**Database Application Development**

**Assignment 2 (30%)**

**Objective:**

In this assignment, you create a simple Retail application using the C++ programming language and Oracle (PL/SQL). This assignment helps students learn a basic understanding of application development using C++ programming and an Oracle database using PL/SQL.

**Submission:**

***Your submission will be a text-based .cpp file including your C++ program and a text-based .sql file including your Oracle stored procedures for the Database Application assignment.***

DBS311sectioncode\_ASS2\_group#.cpp

DBS311sectioncode\_ASS2\_group#.sql

***See the following example:***

DBS311NGG\_ASS2\_group02.cpp

Your submission needs to be commented.

You submit:

The .sql file includes all stored procedures that you are asked to write in the assignment. Write stored procedures in the same order that they are defined in the assignment document. Test all your procedures and make sure that they can be called from the C++ program with no errors.

The .cpp file is the same as application.cpp with two additional functions that you define based on the given instruction in the assignment document. The submitted C++ program must be error free and all functions must work successfully. Test your program before submitting it on Blackboard.

**Marking Scheme**

***The course promotion policy requires all students to complete the assignment successfully to pass the course***.

Submissions with errors will not be considered as completed and will receive zero.

If all stored procedures are not created successfully and completely, the assignment will not be considered as completed and will not receive the full mark.

If all stored procedures are created successfully, the C++ program is completed and works with no errors (All functions are defined and perform their task without generating errors), and answer the questions successfully from the reviewer, the assignment submission is considered as successfully completed and receives the full mark.

|  |  |  |
| --- | --- | --- |
| Tasks | Weight | Total Weight |
| 8 Oracle stored procedures/Functions | 10% each | 80% |
| Two C++ functions | 10% each | 20% |
|  | **Total** | 100% |

You may be asked to complete and submit the assignment again after the assignment is marked. The mark you receive for the assignment will not changed for the new submission. However, you may need to complete and submit the assignment to satisfy the course requirements.

**Instruction**

In this assignment, we use the same database that you have been using for the labs and the assignment 1.

**Note**: For each query in your assignment, you handle the errors and display the proper message including the error code and the error message.

try{

...

}

catch (SQLException& sqlExcp) {

cout << sqlExcp.getErrorCode() << ": " << sqlExcp.getMessage();

}

The following structure is declared before the ***main()*** function:

struct ShoppingCart {

int product\_id;

double price;

int quantity;

};

***Connecting to an Oracle database from a C++ Program***

In your function ***main()***, a connection is created to your database.

We first declare the environment and the connection variables.

Environment\* env = nullptr;

Connection\* conn = nullptr;

We also define and initialize the variable to store the username, password, and the host address.

string user = "username";

string pass = "password";

string constr = "myoracle12c.senecacollege.ca:1521/oracle12c";

You use the same Oracle username and password that you use for your labs and assignments.

After creating the environment and the connection, we make sure to handle any errors may be thrown as the program is executed.

env = Environment::createEnvironment(Environment::DEFAULT);

conn = env->createConnection(user, pass, constr);

We terminate and close the connection and the environment, when the program terminates.

env->terminateConnection(conn);

Environment::terminateEnvironment(env);

After executing the statements, we make sure that the statement is terminated.

conn->terminateStatement(stmt);

You will implement the following Oracle stored procedures and will call them in the following C++ functions:

***Stored Procedures***

*find\_customer (customer\_id IN NUMBER, found OUT NUMBER);*

This procedure has an input parameter to receive the customer ID and an output parameter named found.

This procedure looks for the given customer ID in the database. If the customer exists, it sets the variable *found* to 1. Otherwise, the *found* variable is set to 0.

To check if your query in the ***find\_customer()***procedure returns a row, you need to check the no\_data\_found exception in the EXCEPTION block.

EXCEPTION

WHEN no\_data\_found THEN

found := 0;

To check if your query in the ***find\_customer()***procedure returns multiple rows, you need to check the too\_many\_rows exception in the EXCEPTION block and display a proper message.

To catch any other errors, check the OTHERS exception in the EXCEPTION block and display a proper message.

*find\_product (productId IN NUMBER,*

*price OUT products.list\_price%TYPE,*

*productName OUT products.product\_name%TYPE);*

This procedure has an input parameter to receive the product ID and an output parameter named price.

This procedure looks for the given product ID in the database. If the product exists, it stores the product’s product\_name in the variable *productName* and the product’s list\_price in the variable *price*. If the product does not exist, the productName is set to null and the *price* variable is set to 0.

In November and December, the company gives 10% off on all products in categories 2 and 5. Before storing the list price of products that belong to categories 2 and 5 into the variable price, calculate the new list price if the current month is November or December and send back the new list price to the caller as price.

EXCEPTION

WHEN no\_data\_found THEN

price := 0;

To check if your query in the ***find\_product()***procedure returns multiple rows, you need to check the too\_many\_rows exception in the EXCEPTION block and display a proper message.

To catch any other errors, check the OTHERS exception in the EXCEPTION block. and display a proper message.

*add\_order (customer\_id IN NUMBER, new\_order\_id OUT NUMBER)*

This procedure has an input parameter to receive the customer ID and an output parameter named new\_order\_id.

To add a new order for the given customer ID, you need to generate the new order Id.

To calculate the new order Id, call the function generate\_order\_id(). The procedure add\_order inserts the following values in the orders table

new\_order\_id (as order\_id)

customer\_id (input parameter)

'Shipped' (The value for the order status)

56 (The sales person ID)

sysdate (order date which is the current date)

*generate\_order\_id ()*

***This is an Oracle function with no parameters.*** It finds the maximum order ID in the orders table and increase it by 1 as new order ID. The function ***returns*** the new order ID to the caller.

*add\_order\_item (orderId IN order\_items.order\_id%type,  
 itemId IN order\_items.item\_id%type,   
 productId IN order\_items.product\_id%type,   
 quantity IN order\_items.quantity%type,  
 price IN order\_items.unit\_price%type)*

This procedure has five IN parameters. It stores the values of these parameters to the table order\_items.

*customer\_order (customerId IN NUMBER, orderId IN OUT NUMBER)*

This procedure receives two values as customer ID and Order ID and confirms if there exists any order with this order ID for this customer in the orders table. If the order ID with this customer ID exists, the procedure passes the order ID to the caller. Otherwise, it passes 0 to the caller.

*display\_order\_status(orderId IN NUMBER, status OUT orders.status%type)*

This procedure has an input parameter to receive an order ID and an output parameter to pass the status of the order to the caller. IF the receiving order ID exists, the procedure stores the order status in the *status* variable. If the order ID does not exists, store null in the *status* variable.

*cancel\_order (orderId IN NUMBER, cancelStatus OUT NUMBER)*

This procedure has an input parameter to receive an order ID and an output parameter to pass a value to the caller. IF the receiving order ID exists, the procedure stores the value of the column *status* in a variable *orderStatu*s. If orderStatus is ‘Canceled’, the procedure stores 1 in to the parameter *cancelStatus*. If the *orderStatus* is shipped, the procedure stores 2 in to the parameter *cancelStatus*. Otherwise, it stores 3 in to the parameter *cancelStatus* and updates the status of that order to “Canceled”. If the order ID does not exists, it stores 0 in the cancel variable.

The *cancelStatus* parameter gets the following values:

0: The order does not exit.

1: The order has been already canceled.

2: The order is shipped and cannot be canceled.

3: The order is canceled successfully.

The stored procedure does not print any outputs.

***C++ Functions***

Please read the following functions’ definition to learn how the C++ code work. You need to call the stored procedures that you wrote in the previous section in these functions based on the following instruction. If the stored procedure pass values to the program, define variables or use variables defined in the given program to store passing values from the stored procedure.

**Your task in this section is to add two functions:**

*void displayOrderStatus(Connection\* conn, int orderId, int customerId);*

*void cancelOrder(Connection\* conn, int orderId, int customerId);*

***void displayOrderStatus(Connection\* conn, int orderId, int customerId);***

This function calls the customer\_order stored procedure to confirm the entered order ID belongs to the customer. If the value of the second parameter of the customer\_order procedure is a non-zero value, the order ID belongs to the customer.

If the value of the second parameter of the customer\_order procedure is zero, display the following message:

“Order ID is not valid.”

If the order ID is valid, call the *display\_order\_status* procedure to receive the order status stored in the second parameter of this stored procedure. If the order status is null, display the following message:

“Order does not exist.”

If the status is not null, display a proper message.

See the sample message:

“Order is shipped.”

***void cancelOrder(Connection\* conn, int orderId, int customerId);***

This function calls the *cancel\_order* stored procedure to confirm the entered order ID belongs to the customer. If the value of the second parameter of the customer\_order procedure is a non-zero value, the order ID belongs to the customer.

If the value of the second parameter of the *customer\_order* procedure is zero, display the following message:

“Order ID is not valid.”

0: The order does not exit.

1: The order has been already canceled.

2: The order is shipped and cannot be canceled.

3: The order is canceled successfully.

If the order ID is valid, call the *cancel\_order* procedure to the order with the order ID passed to this function. If the value passed by the second parameter of the *cancel\_order* is 1, display:

“The order has been already canceled.”

If the value passed by the second parameter of the *cancel\_order* is 2, display:

“The order is shipped and cannot be canceled.”

If the value passed by the second parameter of the *cancel\_order* is 3, display:

“The order is canceled successfully.”

The following functions exist in the application.cpp file. You will modify this file to add two C++ functions described above.

*void displayOrderStatus(Connection\* conn, int orderId, int customerId);*

*cancelOrder(Connection\* conn, int orderId, int customerId);*

*int mainMenu();*

The ***mainMenu()***function returns an integer value which is the selected option by the user from the menu. This function displays the following menu options:

1. Login
2. Exit

The program prompts the user to choose an option. If the user enters the wrong value, the program asks the user to enter an option again until the user enters a valid options.

See the following example:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

Enter an option (0-1): 5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

You entered a wrong value. Enter an option (0-1):

If the user chooses option 1, the program asks the user to enter customer ID to login. To see if the customer with the entered ID exists, the program calls the Oracle stored procedure ***find\_customer()***. IF the value of the output parameter in the procedure is 1, the program lets the customer continue. If the value of the output parameter found is 0, the program calls the ***mainMenu()*** function again and asks the customer to login again. The program continues this process until the user chooses the option 0 to exit or enters a valid customer ID.

*int subMenu();*

The ***subMenu()***function returns an integer value which is the selected option by the user from the menu. This function displays the following menu options:

1) Place an order

2) Check an order status

3) Cancel an order

0) Exit

If the user chooses option 1, the user can place an order. If the user chooses option 2, the displayOrderStatus function is called. If the user chooses option 3, the program calls the cancelOrder function. If the user chooses option 0, the user returns to the main menu.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Customer Service Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Place an order

2) Check an order status

3) Cancel an order

0) Exit

Enter an option (0-3):

*int customerLogin(Connection\* conn, int customerId);*

Before calling this function, the program prompts the user to enter the customer ID.

This function is called in the ***main()*** function if the user chooses the login option from the main menu. This function receives an integer value as a customer ID and checks if the customer does exist in the database. This function returns 1 if the customer exists. If the customer does not exists, this function returns 0 and the main menu is displayed.

To validate the customer ID, you call the ***find\_customer()*** stored procedure/function in this function.

See the following example:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

Enter an option (0-1): 1

Enter the customer ID: 1000

The customer does not exist.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

Enter an option (0-1): 1

Enter the customer ID: 44

-------------- Add Products to Cart --------------

Enter the product ID:

*int addToCart(Connection\* conn, struct ShoppingCart cart[]);*

If the customerLogin() functions return 1 (The customer ID exists), this function is called.

This function receives an OCCI pointer (a reference variable to an Oracle database) and an array of type ShoppingCart.

The customer can purchase up to five items in one order.

In a loop the program prompts the user to enter product IDs for the maximum of five products.

When the user enters the product ID in the *addToCart()* function, the findProduct() function is called to check if the product ID exists. IF the product exists, the function findProduct() returns the product’s price. The program displays the product’s price to the user and asks the user to enter the quantity.

If the user enters a valid product ID, the program displays the following message and let the user to enter another product ID.

"Enter 1 to add more products or 0 to checkout:"

If the user chooses 1, the program asks the user to enter the next product ID. Otherwise, the program goes to the next step to checkout. If the user enters 0, the function ***addToCart()***, returns the number of products (items) entered by the user.

For each product ID entered by the customer, the function ***findProduct()*** is called to see if the product ID exists.

If the ***findProduct()*** function returns 0 (The product ID does not exist), the program displays a proper message and lets the user enter the product ID again.

See the following example:

-------------- Add Products to Cart --------------

Enter the product ID: 1000

The product does not exists. Try again...

Enter the product ID: 900

The product does not exists. Try again...

Enter the product ID: 112

Product Price: 808.92

Enter the product Quantity: 3

Enter 1 to add more products or 0 to checkout: 1

Enter the product ID: 115

Product Price: 699.99

Enter the product Quantity: 2

Enter 1 to add more products or 0 to checkout: 0

*double findProduct(Connection\* conn, int product\_id);*

This function receives an OCCI pointer (a reference variable to an Oracle database) and an integer value as the product ID.

When the user enters the product ID in the ***addToCart()*** function, the function ***findProduct()*** is called.

This functions calls the ***find\_product()*** Oracle stored procedure/function. The procedure receives the product ID and returns the price. If the price is 0, the product ID is not valid (does not exist). If the price is a non-zero value, it means the product ID is valid.

*void displayProducts(struct ShoppingCart cart[], int productCount);*

This function receives an array of type ShoppingCart and the number of ordered items (products). It display the product ID, price, and quantity for products stored in the cart array.

This function is called after the function ***AddToCart()*** to display the products added by the user to the shopping cart.

------- Ordered Products ---------

---Item 1

Product ID: 112

Price: 808.92

Quantity: 3

---Item 2

Product ID: 115

Price: 699.99

Quantity: 2

----------------------------------

Total: 3826.74

After displaying the products’ information (product ID, price, and quantity), the program displays the total order amount. To calculate the total order amount, the program first multiply the quantity and the price to calculate the total amount for each product. Next, it sums up products’ total amounts to calculate the total order amount.

*int checkout(Connection \*conn, struct ShoppingCart cart[], int customerId, int productCount);*

This function is called after the function ***displayProduct()***.

This function receives an OCCI pointer (a reference variable to an Oracle database), an array of type ShoppingCart, an integer value as the customer ID, and an integer value as the number of ordered items (products).

First, the program displays the following message:

"Would you like to checkout? (Y/y or N/n) "

If the user enters any values except “Y/y” and “N/n”, the program displays a proper message and asks the user to enter the value again.

"Wrong input. Try again..."

See the following example:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

Enter an option (0-1): 1

Enter the customer ID: 4

-------------- Add Products to Cart --------------

Enter the product ID: 112

Product Price: 808.92

Enter the product Quantity: 3

Enter 1 to add more products or 0 to checkout: 0

------- Ordered Products ---------

---Item 1

Product ID: 112

Price: 808.92

Quantity: 3

----------------------------------

Total: 2426.76

Would you like to checkout? (Y/y or N/n) t

Wrong input. Try again...

Would you like to checkout? (Y/y or N/n) 0

Wrong input. Try again...

Would you like to checkout? (Y/y or N/n) 1

Wrong input. Try again...

Would you like to checkout? (Y/y or N/n) y

The order is successfully completed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

Enter an option (0-1): 0

Good bye...!

If the user enters “N/n”, the function ***checkout()***terminates and returns 0.

If the user enters “Y/y”, the Oracle stored procedure ***add\_order()*** is called. This procedure will add a row in the orders table with a new order ID (See the definition of the ***add\_order()*** procedure.

This stored procedure returns an order ID, which will be used to store ordered items in the table order\_items.

The item\_id for the first product in the array is 1, for the second product is 2, and …

For all products in the array cart (*productCount* is the number of products stored in the array *cart*), the program calls the stored procedure ***add\_order\_item()*** and passes the corresponding values to this stored procedure.

Sample execution:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

Enter an option (0-1): 5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

You entered a wrong value. Enter an option (0-1): 1

Enter the customer ID: 1000

The customer does not exist.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

Enter an option (0-1): 44

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

You entered a wrong value. Enter an option (0-1): 1

Enter the customer ID: 44

-------------- Add Products to Cart --------------

Enter the product ID: 112

Product Price: 808.92

Enter the product Quantity: 2

Enter 1 to add more products or 0 to checkout: 1

Enter the product ID: 115

Product Price: 699.99

Enter the product Quantity: 3

Enter 1 to add more products or 0 to checkout: 0

------- Ordered Products ---------

---Item 1

Product ID: 112

Price: 808.92

Quantity: 2

---Item 2

Product ID: 115

Price: 699.99

Quantity: 3

----------------------------------

Total: 3717.81

Would you like to checkout? (Y/y or N/n) y

The order is successfully completed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

Enter an option (0-1): 1

Enter the customer ID: 44

-------------- Add Products to Cart --------------

Enter the product ID: 110

Product Price: 3192.97

Enter the product Quantity: 2

Enter 1 to add more products or 0 to checkout: 1

Enter the product ID: 116

Product Price: 731.99

Enter the product Quantity: 1

Enter 1 to add more products or 0 to checkout: 1

Enter the product ID: 117

Product Price: 695.99

Enter the product Quantity: 3

Enter 1 to add more products or 0 to checkout: 0

------- Ordered Products ---------

---Item 1

Product ID: 110

Price: 3192.97

Quantity: 2

---Item 2

Product ID: 116

Price: 731.99

Quantity: 1

---Item 3

Product ID: 117

Price: 695.99

Quantity: 3

----------------------------------

Total: 9205.9

Would you like to checkout? (Y/y or N/n) n

The order is cancelled.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1) Login

0) Exit

Enter an option (0-1): 0

Good bye...!