PL/SQL Repetition Exercise 3

2020-09-23

Instructions

- The tasks are based on the sample data used for the SQL Repetition Exercise.
- Extend the package 'electronics_merchant' (header and body) which provides the functionalities listed below.
- **For each task also provide a sample (call) in the test driver!**
- Hand in header, body, test driver and the output of the test driver.
- This is a continuation of the first two PL/SQL repetition exercises so you have to complete these before starting with this.

1 Warehouse Management

The final task is to manage warehouses and our product stock[1].

1.1 Stock value

For bragging reasons our CEO requires the latest information about his company's value at all times. We are tasked to provide PL/SQL functions which he can then call from SQLDeveloper on his phone[2] while on the golf course to get the desired information.

The following information is required:

- For each country
  - The number of warehouses
  - The total stock value
  - This countries percentage of the total stock value
- The total stock value

1.1.1 Remembering Views

Answer the following questions:

1. What is a *View*?
2. How to Create/Modify/Delete a view?
3. Is it part of a package or on the schema level like a trigger?
4. How and when are views (values) updated?
5. Do views improve query performance?

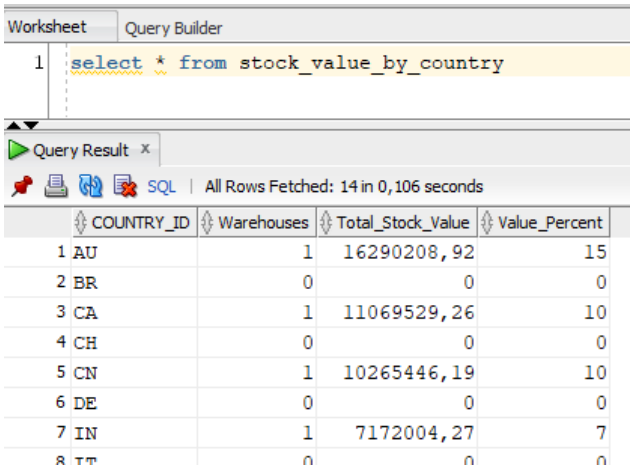
1.1.2 Creating information Views

First, we need to collect some data which can be collected using nothing but pure SQL (no PL/SQL!):

[1] When talking about stock always the number of products in warehouses is meant, not the shares.
[2] Admittedly, the PL/SQL functions being a backend for a webservice would be more realistic...

- The number of warehouses per country
- The stock value in all warehouses of a country
  - Hint: The stock value is based on the quantity and list price of the stored[3] products.

To encapsulate our query and make the results easily reusable we will create a *View* for this job. Give the view the name stock_value_by_country and design it so that a query as shown below is possible (pay attention to rounded values and row order):



	COUNTRY_ID	Warehouses	Total_Stock_Value	Value_Percent
1	AU	1	16290208,92	15
2	BR	0	0	0
3	CA	1	11069529,26	10
4	CH	0	0	0
5	CN	1	10265446,19	10
6	DE	0	0	0
7	IN	1	7172004,27	7
8	IT	0	0	0

Also needed is the total stock value of the company. We will provide that with a view total_stock_value as well. **Use this view in the other one for calculating the percentage!**

1.1.3 FUNCTION get_company_value

- Retrieves data from the views created in 1.1.2.
- Parameter:
  - None.
- Returns a rt_company_value *record*
- rt_company_value fields:
  - country_values: An *associative array* with
    - The country code as key
    - A rt_country_stock *record*
  - total_value: The total value of the company's stock
- rt_country_stock fields:
  - no_of_warehouses: The number of warehouses in the country.
  - total_stock_value: The total value of the products stored in the country.
  - value_percentage: The percentage of this country's products worth compared to that of the whole company. Attention: this is a string value including the '%' sign!

1.2 Performing Inventories

Since we have no real-time stock information (based on receiving and sending shipments) the inventories are especially important.

1.2.1 PROCEDURE process_inventory_counts

- Processes inventory counts of products in a specific warehouse.

[3] We rely only on the inventory snapshots for stock information.

- Parameter:
  - A *list* of inventory counts (= the *ROWTYPE* of the Inventories table)
- Throw an exception if a problem during processing occurs. Use only one exception number, but provide a message with detailed information:
  - Warehouse does not exist
  - Product does not exist
  - Quantity is invalid (< 0)
  - Counts list is null
- Create an additional (package internal) function for performing the validation within the procedure.
  - The function checks for all possible problems at once.
  - Find a way to return what issue (warehouse does not exist, or product does not exist, or ...) caused the validation to fail.
  - How do you recognize a successful validation?
- You have to either update an existing count or insert a new one.

1.2.2 Audit Table

In the inventories table we have but one entry for each warehouse and product combination. But it can be very interesting to look at the historic changes of a product stock (to plan resupply,...). For that we will use a new *Inventories_Audit*[4] table with the following fields:

- Timestamp
- Warehouse ID
- Product ID
- Quantity

Your tasks:

1. Create the table.
2. Find proper datatypes for the fields.
3. Find the correct primary key.
4. Don't forget to define the foreign key constraints.

1.2.3 Populating audit table

Create a *trigger* inventory_audit which adds a new entry to the audit table if:

1. Either a new entry is added to the Inventories table or
2. An existing value is changed in the Inventories table (but *only* if the quantity actually changed!) This includes removing an entry which sets the quantity to 0.

[4] Once again we use a pluralized table name to satisfy conformity with the other sample tables. But please remember that normally you should use singular.