# Assignment 2: Develop and Test a Coded Solution

## Introduction

In this assignment, you’ll examine a business problem and then write a PL/SQL program that meets the business guidelines and restrictions. You will then thoroughly test your coded solution within the parameters laid out in this assignment.

This assignment requires a basic understanding of accounting business terminology. To be successful in this assignment, ensure that you’ve completed the *Accounting* *Terms* activity.

## Equipment and Materials

For this assignment, you will need:

* A Windows computer with a minimum of 16 GB RAM and 250 GB of free disk space, capable of nested virtualization
* Access to ORACLE SQL\*PLUS
* Data files (provided as attachments to the assignment) in Brightspace
* We Keep It Storage (WKIS) files, located in the *Database* folder in Brightspace

## Instructions

1. Review the *Business Problem* and *Evaluation* sections of this document.
2. Working in groups, as directed by your instructor, write a PL/SQL program that addresses the business problem and meets the guidelines and restrictions.
3. Thoroughly test your coded solution within the parameters laid out below.
4. Submit your tested, coded solution to the *Assignment submission* area in Brightspace.

* This file should be an SQL script file (**do not** zip your solution).
* The text file should have the following naming standard: **Group\_group#\_A2.sql**
* Submit one solution per group.

1. Individually, submit your peer evaluation after the assignment has been completed.

## Business Problem

Your client, We Keep It Storage (WKIS) Company, have asked you to write a program for their accounting system, and they have provided you with their data files. They have also provided you with a Readme file outlining their requirements so you can create their WKIS database tables successfully.

**Notes:**

* This is a double-entry accounting system that uses the accounting rules presented in the *Accounting Notes* document in Brightspace.
* Take transactions from a holding table named NEW\_TRANSACTIONS and insert them into the TRANSACTION\_DETAIL and TRANSACTION\_HISTORY tables.
* At the same time, update the appropriate account balance in the ACCOUNT table.
* You need to determine the default transaction type of account (debit (D) or credit (C)) to decide whether to add or subtract when updating account balance.

## Datasets

The following dataset is provided as attachment to this assignment in Brightspace:

* A2\_testDataset\_1-Clean.sql

Use this dataset to help you test your coded solution. Note that they are simply examples of possible data. **Do not code to the dataset, code to the problem**.

It is highly recommended you also create additional test data to ensure your program works regardless of the data provided. Your program should work with any data.

**Note:** Your instructor will evaluate your program with a different set of data.

## Guidelines and Restrictions

When writing your PL/SQL program, follow these guidelines and restrictions.

* Assume that every transaction number is unique (no duplicates) for each transaction.
  + A transaction is a unit made up of more than one row.
  + All rows that represent a single transaction have the same transactional history information (TRANSACTION\_NUMBER, TRANSACTION\_DATE, DESCRIPTION).
* Using two nested cursors makes this problem easier to solve, although you don’t have to use this method. You saw an example of two nested cursors in class.
* It’s a good idea to ensure that the overall structure of your program works with clean data first, Assignment 2 (see the *Datasets* section), and then add error checking and functionality, Assignment 3.
* As long as the debits equal the credits in each transaction, you can assume that the accounting equation for each transaction holds true. For this program, you do **not** have to validate the accounting equation.
* All required tables for this assignment are created with the provided scripts. **Do not** create any additional tables or modify the existing tables (structure or constraints).
* Do **not** use a table of records or any other type of array in your solution. (These aren’t covered in this course, so it’s OK if you don’t know what they are.)
  + **Record data structures are okay**. A table of records is different.
* SELECT INTO cannot be performed against the NEW\_TRANSACTIONS table.
* A SELECT on NEW\_TRANSACTIONS can only be performed by an explicit cursor (use your cursor for any needed values from this table).
* The solution must be performed with **one** anonymous block. Multiple embedded blocks are fine since these are not considered separate anonymous blocks.
  + If multiple anonymous blocks are submitted, **only the first one in the script** will be evaluated by your instructor.
* You **cannot** use stored programs.
* **Do not use** GOTOs, EXITs or SAVEPOINTs.
* CONTINUEs can be used if done appropriately (don’t use as you would a *break* in Java).
* **Do not** hard code values anywhere in your code, except for the transaction type (C, D), which should be hard coded only in the DECLARE section.
* After a transaction has been successfully processed, remove it from the NEW\_TRANSACTIONS table.

## Evaluation

Your instructor will use a separate dataset to evaluate your program based on the criteria below.

#### Evaluation 1

Your program will receive an **automatic zero** if any of the following criteria are true:

* Cannot test application because syntax errors exist.
* A runtime error occurs that prevents any testing of the application.
* Database structure has been modified, resulting in syntax or runtime errors.
* GOTO, EXIT, SAVEPOINT or arrays (table of records/collection) are used.
* Stored programs are included.

#### Evaluation 2

If your program passes the first evaluation, marks will be deducted according to the following rubric.

| **Test Type** | **No Deductions** | **0.5 Mark  Deduction for Each** | **1 Mark  Deduction for Each** | **2 Mark  Deduction** | **Marks** |
| --- | --- | --- | --- | --- | --- |
| **Static White-Box** | All coding restriction guidelines being followed. | * Data values are hard coded in the application (other than values outlined in the assignment). * SELECT INTO on NEW\_TRANSACTIONS is performed or SELECT on NEW\_TRANSACTIONS outside explicit cursors is being performed. * Multiple anonymous blocks used (embedded blocks are different than anonymous blocks). Note: Only first anonymous block will be evaluated (tested). * Documentation (header or inline comments) not present. | * The program does not save data changes (no COMMIT). * The program partially saves data changes (incomplete COMMIT inside of program). | N/A | **/2** |
| **Dynamic Black-Box: Test-to-Pass** | All DML for clean data is successfully performed with mixed clean and erroneous data. | * Not all transactions successfully processed into TRANSACTION\_HISTORY * Not all transactions successfully processed into TRANSACTION\_DETAIL. * Not all transactions successfully removed from NEW\_TRANSACTIONS. * Not all account balances successfully updated. * Some transactions in error are removed from the NEW\_TRANSACTIONS table. | * No transactions successfully processed into TRANSACTION\_ HISTORY * No transactions successfully processed into TRANSACTION\_ DETAIL. * No transactions successfully removed from NEW\_TRANSACTIONS. | * All DML successful if only clean data is in NEW\_ TRANSACTIONS, but when mixed with erroneous data, not all DML is successful for clean data. | **/5** |
| Total | | | | | **/7** |

### Evaluation Summary

**Final Individual Marks**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Team Total**  (7 marks available) | **\*** | **Peer Evaluation Multiplier** (as a percentage) | **=** | **Subtotal** |
|  | \* |  | = |  |
|  | | | | + |
| **Peer Evaluations Completed (3)** | | | |  |
|  | | | | = |
| **Final Marks** | | | | **/10** |