VDM1 Task 1: Automating Data

Sean P Kenney

Western Governors University

WGU Student ID#001041212

A. Real-World Business Report

Polaris Movie Rentals is interested in opening an additional DVD rental store location within the same city as their current rental store with the most DVD rentals. Using the attached data sets and accompanying dictionaries, Polaris Movie Rentals and its stakeholders would like to determine which current rental store location is most popular based on rental count to influence where the new rental location should be opened.

A.1. Data Used For the Report

The data that will be used for creating the business report will need to include address details on the current rental store locations. Additionally, the data will need to include the total rental count for each of the rental store locations. This will allow Polaris Movie Rentals and its stakeholders to see all current store locations and their corresponding rental totals so they can make an informed decision on where to open the new DVD rental store location.

A.2. Specific Tables:

Multiple tables will need to be leveraged from the provided dataset to produce the necessary data for both the detailed and summary sections of the business report. The five specific tables that will need to be used for the business report are the ***rental*** table, ***staff*** table, ***store*** table, ***address*** table, and ***city*** table.

A.3. Specific Fields:

The eight specific fields that will need to be included in the detailed section of the business report are ***rental_id***, ***rental_date***, ***staff_id***, ***store_id***, ***address_id***, ***city_id***, ***address***, and ***city*** fields. The two specific fields that will need to be included in the summary section of the report are ***store_location*** and ***total_rentals*** fields.

A.4. Custom Transformation:

The ***address*** and ***city*** fields within the ***detailed*** section will need to be transformed by concatenating both fields into a single ***store_location*** field. This will allow stakeholders to easily filter the data by the store address and city for circumstances where there might already be multiple rental locations within a single city in the future. The ***address*** and ***city*** fields will be separated in the ***store_location*** field by a comma and space to improve user readability.

A.5. Business Uses:

Polaris Movie Rentals and its stakeholders will use the ***detailed*** section of the report to view all rentals and their corresponding rental store address and city. Stakeholders will be able to filter the ***detailed*** section based on the different fields within the ***detailed*** table and see all corresponding store rental data. They can also use the ***rental_date*** field to review rental totals for a specific day, week, or month. Polaris Movie Rentals and its stakeholders will use the ***summary*** section to get a quick-view of each store location and their corresponding total rental count. Since the ***summary*** table will be ordered by ***total_rentals*** in descending order, stakeholders can quickly see which store is currently the most popular rental location.

A.6. Report Frequency:

Polaris Movie Rentals and its stakeholders will want to initially run the business report to determine where the new store location should be located. Afterwards, as Polaris Movie Rentals continues to expand their customer base with new movie rental locations, the business report should be refreshed every six months after a new location is opened for stakeholders to review an updated business report. Since the ***summary*** section of the business report includes the total rental count for each store location, re-running the report every six months will refresh the data to remain relevant to Polaris Movie Rentals and its stakeholders and allow them to compare total rentals between the existing and new store locations.

B. Creating Tables:

The following is the SQL code that will create and display the empty tables for holding the ***detailed*** and ***summary*** sections of the business report:

```sql
----Create Tables----
----Create Detailed Table----
-- This creates an empty detailed table to be used
-- for the detailed section of the business report

DROP TABLE IF EXISTS detailed;
```

```
CREATE TABLE detailed (
      rental_id integer,
      rental_date timestamp,
      staff_id integer,
      store_id integer,
      address_id integer,
      address varchar (50),
      city_id integer,
      city varchar (50)
);

-- This will display the empty detailed table

SELECT * FROM detailed;

----Create Summary Table----
-- This creates an empty summary table to be used
-- for the summary section of the business report

DROP TABLE IF EXISTS summary;
CREATE TABLE summary (
      store_location varchar (160),
      total_rentals integer
);

-- This will display the empty summary table

SELECT * FROM summary;
```

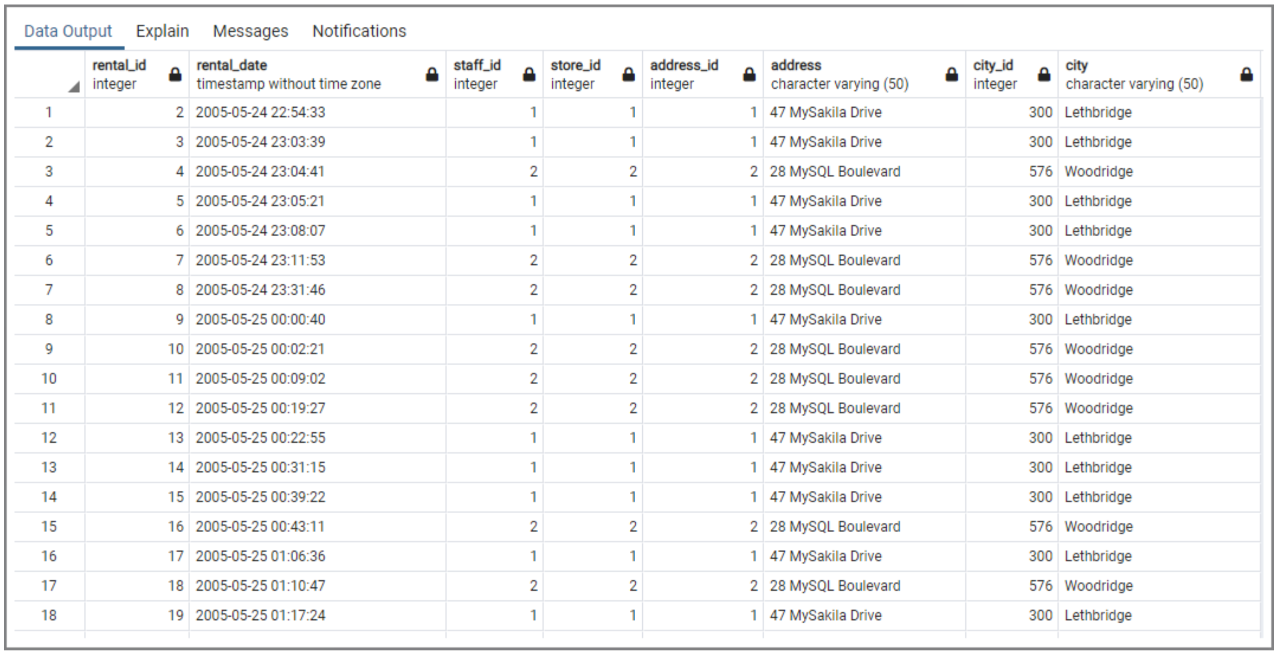
C. Extracting Raw Data:

The following is the SQL query that will extract the raw data needed for the ***detailed*** section of the business report:

```
----Insert Data into the Detailed Table----
-- This will load the data from the rental, staff, store,
-- address, and city tables into the detailed table

INSERT INTO detailed (
      rental_id,
      rental_date,
      staff_id,
      store_id,
      address_id,
      address,
      city_id,
      city
)

SELECT
      rental.rental_id, rental.rental_date,
      staff.staff_id,
      store.store_id,
      address.address_id, address.address,
      city.city_id, city.city
```

```
FROM rental
INNER JOIN staff ON rental.staff_id = staff.staff_id
INNER JOIN store ON staff.store_id = store.store_id
INNER JOIN address ON store.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
;

-- This will display the filled detailed table

SELECT * FROM detailed;
```

Data accuracy can be verified by looking at the data output for the ***detailed*** section of the business report. In reviewing the data output (refer to Figure 1 below), first you can see all eight fields present within the ***detailed*** table. Additionally, all eight fields are in the correct order and have the exact datatypes as defined within the SQL query. Lastly, there are no missing values within the eight fields.



	rental_id integer	rental_date timestamp without time zone	staff_id integer	store_id integer	address_id integer	address character varying (50)	city_id integer	city character varying (50)
1	2	2005-05-24 22:54:33	1	1	1	47 MySakila Drive	300	Lethbridge
2	3	2005-05-24 23:03:39	1	1	1	47 MySakila Drive	300	Lethbridge
3	4	2005-05-24 23:04:41	2	2	2	28 MySQL Boulevard	576	Woodridge
4	5	2005-05-24 23:05:21	1	1	1	47 MySakila Drive	300	Lethbridge
5	6	2005-05-24 23:08:07	1	1	1	47 MySakila Drive	300	Lethbridge
6	7	2005-05-24 23:11:53	2	2	2	28 MySQL Boulevard	576	Woodridge
7	8	2005-05-24 23:31:46	2	2	2	28 MySQL Boulevard	576	Woodridge
8	9	2005-05-25 00:00:40	1	1	1	47 MySakila Drive	300	Lethbridge
9	10	2005-05-25 00:02:21	2	2	2	28 MySQL Boulevard	576	Woodridge
10	11	2005-05-25 00:09:02	2	2	2	28 MySQL Boulevard	576	Woodridge
11	12	2005-05-25 00:19:27	2	2	2	28 MySQL Boulevard	576	Woodridge
12	13	2005-05-25 00:22:55	1	1	1	47 MySakila Drive	300	Lethbridge
13	14	2005-05-25 00:31:15	1	1	1	47 MySakila Drive	300	Lethbridge
14	15	2005-05-25 00:39:22	1	1	1	47 MySakila Drive	300	Lethbridge
15	16	2005-05-25 00:43:11	2	2	2	28 MySQL Boulevard	576	Woodridge
16	17	2005-05-25 01:06:36	1	1	1	47 MySakila Drive	300	Lethbridge
17	18	2005-05-25 01:10:47	2	2	2	28 MySQL Boulevard	576	Woodridge
18	19	2005-05-25 01:17:24	1	1	1	47 MySakila Drive	300	Lethbridge

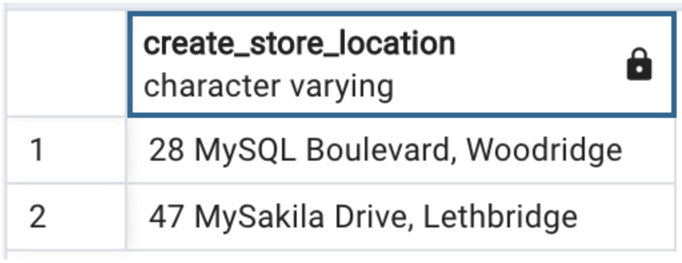
Figure 1: Data Accuracy Check - Screenshot of SQL Query Output

D. Data Transformation:

Below is the SQL function code for transforming the ***address*** and ***city*** fields into a single field, allowing stakeholders to easily filter the data by the store location and improving readability.

```
-- CONCAT FUNCTION --
-- This will input the address and city fields through a
-- concatenating function to create a single field

CREATE OR REPLACE FUNCTION create_store_location(address varchar (50),
city varchar (50))
      RETURNS varchar (100)
      LANGUAGE plpgsql
AS
$$
DECLARE store_location varchar (100);
BEGIN
      SELECT CONCAT_WS(', ', address, city) INTO store_location;
      RETURN store_location;
END;
$$

-- This tests the function with the city and address
-- fields from the detailed table

SELECT DISTINCT create_store_location(address, city) FROM detailed
```

Figure 2 below displays the data output results from testing the ***create_store_location()*** function with the ***detailed*** table, combining the ***address*** and ***city*** fields and separating them with a comma and space for simple readability.



	create_store_location character varying
1	28 MySQL Boulevard, Woodridge
2	47 MySakila Drive, Lethbridge

Figure 2: Verifying create_store_location function

E. Creating Trigger:

To create a trigger that continually updates the ***summary*** table, as data is inserted into the ***detailed*** table, we must first create a trigger function. The trigger will first remove all current data within the ***summary*** table. It will then insert the ***store_location*** field by leveraging the ***create_store_location()*** function to combine the ***address*** and ***city*** fields from the ***detailed*** table (refer to section D: Data Transformation). Next, the trigger will insert the ***total_rentals*** field by counting the total rentals from the ***detailed*** table based on ***store_location***. Lastly, the trigger will group by the ***store_location*** and order the ***summary*** table by ***total_rentals*** in descending order.

The code for the ***update_summary()*** trigger function is:

```
-- CREATE TRIGGER FUNCTION --
-- This creates the trigger function for
-- updating the summary table

CREATE FUNCTION update_summary()
      RETURNS TRIGGER
      LANGUAGE PLPGSQL
AS $$
BEGIN
      -- Removes any previous data from summary table

      DELETE FROM summary;
      INSERT INTO summary (
            store_location,
            total_rentals
      )

      -- This will use the create_store_location function to
      -- create the store_location field in the summary table

      SELECT create_store_location(address, city) AS store_location,
      COUNT (create_store_location(address, city)) AS total_rentals
      FROM detailed
      GROUP BY store_location
      ORDER BY total_rentals DESC;

RETURN NEW;
END;
$$
```

The second component needed is the trigger event function that will automatically call the ***update_summary()*** trigger function whenever new data is inserted into the ***detailed*** tabled. The code for the ***update_summary_trigger*** trigger event function is:

```
-- CREATE TRIGGER EVENT FUNCTION --
-- This event trigger will call the update_summary() trigger
-- function whenever data is inserted into the detailed table

CREATE TRIGGER update_summary_trigger
AFTER INSERT ON detailed
FOR EACH STATEMENT
EXECUTE PROCEDURE update_summary();
```

F. Creating Procedure:

The below code will create/replace the ***tables_refresh()*** procedure. This procedure will first remove all existing data within the ***detailed*** table. Next, it will re-insert all data into the ***detailed*** table from the ***staff***, ***store***, ***address***, and ***city*** tables. Lastly, because this procedure is inserting data into the ***detailed*** table, the ***tables_refresh()*** procedure will trigger the

update_summary_trigger (refer to section E: Creating Trigger). The ***update_summary_trigger*** will call the ***update_summary()*** trigger function, which will remove all existing data within the ***summary*** table and re-insert the ***summary*** table fields.

After Polaris Movie Rentals and its stakeholders initially review the business report to determine the new store location, the ***tables_refresh()*** procedure can be executed six months after the new store location is opened to analyze the ***total_rentals*** between the existing locations and the new location. This will allow Polaris Movie Rentals and its stakeholders to determine how successful the new store location is compared to the existing locations and ultimately help determine where the next store location should be.

```
-- CREATE STORED PROCEDURES --
-- This will create the procedures that will
-- refresh the detailed table which will then
-- trigger the update_summary_trigger to update
-- the summary table

CREATE OR REPLACE PROCEDURE tables_refresh()
LANGUAGE PLPGSQL
AS $$

BEGIN
      -- This clears all data from the detailed table

      DELETE FROM detailed;

      ----Insert Data into the Detailed Table----
      -- This will load the data from the rental, staff, store,
      -- address, and city tables into the detailed table

      INSERT INTO detailed (
            rental_id,
            rental_date,
            staff_id,
            store_id,
            address_id,
            address,
            city_id,
            city
      )

      SELECT
            rental.rental_id, rental.rental_date,
            staff.staff_id,
            store.store_id,
            address.address_id, address.address,
            city.city_id, city.city
```

```
        FROM rental
        INNER JOIN staff ON rental.staff_id = staff.staff_id
        INNER JOIN store ON staff.store_id = store.store_id
        INNER JOIN address ON store.address_id = address.address_id
        INNER JOIN city ON address.city_id = city.city_id
        ;
END;
$$

-- This runs the stored procedure --
-- This should be executed 6 months after a new store
-- location is opened to evaluate new rental data

CALL tables_refresh();
```

F.1. Automatic Scheduler:

Polaris Movie Rentals currently uses PostgreSQL as its database management system. Since this system does not currently offer an integrated job scheduler, Polaris Movie Rentals will need to download and install pgAgent, an independent job scheduler for PostgreSQL, to have the ***tables_refresh()*** stored procedure run on a schedule. Once integrated, Polaris Movie Rentals and stakeholders can create a PgAgent Job with a single step that includes the following code:

```
-- This runs the stored procedure --

CALL tables_refresh();
```

Once the pgAgent Job is created, stakeholders can update the scheduler to automatically run the stored procedure on the same day every month to ensure data freshness.

G. Panopto Video:

Refer to ***VDM1 - Task1 - Automating Data Integration - Sean Kenney*** video file within the Advanced Data Management D191 | D326 (Student Creators) folder in Panopto.

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=92d315aa-f781-4adc-8b9f-af320153e789

H. Web Sources:

No web sources were used to acquire data or segments of third-party code.

I. Sources:

No sources where used.