

Assignment 2-PART2-CPP-PLSQL (20%)

In this assignment, groups will be taking the existing database design created in Project Assignment1 and expanding it using PL/SQL and various tables to better support a business application. Through the process of understanding the business requirements, the business rules, and how the database will work with the software application, learners will be able to identify ways, or solutions, in which the database design can be extended to be an integral part of the software architecture, and not just a storage and retrieval facility.

Groups will be creating the PL/SQL code that will create the required tables in a database to support the business requirements, and business rules, as well as various tables to support the user interface and basic CRUD operations through a secure parameterized method.

Submission

Each group will have 3 parts to their assignment submission (INDIVIDUAL and GROUP submission)

1. A single .cpp file contained all the code of each person's call the PL/SQL procedures by passing the required input argument from C++ and the display the output in C++. This C++ Program should ask choices from user for each of the operation to be performed.
([studentname_a2_cpp.cpp](#))
2. Word document of all individual cpp outputs (studentname_a2_output.docx)
3. COMBINED .cpp file (all student logins should be included- one is active others are commented, code should work in all logins) contained all the code required to call the PL/SQL procedures by passing the required input argument from C++ and the display the output in C++. This C++ Program should ask choices from user for each of the operation to be performed. Each procedure should have a comment of the owner of the procedure. ([groupname_a2_cpp.cpp](#))
4. Word document of all combined cpp outputs (groupname_a2_output.docx)

5. A short demo (20%) ONLINE with the Professor with all group members present at the end of the Week in Nov 27-29. If not present during demo and cannot show the results, **20% will be marked as '0'**. Each person should be involved, presenting the solution, and discussing the roles of the tables in a potential software scenario.

The demo should include things like:

- You can take 5 minutes maximum.
- Each person in the group to introduce about their application, explain their role in creating ER diagram and explain which part was done by the person.
- Each person in the group to explain the SQL procedure in ORACLE SQL DEVELOPER and demo of one procedure that they did for the application
- Each person to show a demo of C++ program execution of one stored procedure that they developed.

Tasks

The assignment is broken into a few tasks and are based on the Project Assignment1 database that has already been provided to you. For the purposes of this assignment, each team member will be working of the table that they created in Project Assignment1 and combine their solutions with other team members:

For each table, create a Stored Procedure that outputs the contents of the table to the script window (using DBMS_OUTPUT) for the standard `SELECT * FROM <tablename>` statement.

1. Individual cpp file

- a. Call SELECT procedure of your project table from cpp file
- b. Call INSERT procedure of your project table from cpp file
- c. Call UPDATE procedure of your project table from cpp file
- d. Call DELETE procedure of your project table from cpp file

2. Group cpp file

- a. If there are 3 group members, the cpp file should combine 4 procedures from each members and there should be 12 options
- b. If there are 2 group members, the cpp file should combine 4 procedures from each members and there should be 8 options

EXAMPLE FOR PL/SQL INSERT

```
CREATE OR REPLACE PROCEDURE spSectionInsert(  
  err_code OUT INTEGER,  
  m_sectionID IN section.sectionid%type,  
  m_professorID section.professorid%type DEFAULT NULL,  
  m_courseID section.courseid%type DEFAULT NULL,  
  m_classNumber IN section.classnumber%type,  
  m_courseTime IN section.coursetime%type  
) AS  
BEGIN  
  INSERT INTO section (sectionid, professorid, courseid, classnumber, coursetime)  
  VALUES (m_sectionID, m_professorID, m_courseID, m_classNumber, m_courseTime);  
  
  err_code := sql%rowcount;  
  COMMIT;  
  
  EXCEPTION  
    WHEN OTHERS  
      THEN err_code := -1;  
END;
```

Example FOR C++ INSERT

```
void insertSection(Connection* conn)
{
    Section section;
    int err = 0;

    cout << "Section ID: ";
    cin >> section.sectionid;
    cout << "Professor ID: ";
    cin >> section.professorid;
    cout << "Course ID: ";
    cin >> section.courseid;
    cout << "Class number: ";
    cin >> section.classnumber;
    cout << "Class time: ";
    cin >> section.coursetime;

    try
    {
        Statement* stmt = conn->createStatement();
        stmt->setSQL("BEGIN spSectionInsert(:1, :2, :3, :4, :5, :6); END;");
        stmt->registerOutParam(1, Type::OCIINT, sizeof(err));
        stmt->setNumber(2, section.sectionid);
        stmt->setNumber(3, section.professorid);
        stmt->setNumber(4, section.courseid);
        stmt->setNumber(5, section.classnumber);
        stmt->setString(6, section.coursetime);

        stmt->executeUpdate();
        err = stmt->getInt(1);

        if (err > 0)
        {
            cout << "\nSUCCESS: New section inserted.\n\n";
        }
        else
        {
            cout << "\nERROR: New section could not be inserted. The entered section ID may already exist.\n\n";
        }

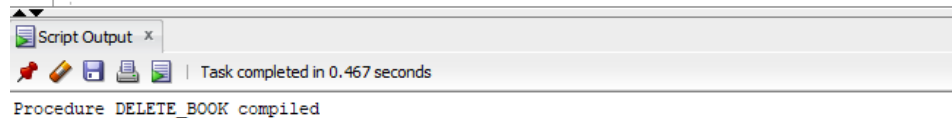
        conn->terminateStatement(stmt);
    }
    catch (SQLException& sqlExcp)
    {
        std::cout << sqlExcp.getErrorCode() << ": "
            << sqlExcp.getMessage();
    }
};
```

Example-

```
--3. delete an existing book's info given the PK
CREATE OR REPLACE PROCEDURE delete_book(bid IN book.book_id%type) AS
    result NUMBER;
BEGIN
    DELETE FROM book
    WHERE book_id = bid;
    result := sql%rowcount;
    DBMS_OUTPUT.PUT_LINE('Rows deleted: ' || result);
EXCEPTION
    WHEN NO_DATA_FOUND
    THEN
        DBMS_OUTPUT.PUT_LINE('Book with ID ' || bid || ' does not exist.');
```

```
END delete_book;
/

BEGIN
    delete_book(6);
END;
```

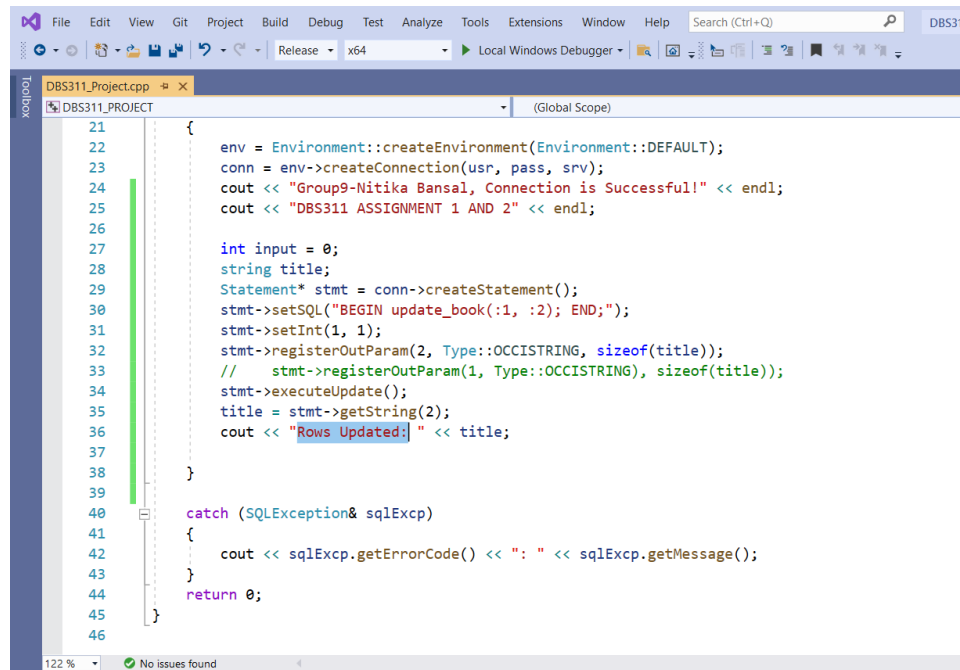


PL/SQL procedure successfully completed.

After executing **delete_book** procedure, the table is shown as below:

	BOOK_ID	TITLE	AUTHOR	CATEGORY_ID	PUBLICATION_YEAR	COPIES_OWned
1	1	Speak	Emma	23	07-06-16	2200
2	2	Rebecca	Goodreads	43	06-11-18	4200
3	3	Atonement	Ian	32	20-04-17	1700
4	4	Persuasion	Jane	55	21-09-13	2000
5	5	Clockers	Richard	14	14-12-16	3500

CPP code- executed **update_book** procedure:



```

21 {
22     env = Environment::createEnvironment(Environment::DEFAULT);
23     conn = env->createConnection(usr, pass, srv);
24     cout << "Group9-Nitika Bansal, Connection is Successful!" << endl;
25     cout << "DBS311 ASSIGNMENT 1 AND 2" << endl;
26
27     int input = 0;
28     string title;
29     Statement* stmt = conn->createStatement();
30     stmt->setSQL("BEGIN update_book(:1, :2); END;");
31     stmt->setInt(1, 1);
32     stmt->registerOutParam(2, Type::OCCISTRING, sizeof(title));
33     // stmt->registerOutParam(1, Type::OCCISTRING), sizeof(title));
34     stmt->executeUpdate();
35     title = stmt->getString(2);
36     cout << "Rows Updated:" << title;
37 }
38
39
40 catch (SQLException& sqlExcp)
41 {
42     cout << sqlExcp.getErrorCode() << ": " << sqlExcp.getMessage();
43 }
44 return 0;
45 }
46

```

OUTPUT-

```

*** HIGH RAISE CONDO MAINTENANCE ***
1) Create an Owner entry
2) Update an Owner Entry
3) Delete an Owner Entry
4) Read an Owner Entry
5) Create an Unit entry
6) Update an Unit Entry
7) Delete an Unit Entry
8) Read an Unit Entry
9) Create an Electricity Usage entry
10) Update an Electricity Usage Entry
11) Delete an Electricity Usage Entry
12) Read an Electricity Usage Entry
0) Exit
Enter an option (0-12): 11
Let's Delete Electricity Usage Record:
Usage ID: 150

SUCCESS: Electricity Usage 150 has been deleted.

```