



Mojtaba Peyrovi

Data Analytics Team Lead

DATA ROLE ASSESSMENT

Content:

- Introduction Page 1-2
 - Format & tools
 - What is included in the shared folder?
 - Naming convention
 - How to follow this research more efficiently?
 - My first impressions/insight from the source data
- Task One Page 3-7
 - Question 1
 - Approach
 - Improvement area
 - Test cases
 - Question 2
 - Approach
 - Test cases
- Task two Page 7-9
 - Approach
 - Table join
 - Custom date dimensions
 - Grain of the dataset
 - Caveat for the analyst(s)
- Final questions..... Page 9-11



Wolt's mission is to make cities better places for customers, merchants and couriers alike. Wolt's platform makes it easy for customers to order whatever they need on one app, for merchants to make additional sales, and for couriers to make meaningful earnings flexibly.

ABOUT MOJTABA: Passionately seeking growth opportunities with proven impact-oriented mindset and influence. An engineer, who has spent several years working with various forms of data, and is thirsty for meaningful insight & analyses, and along the way acquires and learns the most suitable technologies and practices to provide useful solutions. A product minded engineer who has learned to FOCUS ON WHAT REALLY MATTERS to the business growth.

1. Introduction

Dear hiring manager/Team,

Thank you for giving me the opportunity to work on the assessment. I enjoyed working on it so much. I hope this document is close enough to the expectations, although it can definitely be improved and I even get more excited to be given more time to take this project from here and start improving it, making it more cost effective, and pulling out more insight from it.

Before getting into the main task, I believe it will be helpful to share the following points:

Format & tools: The SQL code is written in Postgres database system V15.2.2 and PgAdmin 4 V6.21 is used as the GUI. There are three SQL codes and each one of them returns a table which is stored as a view.

What is included in the shared folder?

- Three SQL scripts to answer all questions related to task 1 and task 2
- A SQL script including all test cases I wrote during the analysis
-

Naming convention: The source data comes in uppercase snake case (e.g. PURCHASE_ID). Any metrics, table generated by me, uses lowercase snake case (e.g. day_of_month_reduced)

How to follow this research more efficiently?

- Use any Postgres engine either cloud based or have an installation on your machine and use localhost to host our sample database.
- The database tables “purchases” and “delivery_radius_log” should be created
- The tables to be seeded by the source data
- The three SQL files to be executed in the following order:
 - question11.sql
 - question12.sql
 - question2.sql

If the above steps are followed properly, we should expect the following three views generated:

1- default_reduces: This table is the answer to the first question of Task 1.

2- radius_change_comparison: This view is used to answer the second question of Task 1, and also to save costs, it is used to generate the dataset for Task 2. Please note that this dataset is not the answer to question 2 of Task 1, but it is used in my query to answer this question located at the second part of the query question12.sql.

3- preprocessed_table_for_question_2: This view is generated as required in the assignment, to be able to answer the three questions mentioned in the Task 2.

My first impressions/insight from the source data:

1- The date range of the datasets don't fully overlap. As shown below, the purchases recorded after 2020-12-24 (7970 purchase records) have no delivery radius information. In this case, my caveat to the analyst would be, the total missed revenue value will not be accurate because it does not encompass the 7970 records.

	dataset text	start_date date	end_date date
1	purchases	2020-01-01	2020-12-31
2	delivery radius log	2020-01-01	2020-12-24

2- I assume the metric named DROPOFF_DISTANCE_STRAIGHT_LINE_METRES in the purchases table is comparable to DELIVERY_RADIUS_METERS from the delivery_radius_log. In this study, anytime the drop off distance goes bigger than the delivery radius, I assumed it falls outside the range and we missed the revenue. The complete conditions will be explained in the next pages.

2. Task One

Question 1: What are the default delivery radiuses for the timeframe provided?

Approach: Since the word “Stable over time” is used to describe the default radiuses, I interpreted it as “the most frequent” radiuses and based on this interpretation, I wrote a query to show me the most frequently used radiuses. The table below shows the results of the query and the query is saved as “question11.sql” file and is already shared in the folder.

	delivery_radius_meters integer	frequency bigint	percent_of_total numeric	radius_status text
1	3500	124	0.46	Default
2	10000	78	0.29	Default
3	6500	58	0.21	Default
4	3000	6	0.02	Temporary
5	2500	4	0.01	Temporary
6	4000	1	0.00	Temporary
7	4500	1	0.00	Temporary

For this assessment, I came up with an arbitrary number of 10% to be the breakpoint between default and temporary radiuses. Of course, it is not the best practice to hardcode such a value and it may return wrong analysis using different datasets, , but since no more information or criteria were shared, I had to settle for this value.



Improvement area: Some alternatives would be finding out if the company has a strict breakpoint predefined, or if this number is floating, finding out if the business specifies the default radius based on other metrics such as the sales amount, the city, the neighborhood, etc. and if a combination of these are being used, we might need ML to define this value based on different conditions in different datasets.

Test cases: In order to make sure we didn’t miss any data while generating the above table, I wrote two quick test cases. (It might sound unnecessary since it was a simple query, but I still include it here to demonstrate my thinking process).

Case 1: sum of the field “frequency” in the target table is equal to the total number of rows in the source table “delivery_radius_log”. The following table shows the test results.

	source_vs_target text	row_count numeric
1	source	272
2	target	272

Case 2: all radius values in the target table are present in the source table and vice versa. We are expecting to see the distinct radius values in both tables equally. The following table shows the test results.

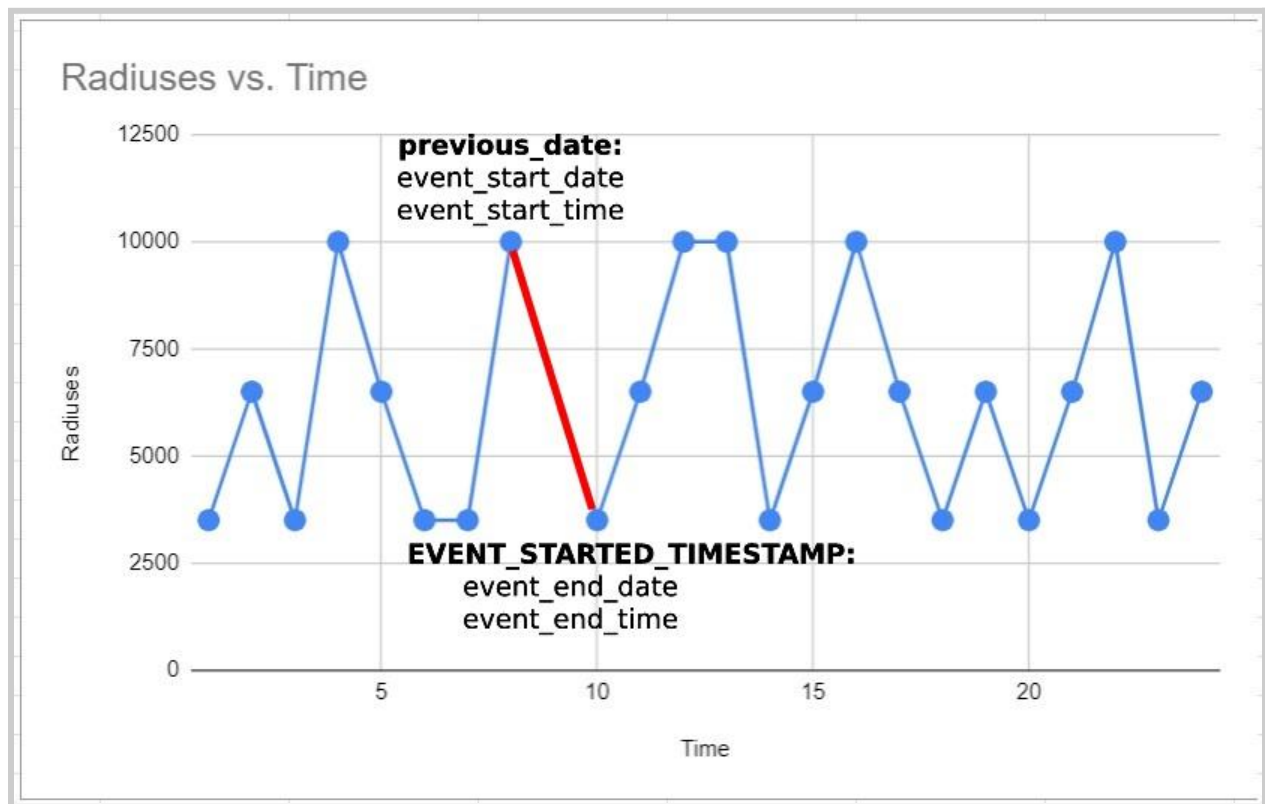
	source integer 	target integer 
1	3500	3500
2	10000	10000
3	6500	6500
4	3000	3000
5	2500	2500
6	4000	4000
7	4500	4500

Question 2: How many hours of radius reductions from the default radiuses have we had during the timeframe provided?

Approach: So far, we understood what radius values are default and which ones are temporary noises. Having the table generated in the previous questions, we can join it to the main delivery_radius_log to tag the radius values. This tagging is needed for our analysis because the assignment specifies *“we would like to understand what the “default” delivery radius is for a given period”*.

Next, I used the window function “LAG” to put the previous “radius” and “change date” values and having the current and previous values, we can specify whether we had a reduction or expansion, and how long the reduction was continued in minutes. The SQL code to generate such a view named “radius_change_comparison”, and query it to answer question 1.2 are stored in the file “question12.sql”.

We also extracted date and time from “EVENT_STARTED_TIMESTAMP” and “previous_date” and the naming convention is show below:



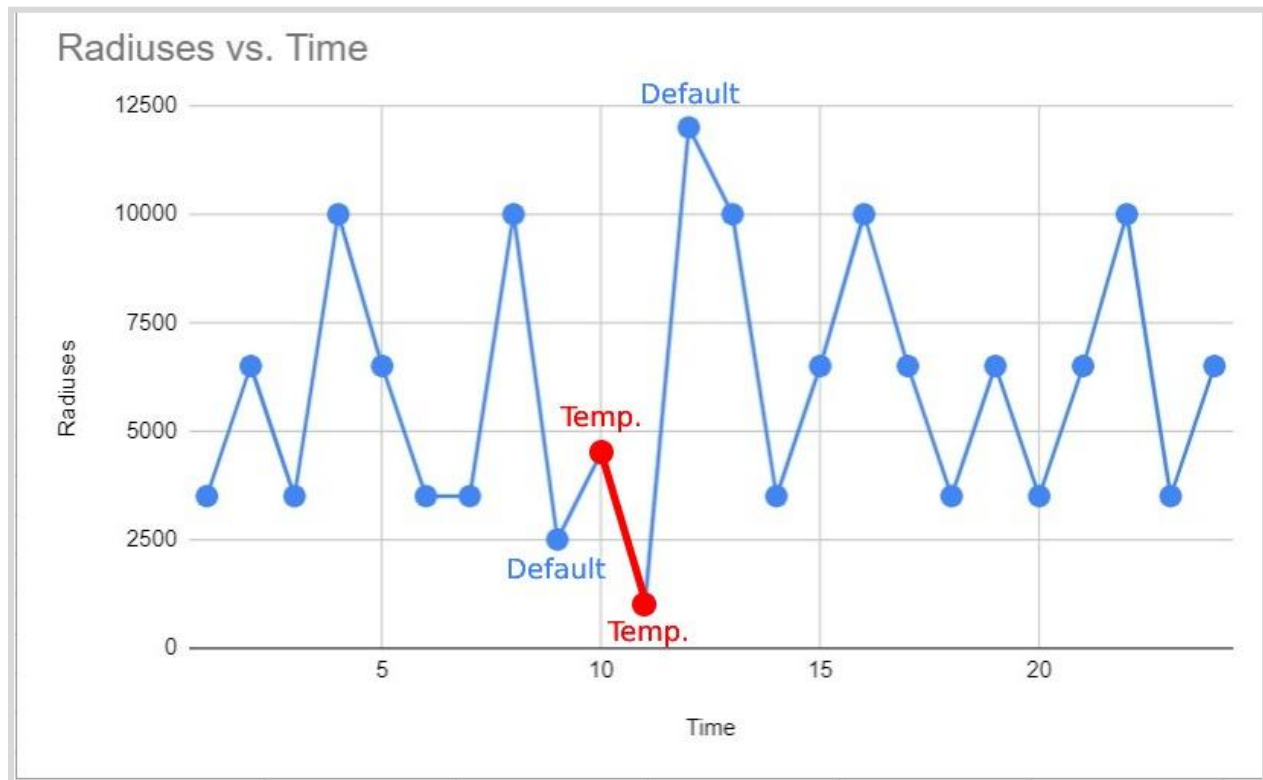
Any time in the graph above, the slope is -1, we have a radius reduction.

My logic for selecting changes from default radius only:

According the table below, among all four possible cases, only in one case we exclude the results from the query:

previous radius (start of reduction)	current radius (end of reduction)	Include/exclude in the query?
Default	Default	Include
Default	Temporary	Include
Temporary	Default	Exclude
Temporary	Temporary	Include

The reason I include Temporary/Temporary also part of the deduction from default is, in case we have multiple temporary changes, at the end they will fall between two defaults, and in this case we need to include them. Although in this dataset we don't have such a case, because it was advised in the assignment description that the query should perform well for any dataset, this can cover that corner case. The photo below shows this corner case in red.



The query results below show the answer to question 1.2 is 7336 hours and 10 minutes of reduction from default radiuses.

	total_mins numeric	hours numeric	minutes numeric
1	440170	7336	10

Test cases: For this table I didn't write any test because the process was straightforward. But in case of having additional time, we can write some tests before updating the table in the ETL layer. For example, we can insert a row into the source with a specific radius (having in mind any corner case such as 0, minus values, or extraordinarily big radiuses) and check if our query correctly extracts the slope (-1, 0, or 1) and if the test fails, we can rollback the update. But I feel it is out of the scope of this research and spending resources on it might not be feasible and highly impactful.

3. Task Two

Please combine the delivery radius dataset with purchases data and design a dataset that can be used by data analysts to answer at least the following questions:

- When do we usually have radius reductions?
- How long do the radius reductions last?
- How much potential revenue (End Amount With VAT Eur) was lost due to delivery radius reductions?

Approach: The first two questions are related to the first dataset (delivery_radius_log) but we need to join it with the “purchases” dataset to be able to tell how much missing revenue has happened due to the radius reduction. Based on my understanding of both datasets, I considered a revenue missing if the below conditions happen:

- The purchase happens between a reduction period (slope = -1)
- Since the radius reduction is not required to be only from Default, we have to include all radiuses
- The distance between the shop and the dropoff point falls outside the radius for each purchase. It means the metric DROPOFF_DISTANCE_STRAIGHT_LINE_METERS should be bigger than DELIVERY_RADIUS_METER while the reduction is happening.
- Because the missing delivery must happen because of a specific reduction, if the dropoff point falls between the previous and current radius, it can be considered a revenue missing condition. It means DROPOFF_DISTANCE_STRAIGHT_LINE_METERS should be bigger than DELIVERY_RADIUS_METER and smaller than “previous_date”.
- Corner case: How about any value bigger than the previous date? It is a revenue miss but not due to this reduction because even if the reduction didn't happen it would have been out of the range for the previous_date anyways. Therefore, we didn't miss the revenue because of the reduction. The below graph shows it:

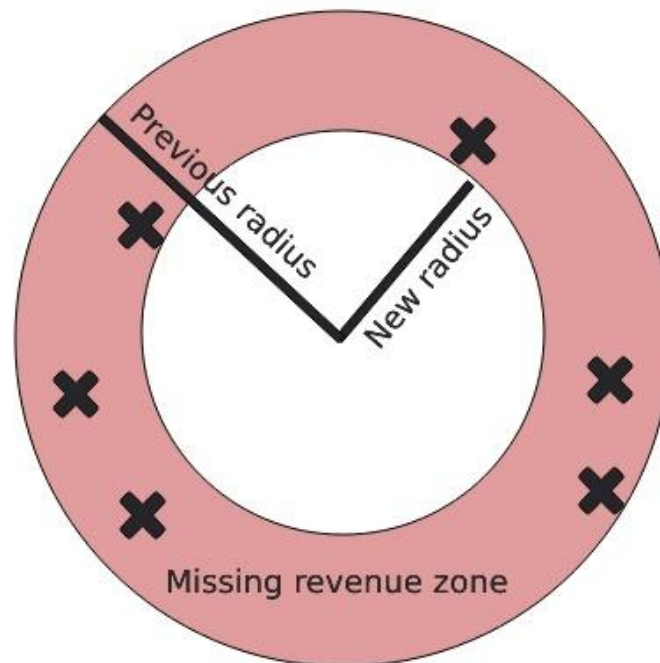


Table join - Date conditions:

- If the purchase date (date_received extracted from TIME_RECEIVED) fall between the start and end of a reduction. Because for this case there are no time conditions, I used an inner join.
- If the purchase date happens on the same day as the start date of the reduction. If the purchase date happens on the same day as the end date of the reduction

For each of these cases created a dataset and at the end, for each one of them added the time condition, and at the end concatenated all three together in order to generate the final dataset.

Please note that for the second and third joints, I used left join to guarantee all revenues are present in the final dataset from the purchases table (left table)

Table join - Time conditions:

- If the order date is between the start and end date of the reduction, no matter what time it happens, it will already be a revenue loss.
- If the order is received on the same day as the reduction start, the time should be after the reduction start time.
- If the order is received on the same day as the reduction end, the time should be before the reduction end time.

At the end, based on the following three conditions, I tagged some purchases as missed revenue.

Custom date dimensions:

I extracted some data from the EVENT START DATETIME. Of course, it is not ideal to have all these dimensions added directly to such a dataset and usually we handle these date-related dimensions inside the DimDate table in a DWH. These dimensions are extracted just for the ease of access to answer question 2.1

Grain of the dataset:

Because calculating the revenue loss depends on the minutes of reduction change as the lowest level (ignoring seconds), we need to keep the grain of the dataset as the most atomic grain which is time of reduction.

Caveat for the analyst(s): Because of the atomic granularity of the dataset, the cardinality of the dataset is increased which means in order for them to answer question 2.1 and 2.2, using the generated dataset, they cannot simply count the reduction hours from this dataset in the lowest grain level. They need to find a way to roll-up the data to an aggregated level, then after knowing there is no duplicated data from the first dataset (delivery_radius_log) they should be able to count the length of the reduction and the count of the reduction cases.

4. Final questions

What assumptions about the data have you made to produce the dataset(s)? Why did you decide to go with this particular approach and what could be the pros and cons of applying it?

This question was answered on page 3 and during the presentation whenever needed. My approach and reasons are:

1- I use CTE instead of subqueries for most cases. These are a few reasons for it:

- CTE is more readable, it is more self-explanatory, and easier to debug.
- CTE can be used multiple times while subquery should be written anytime we need to use the same table.
- With CTE we can encapsulate a specific logic and tackle it separately from other parts of the analysis

2- Because the results of some questions were needed to answer the next questions, I decided to store some tables as views to be able to use them in the next parts and avoid duplicate work and save costs. (manpower, cloud operation cost, opportunity cost)

3- I tried to make the code the most readable and use clear variable names to reduce time for the next developer to understand and debug the code. Also, added comments on the code and tried to use consistent naming convention.

What strategy would you use for updating the dataset from task 2? Consider how often the dataset should be updated, do we need to truncate the table before updating etc.

The update frequency decision can be made when a reasonable understanding of the business user behavior and the cloud costs are in hand but after we decide on the refreshment frequency, we will have two routes ahead.

Assuming the user needs to get the data in real time and we have a sensitivity to update the data on a real time basis, we can use realtime pipeline technologies such as Apache Kafka and event stream processing to automatically make the transformation and update the final dataset in real-time.

But If we need a standard batch processing pipeline, we can use cron based pipelines and we can either truncate the whole table before each update or find the delta and only concatenate the delta to the target table. Each approach has its own pros and cons and also depends on the size of the dataset and the refreshing cost. There is always a trade-off between whether we want to have a simple approach of truncating the table and rewriting it and paying some extra fee versus making a more complex script to find the delta and concatenate it to the bottom, which may involve more engineering work to develop and maintain, and eventually be more expensive in some cases. For this table because the raw data don't come with any natural key as the primary key, and the target table is generated in a denormalized way, and the table is relatively small (< 1M rows), I believe truncating the table and overwriting it is simpler since we don't have to deal with generating primary keys for each table and a surrogate key for the final table and getting into data warehousing techniques.

In case of any data cleanup, preprocessing, and quality checks needed in the source before updating the views, we should consider creating a temporary dataset, cleaning it up and running tests against the temporary table, then executing the update code only if it passed the tests. This way, our production tables which are feeding our reports never get overwritten by incorrect data.

In terms of technology, we can either write our own stored procedures to run tests, updates, triggers, etc. or use a third party BI solution such as Informatica, Talend, etc. which also comes with a trade-off and a management decision but usually I vote for BI tools because they usually are so much simpler to use, a lot cheaper than hiring expensive engineers to code to maintain the pipelines, and many functionalities like data quality testing, version control, and deployment are included in those solutions usually.

How could the solution be improved if given more time and data?

- If I had time, I would have found another more intuitive way of finding the default radius values vs the temporary ones. Currently I used an arbitrary value of 10% of frequency as a breakpoint but we might find datasets where a default value is under 10% but can be considered a default radius. e.g. Maybe the company has a specific value activated for some special occasions such as the Christmas period which has a very low frequency (once a year) but is considered default.
- For each CTE, instead of using *, I would only bring in the fields that are needed for the next step to reduce the amount of data to be analyzed and make the query faster and cheaper.
- I would have written more test cases to have an automated way of testing all my scripts. I currently wrote some tests only for the first dataset.
- I would use query analyzer tools to observe the most costly parts of the query and try to evaluate the query and look for ways to refactor the code in order to minimize or eliminate unnecessary data processes to achieve the same results.
- We could have got more creative and used temporary tables instead of views for some parts, to avoid potential storage cost because if we are dealing with bigger volume of data (which is probably the case for the business scale at Wolt). Instead of storing a view just to be used in an analysis which might impose more costs, we might be able to generate temporary tables which get automatically dropped after the end of the session.
- I would think of more test cases to guarantee the data accuracy and would create some notification for our engineers to look into data inaccuracies if some of them happen. It can save so much time and resources if we automate the data quality checks compared to having the team do it manually. Based on my experience, having an automated data quality system combined with notifications is so powerful to guarantee data accuracy. Ideally they both come out of the box with the technology we use because we won't need to build it ourselves. If not, we can make it ourselves but I would go for developing in house as the very last resort.
- I would enclose this research with a data dictionary.

Thank you for your attention.

Best Regards,
Mojtaba