Samuel Phillips - D326 Advanced Data Management

Business Report

1. Summarize one real-world written business report that can be created from the DVD Dataset from the “Labs on Demand Assessment Environment and DVD Database” attachment.

A pressing business question for any DVD rental company might be: who are our customers who have stopped using our rental service? Management may spend time brainstorming *why* people have stopped renting, but that information is not useful if they don’t have a full list of lapsed customers. Knowing the amount of time since a customer last rented a DVD is crucial for the business to know who exactly to target for advertising to bring them back as active customers. The purpose of my project is to show how much time has passed since a customer rented a DVD. I will add two tables called LastRentalTimeSummary and LastRentalTimeDetailed which will contain two different levels of granularity for data concerning the customer’s last rental time:

1. Identify the specific fields that will be included in the detailed table and the summary table of the report.

2.  Describe the types of data fields used for the report.

| LastRentalTimeSummary Fields | | |
| --- | --- | --- |
|  |  |  |
| Field Name | Data Type | Description |
| customer\_id | smallint | Number value of unique customer ID. |
| days\_since\_last\_rental | smallint | Integer value of number of days since last rental. Calculated by subtracting rental\_date from the present time. |
| last\_update | timestamp | Last time data in column was updated. |

| LastRentalTimeDetailed Fields | | |
| --- | --- | --- |
|  |  |  |
| Field Name | Data Type | Description |
| customer\_id | smallint | Number value of unique customer ID. |
| store\_id | smallint | Store’s unique numeric identifier. |
| create\_date | timestamp | When customer was first added into system. |
| last\_update | timestamp | Last time data in column was updated. |
| rental\_id | integer | Numeric identifier of movie rental. |
| rental\_date | timestamp | When movie was rented. |
| return\_date | timestamp | When movie was returned. |
| days\_since\_last\_rental | smallint | Integer value of number of days since last rental. Calculated by subtracting rental\_date from the present time and converting to integer with floor function. |

3.  Identify at least two specific tables from the given dataset that will provide the data necessary for the detailed table section and the summary table section of the report:

Data is taken from the following tables:

**Rental:** rental\_id, rental\_date, customer\_id.

**Customer:** store\_id, create\_date, last\_update, return\_date

4.  Identify at least one field in the detailed table section that will require a custom transformation with a user-defined function and explain why it should be transformed (e.g., you might translate a field with a value of N to No and Y to Yes).

The days\_since\_last\_rental field requires a custom transformation. Not only, is it a new custom field add to both tables, but it requires a function to subtract the current timestamp from rental\_date and find the number of days in the difference. This function is required because it converts a long timestamp value (typically 10 bytes of information) into a single integer value that is more easily used in advertising applications designed by humans.

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

The summary table will provide the business with an uncomplicated way to fetch data for customers who have not rented a movie in a long time. This list of customers could help create a mailing list with coupons to entice people back to the rental business. The detailed table contains valuable data about customer rental habits and would be valuable to the rental business looking to potentially train AI models to know what times and types of movies customers watch. These models could be used to recommend similar movies to customers in the future, with a bias towards customers who have not rented a DVD in a while.

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

This report should be updated about every week in order to consistently find lapsed customers. It might even be ideal to only update the report every two weeks to avoid using the data too regularly for use in customer outreach. Too much outreach may drive lapsed customers away instead of bringing them back into our rental service.

B.  Provide original code for function(s) in text format that perform the transformation(s) you identified in part A4.

-- Calculate number of days since rental's timestamp

CREATE OR REPLACE FUNCTION calculate\_days(rental\_date TIMESTAMP)

RETURNS SMALLINT AS $$

DECLARE

days\_difference SMALLINT;

BEGIN

days\_difference := CAST(CURRENT\_DATE - rental\_date::DATE AS SMALLINT);

RETURN days\_difference;

END;

$$ LANGUAGE plpgsql;

C.  Provide original SQL code in a text format that creates the detailed and summary tables to hold your report table sections.

DROP TABLE IF EXISTS last\_rental\_time\_summary;

CREATE TABLE last\_rental\_time\_summary (

customer\_id SMALLINT,

rental\_id INTEGER,

days\_since\_last\_rental SMALLINT,

PRIMARY KEY (customer\_id, rental\_id)

);

-- Create LastRentalTimeDetailed table

DROP TABLE IF EXISTS last\_rental\_time\_detailed;

CREATE TABLE last\_rental\_time\_detailed (

customer\_id SMALLINT,

store\_id SMALLINT,

create\_date TIMESTAMP,

rental\_id INTEGER,

rental\_date TIMESTAMP,

return\_date TIMESTAMP,

days\_since\_last\_rental SMALLINT,

PRIMARY KEY (customer\_id, rental\_id)

);

D.  Provide an original SQL query in a text format that will extract the raw data needed for the detailed section of your report from the source database.

-- Insert data into LastRentalTimeDetailed

INSERT INTO last\_rental\_time\_detailed (customer\_id, store\_id, create\_date, rental\_id, rental\_date, return\_date, days\_since\_last\_rental)

SELECT

c.customer\_id,

c.store\_id,

c.create\_date,

r.rental\_id,

r.rental\_date,

r.return\_date,

calculate\_days(rental\_date)

FROM

customer c

JOIN

rental r ON c.customer\_id = r.customer\_id

WHERE

r.rental\_date = (

SELECT MAX(r2.rental\_date)

FROM rental r2

WHERE r2.customer\_id = c.customer\_id

);

E.  Provide original SQL code in a text format 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 function

CREATE OR REPLACE FUNCTION copy\_changes\_to\_summary()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO last\_rental\_time\_summary (customer\_id, rental\_id, days\_since\_last\_rental)

VALUES (NEW.customer\_id, NEW.rental\_id, NEW.days\_since\_last\_rental)

ON CONFLICT (customer\_id, rental\_id) -- If the row already exists, update it

DO UPDATE SET

days\_since\_last\_rental = EXCLUDED.days\_since\_last\_rental;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

F.  Provide an original stored procedure in a text format that can be used to refresh the data in both the detailed table and summary table. The procedure should clear the contents of the detailed table and summary table and perform the raw data extraction from part D.

-- Clear both tables' data and insert original data from dvdrental database

CREATE OR REPLACE PROCEDURE refresh\_data() AS $$

BEGIN

DROP TABLE last\_rental\_time\_summary;

DROP TABLE last\_rental\_time\_detailed;

INSERT INTO last\_rental\_time\_detailed (customer\_id, store\_id, create\_date, rental\_id, rental\_date, return\_date, days\_since\_last\_rental)

SELECT

c.customer\_id,

c.store\_id,

c.create\_date,

r.rental\_id,

r.rental\_date,

r.return\_date,

calculate\_days(rental\_date)

--EXTRACT(DAY FROM CURRENT\_DATE - r.rental\_date)::INTEGER AS days\_since\_last\_rental

--CURRENT\_DATE AS last\_update -- edited this so that last\_update field is only updated after calling trigger

FROM

customer c

JOIN

rental r ON c.customer\_id = r.customer\_id

WHERE

r.rental\_date = (

SELECT MAX(r2.rental\_date)

FROM rental r2

WHERE r2.customer\_id = c.customer\_id

);

END;

$$ LANGUAGE plpgsql;

1. Identify a relevant job scheduling tool that can be used to automate the stored procedure.

We can use the chron tool to automate stored functions. We can take our refresh\_data function and place it into a .sql file and choose to schedule it about every two weeks. This allows the function to run automatically without the need to keep the terminal or another application open.