Automating Data Integration: Configuring ETL Workflows with Postgres

VDM1 Task 1: Automating Data Integration

Advanced Data Managment

SECTION A: Summarize Report Business Report

1. Describe the data used for the report

The data used in this report shows the daily transaction or "sales" data for each store. This report describes the daily sales data set by showing various measures like measures of spread. This is also broken down by each individual store for the management team to easily know what their metrics are. The data is the tranformed and loaded into a separate table show the projected sales for the week based on the previous weeks sales.

2. Identify two or more specific tables in the data-set included in the Report

In this dataset I will be using the payment table and the staff table.

3. Identify the specific fields that will be included in both sections

Detailed Report Fields	Summary Report Fields
day_of_sale *NUMERIC*	month_of_sale *TEXT*
month_of_sale *NUMERIC*	sum_of_sales *MONEY*
sum_of_daily_sales *MONEY*	sale_projections *MONEY*
number_of_sales_per_day *TEXT*	
mean_of_daily_purchase_transaction *MONEY*	
lowest_daily_sales *MONEY*	
highest_daily_sales *MONEY*	
range_of_daily_sales *MONEY*	
store_name *TEXT*	

4. Identify a field in the the detailed_table that requires a custom transformation

A field in the detailed table that will be transformed is the store_name column. This column is an integer and seeing this as a textual representaion of that number helps to easily identify one store FROM another. Understanding what sales happen per store also makes the predition possible.

I will also be making another transformation to 2 columns: month_of_sale and day_of_sale. I will be extracting the day/month out of the timestamp to return an integer representing the day/month. This makes comparisions and operations easier.

5. Explain the different business uses of the detailed and the summary sections of the report.

The business use for the detailed table can be used in summarizing all the sales data for the day along with other statistics for a manager to gague sales performance.

The business use for the summary table will allow the manager to estimate the sales based on the previous 7 days. This is useful to a manager because it lets them schedule correctly and save on labor costs.

6. Explain how frequently your report should be refreshed to remain relevant to stakeholders

> This report should be refreshed weekly to ensure that you have an accurate picture of the next weeks projections.

SECTION B

1. Write a SQL code that creates the tables to hold your report sections.

Detailed Table

```
CREATE TABLE detailed_table (
    store_name TEXT,
    day_of_sale NUMERIC,
    month_of_sale NUMERIC,
    sum_of_daily_sales MONEY,
    number_of_sales_per_day TEXT,
    mean_of_daily_purchase_transaction MONEY,
    lowest_daily_sales MONEY,
    highest_daily_sales MONEY,
    range_of_daily_sales MONEY);
```

Summary Table

```
CREATE TABLE summary_table (
    month_of_sale TEXT,
    sum_of_sales MONEY,
    sale_projections MONEY);
```

SECTION C

> Write a SQL QUERY that will extract the raw data needed for the Detailed section of your report FROM the source database and verify the data's accuracy.

```
INSERT INTO detailed_table(
    SELECT  CASE
          WHEN s.store_id = 1
            THEN 'North Store'
          WHEN s.store_id =2
            THEN 'South Store'
        END case_statement,
        get_day(p.payment_date) as day_of_sale,
        get_month(p.payment_date) as month_of_sale,
        SUM(p.amount) sum_of_daily_sales,
        CONCAT(COUNT(p.payment_date) || ' ' || 'Daily Sales') as
number_of_sales_in_day,
        CAST(AVG(p.amount::NUMERIC) as MONEY) as mean_of_daily_sales,
        MIN(p.amount) as lowest_daily_sale,
        MAX(p.amount) as highest_daily_sale,
        (MAX(p.amount) - MIN(p.amount)) as range_of_daily_sales
FROM    payment p
    INNER JOIN staff s ON p.staff_id = s.staff_id
GROUP BY    1,3,2);
```

SECTION D

> Function(s) that perform the transformation(s) you identified in part A1.

Function 1: Get Day Function

> This get_day function will get the day out of the payment_date (TIMESTAMP WITHOUT TIMEZONE) column in the detailed table and return a numeric value to be used to group the data by.

```
CREATE OR REPLACE FUNCTION get_day(payment_day TIMESTAMP)
    RETURNS int
    LANGUAGE plpgsql
AS $$
DECLARE day_of_sale int;
BEGIN
    SELECT EXTRACT(day FROM payment_day) INTO day_of_sale;
    RETURN day_of_sale;
END;$$
```

Function 2: Get Month Function

> This get_month function will get the month out of the payment_date (TIMESTAMP WITHOUT TIMEZONE) column in the detailed table and return a numeric value to be used to group the data by.

```
CREATE OR REPLACE FUNCTION get_month(payment_month TIMESTAMP)
    RETURNS int
    LANGUAGE plpgsql
AS $$
DECLARE month_of_sale   int;
BEGIN
    SELECT EXTRACT(month FROM payment_month) INTO month_of_sale;
    RETURN month_of_sale;
END;$$
```

SECTION E

> Write a SQL code that creates a trigger on the detailed table of the report that will continually update the summary table as data is added to the detailed table.

Trigger funciton

```
CREATE OR REPLACE FUNCTION insert_summary_trigger_function()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO summary_table(
    SELECT      month_of_sale,
                sum_of_daily_sales,
                AVG(sum_of_daily_sales::numeric)
                OVER(ORDER BY month_of_sale ROWS BETWEEN 7 PRECEDING AND
CURRENT ROW)
                AS projection
                FROM detailed_table);
    RETURN NEW;
END;$$
```

Trigger

```
CREATE TRIGGER summary_insert
    AFTER INSERT
    ON detailed_table
    FOR EACH STATEMENT
```

```
        EXECUTE PROCEDURE insert_summary_trigger_function();
```

SECTION F: Create a stored procedure that can be used to refresh the data

in both your detailed and summary tables. The procedure should clear the contents of the detailed and summary tables and perform the ETL load process FROM part C and include comments that identify how often the stored procedure should be executed.

```
CREATE OR REPLACE PROCEDURE reload_tables()
LANGUAGE plpgsql
AS $$
BEGIN
    TRUNCATE TABLE detailed_table;

    TRUNCATE TABLE summary_table;

    INSERT INTO detailed_table(
    SELECT  CASE
          WHEN s.store_id = 1
            THEN 'North Store';
          WHEN s.store_id =2
            THEN 'South Store';
        END case_statement,
        get_day(p.payment_date) as day_of_sale,
        get_month(p.payment_date) as month_of_sale,
        SUM(p.amount) sum_of_daily_sales,
        CONCAT(COUNT(p.payment_date) || ' ' || 'Daily Sales') as
number_of_sales_in_day,
        CAST(AVG(p.amount::NUMERIC) as MONEY) as mean_of_daily_sales,
        MIN(p.amount) as lowest_daily_sale,
        MAX(p.amount) as highest_daily_sale,
        (MAX(p.amount) - MIN(p.amount)) as range_of_daily_sales
FROM    payment p
    INNER JOIN staff s ON p.staff_id = s.staff_id
GROUP BY    1,3,2);

    INSERT INTO summary_table(
    SELECT      month_of_sale,
                sum_of_daily_sales,
                AVG(sum_of_daily_sales::numeric)
                OVER(ORDER BY month_of_sale ROWS BETWEEN 7 PRECEDING AND
CURRENT ROW)
                AS projection
                FROM detailed_table);

  COMMIT;
```

```
END;$$
```

```
CALL reload_tables();
```

CALL reload_tables();

1. Explain how the stored procedure can be run on a schedule to ensure data freshness.

The reload_tables stored procedure requires a couple of steps to be complete before you can run it. The steps invole downloading an external tool called pgAgent that can schedule stored procedures based on a schedule. This is a popular tool that for the execution of stored procedures, SQL statements, and shell scripts. The best practice is to have this running as a daemon on Linux to check periodically if there are commands to run.

Steps to run a scheduled stored procedure on a linux machine:

1. Install the service itself by running the following command:

```
 > $ sudo apt install pgadmin4 pgadmin4-apache
```

2. Creation of plpgsql procedural language:

```
CREATE  TRUSTED PROCEDURAL LANGUAGE ‘plpgsql’;
                HANDLER plpgsql_call_handler;
                HANDLER plpgsql_validator;
```

3. Installation of pgAgent:

```
> $ sudo apt-get install pgagent
```

4. Creation of the pgAgent Extension:

```
CREATE EXTENSION pageant;
```

5. Define New Job:

In order to define a new job you have to click create on the tab inside of pgAdmin and then define the steps of the job to be able to execute it.

6. Schedule Jobs:

   Schedule the job.

SECTION H: Web Sources

No web sources were used to acquire data or segments of 3rd party code

SECTION I: Sources

- An Overview of Job Scheduling Tools for PostgreSQL read more by: Hugo Dias Hugo Dias is a Guest Writer for Severalnines. He has worked for many years developing software using a variety of programming languages such as C & Dias https://severalnines.com/database-blog/overview-job-scheduling-tools-postgresql