**HW Tasks: applying view and functions**

Task 6. Prepare answers to the following questions

1. Functions in the dvdrental database:

* The **function film\_in\_stock** is returning a list of inventory IDs for a specific film that is currently in stock at a particular store using the returns setof option which allows to return one or more rows based on the predefined structure from the function.
* The **film\_not\_in\_stock function** will return a list of inventory IDs for a specific film that currently is not in stock at a particular store using the returns setof option.
* The **inventory\_in\_stock function** will return a boolean value based on the input parameter which is a specific inventory item in the form of the inventory\_id and determine whether this inventory\_id is in stock. The function declares two variables:
  + v\_rentals – calculates the count of rentals for the specified inventory\_id
  + v\_out – calculates the count of currently rented inventory items that have not been returned meaning the return date is NULL
* The **get\_customer\_balance function** calculates the balance of a customer based on a specified date using the input parameters such as customer\_id and effective date. The customer balance is defined as a sum of rental fees, overdue charges, and payments made. The function declares three variables:
  + v\_rentfees - the total amount the customer has paid for rentals up to the specified date
  + v\_overfees - the total late fee for overdue rentals
  + v\_payments - the total amount of payments made by the customer up to the specified date
* The **inventory\_held\_by\_customer function** will define which customer currently has a specific inventory item rented. The function returns the customer\_id of the customer who has an item, identified by the input parameter when the item has not yet been returned meaning the return\_date is NULL.
* The **rewards\_report function** generates a report of customers who meet a specific purchase criteria within a three month interval. The function identifies customers who have made a minimum number of monthly purchases and spent a minimum dollar amount on their purchases. These thresholds are defined in the input parameters. The function also creates a temporary table which is used to store customer\_id of customers who meet the previously defined purchase criteria and in the next step the query retrieves all the customers related information from the customer table who are present in the temporary table using a loop. After collecting the customer information, the temporary table is dropped to clean up the database.
* The **last\_day function** calculates and returns the last date of the month for a given timestamp input. The function takes a timestamp as an input and returns the last day of the month for the inputted timestamp. It calculates the last day of the month by subtracting 1 day from the first day of the next month and in case of December the function will subtract 1 day from January 1 of the next year. The function is defined as:
  + Immutable - which means that the function’s output is always the same for the same input parameters and doesn’t depend on any database state that could change
  + Parallel unsafe – which means that the function cannot be safely run in parallel.
  + Strict – which means that the function always returns null whenever any of its input arguments are null

1. The rewards\_report function may return 0 rows due the date range that is defined in the code. Retrieving the most recent payment date in the payment table we can establish that it was in 2017:  
   SELECT max(p.payment\_date) as max\_date

FROM public.payment p

Result: 2017-05-31 23:00:00.000 +0200

To modify the function so that it uses the year 2017 and calculates the last three months from a specific date, for example from 2017-05-31, we can replace the current\_date with the fixed date by defining the selected date in the declare section of the function:

DECLARE   
 selected\_date DATE := '2017-05-31';  
 last\_month\_start DATE;   
 last\_month\_end DATE;  
 rr RECORD;   
 tmpSQL TEXT;

We can use this selected\_date to calculate the 3 month interval:

last\_month\_start := selected\_date - '3 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(selected\_date - '1 month'::interval);

1. From the dvd\_rental codebase the last\_day function can potentially be removed. The reason for potentially removing this function from the database is that it calculates the last date of the month for a given timestamp input which can be achieved using date\_trunc and interval functions:

SELECT (date\_trunc('month', current\_date) + INTERVAL '1 month - 1 day')::DATE AS last\_day\_of\_month;

1. The ‘get\_customer\_balance’ function describes a business requirements that wasn’t implemented and it is related to calculating the replacement cost when a rental is overdue by the rental\_duration \* 2. The function should select all the overdue rental days which exceed the rental\_duration \* 2 and then find the replacement cost for each film.

First step is adding a new variable to the declare section of the function

DECLARE  
 v\_replacement\_cost decimal(5,2);

In the next step, case statement will check if the rental is more than rental\_duration \* 2 days overdue. If yes, then the replacement\_cost for that film is added to the v\_replacement\_cost variable:

SELECT COALESCE(SUM(CASE WHEN (r.return\_date - r.rental\_date) > (f.rental\_duration \* 2 \* '1 day'::interval)   
THEN f.replacement\_cost   
ELSE 0  
END), 0)   
INTO v\_replacement\_cost   
FROM public.rental r   
INNER JOIN public.inventory i ON r.inventory\_id = i.inventory\_id   
INNER JOIN public.film f ON i.film\_id = f.film\_id   
WHERE r.rental\_date <= p\_effective\_date AND r.customer\_id = p\_customer\_id;

1. The group\_concat and the \_group\_concat function are connected to each other in the database.   
   The group\_concat creates a aggregation function which concatenates strings while the \_group\_concat function defines the rules of concatanation.
2. The last\_updated function includes a trigger which automatically updates a last\_update column with the current timestamp whenever a table is updated. The last\_update column is part of every table in the dvd\_rental database except the payment table.
3. tmpSQL variable in the ‘rewards\_report’ function is used to define dynamic SQL queries. Dynamic SQL refers to a query that is executed at runtime, rather than being predefined in the function. In the ‘rewards\_report’ function, last\_month\_start and last\_month\_end are dynamic variables and they are calculated dynamically based on the current date. It is not possible to pass last\_month\_start and last\_month\_end as input parameters and that is the reason why they are calculated dynamically based on the current date.  
   The function can’t be rewritten without the use of dynamic SQL as that would require to hardcode the values for dates.