Chapter 14
How to manage transactions and locking

Exercises

1. Write a script that uses an anonymous block to include two SQL statements coded as a transaction to delete the row with a customer ID of 8 from the Customers table. To do this, you must first delete all addresses for that order from the Addresses table.

   If these statements execute successfully, commit the changes. Otherwise, roll back the changes.

2. Write a script that uses an anonymous block that includes these statements coded as a transaction:

   ```sql
   -- hard-code the id values for orders and order_items
   INSERT INTO orders VALUES
   (DEFAULT, 3, NOW(), '10.00', '0.00', NULL, 4, 'American Express', '378282246310005', '04/2013', 4);

   INSERT INTO order_items VALUES
   (DEFAULT, order_id, 6, '415.00', '161.85', 1);

   INSERT INTO order_items VALUES
   (DEFAULT, order_id, 1, '699.00', '209.70', 1);

   If these statements execute successfully, commit the changes. Otherwise, roll back the changes.
Chapter 15
How to create stored procedures and functions

Exercises

1. Write a script that creates and calls a stored procedure named insert_category. First, code a statement that creates a procedure that adds a new row to the Categories table. To do that, this procedure should have one parameter for the category name. Code at least two CALL statements that test this procedure. (Note that this table doesn’t allow duplicate category names.)

2. Write a script that creates and calls a stored function named discount_price that calculates the discount price of an item in the Order_Items table (discount amount subtracted from item price). To do that, this function should accept one parameter for the item ID, and it should return the value of the discount price for that item. Code a SELECT statement that tests this function.

3. Write a script that creates and calls a stored function named item_total that calculates the total amount of an item in the Order_Items table (discount price multiplied by quantity). To do that, this function should accept one parameter for the item ID, it should use the discount_price function that you created in exercise 2, and it should return the value of the total for that item. Code a SELECT statement that tests this function.

4. Write a script that creates and calls a stored procedure named insert_products that inserts a row into the Products table. This stored procedure should accept five parameters, one for each of these columns: category_id, product_code, product_name, list_price, and discount_percent.

   This stored procedure should set the description column to a string that contains a single space, and it should set the date_added column to the current date.

   If the value for the list_price column is a negative number, the stored procedure should raise an error that indicates that this column doesn’t accept negative numbers. Similarly, the procedure should raise an error if the value for the discount_percent column is a negative number.

   Code at least two CALL statements that test this procedure.

5. Write a script that creates and calls a stored procedure named update_product_discount that updates the discount_percent column in the Products table. This procedure should have one parameter for the product ID and another for the discount percent.

   If the value for the discount_percent column is negative, the stored procedure should raise an error that the value for this column must be a positive number.

   Code at least two CALL statements that test this procedure.
Chapter 16
How to create triggers

Exercises

1. Create a trigger named products_before_update that checks the new value for the discount_percent column of the Products table. This trigger should raise an appropriate error if the discount percent is greater than 100 or less than 0.

   If the new discount percent is between 0 and 1, this trigger should modify the new discount percent by multiplying it by 100. That way, a discount percent of .2 becomes 20.

   Test this trigger with an appropriate UPDATE statement.

2. Create a trigger named products_before_insert that inserts the current date for the date_added column of the Products table if the value for that column is null.

   Test this trigger with an appropriate INSERT statement.

3. Create a table named Products_Audit. This table should have all columns of the Products table, except the description column. Also, it should have an audit_id column for its primary key, and the date_added column should be changed to date_updated.

   Create a trigger named products_after_update. This trigger should insert the old data about the product into the Products_Audit table after the row is updated. Then, test this trigger with an appropriate UPDATE statement.