Steven J Hull
D326 - Advanced Data Management
Performance Assessment

A. A1:
The DVD rental business is highly competitive, and staying on top of what the customers truly want to watch is a top priority of all rental companies. The question in any retail setting has always been: What does the customer want? In the DVD rental business, the customer wants to watch movies that interest them, usually new, popular releases. The purpose of this project is to provide information about what the customers deem to be, in general, the best movies to watch. With this information, the organization will be able to see the most popular movies. Using that information, they will be able to stock up on more of the exact same movies. The end result, or course, being to increase revenue for the organization.

This business report, created from the DVD Dataset, will provide a detailed table that provides the Store ID, Inventory ID, Film ID, Movie Title, Customer ID, and Rental Date, in both easily readable format and as a time stamp. It will also provide a summary table containing the Store ID, Movie Title, and Rental Count for the top movie rented at each store over a determined period, in this case 30 days. This information can help the organization extrapolate what movies are most popular during the examined period using the summary table. Other data can also be drawn from the detailed table as needed, such as how often a customer rents movies during that same given time period.

A2:
The data fields used:
The detailed table:
store_id – An integer. This is the identifying number of each store.
inventory_id – An integer. This is the identifying number of the stocked item.
film_id – An integer. This is the identifying number of the movie provided by the film distributor.
title – A variable length string. This id the title of the movie.
customer_id – An integer. This is the identifying number of the customer.
rental_date – A variable length string. This is the string representation of the timestamp in date format.
rental_date_ts – A time stamp. This is the date in necessary timestamp format for sorting purposes.

The summary table:
store_id – An integer. This is the identifying number of each store. Necessary for knowing what movies are most popular at which store.
title – A variable length string. This id the title of the movie.
rentals_count – An integer. This way the organization will know how often a movie was rented.

A3:
I used the rental table, the inventory table, and the film table from the provided dataset to fill my detailed table and summary table with the necessary data for this report. The detailed table contains the customer_id and rental_date from the rental table, the store_id and inventory_id from the inventory table, and the film_id and title from the film table. The summary table contains store_id and title from

the detailed table. It also contains a rental_count that does not source from any table, but rather contains a count of how many times a movie was rented during the given time frame.

A4:
I did a custom transformation on the rental_date of the detailed table with a user defined function to translate it from a time stamp to a variable length string to make it more user-friendly and easily readable. Example: Time stamp “2005-06-24 23:45:39” would be transformed to a much more human-readable form of “Jun 24, 2005”.

A5:
The detailed table provides enough information to be able to determine how many times a customer rents movies and what movies they are, as well as the date and time that a customer rents movies, during the given time period. It also provides the Store ID where the movies are rented from, and the Inventory ID and Film ID for re-ordering or increasing inventory. All of this information can be used to help the organization dynamically adjust inventory to suit the customers needs, which will increase revenue.

The summary table extrapolates the necessary data from the detailed table to provide the organization with information to determine the most popular movie rented from each store during the given time period. In this case, the summary table provided that information for the single top movie, but this can easily be adjusted to accommodate a larger number if desired by the organization. This way, the organization can increase inventory on the most popular movies to increase revenue.

A6:
This report should be refreshed daily to ensure that the data stays relevant for the stakeholders. New movies are released frequently, so daily refreshing will ensure the organization keeps enough popular rentals in inventory. Furthermore, people talk with each other about the movies they like. A daily refreshing will track if a particular movie suddenly becomes more popular due to “word of mouth”, and allow the organization to possibly bring in more inventory to satisfy customers needs, thereby increasing revenue. The refreshing process can be completed automatically using pg_cron as a PostgreSQL extension or cron with shell scripts.

B:
```
-- Section B --
-- Transformation Function --
CREATE OR REPLACE FUNCTION readable_date(rental_date timestamp)
	RETURNS varchar(100)
	LANGUAGE plpgsql
AS
$$
DECLARE clean_date varchar(100);
BEGIN
	SELECT TO_CHAR(rental_date, 'Mon DD, YYYY') INTO clean_date;
	RETURN clean_date;
END;
```

```
$$
```

C:

```
-- Section C --
-- Create Detailed Table --
CREATE TABLE detailed_table (
	store_id int,
	inventory_id int,
	film_id int,
	title varchar(1000),
	customer_id int,
	rental_date varchar(100),
	rental_date_ts timestamp);

-- Create Summary Table --
CREATE TABLE summary_table (
	store_id int,
	title varchar(1000),
	rentals_count int);
```

D:

```
-- Section D --
-- SQL Query to extract raw data from source database and insert into the detailed table --
INSERT INTO detailed_table
	SELECT
		i.store_id,
		i.inventory_id,
		f.film_id,
		f.title,
		r.customer_id,
		readable_date(r.rental_date) AS rental_date,
		r.rental_date AS rental_date_ts
	FROM rental r
	INNER JOIN inventory i ON r.inventory_id = i.inventory_id
	INNER JOIN film f ON i.film_id = f.film_id
	WHERE r.rental_date BETWEEN '2005-05-24' AND '2005-06-24'
	ORDER BY r.rental_date
```

E:

```
-- Section E --
--INSERT INTO trigger function for when data is inserted into the detailed_table --
CREATE OR REPLACE FUNCTION insert_trigger_function()
	RETURNS TRIGGER
	LANGUAGE plpgsql
AS
$$
```

```
BEGIN
	DELETE FROM summary_table;

	WITH top_movies AS (
	SELECT
	dt.store_id,
	dt.title,
			COUNT(*) AS rental_count,
	ROW_NUMBER() OVER (PARTITION BY dt.store_id ORDER BY COUNT(*) DESC) AS
rank
	FROM detailed_table dt
	GROUP BY
			dt.store_id,
			dt.title
	)

	INSERT INTO summary_table
		SELECT
		tm.store_id,
		tm.title,
		tm.rental_count
		FROM top_movies tm
		WHERE tm.rank <= 1;
RETURN NEW;
END;
$$

CREATE TRIGGER summary_table_insert
	AFTER INSERT
	ON detailed_table
	FOR EACH STATEMENT
	EXECUTE PROCEDURE insert_trigger_function();


-- UPDATE trigger function for when existing data is updated in detailed_table --
CREATE OR REPLACE FUNCTION update_trigger_function()
	RETURNS TRIGGER
	LANGUAGE plpgsql
AS
$$
BEGIN
	DELETE FROM summary_table;

	WITH top_movies AS (
	SELECT
	dt.store_id,
	dt.title,
			COUNT(*) AS rental_count,
```

```
        ROW_NUMBER() OVER (PARTITION BY dt.store_id ORDER BY COUNT(*) DESC) AS
rank
        FROM detailed_table dt
        GROUP BY
                        dt.store_id,
                        dt.title
        )

        INSERT INTO summary_table
                SELECT
                tm.store_id,
                tm.title,
                tm.rental_count
                FROM top_movies tm
                WHERE tm.rank <= 1;
RETURN NEW;
END;
$$

CREATE TRIGGER summary_table_update
        AFTER UPDATE
        ON detailed_table
        FOR EACH STATEMENT
        EXECUTE PROCEDURE update_trigger_function();
```

F:

```
-- Section F --
-- Stored procedure that refreshes the data in both the detailed table and summary table --
CREATE OR REPLACE PROCEDURE refresh_data()
LANGUAGE plpgsql
AS
$$
BEGIN
DELETE FROM detailed_table;
DELETE FROM summary_table;
INSERT INTO detailed_table
        SELECT
                i.store_id,
                i.inventory_id,
                f.film_id,
                f.title,
                r.customer_id,
                readable_date(r.rental_date) AS rental_date,
                r.rental_date AS rental_date_ts
        FROM rental r
        INNER JOIN inventory i ON r.inventory_id = i.inventory_id
        INNER JOIN film f ON i.film_id = f.film_id
        WHERE r.rental_date BETWEEN '2005-05-24' AND '2005-06-24'
```

```
        ORDER BY r.rental_date
;

RETURN;
END;
$$

CALL refresh_data();
```

F1:
There are two (2) highly relevant job scheduling tools that can be used to automate a PostgreSQL stored procedure: One is pg_cron implemented as a PostgreSQL extension, and the second is cron with shell scripts.

pg_cron is a PostgreSQL extension that runs scheduled jobs inside the database itself. It allows the user to schedule PostgreSQL commands directly from within the database, providing seamless integration.

Cron (with shell scripts) is a linux utility that allows the user to schedule shell scripts that then execute PostgreSQL stored procedures, at the desired time, using psql.

Both of these scheduling tools are more than adequate to automate the execution of stored procedures.