### **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.**

### 1. film\_in\_stock

### This stored procedure checks whether any copies of a specific film are available in stock at a given store.

### 2. film\_not\_in\_stock

### This stored procedure determines whether all copies of a given film are currently rented out at a specific store

### 3. inventory\_in\_stock

### This function checks if a specific inventory item is available in stock.

### 4. get\_customer\_balance

### This function returns the current amount owing on a specified customer's account as of a given date.

### 5. inventory\_held\_by\_customer

### This function returns the customer ID of the customer who has rented out a specified inventory item.

### 6. rewards\_report

### This stored procedure generates a customizable list of the top customers for the previous month, based on their rental activity.​

### 7. last\_day

### This function returns the last day of the month that contains the specified date.

**Why does ‘rewards\_report’ function return 0 rows? Correct and recreate the function, so that it's able to return rows properly.**

It’s checking payments from *three months ago* instead of *last month.*

CREATE OR REPLACE FUNCTION rewards\_report(min\_monthly\_purchases INT, min\_dollar\_amount\_purchased DECIMAL(10, 2))

RETURNS TABLE (

customer\_id INT,

first\_name VARCHAR,

last\_name VARCHAR,

email VARCHAR,

address\_id INT

) AS $$

DECLARE

last\_month\_start DATE;

last\_month\_end DATE;

tmpSQL TEXT;

BEGIN

-- Sanity checks on input parameters

IF min\_monthly\_purchases <= 0 THEN

RAISE EXCEPTION 'Minimum monthly purchases parameter must be > 0';

END IF;

IF min\_dollar\_amount\_purchased <= 0.00 THEN

RAISE EXCEPTION 'Minimum dollar amount purchased parameter must be > $0.00';

END IF;

-- Calculate the start and end dates of the last month

last\_month\_start := DATE\_TRUNC('month', CURRENT\_DATE) - INTERVAL '1 month';

last\_month\_end := (DATE\_TRUNC('month', CURRENT\_DATE) - INTERVAL '1 day');

-- Create a temporary table to store customer IDs who meet the criteria

CREATE TEMPORARY TABLE tmpCustomer (customer\_id INTEGER NOT NULL PRIMARY KEY);

-- Insert customer IDs into the temporary table based on the conditions

tmpSQL := format('

INSERT INTO tmpCustomer (customer\_id)

SELECT p.customer\_id

FROM payment AS p

WHERE DATE(p.payment\_date) BETWEEN %L AND %L

GROUP BY p.customer\_id

HAVING SUM(p.amount) >= %L AND COUNT(p.customer\_id) >= %L',

last\_month\_start, last\_month\_end, min\_dollar\_amount\_purchased, min\_monthly\_purchases);

EXECUTE tmpSQL;

-- Return customer information for those who meet the criteria

RETURN QUERY

SELECT c.customer\_id, c.first\_name, c.last\_name, c.email, c.address\_id

FROM tmpCustomer AS t

INNER JOIN customer AS c ON t.customer\_id = c.customer\_id;

-- Clean up the temporary table

DROP TABLE tmpCustomer;

RETURN;

END;

$$ LANGUAGE plpgsql;

**Is there any function that can potentially be removed from the dvd\_rental codebase? If so, which one and why?**

inventory\_in\_stock

* It just checks if a DVD (inventory item) is rented out or not.
* It's a super basic query:  
   *"Is this DVD currently rented out? Yes or no?"*
* Instead of calling a whole function, we could just run a small SELECT directly in your app or inside a query.

-- First, drop the function if it exists to avoid duplicates

**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.**

DROP FUNCTION IF EXISTS get\_customer\_balance(INTEGER, TIMESTAMP);

-- Now, create the function

CREATE OR REPLACE FUNCTION get\_customer\_balance(

p\_customer\_id INTEGER,

p\_effective\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

RETURNS NUMERIC(10,2) AS $$

DECLARE

v\_rentfees NUMERIC(10,2) DEFAULT 0; -- rental fees

v\_latefees NUMERIC(10,2) DEFAULT 0; -- late fees ($1 per day late)

v\_replacementfees NUMERIC(10,2) DEFAULT 0; -- replacement cost if way overdue

v\_payments NUMERIC(10,2) DEFAULT 0; -- payments made

v\_exists INTEGER;

BEGIN

-- Validate input: check if customer exists

SELECT COUNT(\*) INTO v\_exists

FROM customer

WHERE customer\_id = p\_customer\_id;

IF v\_exists = 0 THEN

RAISE EXCEPTION 'Customer ID % does not exist', p\_customer\_id;

END IF;

-- 1) RENTAL FEES FOR ALL PREVIOUS RENTALS

SELECT COALESCE(SUM(f.rental\_rate), 0)

INTO v\_rentfees

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.customer\_id = p\_customer\_id

AND r.rental\_date <= p\_effective\_date;

-- 2) ONE DOLLAR PER DAY LATE (AFTER rental\_duration)

SELECT COALESCE(SUM(

GREATEST(

DATE\_PART('day', COALESCE(r.return\_date, p\_effective\_date) - r.rental\_date) - f.rental\_duration,

0

)

), 0)

INTO v\_latefees

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.customer\_id = p\_customer\_id

AND r.rental\_date <= p\_effective\_date;

-- 3) REPLACEMENT COST if overdue more than rental\_duration \* 2

SELECT COALESCE(SUM(

CASE

WHEN DATE\_PART('day', COALESCE(r.return\_date, p\_effective\_date) - r.rental\_date) > (f.rental\_duration \* 2)

THEN f.replacement\_cost

ELSE 0

END

), 0)

INTO v\_replacementfees

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.customer\_id = p\_customer\_id

AND r.rental\_date <= p\_effective\_date;

-- 4) PAYMENTS MADE BEFORE THE DATE

SELECT COALESCE(SUM(p.amount), 0)

INTO v\_payments

FROM payment p

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

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

-- FINAL BALANCE

RETURN (v\_rentfees + v\_latefees + v\_replacementfees) - v\_payments;

END;

$$ LANGUAGE plpgsql;

**How do ‘group\_concat’ and ‘\_group\_concat’ functions work? (database creation script might help) Where are they used?**

**\_group\_concat** takes two strings and joins them with a comma.

**group\_concat** uses \_group\_concat to combine many rows into one single comma-separated string.

**What does ‘last\_updated’ function do? Where is it used?**

The last\_updated function just gives you the current date and time. It's used in the database to automatically fill the last\_update column when you add or change something — like when you insert a new actor, film, customer, etc.

**What is tmpSQL variable for in ‘rewards\_report’ function? Can this function be recreated without EXECUTE statement and dynamic SQL? Why?**

he tmpSQL variable is used to **build a SQL query as a text string**.  
 Later, the function **runs** that text using EXECUTE.

It’s basically like writing a SQL command inside a text and then telling the database: "Hey, run this!"

**Can the function be written without EXECUTE and dynamic SQL?** Yes, 100%.

Because the query itself **never really changes**, only the dates and numbers change.  
 You don't need to build the whole query as a string — you can just write normal SQL and use variables.