# Database Application Development

## Objective:

In this assignment, you create a simple Retail application using the C++ programming language like DBS211 and Oracle (PL/SQL). This assignment will help student learn a basic understanding of application development using C++ programming and an Oracle database using PL/SQL.

## Submission:

***Your submission will be***

***1) a single text based .cpp file including your C++ program for the Database Application assignment.***

*** Your submission needs to be commented.

***2) a video demonstrating a completely working application with all member names appearing at the end.***

## Additional notes:

The submission is

1) The CPP file as mentioned above

2) A simple video showing you testing out the program to prove it works as required.

You should pre-decide how you are going to demonstrate the program working so that you do not keep repeating errors. The video does not need to be a work of art, but a simple, clearly readable screen of your testing the assignment.

3) this assignment is a lot of work, so please do not leave it to the last minute.

Since every time you change the data in your tables, you will need to "fix" your tables back to the original state to re-test your work, do not forget to do a **ROLLBACK**.

Another alternative is to use a copy of the customer, orders and orderline tables and use those until your application is running properly then revert back to the correct tables so I can test it.

# 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, make sure 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();
}
```

Declare the following structure before the ***main()*** function:

```
struct Cart {
    int product_id;
    double price;
    int quantity;
};
```

The following connection may change in future

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

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

First, declare the environment and the connection variables.

```
Environment* env = nullptr;
Connection* conn = nullptr;
```

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";
```

Use the same Oracle username and password that you use for your labs and assignments.
Create the environment and the connection. Make sure you handle any errors may be thrown as your program is executed.

```
env = Environment::createEnvironment(Environment::DEFAULT);
conn = env->createConnection(user, pass, constr);
```

Remember to terminate and close the connection and the environment when your program terminates.

```
env->terminateConnection(conn);
Environment::terminateEnvironment(env);
```

After executing the statements make sure you terminate the statement.

```
conn->terminateStatement(stmt);
```

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

## *Stored Procedures*

*find_customer (customer_id IN NUMBER, found OUT NUMBER);*

This procedure has an input parameter to receive the customer ID. It has 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 can check the no_data_found exception in the EXCEPTION block.

```
EXCEPTION
      WHEN no_data_found THEN
      found := 0;
```

*find_product (product_id IN NUMBER, price OUT products.prod_sell%TYPE, pname OUT products.prod_name);*

This procedure has an input parameter to receive the product ID and an2 output parameters named price and pname. This procedure looks for the given product ID in the database. If the product exists, it stores the product's list_price (prod_sell) in the variable *price*. And the product name in pname. Otherwise, the *price* variable is set to 0.

```
EXCEPTION
      WHEN no_data_found THEN
      price := 0;
```

*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, find the maximum order ID in the orders table and increase it by 1.
This procedure inserts the following values in the ORDERS table:
new_order_id
customer_id (input parameter)
'Shipped' (The value for the order status) (C for closed and shipped, Letter O for open order not shipped yet)
174 (The salesperson ID)
sysdate (order date which is the current date)

*add_orderline (orderId IN orderlines.order_no%type,*
*itemId IN orderlines.line_no%type,*
*productId IN orderlines.prod_no%type,*
*quantity IN orderlines.qty%type,*
*price IN orderlines.price%type)*

This procedure has five IN parameters. It stores the values of these parameters to the table ORDERLINES

## *C++ Functions*

### *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
0) Exit

Prompt the user to choose an option. If the user enters the wrong value, ask the user to enter an option again until the user enters a valid option.

See the following example: The yellow XXXX should be one of your last name ←←

```
*************Main Menu by XXXXXXXXX ********************
1)      Login
0)      Exit
Enter an option (0-1): 5
*************Main Menu by XXXXXXXXX ********************
1)      Login
0)      Exit
You entered a wrong value. Enter an option (0-1):
```

If the user chooses option 1, move to asking the user to enter customer ID to login. To see if the customer with the entered ID exists, call the Oracle stored procedure ***find_customer()***. IF the value of the output parameter in the procedure is 1, allow the customer to continue. If the value of the output parameter found is 0, call the ***mainMenu()*** functions again and ask the customer to login again. Continue this process until the user chooses the option 0 to exit or the user enters a valid customer ID.

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

Before you call this function, prompt the user to enter the customer ID.
Call this function 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 exist, this function returns 0 and the main menu is displayed.
To validate the customer ID call the ***find_customer()*** stored procedure in this function.

See the following example:

```
*************Main Menu by XXXXXXXXX ********************
1)      Login
0)      Exit
Enter an option (0-1): 1

Enter the customer ID: 1000
The customer does not exist.
*************Main Menu by XXXXXXXXX ********************
1)      Login
0)      Exit
Enter an option (0-1): 1

Enter the customer ID: 44
-------------- Add Products to Cart XXXXXXXX --------------
Enter the product ID:
```

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

If the customerLogin() functions return 1 (The customer ID exists), call this function.
This function receives an OCCI pointer (a reference variable to an Oracle database) and an array of type Cart.

The customer can purchase up to 3 items in one order.
Write a loop to prompt the user to enter product IDs for the maximum of 3 products.

When the user enters the product ID in the *addToCart()* function, calls the findProduct() function (see function below for more details) to check if the product ID exists. IF the product exists, the function findProduct() returns the product's price and name of the product. Display the product's price and name to the user and ask the user to enter the quantity.

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

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

If the user chooses 1, ask the user to enter the next product ID. Otherwise, go 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 call the function ***findProduct()*** to see if the product ID exists.

See the following example:

```
-------------- Add Products to Cart XXXXXXXXX --------------
Enter the product ID: 1000
The product does not exist. Please try again...
Enter the product ID: 900
The product does not exist. Please try again...
Enter the product ID: 40100
```

```
Product Price: 165
Enter the product Quantity: 3
Enter 1 to add more products or 0 to checkout: 1
Enter the product ID: 50100
Product Price: 28
Enter the product Quantity: 20
Enter 1 to add more products or 0 to checkout: 0
```

*double findProduct(Connection* conn, int product_id);*

Might need fixing to show name

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 function calls the ***find_product()*** Oracle stored procedure. The procedure receives the product ID and name 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);*

Might be a naming error here for you to fix.

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

Call this function after the function ***AddToCart()*** to display the products added by the user to the shopping cart.

```
------- Ordered Products XXXXXXXXX -----------------------------------
---Item 1   ID: 40100   Star Lite        Price: 165   Quantity: 3
---Item 2   ID: 50100   GO Sport  Bag    Price:  28   Quantity: 2
-----------------------------------------------------------------
Total: 1055
```

---

After displaying the products' information (product ID, name, price, and quantity), display the total order amount. To calculate the total order amount, first multiply the quantity and the price to calculate the total amount for each product. Next, sum up products' total amounts to calculate the total order amount.

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

Call this function 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, display the following message:
`"Are you ready to checkout(Y/y) or Cancel (N/n) "`
If the user enters any values except "Y/y" and "N/n", display a proper message and ask the user to enter the value again.

```
"Wrong input. Please try again..."
```

See the following example:

```
Are you ready to checkout(Y/y) or Cancel (N/n) t
Wrong input. Please try again...
Would you like to checkout? (Y/y or N/n) 7
Wrong input. Please try again...
Would you like to checkout? (Y/y or N/n) y
The order is successfully completed.
*************Main Menu by XXXXXXXXX ********************
1)      Login
0)      Exit
Enter an option (0-1): 0
Thank you --- 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 orders.

The line_no 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*), call the stored procedure ***add_orderlines()*** and pass the corresponding values to this stored procedure.

# Sample execution:

```
*************Main Menu by XXXXXXXXX ********************
1)      Login
0)      Exit
Enter an option (0-1): 5
*************Main Menu by XXXXXXXXX ********************
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 by XXXXXXXXX ********************
1)      Login
0)      Exit
Enter an option (0-1): 1113
*************Main Menu by XXXXXXXXX ********************
1)      Login
0)      Exit
You entered a wrong value. Enter an option (0-1): 1

Enter the customer ID: 1113
-------------- Add Products to Cart XXXXXXXXX --------------
```

```
Enter the product ID: 1000
The product does not exist. Please try again...
Enter the product ID: 900
The product does not exist. Please try again...
Enter the product ID: 40100
Product Price: 165
Enter the product Quantity: 3
Enter 1 to add more products or 0 to checkout: 1
Enter the product ID: 50100
Product Price: 28
Enter the product Quantity: 20

------- Ordered Products XXXXXXXXX -----------------------------------
---Item 1  ID: 40100  Star Lite       Price: 165  Quantity: 3
---Item 2  ID: 50100  GO Sport  Bag   Price:  28  Quantity: 2
-------------------------------------------------------------
Total: 1055
```

---

```
Are you ready to checkout(Y/y) or Cancel (N/n) y
The order is successfully completed.

*************Main Menu by XXXXXXXXX ********************
1)      Login
0)      Exit
Enter an option (0-1): 1
Enter the customer ID: 1040

-------------- Add Products to Cart --------------
Enter the product ID: 60100
Product Price: 9
Enter the product Quantity: 50
Enter 1 to add more products or 0 to checkout: 1
Enter the product ID: 40301
Product Price: 131
Enter the product Quantity: 1
Enter 1 to add more products or 0 to checkout: 1
Enter the product ID: 40302
Product Price: 54
Enter the product Quantity: 32
Enter 1 to add more products or 0 to checkout: 0
------- Ordered Products XXXXXXXXX ---------
---Item 1  Product ID: 60100  Pocket U.V. Alerter Price: 9    Quantity: 50
---Item 2  Product ID: 40301  Pack n' Hike        Price: 131  Quantity: 1
---Item 3  Product ID: 40302  GO Small Waist Pack Price: 54   Quantity: 32
-----------------------------------------------------------
Total: 2309
===========================================================
Are you ready to checkout(Y/y) or Cancel (N/n) n
The order is cancelled.

*************Main Menu by XXXXXXXXX ********************
1)      Login
0)      Exit
```

```
Enter an option (0-1): 0
Thank you --- Good bye...

The members are:
Name              SID         Section
Member name 1   123456789   EE
Member name 2   234567890   GG
```

At the end of the video please n clude all members in your grou that participated fully

Finally, do a select to show orders in 2021.

It will require Customer name, order number, product ordered and total dollars for each order line

Again, since every time you change the data in the tables, you will need to "fix" your tables back to the original state to re-test your work, do not forget to do a **ROLLBACK**. If you do forget and mess up your tables, then rerun the script from week 1 (or copy the tables used to something like customer to custbkup or cust). Remember I do not have these tables so the correct table names have to be enterd to ensure it rus correctly.