2020

BATCH PIPELINE

CHECKOUT.COM

ALEXANDRE MUGNIER

Task summary:

“You have been asked to create a batch pipeline to take into consideration 2 sources of ingested data, join them and deliver a denormalised table/model into Snowflake (or any other database of your choice)”

Table of contents

Data source description:

Users data:

Source

Data Description:

We have an extract process which consumes the users' data on a daily basis around midnight (00:00). The process extracts users data, landing on a table within the Data Warehouse names "users_extract". This table is fully truncated/reloaded on each execution.

Column Name	Data Type	Description
Id	bigint	uniquely identifying each user
Postcode	VARCHAR(32)	Latest position of the user

Destination Table – "users_extract":

Data Description:

- This table is an extract of "Users" data.
- Refreshed every day at midnight.
- User's location, refreshed once a day. **This table contains a single location**

Column Name	Data Type	Description
Id	bigint	uniquely identifying each user
Postcode	VARCHAR(32)	Latest position of the user

Pageviews data:

Source

Data Description:

We have 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 "pageviews_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.

Column Name	Data Type	Description
user_id	bigint	Identify the user that access a specific page
URL	varchar (256)	Url of the page being visited
Pageview_datetime	datetime	Date time when the page is being viewed

Destination Table – "pageview_extract" :

Data Description:

- This table contains a list of the pages accessed by the users
- This table contains only data related to the previous hour before the refresh
- **Records previously loaded are "lost" after the Refresh**

Column Name	Data Type	Description
user_id	bigint	Identify the user that access a specific page
URL	varchar(256)	Url of the page being visited
Pageview_datetime	datetime	Date time when the page is being viewed

Data Warehouse Model/Pipeline:

Requirements

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:

CASE 1 - Number of pageviews, on a given time period (hour, day, month, etc), per postcode based on the current/most recent postcode of a user

CASE 2 - 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.

General preparation: Creating – “tbl_pageviews_history” table:

To respond to both requirements (Cases above), we have been asked to create a structure that allow the user to access the data on a given period. This implies that all the data is available in the Data Warehouse. The 2 original data sources available (“users_extract” & “pageviews_extract”) are fully truncated on a regular basis, the amount of data available in these sources is limited. To meet the requirements, it is necessary to create a new table that would store all the different “screenshots of the data”.

Considering the 2 Data sources Constraints:

- the table “pageviews_extract” contains only 1 hour of records
- the table “users_extract” contains only the latest position of the users each day

Granularity of the “tbl_pageviews_history” :

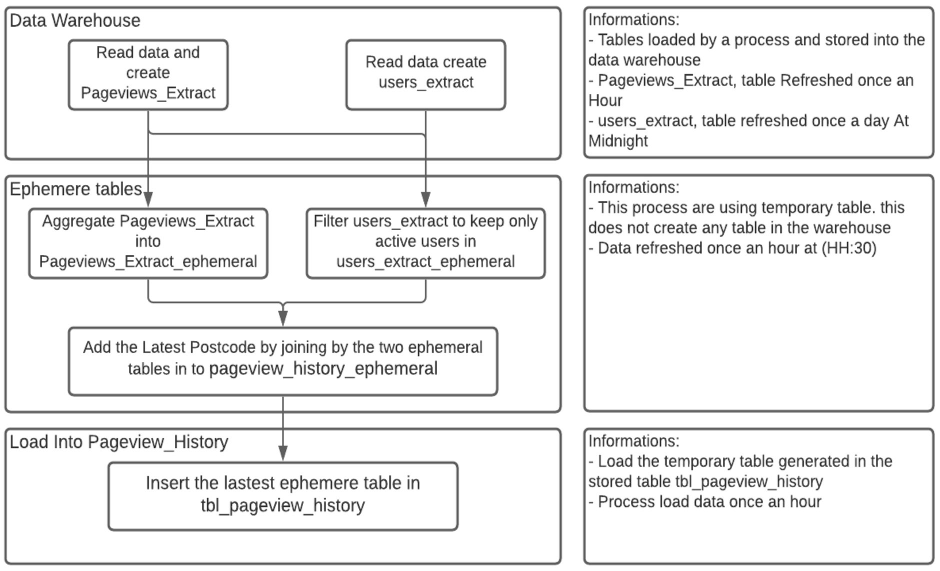
To optimise the storage and the performance of the BI Tool, it is possible to aggregate the original data before loading the table into “tbl_pageviews_history”.

Base on the requirements:

- the time frame can be aggregate by hours for each day. Minutes & Seconds are not a level of detail that the team is interested in. Ex: ‘2020-11-05 16:52:45’ -> **‘2020-11-05 16:00:00’**
- The PageView_URL is not an information that the team is interested. We can aggregate using a COUNT(URL)
- User_ID is kept for this table. This field will also be used to get customer postcodes

Column Name	**Data Type**	**Description**
user_id	bigint	User ID
Pageview_date_hour	datetime	Summarized DateTime (See Above). Minutes and second are removed for a better aggregation
Count_Pageviews	Int	COUNT OF URL Group by user, Pageview_date_hour
Count_Distinct_Pageviews	Int	COUNT OF DISTCINCT URL Group by user ,Pageview_date_hour
Customer_postcode	varchar(32)	Customer_postcode when the table is being built

Proposed Load Mechanism:


Data Warehouse
Read data and create Pageviews_Extract
Read data create users_extract
Informations:
- Tables loaded by a process and stored into the data warehouse
- Pageviews_Extract, table Refreshed once an Hour
- users_extract, table refreshed once a day At Midnight
Ephemere tables
Aggregate Pageviews_Extract into Pageviews_Extract_ephemeral
Filter users_extract to keep only active users in users_extract_ephemeral
Add the Latest Postcode by joining by the two ephemeral tables in to pageview_history_ephemeral
Informations:
- This process are using temporary table. this does not create any table in the warehouse
- Data refreshed once an hour at (HH:30)
Load Into Pageview_History
Insert the lastest ephemere table in tbl_pageview_history
Informations:
- Load the temporary table generated in the stored table tbl_pageweiw_history
- Process load data once an hour

The data source “pageviews_extract” available in the data warehouse stores a screenshot of the customer activity on the website only from the previous hour. Which implies that the data will be replaced once an hour and “lost”. To avoid that, we need to proceed of the load of data in the table “tbl_pageviews_history”

In the diagram above we can divide into 3 distinct parts the process:

- Read the data available in the Data Warehouse
- Create some ephemeral table to process the data
- Load “Pageviews_history_ephemeral” into tbl_pageviews_history

Solutions:

CASE 1 – Solution:

This case can be answered by querying the view created named “view_pageviews_history_Latest_PostCode”. This view is referencing the table history named “tbl_pageviews_history”. By proceeding to a join with the table “users_extract”, it is possible to retrieve the number of pages viewed by any users referencing the latest_postcode of the user on the day of the query.

CASE 2 – Solution:

This case can be answered by querying the table created as detailed above named “tbl_pageviews_history” which is incrementally refreshed once an hour with the new summarized data available in “pageviews_extract”.

Inside this table, the field “customer_postcode” stored represent the postcode when the user accessed the different pages not considering where the user is now.

Working Files:

The files can be accessed into the Github Repository:

https://github.com/amugnier/DBT_AM_CKO/