# Assignment 3 Part 2: Develop and Test a Coded Solution in PL/SQL with Exception Handling

### Objective

### To create a coded solution for a business problem that contains a mix of good and bad data.

## Introduction

In this assignment, you’ll extend the business problem from Part 1 of the assignment to write a PL/SQL program that can handle incorrect input data and handle them appropriately. 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 account (debit (D) or credit (C)) to decide whether to add or subtract when updating account balance.
* Once a transaction is successfully processed, it should be removed from the holding table. A transaction in error should remain in the holding table.

## Datasets

There are two test datasets provided as attachments to this assignment in Brightspace:

* A3\_test dataset\_2 – Clean and Erroneous.sql

Use these datasets to help you test your coded solution. Note that they are simply examples of possible data. **Do not code to the datasets, 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).
* **Only** hard code ‘C’ (credit) and ‘D’ (debit) in your code for this assignment and these should be constants. Values for variables like counters are okay.
* An error in one transaction should not prevent the processing of other transactions (i.e., don’t leave the main looping structure when an error occurs).
* Only the first error in a transaction should be recorded in the error log table (i.e., a specific transaction number should only appear once in the error log table).
  + If the error is a missing transaction number, a single entry can be recorded in the error log table for all rows missing a transaction number, or you can have an entry recorded for each row missing a transaction number.
  + Transactions that produce an error should remain in the NEW\_TRANSACTIONS table.
  + Transactions that did not produce an error should be processed correctly into the corresponding tables: ACCOUNT, TRANSACTION\_DETAIL and TRANSACTION\_HISTORY and removed from NEW\_TRANSACTIONS.
    - The Part 2 code additions around exception handling should NOT affect the normal behavior created in Part 1 of processing clean.
* When your instructor evaluates your program, only errors that appear in the error log table will be evaluated.
  + If you only print the error to the screen, your instructor will consider this error as not being caught.
  + For debugging, you can write errors to the error log table AND print to the screen.

### Errors

Your program should handle the following errors, also writing the transactional history information of the transaction that caused the error and the error message to the WKIS\_ERROR\_LOG table. **Error messages must be descriptive**.

Your program **must** catch and handle the following errors:

* Missing transaction numbers (NULL transaction number)
* Debits and credits that are not equal
* Invalid account numbers (when the account number in the current transaction is not in the account table)
* Negative values given for a transaction amount
* Invalid transaction types (any transaction type except C and D is invalid)

All other errors are considered unanticipated errors. These should be caught as well, because you don’t want your program to crash. For these unanticipated errors:

* The error message should be a system-generated error message, since you can’t provide a customized, descriptive message for this type of error.
* Testing does not need to be performed for this type of error. Your instructor is simply checking that the code exists to handle unanticipated errors.

## 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 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** | **2 Mark  Deduction for Each** | **5 Mark  Deduction** | **Marks** |
| **Processing Transactions without errors** | All data is changed as expected. | * Part of a transaction remains in NEW\_TRANSACTIONS when the entire transaction should be removed. * Not all transactions successfully processed into TRANSACTION\_HISTORY when not in error. * Part of a transaction is being processed into TRANSACTION\_DETAIL when not in error. * Rows have been added or removed from ACCOUNT. * Information other than the account balance has been changed. | * Some transactions remain in NEW\_TRANSACTIONS when they should be removed. * Transaction numbers have been changed from what was given in NEW\_TRANSACTIONS. * Not all transactions successfully processed into TRANSACTION\_DETAIL when not in error. * Not all account balances successfully updated when not in error. | * No transactions removed from NEW\_TRANSACTIONS if no error. * No transaction successfully processed into TRANSACTION\_HISTORY if no error. * No transaction successfully processed into TRANSACTION\_DETAIL if no error. * No account balances successfully updated if no error. | **/20** |
| Subtotal | | | | | **/20** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 is removed from NEW\_TRANSACTIONS when the entire transaction should remain. | * Some transactions are removed from NEW\_TRANSACTIONS when they should remain. | * All transactions are removed from NEW\_TRANSACTIONS when they should remain. | **/10** |
| **Transaction History Data** | Data is changed as expected in TRANSACTION\_HISTORY table. | N/A | * Some transactions processed into TRANSACTION\_HISTORY when they are in error. | * All transactions processed into TRANSACTION\_HISTORY when in error. | **/10** |
| **Transaction Detail Data** | Data is changed as expected in TRANSACTION\_DETAIL table. | * Part of a transaction is being processed into TRANSACTION\_DETAIL when in error. | * Some transactions processed into TRANSACTION\_DETAIL when they are in error. | * All transactions processed into TRANSACTION\_DETAIL when in error. | **/10** |
| **Account Data** | Data is changed as expected in ACCOUNT table. | * Rows have been added or removed from ACCOUNT. | * Some transactions are changing account balances in ACCOUNT when they are in error. | * All transactions are changing account balances in ACCOUNT when they are in error. | **/10** |
| Subtotal | | | | | **/40** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Changes** | **No Deductions** | **1 Mark  Deduction for Each** | **5 Mark  Deduction for Each** | **10 Mark  Deduction** | **Marks** |
| **Missing (NULL) Transaction Number** | Error is caught and a descriptive message is placed in the error log table. | * Missing transaction number error is being caught but the error message is a constraint error rather than a custom, descriptive message. | * Error is only caught and recorded if testing this one transaction in isolation (no other transactions processed with it). | * Missing transaction number error is not being caught. | **/10** |
| **Debits are not equal to credits** | Error is caught and a descriptive message is placed in the error log table. | * Debits not equal to credits error is being caught but the error message is a constraint error rather than a custom, descriptive message | * Error is only caught and recorded if testing this one transaction in isolation (no other transactions processed with it). | * Debits not equal to credits error is not being caught. | **/10** |
| **Invalid Account Number** | Error is caught and a descriptive message is placed in the error log table. | * Invalid account number error is being caught but: a) the error message is a constraint error rather than a custom message and/or b) the transaction is still processed. | * Error is only caught and recorded if testing this one transaction in isolation (no other transactions processed with it). | * Invalid account number error is not being caught. | **/10** |
| **Negative Transaction Amount** | Error is caught and a descriptive message is placed in the error log table. | * Negative transaction amount error is being caught but: a) the error message is a constraint error rather than a custom message and/or b) the transaction is still processed. | * Error is only caught and recorded if testing this one transaction in isolation (no other transactions processed with it). | * Negative transaction amount error is not being caught. | **/10** |
| **Invalid Transaction Type** | Error is caught and a descriptive message is placed in the error log table. | * Invalid transaction type error is being caught but: a) the error message is a constraint error rather than a custom message and/or b) the transaction is still processed. | * Error is only caught and recorded if testing this one transaction in isolation (no other transactions processed with it). | * Invalid transaction type error is not being caught. | **/10** |
| Subtotal | | | |  | **/50** |
| **Team Total** | | | |  | **/110** |

### Individual Mark

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Team Total (110)** | **\*** | **Peer Evaluation Multiplier (as a percentage)** | **=** | **Subtotal** |
|  | \* |  | = |  |
|  | | | | + |
| **Peer Evaluations Completed (10)** | | | |  |
|  | | | | = |
| **Final Mark** | | | | /120 |