# Assignment 3 Part 1: Develop and Test a Coded Solution in PL/SQL with Clean Data

### Objective

### To create a coded solution for a business problem assuming only clean data – no exception handling needed.

## 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 *Double Entry Accounting* exercise.

## Instructions

### This assignment will be completed in your assigned groups.

### The final deliverables will be a coded solution that is submitted to the corresponding Brightspace assignment submission folder. Details on the expectations of this code is provided later in this assignment document.

## Suggestions

* Review the *Business Problem* and *Evaluation* sections of this document.
* Make sure to refresh your data between executions.
* Thoroughly test your coded solution within the parameters laid out in this assignment document.
* The submitted SQL script (text file) should not be zipped.

## Business Problem

Your client, We Keep It Storage (WKIS) Company, has 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 an account (debit (D) or credit (C)) to decide whether to add or subtract when updating the account balance.
* Once a transaction is successfully processed, it should be removed from the holding table.

## Dataset

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

* *A3\_test dataset\_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 row with the same transaction number is part of the same transaction (you will not have more than one transaction with the same number).
  + 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.
* 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 (or SELECT subquery) cannot be performed against the NEW\_TRANSACTIONS table.
* A SELECT on NEW\_TRANSACTIONS can only be performed by an explicit cursor (use your cursor data 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 (EXIT WHEN with a basic loop is fine) 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. Values for variables like counters are okay.

## Evaluation

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

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.

If your program does not fall into an automatic zero criteria, your code will be evaluated based on the following two rubrics.

| **Program Evaluation** | **No Deductions** | **0.5 Mark  Deduction for Each** | **1 Mark  Deduction for Each** |  | **Deduction Total** |
| --- | --- | --- | --- | --- | --- |
| **Code Review Deductions** | All coding restriction guidelines being followed. | * Data values are hard coded in the application (other than values such as counters). * SELECT INTO on NEW\_TRANSACTIONS is performed or SELECT on NEW\_TRANSACTIONS outside explicit cursors is being performed. * 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). |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Changes** | **No Deductions** | **1 Mark  Deduction for Each** | **5 Mark  Deduction for Each** | **10 Mark  Deduction** | **Marks** |
| **New Transactions Data** | Data is changed as expected in NEW\_TRANSACTIONS table. | * Part of a transaction remains in NEW\_TRANSACTIONS when the entire transaction should be removed. | * Some transactions remain in NEW\_TRANSACTIONS when they should be removed. | * No transactions removed from NEW\_TRANSACTIONS | **/10** |
| **Transaction History Data** | Data is changed as expected in TRANSACTION\_HISTORY table. | * Not all transactions successfully processed into TRANSACTION\_HISTORY. | * Transaction numbers have been changed from what was given in NEW\_TRANSACTIONS. | * No transaction successfully processed into TRANSACTION\_HISTORY. | **/10** |
| **Transaction Detail Data** | Data is changed as expected in TRANSACTION\_DETAIL table. | * Part of a transactions is being processed into TRANSACTION\_DETAIL. | * Not all transactions successfully processed into TRANSACTION\_DETAIL. | * No transaction successfully processed into TRANSACTION\_DETAIL. | **/10** |
| **Account Data** | Data is changed as expected in ACCOUNT table. | * Rows have been added or removed from ACCOUNT. * Information other than the account balance has been changed. | * Not all account balances successfully updated. | * No account balances successfully updated. | **/10** |
| **Team Total** | | | | | **/40** |

### Individual Mark

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Team Total (40)** | **\*** | **Peer Evaluation Multiplier (as a percentage)** | **=** | **Subtotal** |
|  | \* |  | = |  |
|  | | | | + |
| **Peer Evaluations Completed (5)** | | | |  |
|  | | | | = |
| **Final Mark** | | | | /45 |