Performance Assessment: Advanced Data Management (VDM1) Automating Data Integration

Logan S. McQuillan

Department of Information Technology, Western Governors University

Advanced Data Management—D191

Professor Barrie Bradley

October 09, 2022

A. Business Report:

The stakeholders of a local DVD rental company have requested data on the seasonality of DVD rentals because they are interested in which categories perform best during different seasons of the year. For example, does the children's category perform better during summer months because children are home from school during the summer break? Does the horror category rent better during the fall season because of Halloween? The stakeholders want to create category-themed displays at the entrance of each store to promote the DVD categories that bring in the highest number of rentals per season, hoping to increase overall rental sales.

A1. Data Description:

The stakeholders identified six specific fields needed in the detailed section and four in the summary section of the business report. In the detailed section, they need to see data representing the film id, film title, film category id, film category, film rental date, and film rental rate. The summary section must include data representing each season, film category, the number of DVDs rented, and the sales amount in dollars. The summary section will aggregate the detailed data and create insights into which season generates the most DVD rentals and has the highest sales.

The columns or fields in the detailed table contain the dvd_id with a data type of NUMERIC, dvd_title with a data type of VARCHAR(50), dvd_category_id with a data type of NUMERIC, dvd_category with a data type of VARCHAR(50), rental_date with a data type of DATE, and rental_rate with a data type of NUMERIC. The dvd_id field represents the film's identification data from the film table. The dvd_category_id field depicts the film's identification data from the film category table. The dvd_category field represents the name of each category from the category table. The rental_date field describes the data from the rental table showing the date the film was rented. The rental_rate field indicates the rental price associated with each DVD from the film table.

The columns or fields in the summary table contain the season with a data type of VARCHAR(20), dvd_category with a data type of VARCHAR(50), qty_rented with a data type of NUMERIC, and sales_in_dollars with a data type of NUMERIC. The data in the season field indicates the season in which the DVD was rented and was found in the rental table. The data in the dvd_category field represents the category name of each film found in the category table. The qty_rented field represents the number of films rented during each season obtained from the rental table. The sales_in_dollars field indicates the total dollar amount in DVD rentals for each season retrieved from the film table.

A2. Identifying Specific Tables:

The rental, inventory, film, film category, and category tables generate the detailed and summary sections of the business report. The JOIN function will combine the five tables listed to obtain the needed data for the business report.

A3. Identifying Specific Fields:

The film id, film title, film category id, film category, rental date, and rental rate fields create the detailed section of the report. The rental date, film category, quantity of films rented, and rental rate fields produce the summary sections of the report.

A4. Field Transformation:

The rental date column in the detailed section of the report must be transformed to represent the specific season using a transformation function to make the table data more succinct and legible. First, the TIMESTAMP data type in the rental table must be transformed to a DATE data type using the SELECT rental_date::TIMESTAMP::DATE transformation function (PostgreSQL - how to extract date from a timestamp, 2018). After transforming the TIMESTAMP to a DATE data type, the summary table will further transform the DATE data type into its appropriate season by using the CASE function (PostgreSQL case, n.d.). Please see section D of the current paper for the written SQL code.

A5. Business Uses:

The detailed section of the report will answer the stakeholders' original business question regarding the seasonality of DVD rentals and help them better understand their customer's needs and viewing habits. The detailed section of the report provides the stakeholders with the needed information to order additional DVDs that are in high demand and generate the highest sales during each season.

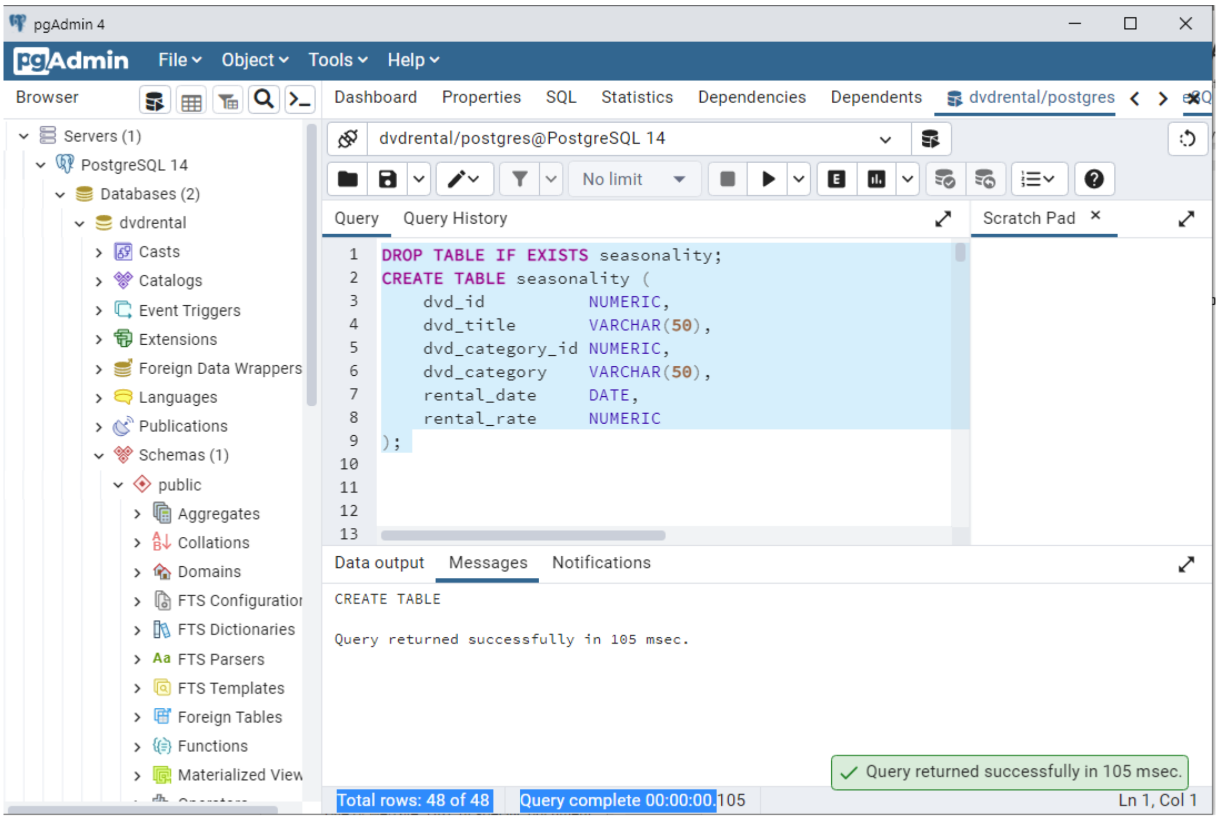
The summary section of the report provides a glimpse into the number of DVDs rented, the specific season, and the sales generated per season. The stakeholders will benefit from the summary section of the report because they can use it to predict future rental habits throughout the year and strategically plan for seasonal events and staffing needs.

A6. Report Freshness:

Because the detailed section of the business report focuses on seasonal DVD rentals, it needs to be refreshed every three months to reflect the appropriate season. At the end of each season, management will need to extract the data from the dvdrental database and update the detailed table with the corresponding data.

B. Creating Tables:

SQL code for the detailed table:



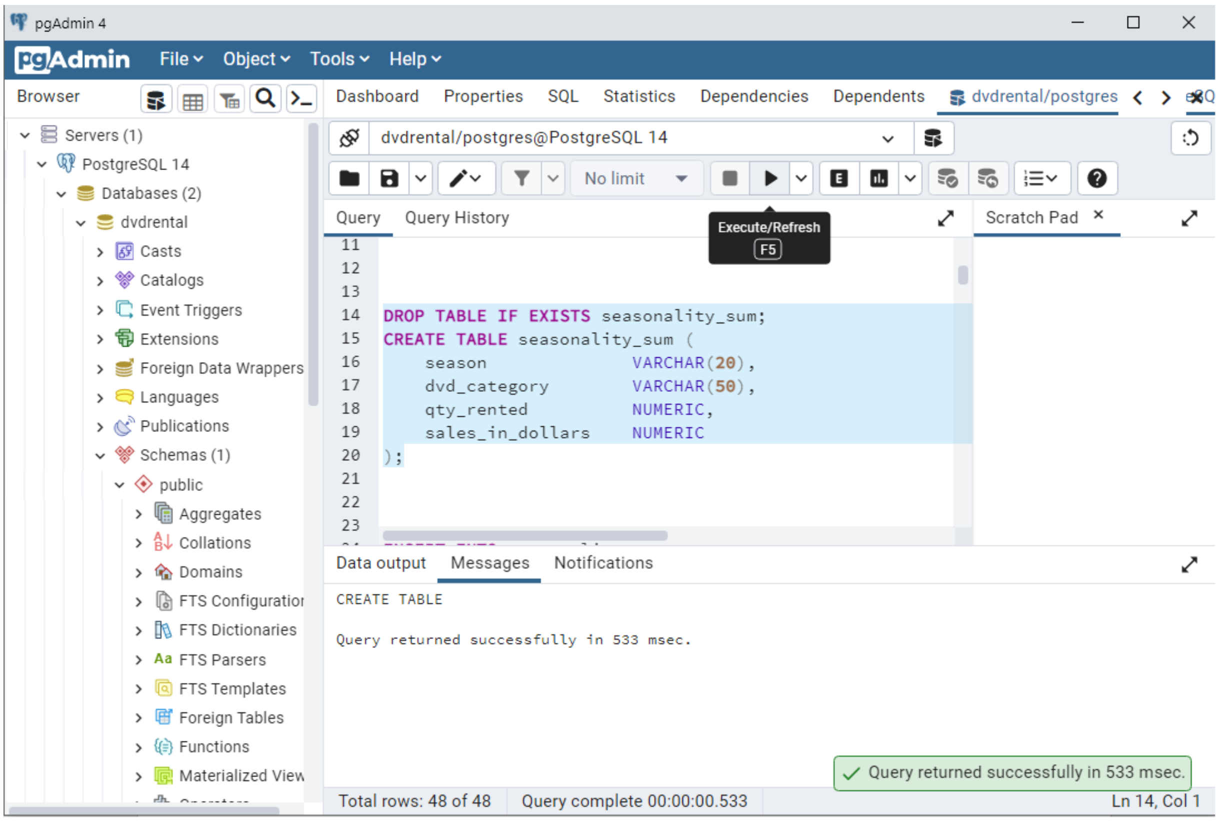
```
DROP TABLE IF EXISTS seasonality;
CREATE TABLE seasonality (
    dvd_id          NUMERIC,
    dvd_title       VARCHAR(50),
    dvd_category_id NUMERIC,
    dvd_category    VARCHAR(50),
    rental_date     DATE,
    rental_rate     NUMERIC
);
```

```
CREATE TABLE

Query returned successfully in 105 msec.
```

Query returned successfully in 105 msec.

Total rows: 48 of 48   Query complete 00:00:00.105   Ln 1, Col 1

SQL code for the summary table:



```
DROP TABLE IF EXISTS seasonality_sum;
CREATE TABLE seasonality_sum (
    season              VARCHAR(20),
    dvd_category        VARCHAR(50),
    qty_rented          NUMERIC,
    sales_in_dollars    NUMERIC
);
```

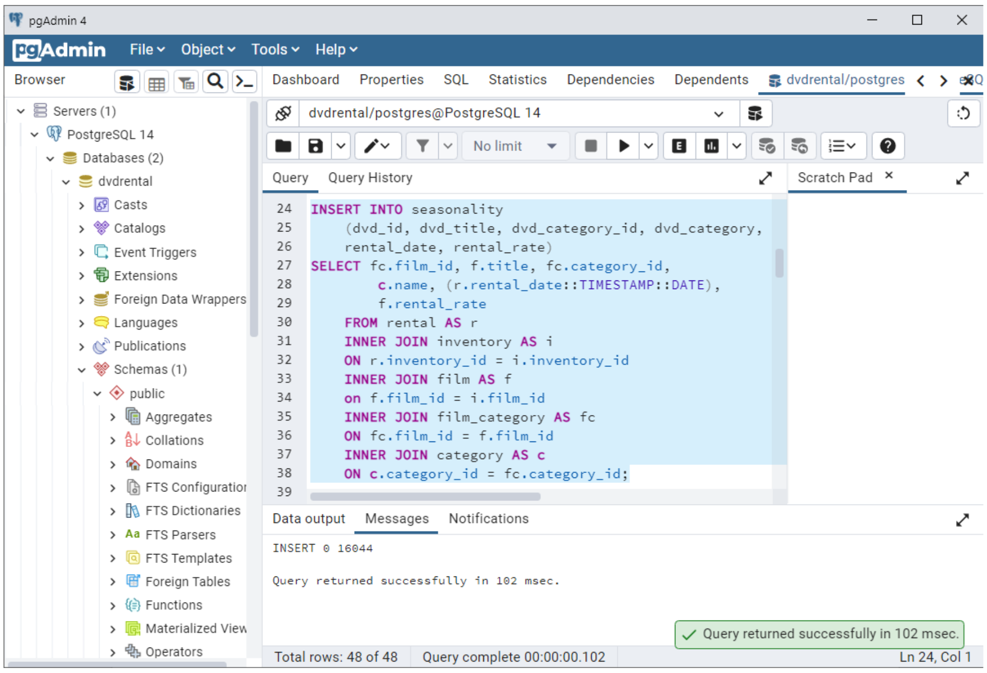
```
CREATE TABLE

Query returned successfully in 533 msec.
```

Query returned successfully in 533 msec.

Total rows: 48 of 48   Query complete 00:00:00.533   Ln 14, Col 1

C. SQL Query:

Please see the image below for the SQL code that generates the detailed section of the report. The query was successfully executed, as indicated by the green notification box near the bottom.



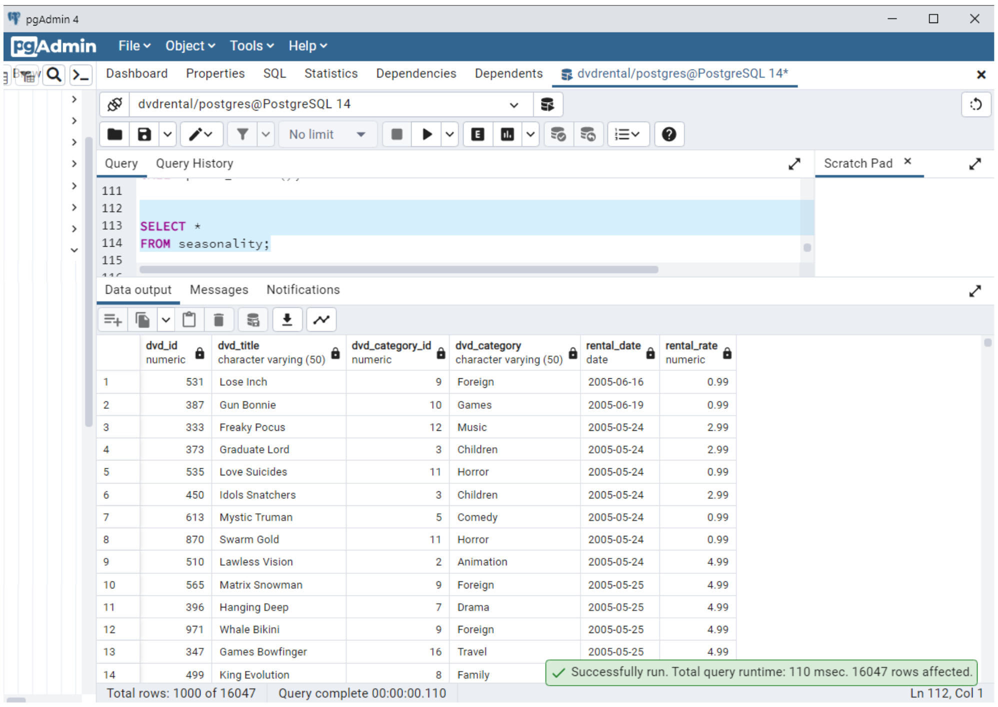
```sql
INSERT INTO seasonality
    (dvd_id, dvd_title, dvd_category_id, dvd_category,
    rental_date, rental_rate)
SELECT fc.film_id, f.title, fc.category_id,
        c.name, (r.rental_date::TIMESTAMP::DATE),
        f.rental_rate
    FROM rental AS r
    INNER JOIN inventory AS i
    ON r.inventory_id = i.inventory_id
    INNER JOIN film AS f
    on f.film_id = i.film_id
    INNER JOIN film_category AS fc
    ON fc.film_id = f.film_id
    INNER JOIN category AS c
    ON c.category_id = fc.category_id;
```

```
INSERT 0 16044

Query returned successfully in 102 msec.
```

Query returned successfully in 102 msec.

Total rows: 48 of 48 Query complete 00:00:00.102 Ln 24, Col 1

All fields in the seasonality table returned accurately, with the appropriate transformation of the rental date data type.



```sql
SELECT *
FROM seasonality;
```

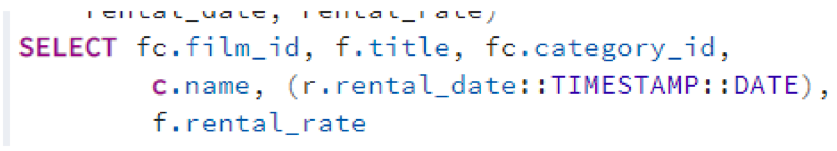
	dvd_id numeric	dvd_title character varying (50)	dvd_category_id numeric	dvd_category character varying (50)	rental_date date	rental_rate numeric
1	531	Lose Inch	9	Foreign	2005-06-16	0.99
2	387	Gun Bonnie	10	Games	2005-06-19	0.99
3	333	Freaky Pocus	12	Music	2005-05-24	2.99
4	373	Graduate Lord	3	Children	2005-05-24	2.99
5	535	Love Suicides	11	Horror	2005-05-24	0.99
6	450	Idols Snatchers	3	Children	2005-05-24	2.99
7	613	Mystic Truman	5	Comedy	2005-05-24	0.99
8	870	Swarm Gold	11	Horror	2005-05-24	0.99
9	510	Lawless Vision	2	Animation	2005-05-24	4.99
10	565	Matrix Snowman	9	Foreign	2005-05-25	4.99
11	396	Hanging Deep	7	Drama	2005-05-25	4.99
12	971	Whale Bikini	9	Foreign	2005-05-25	4.99
13	347	Games Bowfinger	16	Travel	2005-05-25	4.99
14	499	King Evolution	8	Family		

Successfully run. Total query runtime: 110 msec. 16047 rows affected.

Total rows: 1000 of 16047 Query complete 00:00:00.110 Ln 112, Col 1

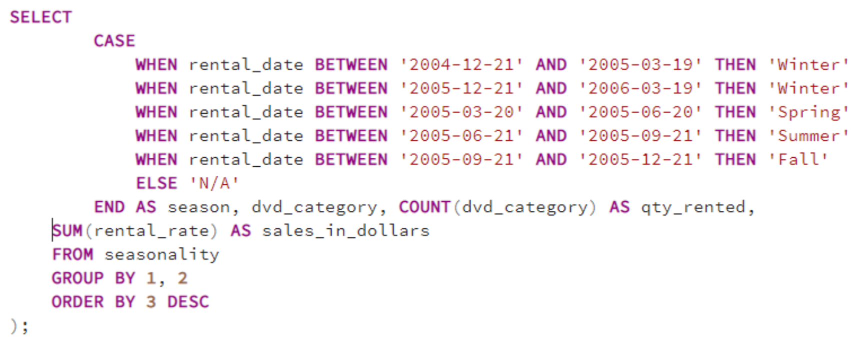
D. Functions:

The SQL code transforms the TIMESTAMP data type to the DATE data type in the detailed table. Please see section A4 for more information and in-text citation.



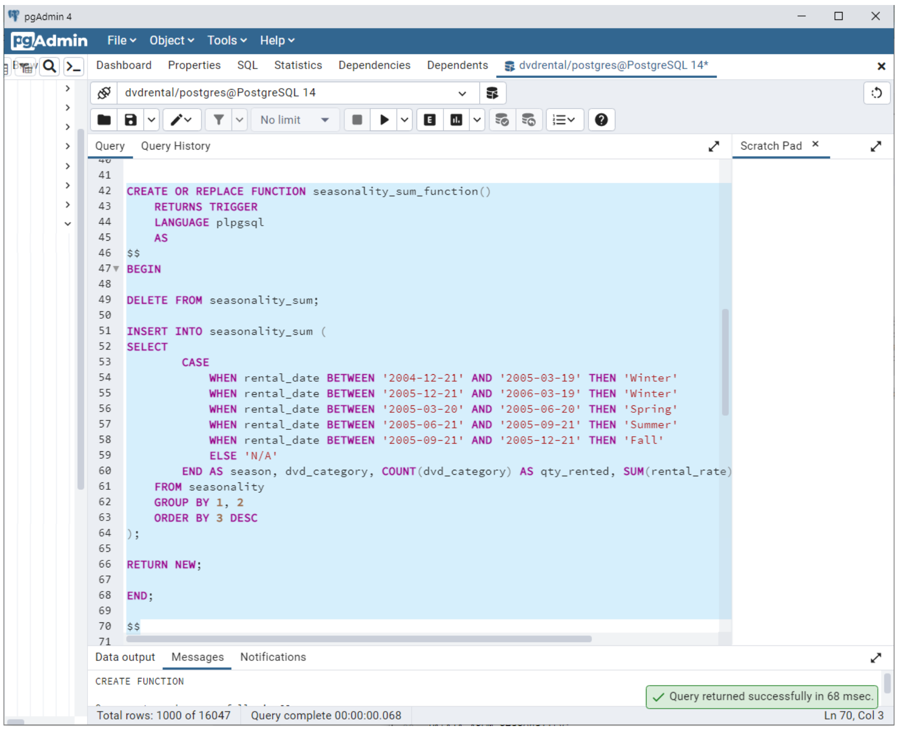
```
SELECT fc.film_id, f.title, fc.category_id,
       c.name, (r.rental_date::TIMESTAMP::DATE),
       f.rental_rate
```

The SQL CASE function transforms the TIMESTAMP data type into appropriate seasons, such as winter, spring, summer, or fall.



```
SELECT
        CASE
            WHEN rental_date BETWEEN '2004-12-21' AND '2005-03-19' THEN 'Winter'
            WHEN rental_date BETWEEN '2005-12-21' AND '2006-03-19' THEN 'Winter'
            WHEN rental_date BETWEEN '2005-03-20' AND '2005-06-20' THEN 'Spring'
            WHEN rental_date BETWEEN '2005-06-21' AND '2005-09-21' THEN 'Summer'
            WHEN rental_date BETWEEN '2005-09-21' AND '2005-12-21' THEN 'Fall'
            ELSE 'N/A'
        END AS season, dvd_category, COUNT(dvd_category) AS qty_rented,
    SUM(rental_rate) AS sales_in_dollars
    FROM seasonality
    GROUP BY 1, 2
    ORDER BY 3 DESC
);
```

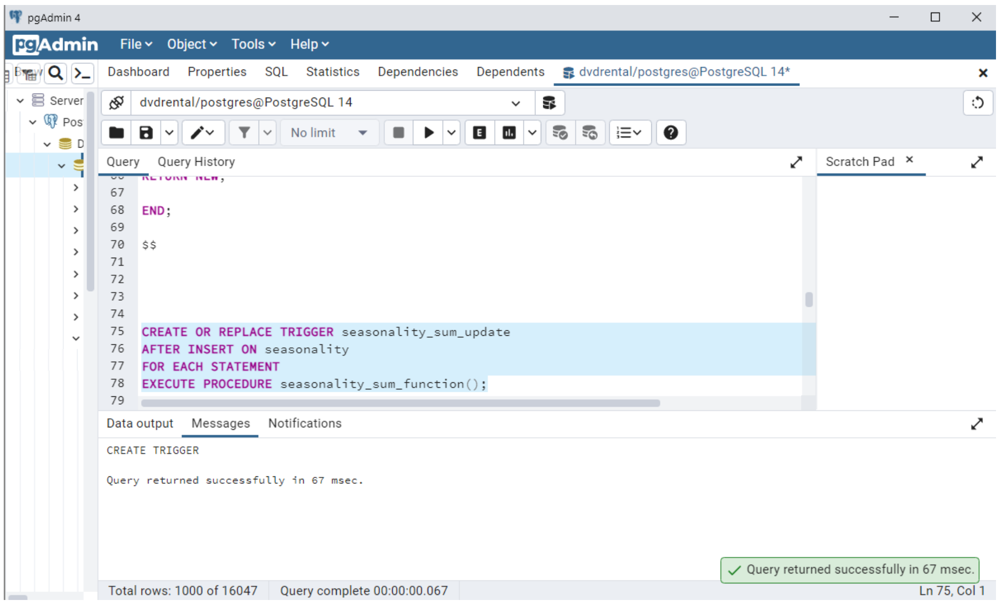
E. Triggers:



```
CREATE OR REPLACE FUNCTION seasonality_sum_function()
    RETURNS TRIGGER
    LANGUAGE plpgsql
    AS
$$
BEGIN

DELETE FROM seasonality_sum;

INSERT INTO seasonality_sum (
SELECT
        CASE
            WHEN rental_date BETWEEN '2004-12-21' AND '2005-03-19' THEN 'Winter'
            WHEN rental_date BETWEEN '2005-12-21' AND '2006-03-19' THEN 'Winter'
            WHEN rental_date BETWEEN '2005-03-20' AND '2005-06-20' THEN 'Spring'
            WHEN rental_date BETWEEN '2005-06-21' AND '2005-09-21' THEN 'Summer'
            WHEN rental_date BETWEEN '2005-09-21' AND '2005-12-21' THEN 'Fall'
            ELSE 'N/A'
        END AS season, dvd_category, COUNT(dvd_category) AS qty_rented, SUM(rental_rate)
    FROM seasonality
    GROUP BY 1, 2
    ORDER BY 3 DESC
);

RETURN NEW;

END;

$$
```

```
CREATE FUNCTION

Query returned successfully in 68 msec.

Total rows: 1000 of 16047    Query complete 00:00:00.068    Ln 70, Col 3
```

The CREATE TRIGGER function will update the seasonality summary table with the data extracted from the seasonality table.



```
RETURN NEW;

END;

$$




CREATE OR REPLACE TRIGGER seasonality_sum_update
AFTER INSERT ON seasonality
FOR EACH STATEMENT
EXECUTE PROCEDURE seasonality_sum_function();
```

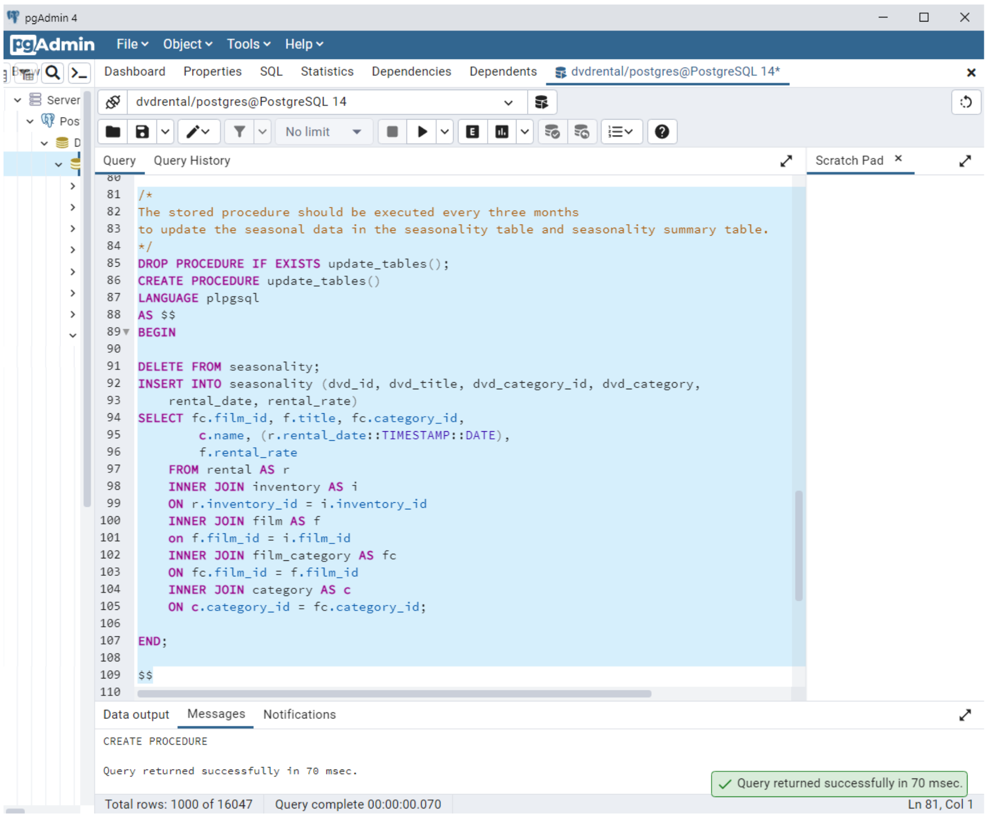
```
CREATE TRIGGER

Query returned successfully in 67 msec.
```

Query returned successfully in 67 msec.

Total rows: 1000 of 16047 Query complete 00:00:00.067 Ln 75, Col 1

This CREATE OR REPLACE TRIGGER will cause the above CREATE TRIGGER function to execute when new data is entered into the seasonality table (PostgreSQL create trigger, n.d.).

F. Stored Procedure:



```
/*
The stored procedure should be executed every three months
to update the seasonal data in the seasonality table and seasonality summary table.
*/
DROP PROCEDURE IF EXISTS update_tables();
CREATE PROCEDURE update_tables()
LANGUAGE plpgsql
AS $$
BEGIN

DELETE FROM seasonality;
INSERT INTO seasonality (dvd_id, dvd_title, dvd_category_id, dvd_category,
    rental_date, rental_rate)
SELECT fc.film_id, f.title, fc.category_id,
        c.name, (r.rental_date::TIMESTAMP::DATE),
        f.rental_rate
    FROM rental AS r
    INNER JOIN inventory AS i
    ON r.inventory_id = i.inventory_id
    INNER JOIN film AS f
    on f.film_id = i.film_id
    INNER JOIN film_category AS fc
    ON fc.film_id = f.film_id
    INNER JOIN category AS c
    ON c.category_id = fc.category_id;

END;

$$
```

```
CREATE PROCEDURE

Query returned successfully in 70 msec.
```

Query returned successfully in 70 msec.

Total rows: 1000 of 16047 Query complete 00:00:00.070 Ln 81, Col 1

The CREATE PROCEDURE update_tables() statement will refresh the data in both the seasonality table and the seasonality summary table. The procedure will first delete the content of the two tables, then perform the extraction, transformation, and load process described in part C of this paper (PostgreSQL create procedure, n.d.). When the two tables require updating, the CALL statement will be invoked, triggering the procedure, and refreshing the detailed and summary tables.

F1. Data Freshness:

The stored procedure must be run at least once every three months to keep the data on a seasonal schedule. The seasonality and seasonality summary tables in the business report emphasize which DVD category generates the most rentals and highest rental sales for each season, so both tables need to be refreshed each season, e.g., every three months or four times per year.

G. Provide a Panopto video.

The Panopto video will be uploaded to the Panopto drop box, and then the URL will be uploaded with my final submission.

H. Record the web sources you used to acquire data or segments of third-party code to support the application if applicable.

I downloaded PostgreSQL on my computer. I did not use the Labs on Demand Assessment Environment and DVD Database to complete the assessment. I also downloaded the required DVD Rental Sample Database from the PostgreSQL Tutorial website and the printable ER diagram to help design the table entities, attributes, and relationships (PostgreSQL sample database, n.d.).

References

PostgreSQL case. PostgreSQL Tutorial. (n.d.). Retrieved October 7, 2022, from https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-case/

PostgreSQL create procedure. PostgreSQL Tutorial. (n.d.). Retrieved October 8, 2022, from https://www.postgresqltutorial.com/postgresql-plpgsql/postgresql-create-procedure/

PostgreSQL create trigger. PostgreSQL Tutorial. (n.d.). Retrieved October 6, 2022, from https://www.postgresqltutorial.com/postgresql-triggers/creating-first-trigger-postgresql/

PostgreSQL - how to extract date from a timestamp? TablePlus. (2018, July 25). Retrieved October 5, 2022, from https://tableplus.com/blog/2018/07/postgresql-how-to-extract-date-from-timestamp.html

PostgreSQL sample database. PostgreSQL Tutorial. (n.d.). Retrieved October 8, 2022, from https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/