1. Operations Performed by Functions

Let's break down the expected operations of each function in the dvd\_rental database. However, without access to the actual database or function definitions, the following operations are deduced based on typical naming conventions:

1. film\_in\_stock:

- Likely returns a list of films that are currently available in the inventory (i.e., films that have stock).

- It would probably query the inventory table, checking the number of items available for each film.

2. film\_not\_in\_stock:

- Likely returns a list of films that are currently not available in the inventory (i.e., films that are out of stock).

- It would likely query the inventory table, checking for films where the stock count is 0 or unavailable.

3. inventory\_in\_stock:

- Likely returns a list of all items in the inventory table that are available for rent. This might check if items have been rented or if the inventory is active.

- It would query the inventory table with conditions such as "quantity > 0" or "inventory status = available."

4. get\_customer\_balance:

- This function most likely calculates and returns the balance for a customer, considering various factors like unpaid rentals, late fees, or pre-paid amounts.

- It would likely query tables such as payment, rental, and customer to compute the balance.

5. inventory\_held\_by\_customer:

- This function would likely return a list of films that a customer currently has rented, based on the rental table.

- It might query for rental records that are not yet returned, joining inventory and customer to filter by a specific customer.

6. rewards\_report:

- Likely generates a report of rewards or loyalty points earned by customers based on their rental history.

- It would query payment, customer, and possibly rental and reward tables.

7. last\_day:

- Likely returns the last date of a specific time period (such as the last day of the current month, quarter, or year).

- This function would likely use date/time functions to calculate the last day of the relevant period.

---

2. Why does the rewards\_report function return 0 rows?

The function rewards\_report likely returns 0 rows because the dynamic SQL (possibly within EXECUTE) isn't correctly fetching the data or doesn't match the expected conditions.

To fix this, you would need to check the following:

- Ensure the correct WHERE conditions are used in the dynamic query.

- Check if the tmpSQL variable is correctly defined and used to query the correct tables.

- Ensure that there is data matching the query criteria.

You can modify the function to ensure it's fetching the correct data. Here's an example of how to improve the function:

CREATE OR REPLACE FUNCTION rewards\_report()

RETURNS TABLE (

customer\_id INT,

customer\_name TEXT,

reward\_points INT

) AS $$

DECLARE

tmpSQL TEXT;

BEGIN

-- Build dynamic SQL to calculate rewards

tmpSQL := 'SELECT c.customer\_id, c.first\_name || '' '' || c.last\_name AS customer\_name, SUM(r.reward\_points) AS reward\_points

FROM rental r

INNER JOIN customer c ON r.customer\_id = c.customer\_id

GROUP BY c.customer\_id';

-- Execute the query

RETURN QUERY EXECUTE tmpSQL;

END;

$$ LANGUAGE plpgsql;

3. Potential Function for Removal

Without access to the specific function definitions, it would be challenging to determine if any function can be removed. However, you could consider removing any function that is:

- Not used in the codebase (unused functions).

- Duplicating the functionality of other functions.

- Inactive due to changes in the database schema or business requirements.

For example, if film\_not\_in\_stock and inventory\_in\_stock are providing redundant information (e.g., one just returns the inverse of the other), you could potentially remove one of them.

---

4. Modifications to get\_customer\_balance

The function likely calculates customer balances but doesn't fully implement all the business requirements. To adjust it, you'd need to:

- Ensure that it accounts for late fees, payment history, and any applicable discounts.

- Modify the query to include all relevant payment, rental, and fee tables.

Here’s an example of an enhanced function:

CREATE OR REPLACE FUNCTION get\_customer\_balance(customer\_id INT)

RETURNS NUMERIC AS $$

DECLARE

balance NUMERIC;

BEGIN

-- Calculate balance, including payments and rental fees

SELECT SUM(p.amount) - COALESCE(SUM(f.fee), 0) INTO balance

FROM payment p

LEFT JOIN rental r ON p.rental\_id = r.rental\_id

LEFT JOIN fee f ON r.rental\_id = f.rental\_id

WHERE r.customer\_id = customer\_id

GROUP BY r.customer\_id;

RETURN balance;

END;

$$ LANGUAGE plpgsql;

---

5. How do group\_concat and \_group\_concat work?

The group\_concat function (or \_group\_concat in some databases) is used to concatenate multiple values from rows into a single string. It is commonly used in databases like MySQL.

In PostgreSQL, string\_agg is often used instead. Here’s an example of how they work:

SELECT string\_agg(column\_name, ', ') FROM table\_name GROUP BY some\_column;

This function concatenates all values of column\_name into a single string, separated by a comma.

---

6. What does the last\_updated function do?

The last\_updated function likely returns the last modification timestamp of a table or record. It could check the updated\_at column in a table and return the most recent value.

Example:

CREATE OR REPLACE FUNCTION last\_updated(table\_name TEXT)

RETURNS TIMESTAMPTZ AS $$

DECLARE

last\_update TIMESTAMPTZ;

BEGIN

EXECUTE format('SELECT MAX(last\_update) FROM %I', table\_name) INTO last\_update;

RETURN last\_update;

END;

$$ LANGUAGE plpgsql;

---

7. Purpose of tmpSQL in rewards\_report Function

The tmpSQL variable holds the dynamically constructed SQL query that is executed using EXECUTE. This is often necessary when the structure of the query depends on variables or when table/column names need to be determined at runtime.

Can the function be recreated without dynamic SQL?

- Yes, but only if the structure of the query can be statically defined. If the query requires runtime calculation of table names or conditions, then dynamic SQL is required.

---

Conclusion

Each function plays a critical role in handling various business logic related to rentals, sales, and customer data in the DVD rental system. If the rewards\_report function is returning 0 rows, it may require an adjustment to its dynamic query. You can remove redundant functions if their functionality overlaps with others, and ensuring that business logic is correctly implemented in functions like get\_customer\_balance is essential.