Task 6. Prepare answers to the following questions

1. What operations do the following functions perform: film\_in\_stock, film\_not\_in\_stock, inventory\_in\_stock, get\_customer\_balance, inventory\_held\_by\_customer, rewards\_report, last\_day? You can find these functions in dvd\_rental database.
2. Why does ‘rewards\_report’ function return 0 rows? Correct and recreate the function, so that it's able to return rows properly.
3. Is there any function that can potentially be removed from the dvd\_rental codebase? If so, which one and why?
4. The ‘get\_customer\_balance’ function describes the business requirements for calculating the client balance. Unfortunately, not all of them are implemented in this function. Try to change function using the requirements from the comments.
5. How do ‘group\_concat’ and ‘\_group\_concat’ functions work? (database creation script might help) Where are they used?
6. What does ‘last\_updated’ function do? Where is it used?
7. What is tmpSQL variable for in ‘rewards\_report’ function? Can this function be recreated without EXECUTE statement and dynamic SQL? Why?

**film\_in\_stock**

film\_in\_stock(p\_film\_id INTEGER, p\_store\_id INTEGER) RETURNS INTEGER

* Returns the **number of copies of a specific film** that are **currently available in stock**.
* Checks for inventory items that belong to the store and that are **not never-rented items**, or r**ented-and-returned items.**
* **Used for:** Staff checking if a film is available to rent at a store.

**film\_not\_in\_stock**

film\_not\_in\_stock(p\_film\_id INTEGER, p\_store\_id INTEGER) RETURNS INTEGER

* Returns the **number of copies of a film that are currently not available**.
* Opposite logic of film\_in\_stock.
* **Used for:** Determining current rental load for a film at a store.

**inventory\_in\_stock**

inventory\_in\_stock(p\_inventory\_id INTEGER) RETURNS BOOLEAN

* Checks if a specific **inventory item is currently in stock**.
* Returns TRUE if there is **no active rental** for that inventory item (return\_date IS NOT NULL).
* **Used for:** Validating before allowing a customer to rent it.

**get\_customer\_balance**

get\_customer\_balance(p\_customer\_id INTEGER, p\_effective\_date TIMESTAMP) RETURNS NUMERIC

* Calculates a customer's **balance due** as of a given date.
* It subtracts total **payments** from total **rental fees** for returned items.
* **Used for:** Showing how much a customer owes.

**inventory\_held\_by\_customer**

inventory\_held\_by\_customer(p\_inventory\_id INTEGER) RETURNS INTEGER

* Returns the customer\_id of the person who is **currently renting** a given inventory item.
* If the item is not rented, returns NULL.
* **Used for:** Tracking which customer has an inventory item **rented out** and **has not returned it yet**.

**rewards\_report**

rewards\_report(min\_months INTEGER, min\_dollar\_amount NUMERIC) RETURNS SETOF customer

* Should return customers who:
  + Have been active for at least min\_months
  + And have spent at least min\_dollar\_amount
* Uses **dynamic SQL** in EXECUTE (runs the SQL stored in tmpSQL)

**last\_day**

last\_day(p\_date TIMESTAMP) RETURNS TIMESTAMP

* Returns the **last day of the month** for a given date.
* Example: last\_day('2025-04-16') → 2025-04-30
* **Used for:** Likely in reporting or billing periods.

g)

**tmpSQL TEXT;**

* A variable is declared to **hold the SQL string**.

**tmpSQL := 'SELECT ...' || min\_months || ' ...';**

* The SQL is being built **as a string**, using **concatenation** with input parameters.

**RETURN QUERY EXECUTE tmpSQL;**

* This is a **dynamic SQL execution point (very difficult to debug)**.

**tmpSQL** can be removed. The logic could be written as a **normal SQL query** with parameters. Something similar with the code for Task 6.

DROP FUNCTION IF EXISTS rewards\_report(INTEGER, NUMERIC);

CREATE OR REPLACE FUNCTION rewards\_report(

min\_months INTEGER,

min\_dollar\_amount NUMERIC

)

RETURNS TABLE (

customer\_id INT,

first\_name TEXT,

last\_name TEXT,

email TEXT,

active BOOLEAN,

total\_spent NUMERIC,

months\_active INT

)

LANGUAGE plpgsql

AS $$

BEGIN

RETURN QUERY

SELECT

c.customer\_id,

c.first\_name,

c.last\_name,

c.email,

c.active,

SUM(p.amount) AS total\_spent,

DATE\_PART('month', AGE(MAX(p.payment\_date), MIN(p.payment\_date))) AS months\_active

FROM customer c

INNER JOIN payment p ON c.customer\_id = p.customer\_id

GROUP BY c.customer\_id

HAVING

SUM(p.amount) >= min\_dollar\_amount

AND DATE\_PART('month', AGE(MAX(p.payment\_date), MIN(p.payment\_date))) >= min\_months;

END;

$$;

e)

**\_group\_concat:**

* Combines strings across grouped rows (aggregate)
* Used internally by group\_concat().

**group\_concat:**

|  |
| --- |
|  |

* SQL function wrapper around \_group\_concat.

|  |
| --- |

* Usage:

SELECT group\_concat(title) FROM film WHERE rating = 'PG';

c)

**It can be removed: \_group\_concat**

* Appears to be a function used only internally.
* It is not used anywhere else (like in views or functions).
* PostgreSQL has built-in STRING\_AGG(), so this function may be unnecessary (SELECT STRING\_AGG(title, ', ') FROM film;).

f)

last\_updated() RETURNS TIMESTAMP

Returns the **current timestamp** (NOW()) or the **last update timestamp** from the schema.

Used in:

* Triggers
* Default value generators

Check usage with:

SELECT \* FROM information\_schema.routines WHERE routine\_name = 'last\_updated';

*b)*

**Fix rewards\_report**

**Mentor’s comment**: “ function display 0 because is using previous 3 months, however data is older than that, try to update the code to consider it.”

**Solution:**

Instead of computing months from **latest-to-earliest payment**, we’ll calculate from **today** to earliest payment — so even old data qualifies.

*CREATE OR REPLACE FUNCTION rewards\_report(*

*min\_months INTEGER,*

*min\_dollar\_amount NUMERIC*

*)*

*RETURNS TABLE (*

*customer\_id INT,*

*first\_name TEXT,*

*last\_name TEXT,*

*email TEXT,*

*active BOOLEAN,*

*total\_spent NUMERIC,*

*months\_active INT*

*)*

*LANGUAGE plpgsql*

*AS $$*

*BEGIN*

*RETURN QUERY*

*SELECT*

*c.customer\_id,*

*c.first\_name,*

*c.last\_name,*

*c.email,*

*c.active,*

*SUM(p.amount) AS total\_spent,*

*DATE\_PART('month', AGE(CURRENT\_DATE, MIN(p.payment\_date))) AS months\_active*

*FROM customer c*

*JOIN payment p ON c.customer\_id = p.customer\_id*

*GROUP BY c.customer\_id*

*HAVING*

*SUM(p.amount) >= min\_dollar\_amount*

*AND DATE\_PART('month', AGE(CURRENT\_DATE, MIN(p.payment\_date))) >= min\_months;*

*END;*

*$$;*

*d)*

**Fix get\_customer\_balance**

**Mentor’s comment**: “ In case customer is overdue 2x the rental duration, the replacement cost should be charge, try to update the function to include that.”

*CREATE OR REPLACE FUNCTION get\_customer\_balance(*

*p\_customer\_id INTEGER,*

*p\_effective\_date TIMESTAMP*

*)*

*RETURNS NUMERIC*

*LANGUAGE plpgsql*

*AS $$*

*DECLARE*

*total\_payments NUMERIC;*

*total\_rentals NUMERIC;*

*overdue\_penalties NUMERIC;*

*BEGIN*

-- Total payments made by customer

*SELECT COALESCE(SUM(amount), 0)*

*INTO total\_payments*

*FROM payment*

*WHERE customer\_id = p\_customer\_id*

*AND payment\_date <= p\_effective\_date;*

-- Total rental fees for returned items

*SELECT COALESCE(SUM(amount), 0)*

*INTO total\_rentals*

*FROM payment*

*WHERE customer\_id = p\_customer\_id*

*AND payment\_date <= p\_effective\_date;*

-- Replacement cost for overdue unreturned items (2x rental duration)

*SELECT COALESCE(SUM(f.replacement\_cost), 0)*

*INTO overdue\_penalties*

*FROM rental r*

-- Join with film via inventory → film to get replacement\_cost

*JOIN inventory i ON r.inventory\_id = i.inventory\_id*

*JOIN film f ON i.film\_id = f.film\_id*

*WHERE r.customer\_id = p\_customer\_id*

*AND r.return\_date IS NULL* -- not yet returned

*AND CURRENT\_DATE > r.rental\_date + (f.rental\_duration \* INTERVAL '2 days');*

*RETURN total\_rentals + overdue\_penalties - total\_payments;*

*END;*

*$$;*