A

Real-World Report Created from DVD Dataset 2

A1 & A2

Customer Movie Preference Table Fields and Types 2

A3

Source Tables 3

A4

Custom Transformation & Functions 4

A5

Business Uses for Customer Movie Preferences Reporting 4

A6

Frequency of Reporting 5

B

Function & Transformation Code 5

C

Table Creation Code 5

D

Source Data Extraction Code for Detailed Table 6

E

Summary Table Trigger Function 6

F

Stored Procedure to Update Report 7

F1

Job Scheduling Tools 9

G

Live Code Demonstration 9

H

Works Referenced 9

Supplementary

Appendix: Complete, Annotated SQL 10

**Managing Advertisements in a DVD Rental Business**

Within any business, there are many possible uses of reporting functionality in a database. In this scenario, the owner of a small DVD rental chain known as BluckBoster feels that an advertising campaign will increase the quantity of week-over-week rentals, indirectly improving the company’s revenue. The owner hires an outside marketing firm to improve their advertising. To better understand BluckBoster’s background, the marketing firm first decides to review historical data to gain insight on possible areas for growth.

# A: Real-World Report Created from DVD Dataset

Businesses often use data to identify trends in purchasing, which can then be used to make marketing more effective and increase revenue. At BluckBoster, this thought process can be implemented by reviewing clients’ rental history. The marketing firm decides to identify each customer’s preferred movie genre for the purpose of producing more relevant advertisements. A data analyst is tasked with manipulating the DVD rental dataset to provide this information.

The first portion of the report will be a detailed table comprised of client identification numbers, names, emails, activity status and the customer’s most-commonly rented (and likely, favorite) genre. Their preferences can be used to create targeted advertisements. In contrast, a second table will be designed to provide a big-picture view of all BluckBoster’s clients’ preferred film categories. That table will aggregate the data from the first one, showing each genre with a count of customers whose rental history indicates a preferences for that genre.

The data analyst considers two possible barriers before undertaking this project. Their first concern is that some clients have rented two or more film categories equally. The data analyst understands that knowing this information may be helpful for more specific advertising, but including more than one genre for only some customers may make it difficult to manipulate that data and automate its usage later. Considering the benefits of data uniformity, the data analyst decides to use an aggregate function, mode(), which will select only one of the multiple film categories in this circumstance. The second issue in creating this data is the risk of old datapoints skewing the results even though more-recent data is likely more useful for marketing purposes. After reviewing the data, the analyst determines that BluckBoster’s existing database only holds data related to rentals from May 24, 2005 – February 14, 2006. It may be necessary to filter for recency in future iterations, but considering the small sample size available to the analyst at the time of creation, they decide to incorporate all existing data with no date filter. With these guidelines in mind, they set forth to create the two tables that will comprise the BluckBoster Customer Movie Preference report.

# A1 & A2: Customer Movie Preference Report Fields and Types

The detailed table described will contain the following fields and their data types:

* **Customer\_id –** INT – from “customer\_id” in customer table
* **Customer\_name –** VARCHAR(255) – concatenated from “first\_name” and “last\_name” fields in customer table
* **Email –** VARCHAR(50) – from “email” field in customer table
* **Favorite\_genre –** VARCHAR(25) – from “name” field in category table
* **Active\_text –** VARCHAR(8) – from “active” field in customer table

The summary table will contain the following fields and their data types:

* **Favorite\_genre –** VARCHAR(25) – from “name” field in category table
* **Customer\_count –** INT – counts “customer\_id” field in customer table

# The data types listed above are described as:

* **VARCHAR(x) –** a field with varying characters including letters, symbols and numbers, with a maximum length of ‘x’ characters.
* **INT –** a numeric field allowing integers between -214,748,3648 and 214,748,3647, meaning whole numbers with no decimals. Though this data type technically allows negative numbers, negative numbers in the output would indicate an error in the SQL code.

# A3: Source Tables

The following tables and fields will be used to source the data for the detailed report:

* customer
  + customer\_id
  + first\_name
  + last\_name
  + email
  + active
* rental (linking other tables but not in final output)
  + customer\_id
  + inventory\_id
* inventory (linking other tables but not in final output)
  + inventory\_id
  + film\_id
* film\_category (linking other tables but not in final output)
  + film\_id
  + category\_id
* category
  + category\_id
  + name

The following tables and fields will be used to source the data for the summary report:

* moviepreferences\_detail
  + favorite\_genre
  + customer\_id
  + active\_text
* customer (used indirectly; populates moviepreferences\_detail)
  + customer\_id
  + active
* rental (used indirectly; populates moviepreferences\_detail)
  + customer\_id
  + inventory\_id
* inventory (used indirectly; populates moviepreferences\_detail)
  + inventory\_id
  + film\_id
* film\_category (used indirectly; populates moviepreferences\_detail)
  + film\_id
  + category\_id
* category (used indirectly; populates moviepreferences\_detail)
  + category\_id
  + name

# A4: Custom Transformation & Functions

Within the database, the “active” field on the customer table shows client status in binary, with active status represented by 1s and inactivity represented by 0s. This status will be useful to include in the detailed table, since the marketing firm would prefer to advertise differently to active and inactive clients. The binary within the column may be confusing to laypersons, so adjusting it to a string that states “active” or “inactive” will make it more approachable.

# A5: Business Uses for Customer Movie Preferences Reporting

The marketing firm suggests several possible uses for the new report to increase BluckBoster’s sales. The detailed version will change how the store handles advertising to specific clients to increase repeat business. The summary will assist in improving mass marketing techniques like public advertisements and merchandising to increase overall sales.

The marketing firm suggests several possible ways for each table to support those goals. First, they intend to improve the relevancy of the weekly advertising email sent to BluckBoster’s clients every Friday morning. The current version of the email sends the same message with new releases and special features to all clients. The updated version of the email will use the new report to send targeted emails based on the client’s most-rented genre. Inactive customers will receive the personalized email with an attached coupon or other enticing offer each week to encourage their return to the store. Regardless of activity status, the emails will still include an option to unsubscribe from the emails. The subscriber email list will be maintained within

In addition to emails, Bluckboster’s new report will also be used within the store to improve recommendations by store employees. While employees were always able to view purchase history when scanning the client’s rewards card, it was difficult to determine each client’s overall preferences quickly during transactions to upsell new films. The new report will connect the rewards card ID number to the customer\_id field within the detailed table to display the customer’s most-rented genre at the point-of-sale. This will allow employees to create a natural interaction in which they recommend a new release inspired by the client’s preference.

In harmony with the improvements in targeted advertising, Bluckboster’s the summary data will guide purchasing, merchandising, and marketing decisions. For example, if the summary view shows that horror is the most popular genre, the store will increase the variety and quantity of horror movies. Horror-related memorabilia would be stocked near the register to encourage impulse purchases, and featured horror movies would be displayed prominently on a rack near the entrance of the store. The benefits of those changes could also be maximized by pulling in more customers with similar preferences. For instance, BluckBoster could enter a partnership with a Halloween pop-up, and ask them to give out coupons or a spooky ad with each purchase. If those potential clients choose to visit the DVD rental store, they may be more likely to rent the featured films near the entrance or purchase the themed memorabilia.

# A6: Frequency of Reporting

In consideration of the many goals outlined in subsection A5, the marketing firm determines that the report should be scheduled based on the deliverable that will require the most frequent updates. The store may not need to re-merchandise every week, but it would benefit from updating the report prior to the advertising emails that are sent on Friday mornings. These emails are intended to drive weekend traffic, so it is important to capture all new clients and preferences each week. Based on this need, the data analyst plans to update the report at the end of business hours each Thursday. The data will be referenced as-needed for monthly purchase orders and quarterly merchandising decisions.

# B: Function & Transformation Code

The following SQL function completes the transformation described in sub-section A4:

CREATE OR REPLACE FUNCTION ACTIVE\_AS\_TEXT(active INT)

RETURNS VARCHAR(8)

LANGUAGE plpgsql

AS

$$

DECLARE active\_text VARCHAR(8);

BEGIN

active\_text:='active';

IF active=0 THEN

active\_text:='inactive';

END IF;

RETURN active\_text;

END;

$$;

# C: Table Creation Code

The following SQL statement creates the detailed table for the Customer Movie Preferences report:

CREATE TABLE IF NOT EXISTS moviepreference\_detail (

customer\_id INT,

customer\_name VARCHAR(255),

email VARCHAR(50),

favorite\_genre VARCHAR(25),

active\_text VARCHAR(8)

);

The following SQL statement creates the summary table for the Customer Movie Preferences report:

CREATE TABLE IF NOT EXISTS moviepreference\_summary (

favorite\_genre VARCHAR(25),

customer\_count INT

);

# D: Source Data Extraction Code for Detailed Table

The following SQL code extracts the data from the source data when data is inserted into detailed table for the first time:

INSERT INTO moviepreference\_detail (customer\_id, customer\_name, email, favorite\_genre,active\_text)

SELECT

customer.customer\_id,

CONCAT(customer.first\_name,' ', customer.last\_name) AS customer\_name,

customer.email,

mode() WITHIN GROUP (ORDER BY category.name) AS favorite\_genre,

ACTIVE\_AS\_TEXT(customer.active) AS active\_text

FROM customer

LEFT JOIN rental

ON customer.customer\_id = rental.customer\_id

INNER JOIN inventory

ON rental.inventory\_id = inventory.inventory\_id

INNER JOIN film\_category

ON inventory.film\_id = film\_category.film\_id

INNER JOIN category

ON film\_category.category\_id = category.category\_id

GROUP BY customer.customer\_id, customer\_name, email,active\_text

ORDER BY customer\_id ASC;

All future data extractions for the detailed table will be managed by calling the stored procedure listed in sub-section F.

# E: Summary Table Trigger Function

# Unlike the report’s detailed table, the summary table’s data extraction process has no independent “insert into” statement. Instead, the “insert into” data extraction statement is nested within a trigger that updates the summary table whenever rows are inserted into the detailed table. The SQL code that creates this trigger is listed below:

CREATE OR REPLACE FUNCTION summary\_reset()

RETURNS TRIGGER

LANGUAGE plpgsql

AS

$$

BEGIN

SET client\_min\_messages TO NOTICE;  
TRUNCATE TABLE moviepreference\_summary;

INSERT INTO moviepreference\_summary (favorite\_genre, customer\_count)

SELECT

moviepreference\_detail.favorite\_genre AS favorite\_genre,

COUNT(moviepreference\_detail.customer\_id) AS customer\_count

FROM moviepreference\_detail

WHERE NOT moviepreference\_detail.active\_text = 'inactive'

GROUP BY favorite\_genre

ORDER BY favorite\_genre ASC;

RAISE NOTICE 'Trigger Function Executed Successfully';

RETURN NEW;

END;

$$;

CREATE TRIGGER summary\_trigger

AFTER INSERT

ON moviepreference\_detail

FOR EACH STATEMENT

EXECUTE FUNCTION summary\_reset();

# F: Stored Procedure to Update Report

Once the Customer Movie Preference reports are created and populated the first time, future updates will only query a stored procedure that refreshes both the detailed and summary tables. This should be completed weekly, since the data will be used to increase the efficacy of BluckBoster’s pre-weekend email advertisement and drive weekend sales. The following stored SQL procedure will be called every Thursday at closing time to update the tables within the report so the Friday morning email advertisements can be prepared:

CREATE OR REPLACE PROCEDURE full\_reset()

LANGUAGE plpgsql

AS

$$

BEGIN

SET client\_min\_messages TO NOTICE;

TRUNCATE TABLE moviepreference\_detail;

INSERT INTO moviepreference\_detail (

customer\_id,

customer\_name,

email,

favorite\_genre,

active\_text

)

SELECT

customer.customer\_id,

CONCAT(customer.first\_name,' ', customer.last\_name) AS customer\_name,

customer.email,

mode() WITHIN GROUP (ORDER BY category.name) AS favorite\_genre,

ACTIVE\_AS\_TEXT(customer.active) AS active\_text

FROM customer

LEFT JOIN rental

ON customer.customer\_id = rental.customer\_id

INNER JOIN inventory

ON rental.inventory\_id = inventory.inventory\_id

INNER JOIN film\_category

ON inventory.film\_id = film\_category.film\_id

INNER JOIN category

ON film\_category.category\_id = category.category\_id

GROUP BY customer.customer\_id, customer\_name, email,active\_text

ORDER BY customer\_id ASC;

TRUNCATE TABLE moviepreference\_summary;

INSERT INTO moviepreference\_summary (favorite\_genre, customer\_count)

SELECT

moviepreference\_detail.favorite\_genre AS favorite\_genre,

COUNT(moviepreference\_detail.customer\_id) AS customer\_count

FROM moviepreference\_detail

WHERE NOT moviepreference\_detail.active\_text = 'inactive'

GROUP BY favorite\_genre

ORDER BY favorite\_genre;

RAISE NOTICE 'Full Reset Procedure Executed Successfully';

END;

$$;

# F1: Job Scheduling Tools

Since BluckBoster’s owner is not familiar with relational databases or SQL, the marketing firm suggests that a job scheduling tool be used to automate the weekly update for the report. To execute this request, the data analyst downloads Stack Builder from the Postgres website and uses it to install pgAgent. Once installed, pgAgent Jobs is accessed in pgAdmin and is used to create a “job” that will be scheduled every Thursday at closing time. The job runs SQL code which calls the stored procedure to refresh the tables, then exports the tables to two user-friendly CSV files.

# G: Live Code Demonstration

As the BluckBoster project draws to a close at the marketing firm, the database administrator creates a video demonstrating the functionality of the SQL code deliverable. That live demonstration can be [viewed on Panopto.](https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=0b0038fc-43fd-4c54-94cd-afc4001ee9aa)

**H: Works Referenced**

No outside resources were used to write this code or create this report.

**Appendix: Complete, Annotated SQL**

*--Notes related to each code block are listed in italics above the relevant code section*

*--Create the detailed table*

CREATE TABLE IF NOT EXISTS moviepreference\_detail (

customer\_id INT,

customer\_name VARCHAR(255),

email VARCHAR(50),

favorite\_genre VARCHAR(25),

active\_text VARCHAR(8)

);

*--Create the summary table*

CREATE TABLE IF NOT EXISTS moviepreference\_summary (

favorite\_genre VARCHAR(25),

customer\_count INT

);

*--Create a function to turn the 0s and 1s from customer.active to inactive*

*--and active respectively*

CREATE OR REPLACE FUNCTION ACTIVE\_AS\_TEXT(active INT)

RETURNS VARCHAR(8)

LANGUAGE plpgsql

AS

$$

DECLARE active\_text VARCHAR(8);

BEGIN

active\_text:='active';

IF active=0 THEN

active\_text:='inactive';

END IF;

RETURN active\_text;

END;

$$;

*--Create the trigger to update the summary table*

CREATE OR REPLACE FUNCTION summary\_reset()

RETURNS TRIGGER

LANGUAGE plpgsql

AS

$$

BEGIN

SET client\_min\_messages TO NOTICE;

TRUNCATE TABLE moviepreference\_summary;

INSERT INTO moviepreference\_summary (favorite\_genre, customer\_count)

SELECT

moviepreference\_detail.favorite\_genre AS favorite\_genre,

COUNT(moviepreference\_detail.customer\_id) AS customer\_count

FROM moviepreference\_detail

WHERE NOT moviepreference\_detail.active\_text = 'inactive'

GROUP BY favorite\_genre

ORDER BY favorite\_genre ASC;

RAISE NOTICE 'Trigger Function Executed Successfully';

RETURN NEW;

END;

$$;

CREATE TRIGGER summary\_trigger

AFTER INSERT

ON moviepreference\_detail

FOR EACH STATEMENT

EXECUTE FUNCTION summary\_reset();

*--Insert data into the detailed table*

INSERT INTO moviepreference\_detail (customer\_id, customer\_name, email, favorite\_genre,active\_text)

SELECT

customer.customer\_id,

CONCAT(customer.first\_name,' ', customer.last\_name) AS customer\_name,

customer.email,

mode() WITHIN GROUP (ORDER BY category.name) AS favorite\_genre,

ACTIVE\_AS\_TEXT(customer.active) AS active\_text

FROM customer

LEFT JOIN rental

ON customer.customer\_id = rental.customer\_id

INNER JOIN inventory

ON rental.inventory\_id = inventory.inventory\_id

INNER JOIN film\_category

ON inventory.film\_id = film\_category.film\_id

INNER JOIN category

ON film\_category.category\_id = category.category\_id

GROUP BY customer.customer\_id, customer\_name, email,active\_text

ORDER BY customer\_id ASC;

*--Create stored procedure to update both the detailed and summary tables*

*--Note: for purposes of the assignment, the summary table will be updated twice*

*--when this procedure runs due to the combination of the table refresh code in the*

*--procedure and the trigger function above*

CREATE OR REPLACE PROCEDURE full\_reset()

LANGUAGE plpgsql

AS

$$

BEGIN

SET client\_min\_messages TO NOTICE;

TRUNCATE TABLE moviepreference\_detail;

INSERT INTO moviepreference\_detail (

customer\_id,

customer\_name,

email,

favorite\_genre,

active\_text

)

SELECT

customer.customer\_id,

CONCAT(customer.first\_name,' ', customer.last\_name) AS customer\_name,

customer.email,

mode() WITHIN GROUP (ORDER BY category.name) AS favorite\_genre,

ACTIVE\_AS\_TEXT(customer.active) AS active\_text

FROM customer

LEFT JOIN rental

ON customer.customer\_id = rental.customer\_id

INNER JOIN inventory

ON rental.inventory\_id = inventory.inventory\_id

INNER JOIN film\_category

ON inventory.film\_id = film\_category.film\_id

INNER JOIN category

ON film\_category.category\_id = category.category\_id

GROUP BY customer.customer\_id, customer\_name, email,active\_text

ORDER BY customer\_id ASC;

TRUNCATE TABLE moviepreference\_summary;

INSERT INTO moviepreference\_summary (favorite\_genre, customer\_count)

SELECT

moviepreference\_detail.favorite\_genre AS favorite\_genre,

COUNT(moviepreference\_detail.customer\_id) AS customer\_count

FROM moviepreference\_detail

WHERE NOT moviepreference\_detail.active\_text = 'inactive'

GROUP BY favorite\_genre

ORDER BY favorite\_genre;

RAISE NOTICE 'Full Reset Procedure Executed Successfully';

END;

$$;

*--Select statement to view detailed table*

*--SELECT \* FROM moviepreference\_detail;*

*--Select statement to view summary table*

*--SELECT \* FROM moviepreference\_summary;*

*--Call the stored procedure to initiate refresh of tables*

*--CALL full\_reset();*