PL/SQL Repetition Exercise 2

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 PL/SQL repetition exercise so you have to complete that before starting with this.

1 Order Management

1.1 Adding a new order

To make money the company needs to take orders, so let's add this feature. Don't forget that orders are composed of order items.

1.1.1 PROCEDURE add_order

- Inserts a new order into the database.
- Parameter:
  - A rt_new_order record with information about the new order.
- rt_new_order fields:
  - customer_id: The ID of the customer placing the order.
  - salesman_id: The ID of the employee who made the sale.
  - order_items: A *list (Nested Table)* of rt_order_item records.
- rt_order_item fields:
  - product_id: The ID of the product.
  - quantity: The number of pieces ordered.
  - unit_price: The price per piece.
- Uses the function 1.1.2 to determine if the supplied order is valid.
- If the supplied order is not valid raise an application error.

1.1.2 FUNCTION validate_order

- Checks if the supplied data for a new order is valid.
- This includes:
  - Checking if all foreign key constraints can be satisfied (= ID's exist in referenced tables).
  - Checking if the quantity and unit price have reasonable values.
- Parameter:
  - A rt_new_order record with information about the new order.
- Returns a boolean value indicating if any problems were found (to make things easier you do not have to list the problems individually).

- Important: You *have* to use at least one VARRAY in the function implementation.

1.2 Updating an order

1.2.1 PROCEDURE update_order

- Updates an existing order.
- Be careful: you may have to either update the status or the salesman or even both.
- Parameter:
  - The ID of the order to update.
  - The new status of the order (may be NULL).
  - The new responsible salesman (may be NULL).
- Do not forget to raise a proper exception if
  - The order does not exist
  - The status is unknown
  - The salesman does not exist
- Important: the salesman *can* be NULL in the orders table but to simplify the distinction between 'update only status' and 'update only salesman' we assume that we never set a salesman to NULL.

1.2.2 PROCEDURE update_order_item

- Adds, updates or deletes an order item.
- Parameter:
  - A rt_order_item_update record with information about the order item.
- rt_order_item_update fields:
  - order_id: The ID of the order.
  - item_id: The ID of the item to update or delete – NULL if adding.
  - do_delete: Flag indicating if this is a delete operation.
  - do_add: Flag indicating if this is an add operation.
  - product_id: The ID of the product – only set when adding.
  - new_quantity: The new quantity – NULL when deleting.
  - new_unit_price: The new unit price – NULL when deleting.
- Do not forget to validate
  - Setting a negative quantity
  - Updating a non existent entry
  - Adding an entry for a product which already exists
  - ...

1.3 Offering bonus payments

Employees are much more motivated if they can profit from the company's success (in addition to not being fired). So we will provide a bonus to everyone who lands a sale.

1.3.1 Extending the Compensations table

- Add a new column 'bonus' (NUMBER(7,2)) to the table.
- Find proper defaults for the existing employees.
- What has to happen if a new employee is added? If changes are necessary apply them.

1.3.2 FUNCTION calc_bonus

- Calculates the bonus an employee receives following these rules:
  - The base value is the total revenue[1] the company made thanks to the employees sales[2].
  - Sales are determined by the salesman_id column in the orders table.
  - Total revenue is calculated via the order items (quantity * price).
  - Of this base value 0.75% are calculated – this is the bonus the employee receives.
- Parameter:
  - The ID of the employee for whom the bonus should be calculated.
- Returns the calculated bonus.

1.3.3 Storing the bonus

So far we only calculated the bonus value. Now you need to set & update the value where necessary.

- Create a procedure for setting the bonus for an employee.
  - No detailed instructions this time. Try to create a procedure meeting all the requirements yourself.
- The procedure has to use the function created in 1.3.2.
- You'll need to call it in several places.
- Do not add the procedure to the header of the package – this is an internal functionality we do not want to expose.
  - Hint: to test a private method you'll have to call a public method utilizing it.
- Think about the transactions updating our data: where and when do you have to commit and when not?

1.3.4 Limiting the bonus

Management realized that the company is paying vast amounts of money to employees. So they made two decisions:

1. Employee bonuses now have an upper limit of 20000.
2. The changes cannot be made in the PL/SQL package, because that has already been approved for production.

Thanks to the second brilliant idea of your manager you can't simply fix the 1.3.2 function but need to hot-patch the application with a *trigger*. Of course the correct value has to be inserted right away (same transaction) to not let any errors slip through. *Come up with a solution for the problem and create an appropriate trigger.*

[1] Calculating the actual profit by subtracting our own costs from the sales value would be more sensible but we'll just ignore that, because at school we may do so while at work you will not :)

[2] Usually this would be limited to a certain time range, e.g. the current year. But to work well with our test data we just use all orders no matter when they were placed.