Assignment 2 (30%)

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. 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)
3. A single .sql file containing all the person's code required to create the PL/SQL procedures required for the assignment. A header should be included in SQL comment style to have student name, email. (studentname_a2_SQL.txt)
4. A combined .sql file containing all the code required to create the PL/SQL procedures required for the assignment. A header should be included in SQL comment style to say who created this procedure. (groupnumber_a2_SQL.txt)
5. A users guide (groupnumber_a2_USERGUIDE.DOCX) to the created PL/SQL procedures listing each object created including:
    - Clearly stating who are the team members and listing the contributions of each team member. If someone didn't contribute don't do their work. Mention they didn't contribute
    - PL/SQL procedure
        i. Required input parameters (type, size, and meaning)
        ii. expected outputs,
        iii. potential error codes,
        iv. a description of its' purpose, and
        v. an example of non-saved procedural code to execute the given object.
        vi. An example output from C++ call to this procedure.

6. A short demo (20%) ONLINE with the Professor with all group members present at the end of the Week in Aug 1-5. 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:

- `TeamMember#1 table (the tables created and inserted in Assignment1)`
- `TeamMember#2 table (the one created and inserted in Assignment1)`
- `TeamMember#3 table (the one created and inserted in Assignment1)`

- For each table, create Stored Procedures to cover the 4 basic CRUD tasks.

  a. INSERT a new record using PK, the SP returns if the INSERT was successful if it is a new PK or failed if it exists
  b. UPDATE an existing record given the PK. Result as successful or failed based on PK.
  c. DELETE an existing record given the PK. Result as successful or failed based on PK.
  d. SELECT return all fields from a table given a PK. Result as successful then displays the entire record or failed with corresponding error message.

  - Name the Stored Procedures using the following guide: spTableNameOPERATION (example `spPlayersInsert`)
  - Do not use DBMS_OUTPUT in the procedures in any way.
  - All SPs must have appropriate exception handling specific to the method and table.

- Use error codes of the same type and size of the PK to return values that can be clearly determined to indicate an error (example: -1 might indicate no record was found)

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.

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::OCCIINT, 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 OF USERGUIDE

Update Table...

DBS311 Group 1 Project - College Registration

STUDENT A -> Financial Table

STUDENT B -> Section Table

STUDENT C -> Course Table

Overview

This application is on college registration and the focus is on the tables Financial, Section, and Course. It displays a menu of 13 options, which are 4 functions for each table and an exit option. The user input starts with picking an option (0 to 12) and then performing create, read, update, delete, and display all functions depending on each table's constraints and expected input. Selecting 0 in the menu means quitting the application.

User Guide – Table of Contents_Toc100219839

DBS311NFF Group 1 Assignment 2 User Guide

Financial Table

Financial table contains

- financialID (number and primary key) – bank account number
- balance (double) – the amount of money in a bank account
- bankName (varchar2) – name of the bank where an account is at
- bankAccNum (number) – the bank account number
- cardNum (number) – the card number for the account

financialID is a foreign key in the professor and student tables.

Create

Insert a row in the financial table for all five columns. balance and bankName are mandatory.

Read

Correct input of financialID will help retrieve a row. The procedure produces an error code which is 0 if input fails or greater than 0 if input succeeds.

Update

Correct input will help update any of the columns. The procedure produces an error code which is 0 if input fails or greater than 0 if input succeeds.

Delete

Correct input of financialID will help delete a row. The procedure produces an error code which is 0 if input fails or greater than 0 if input succeeds.

Display all financial accounts

This is a select-all statement and displays all the rows in this table.

Example non-saved procedural call:

```
DECLARE
err_code INTEGER;
BEGIN
    spsectioninsert(err_code, 50, 7250, 105, 5, 'Tuesday');
END;
```

Sample output:

```
------------ College Registration ------------
1.  Create a section entry
2.  Read a section entry
3.  Update a section entry
4.  Delete a section entry
5.  Create a financial entry
6.  Read a financial entry
7.  Update a financial entry
8.  Delete a financial entry
9.  Create a course entry
10. Read a course entry
11. Update a course entry
12. Delete a course entry
0. Exit

Enter an option: 1
Section ID: 99
Professor ID: 7255
Course ID: 144
Class number: 05
Class time: Sunday

SUCCESS: New section inserted.
```

DBS311-mahmed224 | SECTION

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Deta

Sort.. | Filter:

	SECTIONID	PROFESSORID	COURSEID	CLASSNUMBER	COURSETIME
1	420	7253	311	90	Thursday
2	75	7252	144	65	Thursday
3	99	7255	144	5	Sunday
4	90501	7250	144	12	Monday
5	90502	7251	100	11	Tuesday
6	90503	7252	150	10	Wednesday
7	90504	7253	210	9	Thursday
8	90505	7254	244	8	Friday
9	90506	7255	311	7	Monday
10	90507	7256	140	6	Tuesday
11	90508	7257	105	5	Wednesday
12	90509	7258	123	4	Thursday
13	90510	7259	168	3	Friday