Data Warehousing and Concepts

Part A – Table Creation

Please see **Part A – Table Creation.sql** in attached for more details. The code was written in T-SQL with the following considerations.

* Used the IDENTITY function on the primary keys (user\_id and order\_id) to set the automatic increment of 1 when a new row is inserted.
* Applied PRIMARY KEY constraint on the same fields meaning that these are unique and always populated whenever a row is inserted.
* Applied a FOREIGN KEY constraint on the user\_id in ORDERS table to reference user\_id in USERS. This translates that the ORDERS table can have one or more records associate with a user in USERS table (1 to Many relationship).
* By specifying that ORDERS.user\_id is not null, we want to ensure that an order is always associated with a user.

Part B – Select Queries

Please see **Part B – Select Queries.sql** in attached for more details. The code was written in T-SQL with the following considerations.

For all queries, the DISTINCT function was used to count **the number of orders** for a given user under the assumption that the order can contain multiple items from different shops. This means that the order can have multiple records so therefore taking the count distinct will ensure that we are considering it once.

Additional comments:

1. Used a LEFT JOIN to ensure that all users are returned regardless their number of orders. Used the COALESCE function to return 0 for users with no order.
2. Used the query above as a CTE to answer the question.
3. Used INNER JOIN to return only matching users that have a transaction amount defined in the WHERE clause of the query.

Part C– Transforming and moving data

1. Please refers to **Part C – Transforming and moving data.sql** in attached for more details.
2. Instead of a running full refresh to update the table, the incremental update is a beneficial option as this strategy involves adding only new or modified records since the last run. In the absence of a timestamp field to track these records, we can rely on the primary key in our source ORDERS table to determine new records (Both fields were implemented to update incrementally) and then add them in the target user\_orders table. The MERGE statement in T-SQL was used for this purpose. Reference to **Part C – Transforming and moving data.sql** for more details.
3. We used a conjunction of T-SQL functions SUBSTRING (extract the first initial from the first letter of the name) and CONCAT to attach the second string. As one-off operation, the UPDATE clause was used to update the user\_orders table. Reference to **Part C – Transforming and moving data.sql** for more details.

Part D– Data warehousing concepts

1. To answer these questions, we highlight some key differences between OLTP and OLAP systems.

* OLTP are designed to support the execution of the business processes whereas the purpose of a data warehouse focused is evaluating the business processes. As it is in the present use case, the source order system is mostly concerned about constructing a record of activities about orders. This is translated by the various events of insert, update and delete in the system.
* Data is likely to change as things update in OLTP as opposed to Analytics systems designed to store efficiently historical data.
* OLTP are designed to be highly normalised (third normal form) which makes it not the best place to perform aggregated queries to answer business questions. On the other side, Analytics systems are designed on the principles of dimensional design to provide ease of use, high performance queries on large volume of data.

For these reasons, creating the denormalized table in an OLTP database could result in performance issues interacting with the system and slow it down. Business users would not have the historical data needed to answer critical questions on performance and not the best experience when it comes to query the denormalized table.

For these reasons, it is advisable to build and store the denormalized table in an analytics system to answer business questions in the most efficient way.

1. Using the CDC implementation is to facilitate real-time or near real-time integration between the sources systems and the data warehouse. Unlike the batch load approach taken in part C (loading the table on a daily basis ), the CDC approach enable to continuously capturing and delivering changes as they occur. Businesses benefit greatly from accessing the most current for decisions and risk mitigations in a fast-moving environment.

Importance of event type and event\_timestamp in the target table

* Event type: plays a role of filter as it enables us to filter based on the type of event (insert, update, delete).
* Event timestamp: plays the role of a sorting key as it enables to sort events to provide ordering. For example, we can track a user activity based on this attribute.

Other tables to build and how to keep them updated

We could build a final table in the data warehouse on the top of the staging table for analytics or track slowly changing dimensions. All downstream tables eventually built in this case will be required to have an orchestration flow to refresh them at a defined interval.

Data Modelling

Solutions presented for this use case are implemented in BigQuery to align with the requirements of the role and the data stack used by the team at GoHenry.

0.Raw Data from Extract Load Process

Depending on the initial format of the data and the agreement with the engineering team, the JSON data might be stored into BigQuery in 2 ways:

* Structured tables (with nested data and columns as record type)
* ‘Stringfied’ JSON data

Although BigQuery offers the tools to pre-process the raw data into format that can be queried in both cases (for example JSON\_VALUE, ARRAY\_TO\_STRING functions to convert the ‘stringfield’ data and UNNEST function for the record type data), we are making the initial assumption that the data is stored as tables with nested structures.

Also it’s worth mentioning that for the purpose of this exercise, we are developing and deploying in one schema ‘dbt\_souare’ and in the same environment (GCP project).

The following 3 tables have been created in BigQuery based on the source JSON data.

A close-up of a computer screen

Description automatically generated with low confidence

Campaigns

A screenshot of a computer

Description automatically generated with low confidenceA screenshot of a computer

Description automatically generated with medium confidence

Campaign\_statistics

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

We can see here that the platforms attribute is a parent attribute as record type and as well as its children in the table campaign\_statistics.

Creatives

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated

1.Build a staging layer

This initial step involves 2 main actions.

* 1. Building an initial layer of the transformation process with a 1 to 1 relationship with the source tables.

In this initial step, we have renamed some fields (mainly ids) and converted timestamp to date to make them consistent in later stages. A decision was also taken to disregard the attribute ‘created’ (timestamp) from the campaign table as it is considered as a metadata field.

For the purpose of this exercise, we have created 3 views to mirror our 3 sources tables with a prefix ‘stg’.

All the subsequent models will be built based on the top of these.

A screenshot of a computer

Description automatically generated with medium confidence

A significant action taken here was to flatten the view stg\_campaign\_statistics to ensure that the platforms attribute (record type) and its nested columns can be queried in downstream steps.

A screenshot of a computer

Description automatically generated with low confidence

1.2 Data quality tests and granularity

This steps is to help us understand the grain of the data and validate our assumptions by running basic tests on the tables including not null and unique on primary keys.

stg\_creatives

The query below shows that:

* creative\_creativeid as the primary key (natural key)
* Each row records information on a unique creativeid

A screenshot of a computer

Description automatically generated with medium confidence

Stg\_campaigns

The query below shows that:

* creative\_creativeid as the primary key (natural key)
* Each row records information on a unique creativeid

A screenshot of a computer

Description automatically generated

Stg\_campaign\_statistics

The query below shows that each row records information on a unique campaign and creativeid. In other words, each record shows information on one campaign and one creative. This is crucial information to consider in our mart layer because this view also records facts at the platform level (google and facebook).

A screenshot of a computer

Description automatically generated

2.Build a Mart layer

Having explored the structure of the data and for the purpose of meeting the requirements by the analysts, it is important to organize the data for faster retrieval, hence minimize joins at the semantic layer in our BI tool and most importantly to respond questions on the campaign performance at various level of detail.

Our design must also consider how to respond to a wide range of unanticipated questions (scalability).

To accomplish this, we are adopting a dimensional design technique to model our mart layer. Central to do will be organising our staged data into dimensions tables. The expected model will be presented in a form of a start schema or multi-fact star schema.

Key design principles:

* All our objects within the mart layer are materialized as tables (end use experience)
* Dimensions are prefixed with ‘dim’ and fact tables with ‘fct’
* Dimensions are built first and then fact table
* Natural keys in fact tables are disregarded. There exist different opinions on this but our guidance here is to disregard them.
* Surrogate keys are built by hashing the natural keys in the tables. We used the SH256 hashing function in BigQuery to achieve this.
* Surrogate keys are suffixed with ‘key’

2.1 Create the dimension tables

To follow best practices, we will start by building our dimensions tables so that we have updated and complete keys to link with our facts tables.

Reference is being made to surrogate keys that will created as part of this process and will be unique identifiers to all records in the new dimensions tables.

**Dim\_campaign**

The dim\_campaign dimension was created with all its attributes and a surrogate key. Reference to **dim\_campaign.sql**in attachedfor more details.

A screenshot of a computer

Description automatically generated with low confidence A screenshot of a computer

Description automatically generated with medium confidence

**Dim\_creative**

The dim\_creative dimension was created with all the existing attributes and a surrogate key. Reference to **dim\_creative.sql**in attachedfor more details.

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

**Dim\_platform**

As briefly pointed out in the previously stages, the stg\_campaign\_stats table records facts both at the campaign and platforms level so we build the platform table on its own dimension to enable slcice and dice of the data.

Although it’s tempting to include the platform within dim\_campaign, the platform attribute is also linked to dim\_creative, hence the decision to build this dimension. Reference to **dim\_platform.sql** for more details. This will also make easier to filter the data.

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

**Dim\_date**

Our dim\_date dimension has one row per date starting from the 2022-01-16 which is the earliest day of all campaigns. It has also a surrogate key built based on the full\_date. Reference to **dim\_date.sql** in attachedfor more details.

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated

2.2 Create the fact tables

**Fct\_campaign\_stats**

This table records only facts at the campaign and creative level and all surrogate keys of the related dimensions. Reference to **fct\_campaign\_stats.sql** in attached for more details.

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

**Fct\_platform\_stats**

Despite that all facts recorded in stg\_campaign\_stats seem to be taking place simultaneously, it’s clear that they measure campaign performance at different level of granularity as shown in the image below. Keeping the these in the same table would violate one of the principles of the dimensional design and hamper analysis.

A screenshot of a computer

Description automatically generated with low confidence

Reference to **fct\_platform.sql** in attached for more details.

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

3.Model Documentation

Our final model is a multi-fact star schema that includes 2 facts tables, 2 conformed dimensions and an additional dimension. This model provides ease of use for the analysts and allow to measure the performance at campaign level as well as drill down by platform.

Below are overview of the model and the project structure in our data warehouse.

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text, diagram, plan, technical drawing

Description automatically generated

4. Areas of Improvement

Given the data type of natural keys (integer preferred over string data type), exploring other options to generate the surrogate keys would be worth the efforts.

3.Alternatives Approach

An alternative modelling technique would have been the One big table or wide table which consists of using one single and large denormalized table to store all the information.

As a columnar database, BigQuery offers great capabilities to accomplish this, however, this technique is best suited for tracking specific item.

For example, country of performance for our campaigns. The analysis would be focused on information such as region of the country, size, population etc…

Another reason is that management and maintenance can become difficult if the data becomes complex.

Finally, performing joins on data with different granularity can potentially produce inconsistent results when using OBT at scale.