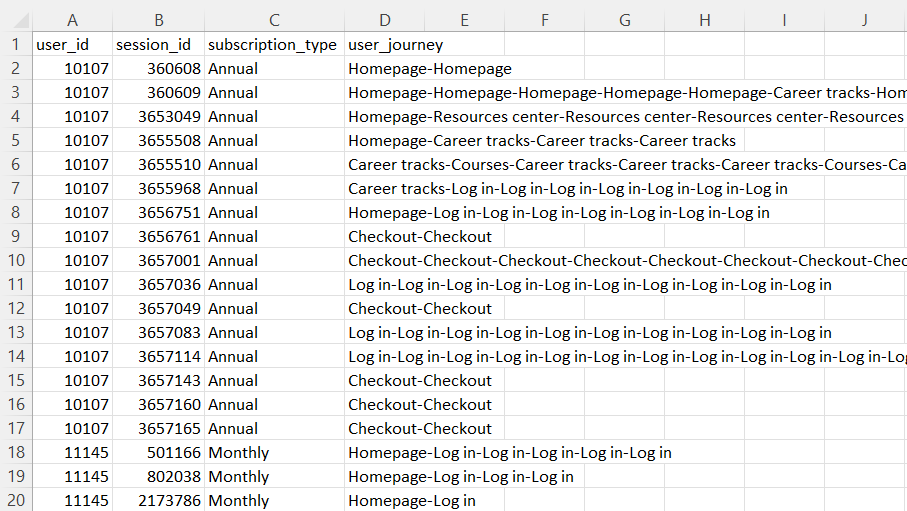
**Extracting User Journey Data Using SQL Project**

**Extracting the Data**

Your objective in this project is to write up a query to extract data for a subsequent user journey analysis. The idea is that someone in your company wants to analyze the sequence of pages visited leading to a purchase. For that purpose, you need to group all the pages visited in a session into one string. This shouldn’t be done for every visitor but only for paying customers. In other words, you should extract the user journey of only those that have purchased a subscription plan.

In addition, you should only consider users that made their first purchase between January 1, 2023, and March 31, 2023, inclusive (i.e., Q1 2023). To give more options for the analyst, later on, you also need to include what subscription plan the user purchased.

Note the screenshot below of an example CSV generated through the SQL code:



The expected format for the data extract includes four columns:  user\_id  ,  session\_id  ,  subscription\_type  **,** and  user\_journey  . As you see, each session should have a separate user journey string. And all the pages are combined into one string separated with a hyphen -. You can use a different separator but don’t go for a comma since that will clash with the CSV file.

There are also many repeating pages in the strings because the visitor scrolled or took other measures on the page that didn’t redirect them to a different page. For this task, you’re not required to clean this up. You can safely squeeze all the pages from the ‘front\_interactions’ table together without removing duplicates.

Note some additional requirements below.

First, you need to consider when to cut off the user journey. The analysis is based on the pages a user visited before purchasing because there’s no need to analyze the ‘thank you’ and ‘purchase confirmation’ pages. So, you must exclude all visited (or interacted) pages after the purchase timestamp.

Another consideration is that there might be some test records in the database. You should filter them out by considering that a real user wouldn’t pay $0 for the product, or the company would quickly go bankrupt.

Finally, all pages are listed through their respective URLs in the database. You would need to assign an alias to the most essential URLs. For example, instead of including https://365datascience.com/ in the user journey string, only Homepage should appear. You can find the list of URLs and aliases as an Excel file in the provided files for this project.

Great! Let’s have a quick recap of everything you need to do.

**Tasks Summary:**

* Consider all users that purchased a subscription plan **for the first time** between January 1 and March 31, 2023 (inclusive).
* Consider all their page interactions before their purchase date.
* Remove test users (ones that paid 0 dollars).
* Create aliases (nicknames) for the URLs.
* Combine all the pages of each session into a single user journey string.
* Export all this data as a CSV with the **user\_id**, **session\_id**, **subscription\_type,** and **user journey**

Note the following tips to help guide you in executing these points above.

The easiest way to achieve all this is to use a statement with a WITH clause (CTE). This allows us to create several small queries that are easier to work with, and we can test them out independently.

With that in mind, the first query can list all the users relevant to this task; let’s call it paid\_users. This will allow us to cross-reference them with their visitor ID and extract their sequence of visited pages. You can even filter out the test users (those with purchase\_price of 0) directly here, considering only the relevant date ranges. This query should have four columns: user\_id, first\_purchase\_date, subscription\_type (you can apply the actual names "Monthly", "Quarterly" and "Annual" to the numerical codes here, for clarity) and purchase\_price. Consider how your filtering interacts with grouping.

The next common table expression in the CTE structure can take all the users from the first query and extract all the relevant entries from the front\_interactions table. (Recall that in a statement with a WITH clause, you can use all the above subqueries directly in the current one.) I would consider all columns from the interactions table except the event\_name which is not relevant for this query. Be sure to include only the events before the user subscribes for the first time.

At this point, you should have a table considering all relevant users, their subscription type and listing all the relevant sessions and the URLs of the source and destination pages. So, for the overall objective, the only thing left to do is concatenate those pages, one after another, for every session. However, at this point, all those pages are represented through URLs, which are long strings and unnecessarily cluttered. So, for clarity and readability, replace all those URLs with short nicknames or aliases. For that purpose, you can use the aliases file provided.

Therefore, the next query in the WITH clause needs to select all the columns from the previous one and change the strings in the two URL columns. This change can be achieved using CASE. One detail to consider here is that URLs like https://365datascience.com/resources-center/ reference that section’s main page which contains many different resources. But every specific resource resides on a page with a URL that starts like the example one but adds more elements after the last backslash. So, when assigning the nicknames, consider all URLs starting with the provided strings, not the exact strings—except for the homepage; you can look for an exact match there.

Great.

You can now combine all the pages, right? Well, yeah, but it’s not as simple to do in one step. You’ll need two subqueries for this: the first should combine the two columns (source page and destination page), and the second is supposed to include all the strings that belong to a single session.

You can use the CONCAT() function to combine two strings from two columns into one in MySQL. And that’s precisely what you should do in the following subquery—just be sure to pick a sensible separator. I use a dash -. You can find more details on how to use the CONCAT() function in the official documentation.

So, at this point, you have only one column containing the pages instead of two. Thus, you can move on to the last query on this enormous WITH statement that will combine all the strings of this one column for each session. This is achieved with the GROUP\_CONCAT()function . Since that’s an aggregate function, you actually need to group by session\_id. Be sure to use the same separator in this function as in the previous CONCAT().

Unfortunately, there is one small but essential detail when working with GROUP\_CONCAT(). By default, this function’s maximum string length is 1,024 symbols. You may notice that many user journeys in our data are longer than that. In that case, the string is truncated and stops in the middle of a page. So, you must increase this limit to include all pages—100,000 symbols should work fine. (Check the official documentation to see how to do this.)

And that’s it. You can now order the result by user\_id and session\_id and export it in a CSV.

**Quiz**

**Question 1:**

In SQL, if you want to obtain all users whose first purchase date is in a given range, what construct would you use?

GROUP BY + WHERE

GROUP BY + HAVING

WHERE + HAVING

JOIN + WHERE

**Question 2:**

In SQL, if you want to consider all strings that contain an example\_string in a column, which construct would you use?

*column* CONTAINS 'example\_string'

*column* HAS 'example\_string'

*column* LIKE 'example\_string'

*column* LIKE '%example\_string%'

**Question 3:**

What is the purpose of the group\_concat\_max\_len parameter?

It states the maximum amount of times a string can be concatenatedIt states the maximum length of a single string to be considered for concatenation

It states the maximum amount of symbols for a string the GROUP\_CONCAT function can produce

It states the maximum amount of sessions GROUP\_CONCAT should consider

**Question 4:**

What is the user\_id of the first record in the data extract (when ordered by user\_id and session\_id in ascending order)?

1516339520264205

**Question 5:**

What page is the first one in the user journey of the 3rd record in the extract?

Log inHomepageCheckoutOther