DVD Rental Business Insights

This comprehensive report provides an analysis of our DVD rental business, encompassing a variety of critical metrics and insights, reflecting our operations’ health, customer engagement, and overall financial performance. The contents are structured to offer a summarized breakdown of customer activity, revenue generation, inventory utilization, and popularity metrics of our rental inventory by rental category.

**Customer Activity Analysis**: The report begins with an in-depth exploration of customer rental patterns, highlighting trends in customer preferences and frequency of rentals. This section provides a clearer understanding of our customer base and their engagement levels throughout the year.

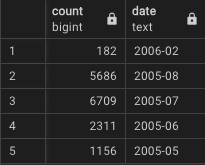
**Revenue Overview**: A detailed examination of the revenue streams from rentals. This includes analysis of peak periods, comparison of month over month performance, and identification of the employees responsible for that revenue.

**Inventory Utilization And Performance**: Our analysis on inventory utilization offers a look into the efficiency of our current stock management. It identifies the rates at which different categories of DVDs are rented out, helping pinpoint areas where inventory adjustments may be beneficial. This section will also look at the most and least popular DVDs based on rental frequency. By understanding customer preferences, we can tailor our purchasing and promotional strategies to better align with demand.

**Procedures and Queries**: To ensure transparency and reproducibility of our findings, the report concludes with a detailed description of the procedures and SQL queries used to extract data from the “dvdrental” database. This part serves as a valuable resource for those interested in diving deeper into the data used to create this report. .

This report aims to equip stakeholders with the knowledge required to make informed decisions and to continue enhancing our service offerings to meet customer demands effectively.

**Customer Activity Analysis**

2005 - 2006 saw a total of 599 customers that took out 16,044rentals, with the most rentals being in July ‘05 and the least rentals being February ‘06. A detailed month over month breakdown by customer is available in the database as a separate detailed report. Information on how to access this data is in the Procedures and Queries section.

5686 rentals in August ’05

6709 rentals in July ’05

2311 rentals in June ’05

1156 rentals in May ’05

182 rentals in February ‘06

**Revenue Overview**

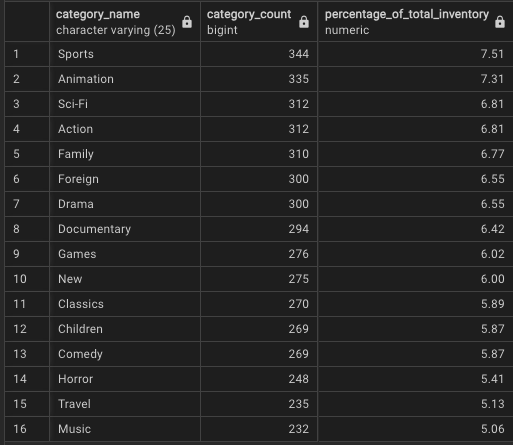
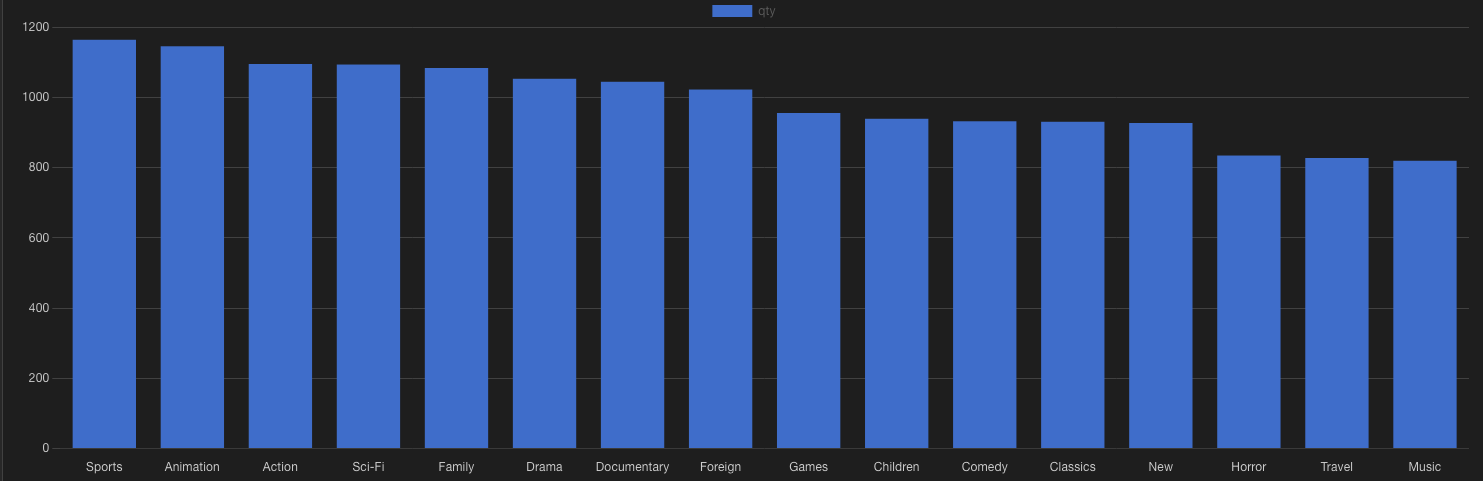
This same period saw a total revenue of 61,312 with $30,252 coming from store 1 and $31,059 coming from store 2. Each store only has 1 employee wich is also the manager of that store making Mike Hillyer responsible for store 1, and Jon Stephens responsible for store 2.

Note: store 1 total inventory is 2270, store 2 is 2311. Even accounting for inventory difference store 2 has slightly higher overall performance. Store 1 ultimately had generated $13.32 per dvd in their inventory while store 2 generated $13.43

**Inventory Utilization**

Company wide, 99.97% of the dvd inventory was rented out at least once, with the only non-utilized inventory being a single copy of ‘Academy Dinosaur’. I do believe this to be random happenstance but would also like to take the opportunity to point out the importance of ensuring a proper rotation of inventory at all times to maximize the longevity of our inventory. I would consider anything above 95% to be excellent.

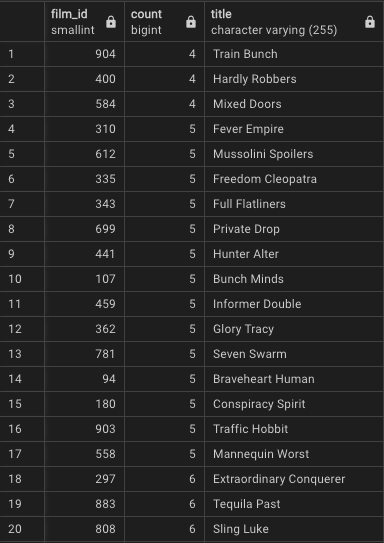
Overall the company had a very predictable spread in terms of rental popularity as the most popular rental categories also had the most available inventory across all stores. There are a few exceptions to this however such as action being 0.18% more popular than science fiction even though they have the same inventory, and Drama being 2.94% more popular than Foreign.

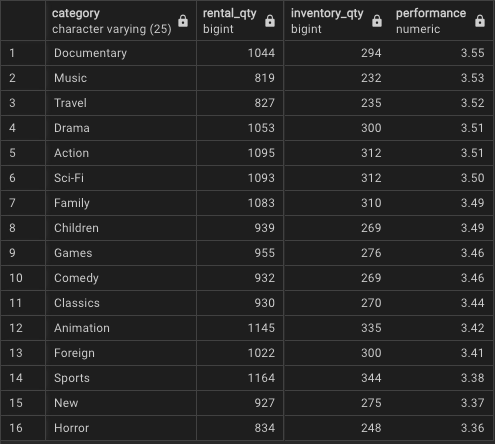
*Count of rentals by category*

*Inventory Breakdown By Category*

*Count of rentals by category*

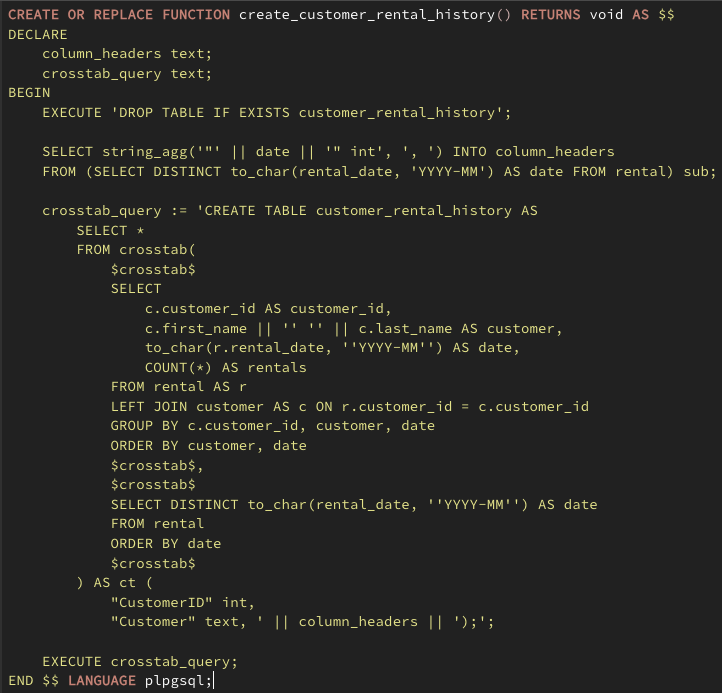
Next if we look at popularity by individual titles, the least Popular Rentals of our current inventory are Hardy Robbers, Mixed Doors, and Train Bunch, all with 4 rentals each and the overall most popular rental was Bucket Brotherhood with 34 rentals.

*Top 20*   *Bottom 20*

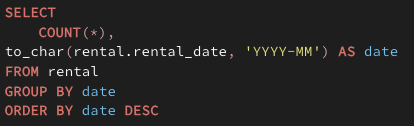
Lastly lets look at a performance metric for each category. This “performance” number is derived by dividing the number of rentals by the available inventory for that category to create a ratio of rentals:inventory per category. This data can be used to determine wich categories we should focus on expanding the most moving foreward as a higher number means the movies are rented out more frequently relative to their available inventory. For example horror is a low performer and a low popularity, ranking 14th by customer rental count and 16th by performance with it being 14th on our inventory percentage list. By contrast Music is a category that stands out as something we should consider increasing the total inventory of. Even though it ranks dead last on customer rental quantity, this is likely due to low availability as it is the 2nd highest performing category and the smallest percentage of the overall inventory, meaning it would be worth looking into increasing the overall inventory of music related films.

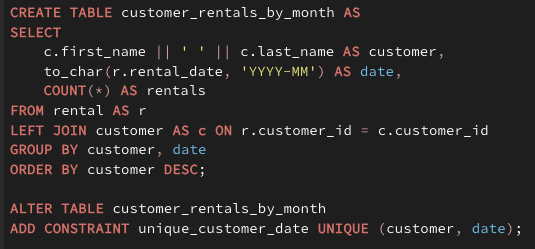
**Procedures And Queries**

Analysis of the DVD Rental Database begins with adding the ‘tablefunc’ module for PostgreSQL databases to have access to the crosstab feature, allowing the use of pivot tables.Screenshot 2024-05-19 at 2.30.23 PM.png

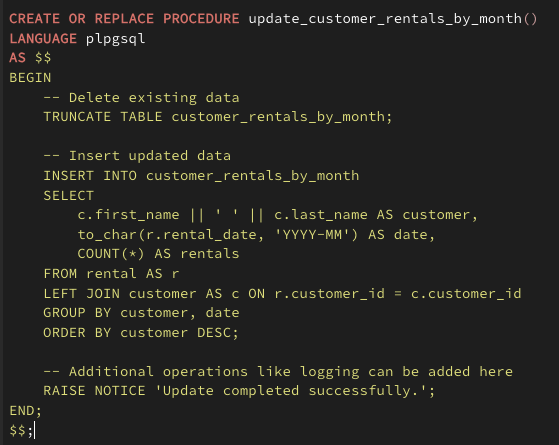
Next create the pivot table.

The query pictured above creates a pivot table from a dataset consisting of customer rental data aggregated by customer and month of activity. First column is the customers ID followed by their name, concatenated for ease of reading, and 1 column for each month in wich there is rental data with the customers rental count for that month. Now while seeing each individual customers rental history is a useful starting point for a lot of reports and does simplify reviewing account history on a customer by customer basis, this table can be aggregated further to provide a month over month report for total number of rentals per month, and if this report were to be run every year it could be used to provide important data on the habits of individual customers over time.

However if you are only interested in the total number of rentals month over month this can be achieved from the base table “public.rental” with the following query.

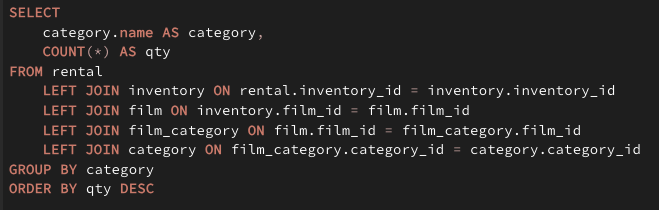
The basis of the pivot table was made after aggregating customer data into a new table called customer\_rentals\_by\_month wich was created with the following query:

This table has a stored procedure to update it over time although it is ultimately non-essential to the pivoted table.



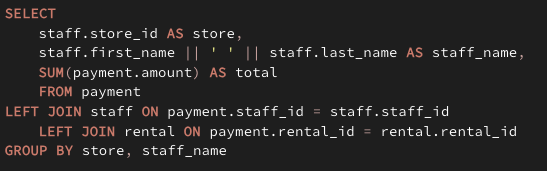
The customer\_rentals\_by\_month table is intended to provide long term year over year tend analysis of customer rentals on a per customer basis wich would be most useful for creating some kind of loyalty rewards points system based on number of rentals over time.

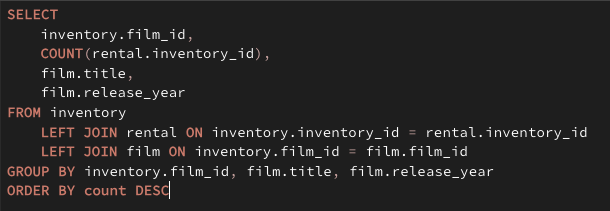
For the category breakdowns listed in the report summary there are a few queries that will be useful.

Rental Count By Category:

Performance by category:

And the 2 of those combined is what I would use to interpret categorical performance in the future:

Financial Performance By Employee:

Count of rentals by film:

Note: there appears to be a major incongruity in the database in that every film in the film table has a release year of 2006, even the films that were rented in 2005. Also the payment dates for rentals in 2005 are all registered in 2007 wich again raises some concerns about the integrity of the data. I had intended to do a more in-depth financial breakdown for 2005, but as the transaction data dates do not line up with rental dates that was not possible to accurately compose.

**Query Appendix**

**—Count of rentals by film**

SELECT

inventory.film\_id,

COUNT(rental.inventory\_id),

film.title,

film.release\_year

FROM inventory

LEFT JOIN rental ON inventory.inventory\_id = rental.inventory\_id

LEFT JOIN film ON inventory.film\_id = film.film\_id

GROUP BY inventory.film\_id, film.title, film.release\_year

ORDER BY count DESC

**—Inventory Performance By Category**

SELECT

c.name AS category\_name,

COUNT(i.inventory\_id) AS category\_count,

ROUND((COUNT(i.inventory\_id) \* 100.0 / total.total\_inventory), 2) AS percentage\_of\_total\_inventory

FROM

inventory i

JOIN film\_category fc ON i.film\_id = fc.film\_id

JOIN category c ON fc.category\_id = c.category\_id,

(

SELECT

COUNT(inventory\_id) AS total\_inventory

FROM

inventory

) AS total

GROUP BY

c.name,

total.total\_inventory

ORDER BY

percentage\_of\_total\_inventory DESC

**—Count rentals by month**

SELECT

to\_char(r.rental\_date, 'YYYY-MM') AS date,

COUNT(\*) AS rentals

FROM rental AS r

GROUP BY date

ORDER BY date DESC

**—Count rentals by category**

SELECT

category.name AS category,

COUNT(\*) AS qty

FROM rental

LEFT JOIN inventory ON rental.inventory\_id = inventory.inventory\_id

LEFT JOIN film ON inventory.film\_id = film.film\_id

LEFT JOIN film\_category ON film.film\_id = film\_category.film\_id

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

GROUP BY category

ORDER BY qty DESC

**—Count rentals by film**

SELECT

inventory.film\_id,

COUNT(rental.inventory\_id),

film.title

FROM inventory

LEFT JOIN rental ON inventory.inventory\_id = rental.inventory\_id

LEFT JOIN film ON inventory.film\_id = film.film\_id

GROUP BY inventory.film\_id, film.title

ORDER BY count DESC

**—Revenue per employee**

SELECT

staff.store\_id AS store,

staff.first\_name || ' ' || staff.last\_name AS staff\_name,

SUM(payment.amount) AS total

FROM payment

LEFT JOIN staff ON payment.staff\_id = staff.staff\_id

LEFT JOIN rental ON payment.rental\_id = rental.rental\_id

GROUP BY store, staff\_name

**—Create Category/Inventory Performance Table**

**\***note: not really needed since the procedure already does this but its not a bad idea to create the table first or to have a manual backup for reference in case someone edits the procedure incorrectly.

CREATE TABLE inventory\_performance AS

WITH

inventory\_count AS (

SELECT

c.name AS category\_name,

COUNT(i.inventory\_id) AS category\_count,

ROUND(

(COUNT(i.inventory\_id) \* 100.0 / total.total\_inventory), 2

) AS percentage\_of\_total\_inventory

FROM

inventory i

JOIN film\_category fc ON i.film\_id = fc.film\_id

JOIN category c ON fc.category\_id = c.category\_id,

(

SELECT

COUNT(inventory\_id) AS total\_inventory

FROM

inventory

) AS total

GROUP BY

c.name,

total.total\_inventory

ORDER BY

percentage\_of\_total\_inventory DESC

),

rental\_count AS (

SELECT

category.name AS category,

COUNT(\*) AS qty

FROM

rental

LEFT JOIN inventory ON rental.inventory\_id = inventory.inventory\_id

LEFT JOIN film ON inventory.film\_id = film.film\_id

LEFT JOIN film\_category ON film.film\_id = film\_category.film\_id

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

GROUP BY

category

ORDER BY

qty DESC)

SELECT

rc.category,

rc.qty AS rental\_qty,

ic.category\_count AS inventory\_qty,

ROUND(CAST(rc.qty AS NUMERIC) / ic.category\_count, 2) AS performance

FROM

rental\_count AS rc

LEFT JOIN inventory\_count AS ic ON rc.category = ic.category\_name

ORDER BY performance DESC;

**— Create update\_inventory\_performance Stored Procedure**

CREATE OR REPLACE PROCEDURE update\_inventory\_performance()

LANGUAGE plpgsql

AS $$

BEGIN

EXECUTE 'DROP TABLE IF EXISTS inventory\_performance';

EXECUTE '

CREATE TABLE inventory\_performance AS

WITH

inventory\_count AS (

SELECT

c.name AS category\_name,

COUNT(i.inventory\_id) AS category\_count,

ROUND((COUNT(i.inventory\_id) \* 100.0 / total.total\_inventory), 2

) AS percentage\_of\_total\_inventory

FROM

inventory i

JOIN film\_category fc ON i.film\_id = fc.film\_id

JOIN category c ON fc.category\_id = c.category\_id,

( SELECT

COUNT(inventory\_id) AS total\_inventory

FROM

inventory

) AS total

GROUP BY

c.name,

total.total\_inventory

ORDER BY

percentage\_of\_total\_inventory DESC ),

rental\_count AS (

SELECT

category.name AS category,

COUNT(\*) AS qty

FROM

rental

LEFT JOIN inventory ON rental.inventory\_id = inventory.inventory\_id

LEFT JOIN film ON inventory.film\_id = film.film\_id

LEFT JOIN film\_category ON film.film\_id = film\_category.film\_id

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

GROUP BY

category

ORDER BY

qty DESC)

SELECT

rc.category,

rc.qty AS rental\_qty,

ic.category\_count AS inventory\_qty,

ROUND(CAST(rc.qty AS NUMERIC) / ic.category\_count, 2) AS performance

FROM

rental\_count AS rc

LEFT JOIN inventory\_count AS ic ON rc.category = ic.category\_name

ORDER BY

performance DESC;

';

END;

$$;

**—Create customer\_rentals\_by\_month table**

* *Note: this was not needed by anything in the end and I only left it here because it was already done, and to serve as a backup example of stored procedures that is simpler than the inventory\_performance.*

CREATE TABLE customer\_rentals\_by\_month AS

SELECT

c.first\_name || ' ' || c.last\_name AS customer,

to\_char(r.rental\_date, 'YYYY-MM') AS date,

COUNT(\*) AS rentals

FROM rental AS r

LEFT JOIN customer AS c ON r.customer\_id = c.customer\_id

GROUP BY customer, date

ORDER BY customer DESC;

ALTER TABLE customer\_rentals\_by\_month

ADD CONSTRAINT unique\_customer\_date UNIQUE (customer, date);

**—Update customer rentals by month - Stored Procedure**

CREATE OR REPLACE PROCEDURE update\_customer\_rentals\_by\_month(new\_customer\_id int)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO customer\_rentals\_by\_month (customer, date, rentals)

SELECT

c.first\_name || ' ' || c.last\_name AS customer,

to\_char(NOW(), 'YYYY-MM') AS date, -- Assume we're updating for the current month

COUNT(\*) AS rentals

FROM rental AS r

LEFT JOIN customer AS c ON r.customer\_id = c.customer\_id

WHERE c.customer\_id = new\_customer\_id

GROUP BY customer, date

ON CONFLICT (customer, date) DO UPDATE

SET rentals = EXCLUDED.rentals;

RAISE NOTICE 'Update completed successfully.';

END;

$$;

**—Add in crosstab functionality**

CREATE EXTENSION IF NOT EXISTS tablefunc;

**— Add in pg\_cron for task scheduling on Linux and MacOS**

* *Note: pg\_cron works on Mac an linux but I was unable to make it work in windows, so instead I added the procedure to the trigger function that executes on insert into the rental table for this procedure.*

CREATE EXTENSION IF NOT EXISTS pg\_cron;

SELECT cron.schedule('daily\_update', '0 0 \* \* \*', $$

BEGIN

PERFORM update\_customer\_rentals\_by\_month();

END $$);

**—Create scheduled task on Windows**

1. Create a file called psql\_scheduled\_tasks.sql and add the following content to that file:

CALL update\_inventory\_performance();

CALL update\_customer\_rentals\_by\_month();

2. Create a batch file called execute\_stored\_procedures.bat with the following content:

@echo off

psql -U your\_username -d your\_database -f “path\to\update\_inventory\_performance.sql"

3. Open Windows Task Scheduler, create a new task to run at midnight evey dat, and set the “Action” to start the batch file you just created.

**— Create function to create customer\_rental\_history crosstab/pivot table**

CREATE OR REPLACE FUNCTION create\_customer\_rental\_history() RETURNS void AS $$

DECLARE

column\_headers text;

crosstab\_query text;

BEGIN

EXECUTE 'DROP TABLE IF EXISTS customer\_rental\_history';

SELECT string\_agg('"' || date || '" int', ', ') INTO column\_headers

FROM (SELECT DISTINCT to\_char(rental\_date, 'YYYY-MM') AS date FROM rental) sub;

crosstab\_query := 'CREATE TABLE customer\_rental\_history AS

SELECT \*

FROM crosstab(

$crosstab$

SELECT

c.customer\_id AS customer\_id,

c.first\_name || '' '' || c.last\_name AS customer,

to\_char(r.rental\_date, ''YYYY-MM'') AS date,

COUNT(\*) AS rentals

FROM rental AS r

LEFT JOIN customer AS c ON r.customer\_id = c.customer\_id

GROUP BY c.customer\_id, customer, date

ORDER BY customer, date

$crosstab$,

$crosstab$

SELECT DISTINCT to\_char(rental\_date, ''YYYY-MM'') AS date

FROM rental

ORDER BY date

$crosstab$

) AS ct (

"CustomerID" int,

"Customer" text, ' || column\_headers || ');';

EXECUTE crosstab\_query;

END $$ LANGUAGE plpgsql;

**—Execute function to create table**

SELECT create\_customer\_rental\_history()

**— Select data from newly created table**

SELECT \* FROM customer\_rental\_history;

**—Create trigger function to call stored procedure and recreate the customer\_rental\_history**

CREATE OR REPLACE FUNCTION call\_update\_customer\_history() RETURNS trigger AS $$

BEGIN

CALL update\_customer\_rentals\_by\_month(NEW.customer\_id);

PERFORM create\_customer\_history();

RETURN NEW;

END $$ LANGUAGE plpgsql;

**—Create event trigger to call trigger function on insert into rental table**

DROP TRIGGER IF EXISTS update\_customer\_history\_after\_insert ON rental;

CREATE TRIGGER update\_customer\_history\_after\_insert

AFTER INSERT ON rental

FOR EACH ROW

EXECUTE FUNCTION call\_update\_customer\_history();

**— Insert new data into rental table to test if update functions work:**

INSERT INTO rental (customer\_id, rental\_date, inventory\_id, staff\_id) VALUES (1, '2024-05-01', 1, 1);

**Requirements Breakdown**

*\*Note: Emphasis is placed on the Category Performance aspect of the report to meet grading requirements.*

A. Real World Written Business Report: General Insights on inventory performance and customer rental history over time.

A1. category.name will be in the final summarized table of inventory\_performance as well as the temp tables used to create that table.

A2. Data types of these fields are category - varchar(25), rentaly\_qty - bigint, inventory\_qty - bigint, and performance - numeric

A3. The tables used in inventory\_performance are: inventory, film\_category, category, rental, and film.

A4. Field that required specific manipulation was rc.qtywich is field from the temp table rental\_count in the query to create the report. It was bigint but needed to be cast to numeric so I could get a non-integer value when diving rc.qty by ic.category (comes from temp table inventory\_count). Since if you divide 2 integers psql will only output an integer and I wanted to display a float/decimal. If I hadn’t not done this every category would have had the same performance of 3 or 4 when that was not technically accurate.

A5. This report can be used to determine what rental categories are performing the best/worst and could be used to decide wich categories in the overall inventory should be expanded or refreshed the most. Also can be used to identify if any part of the inventory has lower performance than others indicating areas that would benefit the most from focussed improvement efforts.

A6. The table is set to be recreated every night at midnight using Windows Task Scheduler if your on windows, or pg\_cron if your on Linux or MacOS

Note on section B: There are not permanent detailed view tables as they truly served no purpose other than to de-normalize the data wich is why I decided to create them solely as temp tables within the query to create the simplified view. If for any reason the detailed view of either of those tables is needed it is very easy to extract those queries and run them separately. This is a technique I use on a regular basis in my work to not clutter databases with duplicate de-normalized data.

All queries used to create this report and these tables, including the stored procedures and its triggers are located in the following GitHub repository:

https://github.com/cobalt88/DVD-Rental-EOY-2005