Task 6 / 1

Basically, two option to check what does the function. One is the script below, the other is finding it inside dbeaver navigator panel.

**SELECT** pg\_get\_functiondef(**'rewards\_report'**::**regproc**);

by checking thorugh, i found out the following answers:

**film\_in\_stock:** Checks if a particular film is in stock at a specific store. It returns the number of copies of the film that are currently in stock.

**film\_not\_in\_stock:** This function is likely the opposite of film\_in\_stock. It checks if a particular film is not in stock at a specific store. It might return the number of copies of the film that are currently rented out or not available.

**inventory\_in\_stock:** This checks the inventory status of all films in a store. It might return a list of films along with their stock status.

**get\_customer\_balance:** Returns the current balance of a specific customer’s account1. It might consider all the rentals and payments made by the customer up to a certain date.

**inventory\_held\_by\_customer:** This function returns the customer\_id of the customer who has rented out the specified inventory item2. If the item is in stock, it returns NULL.

**rewards\_report:** It generates a report of customers who meet certain purchase requirements from where basically rewards can be given to loyal customers.

**last\_day:** It takes a timestamp with time zone as an input and returns the date of the last day of the month for the given timestamp.

Task 6/2

I have modified the code, and then with it I could see an output, however there is nothing appears inside the report when calling the function

**SELECT** pg\_get\_functiondef(**'rewards\_report'**::**regproc**);

**CREATE** **OR** **REPLACE** **FUNCTION** public.rewards\_report(min\_monthly\_purchases **integer**, min\_dollar\_amount\_purchased **numeric**)

**RETURNS** **TABLE** (

customer\_id **INTEGER**,

first\_name **TEXT**,

last\_name **TEXT**

)

**LANGUAGE** plpgsql

**AS** **$function$**

**DECLARE**

last\_month\_start **DATE**;

last\_month\_end **DATE**;

**BEGIN**

**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 monthly dollar amount purchased parameter must be > $0.00'**;

**END** **IF**;

last\_month\_start := **CURRENT\_DATE** - **'1 month'**::**interval**;

last\_month\_start := **to\_date**((**extract**(**YEAR** **FROM** last\_month\_start) || **'-'** || **extract**(**MONTH** **FROM** last\_month\_start) || **'-01'**),**'YYYY-MM-DD'**);

last\_month\_end := LAST\_DAY(last\_month\_start);

**RETURN** QUERY

**SELECT** c.customer\_id, c.first\_name, c.last\_name

**FROM** payment **AS** p

**INNER** **JOIN** customer **AS** c **ON** p.customer\_id = c.customer\_id

**WHERE** **DATE**(p.payment\_date) **BETWEEN** last\_month\_start **AND** last\_month\_end

**GROUP** **BY** c.customer\_id, c.first\_name, c.last\_name

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

**AND** **COUNT**(p.customer\_id) > min\_monthly\_purchases;

**END**

**$function$**

And I tried the following call:  
**SELECT** \* **FROM** rewards\_report(1, 10.00);

Task 6/3

Okay, so I think that the function “film\_in\_stock” is unnecessary since ther exist a “film\_in\_stock\_by\_title” function from where if we did not add any parameters, it should list us all the films that are in stock.

There is two function, the “get\_sales\_revenue\_by\_category\_qtr (Date)” and “get\_sales\_revenue\_by\_category\_qtr (int4, int4) which I think both is unnecessary to have, one should be enough.

“Inventory\_held\_by\_customer” is makes some sense to have, but I don’t think it is necessary, because it can be simply SELECTED. But, maybe make sense if the function works in a way that I only have to add the customer name, and then represent the inventory. Actually, it is useful, nevermind.

Task 6/4

**CREATE** **OR** **REPLACE** **FUNCTION** public.get\_customer\_balance(p\_customer\_id **integer**, p\_effective\_date **timestamp** **with** **time** **zone**)

**RETURNS** **numeric**

**LANGUAGE** plpgsql

**AS** **$function$**

**DECLARE**

v\_rentfees **DECIMAL**(5,2); --#FEES PAID TO RENT THE VIDEOS INITIALLY

v\_overfees **DECIMAL**(5,2); --#LATE FEES FOR PRIOR RENTALS

v\_replacefees **DECIMAL**(5,2); --#REPLACEMENT FEES FOR VERY LATE RENTALS

v\_payments **DECIMAL**(5,2); --#SUM OF PAYMENTS MADE PREVIOUSLY

**BEGIN**

**IF** p\_customer\_id **IS** **NULL** **OR** p\_effective\_date **IS** **NULL** **THEN**

**RAISE** **EXCEPTION** **'Customer ID and effective date must be provided'**;

**END** **IF**;

**SELECT** **COALESCE**(**SUM**(film.rental\_rate),0) **INTO** v\_rentfees

**FROM** film, inventory, rental

**WHERE** film.film\_id = inventory.film\_id

**AND** inventory.inventory\_id = rental.inventory\_id

**AND** rental.rental\_date <= p\_effective\_date

**AND** rental.customer\_id = p\_customer\_id;

**SELECT** **COALESCE**(**SUM**(**CASE**

**WHEN** (rental.return\_date - rental.rental\_date) > (film.rental\_duration \* **'1 day'**::**interval**)

**THEN** **EXTRACT**(epoch **FROM** ((rental.return\_date - rental.rental\_date) - (film.rental\_duration \* **'1 day'**::**interval**)))::**INTEGER** / 86400 -- \* 1 dollar

**ELSE** 0

**END**),0)

**INTO** v\_overfees

**FROM** rental, inventory, film

**WHERE** film.film\_id = inventory.film\_id

**AND** inventory.inventory\_id = rental.inventory\_id

**AND** rental.rental\_date <= p\_effective\_date

**AND** rental.customer\_id = p\_customer\_id;

**SELECT** **COALESCE**(**SUM**(**CASE**

**WHEN** (rental.return\_date - rental.rental\_date) > (film.rental\_duration \* 2 \* **'1 day'**::**interval**)

**THEN** film.replacement\_cost

**ELSE** 0

**END**),0)

**INTO** v\_replacefees

**FROM** rental, inventory, film

**WHERE** film.film\_id = inventory.film\_id

**AND** inventory.inventory\_id = rental.inventory\_id

**AND** rental.rental\_date <= p\_effective\_date

**AND** rental.customer\_id = p\_customer\_id;

**SELECT** **COALESCE**(**SUM**(payment.amount),0) **INTO** v\_payments

**FROM** payment

**WHERE** payment.payment\_date <= p\_effective\_date

**AND** payment.customer\_id = p\_customer\_id;

**RETURN** v\_rentfees + v\_overfees + v\_replacefees - v\_payments;

**END**

**$function$**

Task 6/5

\_Group\_concat function: This concatenates two values with a comma. It can be used where you want to concatenate text fields from multiple rows into a signle text string. But it only concatenates TWO text value.

Group\_concat function:

It is similarly concatenates, but it does with data from multiple rows into one field and returns a string value if the group contains at least one non-NULL value, otherwise it is NULL. Well it should be used when you want to aggregate data from multiple rows into a single field. For example it can be used to create a list of all items purchased by each customer in an e-commerce database..

Task 6/6

Last\_updated function, it triggers as a machine gun if a row is inserted or updated in a table. It’s purpose is to update automatically the last\_update field in a table with the current timestamp whenever a row is inserted or updated, it makes it easier to track when each row was updated.

Task 6/7

I am having issues previously with the rewards\_report function, but to answer the question about the tmpSQL variables role. tmpSQL is build an INSERT query that inserts customer IDs into a temporary table. In this case, by using the min\_monthly\_purchases and min\_dollar\_amount\_purchased parameters and of course the last\_month\_start and last\_month\_end variables. The execute statement is used to execute the dynami SQL query.  
Technically, it should be created without the EXECUTE statement and dynamic sql. If there is no need to build queries based on variables or conditions that are not known until runtime, we can use static SQL.