**D326 DVD Rental Business Report**

Harpaal Gohlwar

College of Information Technology, Western Governors University

Instructor: Bruce Johnson

November 11, 2023

**D326 DVD Rental Business Report**

This business report will be using data from a DVD rental sample database. The question I aim to answer is, “which customers generated the most revenue in the last quarter?” I will be limiting this amount to only the top ten customers, as I feel that data would be the most valuable to stakeholders. The DVD rental business will surely be able to benefit from this report. It highlights the top ten revenue generating customers based on a very functional metric of time. This information will allow DVD rental business owners to focus their marketing efforts on customers in this list. Owners can choose to offer special discounts via email to said customers, increasing client return rates and loyalty. Referral bonuses can be sent as well since these customers are the most likely to refer others. Ultimately, this data will help businesses establish firm relationships with regular clients, increase revenue, and potentially gain more loyal customers.

**A1**

To start with the logistics of this report, I will be going over all fields included in my detailed and summary tables. All data I will be discussing in this report was sourced from the PostgreSQL DVD rental sample database. The query tool I used to work with this data was pgAdmin4, a software offering PostgreSQL tools. The fields, also known as columns, of the detailed table are customer\_id, first\_name, last\_name, email, payment\_id, rental\_id, amount, and payment\_date. The fields of the summary table include name, revenue, and email.

**A2**

Now that the detailed and summary table data fields have been established, I would like to discuss their individual data types. Listed below are both tables, their columns, as well as those columns’ data types. As a refresher, the INTEGER data type is simply a positive or negative whole number. VARCHAR is short for variable character. This is just a line of text with a variable length and can be limited with a number in parentheses, which is followed right after the data type declaration. The NUMERIC data type is essentially a decimal value. It is followed by two numbers inside of parentheses. The first value indicates the total amount of significant digits, and the second value indicates how many digits are to be shown after the decimal point.

Detailed Table Summary Table

|  |  |
| --- | --- |
| customer\_id | INTEGER |
| first\_name | VARCHAR(45) |
| last\_name | VARCHAR(45) |
| email | VARCHAR(50) |
| payment\_id | INTEGER |
| rental\_id | INTEGER |
| amount | NUMERIC(5,2) |
| payment\_date | DATE |

|  |  |
| --- | --- |
| name | VARCHAR(100) |
| revenue | NUMERIC(5,2) |
| email | VARCHAR(50) |

**A3**

Of course, without inserting any data from the sample database, the detailed and summary tables would be empty. To populate both tables, the customer and payment relations from the raw database were used. The fields from these tables were directly populated into the detailed table. However, to populate data into the summary table, only the detailed table data was used, since it required all necessary values.

**A4**

Regarding the detailed table, there are two fields that will need to be transformed to correctly populate into the summary table. The first\_name and last\_name columns of the detailed table would not look presentable ‘as is’ in the summary table. One designated name column in the summary table would be more fitting. To do this, I have concatenated the first\_name and last\_name fields of the detailed table. This is simply done by using PostgreSQL’s built in CONCAT function.

Given that the sample database only provided three months of data, it was not necessary to include a WHERE clause in my trigger function. However, I genuinely wanted to ensure that a business would be able to use this report regardless of their size and revenue. To do so, I included a WHERE clause in my code that can select only transactions which have been processed in the last three months. This guarantees that any sized business with any date range of data can run this report and generate a usable summary table that provides a look into their last quarter. The function I used to produce these values is DATE\_TRUNC. I implemented DATE\_TRUNC in this clause to limit the query to the previous three months in relation to the current date of running the trigger. This code is shown below at the end of my report on lines 64-65.

**A5**

Now that all fields and data types have been addressed, it is important to understand the different use cases of the detailed table versus the summary table. The detailed table provides a broader outlook of the business question. This table supplies stakeholders with all relevant information regarding customers purchases. With the inclusion of the rental\_id and payment\_id fields, owners can further investigate transactions depending on specific cases. The summary table produces only pertinent data that directly answers the business question. This table can ultimately be used to target these customers with a variety of marketing tactics, a few of which I have aforementioned.

**A6, F1**

This report should be refreshed every quarter to remain relevant to stakeholders. To do so, the stored procedure should be automated. This can be performed by utilizing pgAgent, a scheduling tool available for PostgreSQL. This tool would be able to execute the stored procedure every quarter and populate fresh usable data. It is important to note that pgAgent is not included with pgAdmin4, therefore, download and installation prior to setup of automation would be necessary (Dias, 2020). Refreshing data every quarter would be the most favorable solution for this scenario due to a few significant reasons. While it is just as valid, and probably more usual, to refresh this data every month, the output of that automation would only provide a limited scope into a business’ customer base. The use of that data would be more practical to find short-term regular clients. The question I aim to answer with this business report requires more raw data that just one month. My objective is to find the most loyal and potentially profitable customers. One quarter is the quintessential interval of time to answer this question, especially given the size of a standard DVD rental business.

**Code**

--A. Business Problem: Which customers generated the most revenue in the last quarter?

/\*C. Provide original SQL code that creates the detailed and summary tables

to hold your report table sections.\*/

-- Detailed Table

CREATE TABLE detailed\_table (

customer\_id INT,

first\_name VARCHAR(45),

last\_name VARCHAR(45),

email VARCHAR(50),

payment\_id INT,

rental\_id INT,

amount NUMERIC(5,2),

payment\_date DATE

);

-- Verifying successful table creation

DROP TABLE detailed\_table;

SELECT \* FROM detailed\_table;

-- Summary Table

CREATE TABLE summary\_table (

name VARCHAR(100),

revenue NUMERIC(5,2),

email VARCHAR(50)

);

-- Verifying successful table creation

DROP TABLE summary\_table;

SELECT \* FROM summary\_table;

/\*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 INTO detailed\_table

(customer\_id, first\_name, last\_name, email, payment\_id, rental\_id, amount, payment\_date)

SELECT c.customer\_id, c.first\_name, c.last\_name, c.email, p.payment\_id, p.rental\_id, p.amount, p.payment\_date

FROM customer c

LEFT JOIN payment p ON c.customer\_id = p.customer\_id;

-- Verifying successful insertion of data and validity

SELECT \* FROM detailed\_table;

/\*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.\*/

/\*B. Provide original code for function(s) that perform the transformation(s) you identified

in part A4.\*/

CREATE OR REPLACE FUNCTION insert\_trigger\_function()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

DELETE FROM summary\_table;

INSERT INTO summary\_table (name, revenue, email)

-- B. Transformation and Aggregate Functions

SELECT CONCAT(first\_name,' ',last\_name) as name, SUM(amount) as revenue, email

FROM detailed\_table

/\* Below is an example of a WHERE clause that would actually run the previous quarters values

based on the current date of running this trigger.

Since we are working with a sample database that ONLY supplies 3 months of data,

no "WHERE payment\_date..." clause is necessary. The below WHERE clause ensures that ANY sized business

with any date range of data can select the top 10 customers from their previous quarter. \*/

/\* WHERE payment\_date >= DATE\_TRUNC('month', CURRENT\_DATE) - INTERVAL '3 months'

AND payment\_date < DATE\_TRUNC('month', CURRENT\_DATE) \*/

GROUP BY name, email

ORDER BY revenue DESC

LIMIT 10;

RETURN NEW;

END;

$$

CREATE TRIGGER new\_summary\_table

AFTER INSERT

ON detailed\_table

FOR EACH STATEMENT

EXECUTE PROCEDURE insert\_trigger\_function();

-- Dropping trigger function to ensure proper function of refresh procedure.

DROP TRIGGER new\_summary\_table ON detailed\_table CASCADE;

-- Testing trigger by inserting data into detailed table

INSERT INTO detailed\_table

VALUES (1, 'Harpaal', 'Gohlwar','[hgohlw1@wgu.edu](mailto:hgohlw1@wgu.edu)', 12, 13, 212.00, '2007-03-21');

-- Validating data insertion into detailed table and proper trigger functionality

DELETE FROM detailed\_table

WHERE first\_name = 'Harpaal';

SELECT COUNT(\*) FROM detailed\_table; --14596 rows of raw data before any insertion

SELECT \* FROM summary\_table;

/\*F. Provide an original stored procedure that can be used to refresh the data in both the

detailed and summary tables. The procedure should clear the contents of the detailed table

and summary table and perform the raw data extraction from part D.\*/

CREATE OR REPLACE PROCEDURE refresh\_tables()

LANGUAGE plpgsql

AS

$$

BEGIN

DELETE FROM detailed\_table;

DELETE FROM summary\_table;

INSERT INTO detailed\_table

(customer\_id, first\_name, last\_name, email, payment\_id, rental\_id, amount, payment\_date)

SELECT c.customer\_id, c.first\_name, c.last\_name, c.email, p.payment\_id, p.rental\_id, p.amount, p.payment\_date

FROM customer c

LEFT JOIN payment p ON c.customer\_id = p.customer\_id;

INSERT INTO summary\_table (name, revenue, email)

SELECT CONCAT(first\_name,' ',last\_name) as name, SUM(amount) as revenue, email

FROM detailed\_table

GROUP BY name, email

ORDER BY revenue DESC

LIMIT 10;

END;

$$

-- Verifying refresh procedure functionality and data validity

SELECT \* FROM detailed\_table;

SELECT COUNT(\*) FROM detailed\_table; -- Should be back to 14596 after calling refresh\_tables procedure

SELECT \* FROM summary\_table; -- Should delete row "Harpaal Gohlwar" after calling refresh\_tables procedure

CALL refresh\_tables();

**References**

Dias, H. (2020). *An Overview of Job Scheduling Tools for PostgreSQL.*

<https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/>

Johnson B. (2021). *D191 Functions Tutorial* [Video]. Western Governors University.

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=0eabc1aa-1e70-43ac-bb1e-addb013a26e8>

Johnson B. (2022). *Triggers & Stored Procedures* [Video]. Western Governors University.

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=cdeef825-e36a-4e9a-a157-aeed0139b592>

PostgreSQL Tutorial. (2011). *PostgreSQL Sample Database.*

<https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/>