro ON co.order_id = ro.order_id
WHERE
    COALESCE(ro.cancellation, '') = ''
    AND ro.distance_kms IS NOT NULL
GROUP BY
    co.customer_id
ORDER BY
    co.customer_id;
```

customer_id	average_distance
101	20.0000000000000000

customer_id	average_distance
102	16.7333333333333333
103	23.4000000000000000
104	10.0000000000000000
105	25.0000000000000000

- **Q14: Average Distance for Each Customer**
  - **Question:** What was the average distance travelled for each customer?
  - **Logic:** Calculates the average distance for each customer, excluding null distances and canceled orders.

Query #22

```
SELECT
    MAX(ro.duration_mins::numeric) - MIN(ro.duration_mins::numeric) AS
delivery_time_difference
FROM
    runner_orders ro
WHERE
    COALESCE(ro.cancellation, '') = ''
    AND ro.duration_mins IS NOT NULL;
```

delivery_time_difference
30

- **Q15: Difference Between Longest and Shortest Delivery Times**
  - **Question:** What was the difference between the longest and shortest delivery times for all orders?
  - **Logic:** Calculates the difference between the longest and shortest delivery times, excluding null durations and canceled orders.

Query #23

```
SELECT
    ro.runner_id,
    ro.order_id,
    ro.distance_kms,
    ro.duration_mins,
    CASE
        WHEN ro.duration_mins::numeric > 0 THEN ro.distance_kms::numeric /
ro.duration_mins::numeric
        ELSE NULL
    END AS average_speed
FROM
    runner_orders ro
WHERE
    COALESCE(ro.cancellation, '') = ''
    AND ro.distance_kms IS NOT NULL
    AND ro.duration_mins IS NOT NULL;
```

runner_id	order_id	distance_kms	duration_mins	average_speed
1	1	20	32	0.62500000000000000000
1	2	20	27	0.74074074074074074074
1	3	13.4	20	0.67000000000000000000
2	4	23.4	40	0.58500000000000000000
3	5	10	15	0.66666666666666666667
2	7	25	25	1.00000000000000000000
2	8	23.4	15	1.5600000000000000
1	10	10	10	1.00000000000000000000

- **Q16: Average Speed for Each Runner for Each Delivery**
  - **Question:** What was the average speed for each runner for each delivery, and do you notice any trends for these values?
  - **Logic:** Calculates the average speed for each runner for each delivered order, excluding null distances and durations.

Query #24

```
SELECT
    ro.runner_id,
    COUNT(*) AS total_deliveries,
    SUM(CASE WHEN COALESCE(ro.cancellation, '') = '' THEN 1 ELSE 0 END) AS
successful_deliveries,
    (SUM(CASE WHEN COALESCE(ro.cancellation, '') = '' THEN 1 ELSE 0 END) * 100.0 /
COUNT(*)) AS success_percentage
FROM
    runner_orders ro
GROUP BY
    ro.runner_id;
```

runner_id	total_deliveries	successful_deliveries	success_percentage
3	2	1	50.0000000000000000
2	4	3	75.0000000000000000
1	4	4	100.0000000000000000

- **Q17: Successful Delivery Percentage for Each Runner**
  - **Question:** What is the successful delivery percentage for each runner?
  - **Logic:** Calculates the total deliveries, successful deliveries, and success percentage for each runner.