Advanced Data Management — D191

VDM2 — VDM2 Task 1: Data Analysis

Ned Imhoff
Student ID: 005917082

Section's A-H are listed below

A. 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.

Our business "Essential Films" would like to know what our rental revenue is by month. This information will help us plan promotional campaigns to boost sales.

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

The report will include a detailed table and a summary table.

The detailed table will include the following fields:

- "rental_id" from the "rental" table
- "rental_date" from the "rental" table
- "payment_id" from the "payment" table
- "amount" from the "payment" table
- "payment_date" from the "payment" table

The summary table will include the following fields:

- "month" to display the calendar year months for which there is data.
- "monthly_revenue" to display the total revenue from rentals in that month.

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

The report will display a detailed table and a summary table with specific data types in each field.

The detailed table data fields are listed as follows:

- "detail_id" (INTEGER)
- "rental_id" (INTEGER)
- "rental_date" (TIMESTAMP - format: YYYY-MM-DD hh:mm:ss)
- "payment_id" (INTEGER)
- "amount" (NUMERIC (5,2))
- "payment_date" (TIMESTAMP - format: YYYY-MM-DD hh:mm:ss)

The summary table data fields are listed as follows:

- "month" (VARCHAR(255)). This is for displaying the name of the month.
- "monthly_revenue" (NUMERIC (10,2)). This is for displaying the total sum of rental revenue per the specific month.

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.

The detailed table will require fields from the following tables:

- "*rental_id*" from the "rental" table
- "*rental_date*" from the "rental" table
- "*payment_id*" from the "payment" table
- "*amount*" from the "payment" table
- "*payment_date*" from the "payment" table

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*).

Both the *rental_date* field and *payment_date* field will be transformed from a TIMESTAMP data type to a VARCHAR data type to display the corresponding month name. This is necessary to allow easy reading and grouping of films rented according to month. It will also allow a method of ensuring a rented film was paid for.

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

The report has two separate tables: the detailed table and the summary table.

The detailed table is intended to provide specific, yet extensive data to help answer the business question. It will contain fields from various tables that are relevant to the business question. This table is intended only for parties conducting the analysis and although it has necessary data, there are no insights gleaned from this table. This table is used to allow for analysis in order to discover the insights to answer the business question. The analysis of this detailed table is put into the summary table.

The summary table is the result of analysis from the detailed table. It holds specific fields that allow stakeholders to easily read the results and directly apply the results to the business question. In the

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

The report looks at monthly revenue by rentals, therefore it should be refreshed at the beginning of every month to gather the new data from the previous month. It would not be necessary at this time to increase the frequency.

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

The following is a function that takes the date value from a field and transforms it into a string value:

```
CREATE OR REPLACE FUNCTION ConvertDateToMonthString(InputDate
TIMESTAMP WITHOUT TIME ZONE)
RETURNS VARCHAR AS $$
BEGIN
    RETURN TO_CHAR(InputDate, 'Month');
END;
$$ LANGUAGE plpgsql;
```

The following are queries used to verify the ‘ConcertDateToMonthString’ function produced the expected results:

```
SELECT ConvertDateToMonthString(payment_date) AS MonthPaymentString
FROM payment;

SELECT ConvertDateToMonthString(rental_date) AS MonthRentalString
FROM rental;
```

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

The following creates the detailed table:

```
CREATE TABLE detail_table (
    detail_id SERIAL PRIMARY KEY,
    rental_id SERIAL,
    rental_date TIMESTAMP,
    payment_id INTEGER,
    amount NUMERIC(5,2),
    payment_date TIMESTAMP
);
```

The following creates the summary table:

```
CREATE TABLE summary_table (
    month_string VARCHAR(255) UNIQUE,
    monthly_revenue NUMERIC(10,2)
);
```

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.

The following extracts the raw data fields from the necessary tables and places them in the detail_table:

```
INSERT INTO detail_table (rental_id, rental_date, payment_id, amount,
payment_date)
SELECT
    r.rental_id,
    r.rental_date,
    p.payment_id,
    p.amount,
    p.payment_date
FROM
    rental r
JOIN
    payment p ON r.rental_id = p.rental_id;
```

After extracting the raw data fields, the detail_table can be checked by running the following:

```
SELECT * FROM detail_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.

The following is a function to enter data into summary_table:

```
CREATE OR REPLACE FUNCTION UpdateSummaryTable()
RETURNS VOID AS $$
BEGIN
	TRUNCATE summary_table;
	INSERT INTO summary_table (month_string, monthly_revenue)
	SELECT
	ConvertDateToMonthString(payment_date) AS month_string,
	SUM(amount) AS monthly_revenue
	FROM
	detail_table
	GROUP BY
	ConvertDateToMonthString(payment_date);
END;
$$ LANGUAGE plpgsql;
```

The following is a trigger function:

```
CREATE OR REPLACE FUNCTION UpdateSummaryTableTrigger()
RETURNS TRIGGER AS $$
BEGIN
	PERFORM UpdateSummaryTable();
	RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

The following creates a trigger that is executed AFTER an INSERT, UPDATE, or DELETE SQL code interacts with the detail_table. The trigger executes the UpdateSummaryTableTrigger function for each statement:

```
CREATE TRIGGER update_summary_trigger
AFTER INSERT OR UPDATE OR DELETE
ON detail_table
FOR EACH STATEMENT
EXECUTE FUNCTION UpdateSummaryTableTrigger();
```

The following can be used to test the trigger:

```
INSERT INTO detail_table (rental_date, payment_id, amount, payment_date)
VALUES ('2023-11-09 12:00:00', 101, 5.99, '2023-11-09 12:30:00');


SELECT * FROM summary_table;
```

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.

The following creates a stored procedure that clears the data from the detail_table and then loads the refined data into the summary_table:

```
CREATE OR REPLACE PROCEDURE RefreshData()
LANGUAGE plpgsql AS $$
BEGIN
    TRUNCATE detail_table;

    INSERT INTO detail_table (rental_id, rental_date, payment_id, amount,
payment_date)
    SELECT
    r.rental_id,
    r.rental_date,
    p.payment_id,
    p.amount,
    p.payment_date
    FROM
    rental r
    JOIN
    payment p ON r.rental_id = p.rental_id;

    TRUNCATE summary_table;
    INSERT INTO summary_table (month_string, monthly_revenue)
    SELECT
    ConvertDateToMonthString(payment_date) AS month_string,
    SUM(amount) AS monthly_revenue
    FROM
    detail_table
    GROUP BY
    ConvertDateToMonthString(payment_date);
END;
$$;
```

The following refreshed the data:

```
CALL RefreshData();
```

The following allows us to view the data in detail_table and summary_table following the RefreshData() procedure:

```
SELECT * FROM detail_table;
SELECT * FROM summary_table;
```

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

A relevant job scheduling tool to automate the stored procedure is *Linux crontab.*

G. Provide a Panopto video recording that includes the presenter and a vocalized demonstration of the functionality of the code used for the analysis.

The following link will take you to the Advanced Data Management D191 folder and link titled “D191 Ned Imhoff”. This is a video recording demonstrating the functionality of the code shown above to answer the business question. The link has also been added to the “link” section of the submission.

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=cf98b2a5-ff7e-40da-9df5-b0c001435afc

H. Acknowledge all utilized sources, including any sources of third-party code, using in-text citations and references. If no sources are used, clearly declare that no sources were used to support your submission.

Dias, H. (2020, February 3). An Overview of Job Scheduling Tools for PostgreSQL. Severalnines. Retrieved from https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/