**Batch Pipeline**

**Pipeline overview**

The proposed solution available at <https://github.com/andrei-glk/checkout> creates a continuous data pipeline that takes into consideration 2 sources of ingested data, joins them and delivers a denormalised table into cloud-based data warehouse Snowflake. As our choice of DWH is Snowflake a decision has been taken to develop a data pipeline using new Snowflake object types:

* Stream
* Tasks

A stream is created on top of a base table (effectively a new table although it doesn’t appear in the UI as a table) and provides change data capture (CDC) capabilities to track the delta of changes in a table, including inserts and data manipulation language (DML) changes.

A task is a new Snowflake object type that is used to schedule actions. They can run based on frequency (e.g. every minute) or time based (e.g. 7am daily). They may also contain a Boolean when clause to evaluate whether the task should trigger:

* This can be used to check if a **stream** has data
* when system$stream\_has\_data(‘stream\_name’)
* If the result is false then the task does not run

In this solution, tasks use streams to provide a convenient way to continuously process new or changed data. By using a stream, a task can verify whether a stream contains changed data for a table and either consume the changed data or skip the current run if no changed data exists.

**Assumptions**

This pipeline is working under the assumption that there is an external ETL tool where two jobs have been scheduled to perform these tasks:

1. an extract process consumes the users data on a daily basis around midnight (00:00). The process fully extracts users data, landing the data on a table within the Data Warehouse (Snowflake) named “user\_daily\_extract”. This table is fully truncated/reloaded on each execution.
2. an extract process which consumes the pageviews data on an hourly basis. The process incrementally extracts pageviews data, landing the data on a table within the Data Warehouse named “pageview\_extract”. On each execution of the extract process, this table is fully truncated and subsequently loaded only with the pageviews data relative to the previous hour.

**Sample Data**

For testing purposes, a python script has been developed to generate sample data:

**user\_extract:**

* An id, uniquely identifying each user. Example: 1234
* A postcode, indicating where a user is at the moment. This attribute may change regularly based on the user’s location. Example: SW19

**Pageview\_extract**

* A user\_id, uniquely identifying a user. This matches the id on the users table. Example: 1234
* An url of the page being visited. Example: www.website.com/index.html
* A pageview\_datetime when the pageview occurred. Example: 2019-10-11 14:55:23

The sample data is available at <https://github.com/andrei-glk/checkout/tree/master/sample_data>

**Data Warehouse Model**

Our end goal is to build the Data Warehouse tables/structures which will allow our BI tool to easily and in a performant way answer 2 questions:

* Number of pageviews, on a given time period (hour, day, month, etc), per postcode based on the current/most recent postcode of a user.
* Number of pageviews, on a given time period (hour, day, month, etc), per postcode based on the postcode a user was in at the time when that user made a pageview.

Our model is enclosed with three layers

1. Staging layer (STG\_USERS, STG\_PAGEVIEWS) where all the data from different sources are situated
2. Transformation layer (TRANSFORM\_ALL) where the data undergoes ETL processing
3. Datamart layer where the front-end tools are employed as per the users’ convenience.

To set-up an environment in Snowflake we need to run the scripts available at <https://github.com/andrei-glk/checkout/tree/master/environment> in the following order:

1. set\_up\_environment.sql
2. create\_tables.sql

Once the environment has been created, run the script provided in <https://github.com/andrei-glk/checkout/tree/master/transformation>. This will create a view v\_pageviews in the Transformation layer. A view allows the result of a query to be accessed as if it were a table. At this stage, the query will return no records due to empty tables. There are two ways to populate newly created staging tables with data:

1. Using Snowflake web interface.
2. Bulk loading using the COPY command. Use this option if you have an AWS account.

A set of CSV files, as well as copy script, are available at <https://github.com/andrei-glk/checkout/tree/master/sample_data>.

**Implementation of a mechanism for scheduling the Transform pipeline**

At this stage, we have several staging tables being populated with sample data. Now we will develop a continuous data pipeline. This can be achieved by running a script available at <https://github.com/andrei-glk/checkout/tree/master/pipeline>. This script creates several streams on top of a base tables and schedules the tasks to execute SQL statements. The tasks running every 60 minutes can verify whether a stream contains changed data for a table and either consume the changed data or skip the current run if no changed data exists. The final task in the pipeline overwrites any existing data in the target denormalised table (pageview) held in DataMart layer after the successful completion of the parent task if a specified stream contains change data capture (CDC) records.