This document servers as the Resource Guide for the DE02 (Data Engineering 02) Case Study proposed solution, Credit Card Management (CCM) System: Operational Database. It provides summary information as input to test team creating a test plan to verify the components satisfy the information system processing application requirements.

Below each section contains summary information about the resources developed to satisfy the requirement. In this document, the document requirement identifier serves as the section header.

**Core Java**

The file Requirement211TransactionDetailsModule.zip contain the Java code for the solution to interact with the cw\_sapp database in the MySQL RDBMS using JDBC.

The Credit Card Management (CCM) System development in the solution focused on delivering Processors which display the required reports to the End-user.

The solution consists of three Java classes as processors interacting with the End-user. They are:

1. Process Operational Data Requests (ProcessOperationalDataRequests),
2. Process Customer Detail (ProcessCustomerDetail), and
3. Process Transaction Detail (ProcessTransactionDetail).

Each processor in the CCM System receives requests and coordinates outcomes. Process Customer Detail and Process Transaction Detail are responsible for performing activities that deliver reports. Below is a summary of the overall CCM System process flow data flow when the End-user interacts with the process. An overall processing approach is to allow the End-user to focus on customer detail activities or transaction detail activities.

After the End-user initiates the processing by selecting to access the CCM System, the high-level steps are the following:

1. Display the Request Menu for Process Operational Data Requests.
2. IF the End-user enters menu option 1, GO TO Step 05, ELSE IF the End-user enters menu option 2, GO TO Step 03, ELSE GO TO Step 04.
3. Perform Process Customer Detail activities which including the following:
   1. Display Report Selection Menu;
   2. IF the End-user enters 0, GO TO Step 03.6, ELSE GO TO Step 03.3 when a report number on the menu.
   3. Retrieve report parameters from the End-user.
   4. Perform the requested activities for the selection.

Comment: Please see the comments in the Java class for more details. They include the Steps for interacting with the CDW\_SAPP database.

* 1. GOTO Step 03.1.
  2. Display end of processing message.
  3. GOTO Step 01.

1. Perform Process Transaction Detail activities which including the following:
   1. Display Report Selection Menu;
   2. IF the End-user enters 0, GO TO Step 04.6, ELSE GO TO Step 04.3 when a report number on the menu.
   3. Retrieve report parameters from the End-user.
   4. Perform the requested activities for the selection.

Comment: Please see the comments in the Java class for more details. They include the Steps for interacting with the CDW\_SAPP database.

* 1. GOTO Step 04.1.
  2. Display end of processing message.
  3. GOTO Step 01.

1. Display end of processing message
2. Exit CCM System.

Exhibit 1 contains the overall process flow diagram

Exhibit 1



The file Requirement211CustomerAndTransactionDetailsModule.zip contains the Java class for processes 2.1.1 and 2.1.2. Also, it has a JDBC\_Test class for testing the DQL and DML submissions and the results returns between the CCS System and the database cdw-sapp in the MySQL RDMS.

To perform a test of the JDBC related code for a DQL or DML, the Tester should complete the following steps. They are:

1. Copy the content of the SQL variable line under Step 2 of the case statement.
2. Paste the content under the comment line, Step 2 in the JDBC\_Test.java class file.
3. Copy the content of the prepareStatement lines under Step 2 of the case statement.
4. Paste the content of the prepareStatement lines under the comment line “Remove statements above before going into production” in Step 2 in the JDBC\_Test.java class file
5. Copy the content of the while loop block in Step 4 of the under the line starting with “System.out.println”.
6. Paste the content of the while loop block into the JDBC\_Test.java class file under the comment Step 4 by replacing the lines between between the lines “rowCount = 0;” and the line starting with “System.out.println”.

The solution does not include the following:

1. Security authentication and verification for the End-user privileges to use the Operational Database of the CCM System.
2. Specific Java classes to house common methods for each of the higher level processors were not created as part of this solution, given the time constrains. Further modularization can occur as a process improvement

**RDBMS/mySQL Description**:

The file Requirement211CustomerAndTransactionDetailsModule.zip contains a copy of the CDW\_SAPP with added records to test the code for the solution under Requirement 2.2.4 Process Optimization Module in the **Oozie (Sqoop and Hive optimized)** section of this document.

The latest insert of data into all tables was on 06/20/2018. If a Tester intends to import incremental records into the HDFS directories or the Hive Data Warehouse, the person must insert records into the tables.

**Hadoop/hdfs/dataware housing**

The file Requirement221DataExtractionAndTransportationWithSqoop.zip contains the files used in the creation of the solution for the initial ETL (Extract, Transfer, and Load) Sqoop jobs to import data into the HDFS file system directory. It includes copy of sql files used to verify MySQL returns the same information in the --query clause of each Sqoop import job.

The content of each file was entered by copy and paste using a terminal app, and the Sqoop execute command was given to run each job.

**Hive and Partition**

The file Requirement222DataLoadingModule.zip contains the files used to create the Hive tables and store the data in the Hive Data Warehouse. For implementation of this solution the cdwod database contain the following tables. They are:

1. CDW\_SAPP\_D\_BRANCH,
2. CDW\_SAPP\_F\_CREDIT\_CARD,
3. CDW\_SAPP\_D\_TIME, and
4. CDW\_SAPP\_D\_CUSTOMER.

The CDW\_SAPP\_D\_CUSTOMER is in a Managed partition table by state. For this solution, a partition exists for the CDW\_SAPP\_D\_CUSTOMER table only. The assumption is other tables are small and would not change often. The assessment is creating partitions for small tables could have a negative impact on processing time efficiency. A Hive script exist to create the CDW\_SAPP\_D\_CUSTOMER partition table and store the data.

**Oozie (Sqoop and Hive)**

The file Requirement223AutomatingTheProcessWithOozie.zip contains the files in the solution to automatically perform the following:

1. Import and store current table data from the CDW\_SAPP database into the MySQL RDBMS into the Hadoop HDFS directory under /Credit\_Card\_System.
2. Create a data table for each imported file and store the data in a Hive directory.

This solution sequentially execute each Sqoop import job to store the extracted in a HDFS file directory, then after the successful completion of all the Sqoop import jobs, the Hive scripts sequentially execute to create the tables in the cdwod database, and to store the data in the Hive Data Warehouse tables.

An alternative approach could be to start the workflow, establish a fork node and each branch of the node execute a Sqoop import job followed by a Hive script in parallel then flow to a join node then joining back after all successfully complete and end the workflow.

The alternative may ensure no contention for datanode resources but have longer processing time

**Oozie (Sqoop and Hive optimized)**

The file Requirement224ProcessOptimizationModule.zip contains the files necessary to append new data in the HDFS and the Hive Data Warehouse for the cdwod database tables.

This solution is implements a process to import and store only new data. It deletes all data currently in the Hive Data Warehouse.

All the Sqoop action nodes and job files in the Oozie workflow have the word “Incremental” to identify what has changed in the original workflow.

The “Incremental” workflow uses the sequential approach was used to first execute all the Sqoop jobs then the Hive scripts.

**Visualization**

The file Requirement225DataVisualization.zip contains the scripts in the solution for querying the cdw\_sapp\_branch and cdw\_sapp\_creditcard tables to return the following:

1. The total transaction value for the top 20 branches, and w
2. The total transaction value in each Quarter of 2018 for each transaction type.

The file includes a word document containing screen images for each visualization.

**Pig Aggregate Query.**

The file CurrentBranchCountUsingPig.zip contains the list of current branches in the cdw\_sapp\_branch table, the Pig script for a query to return a count of the current branches, and a screen image of a listing that contains a directory with data files for another exercise. The screen image is to depict how multiple files of data would display in the interface of the Ambari sandbox app.

The results were achieved by performing the following steps. They are:

1. Export the cdw\_sapp\_branch table to a csv file,
2. Upload the data to a HDFS directory,
3. Copy the query into the Pig view interface, and
4. Execute the query.

**Closing Summary and Contact Information**

**The descriptions in this document are summaries, except for the files containing the code for the Java classes, and some comments in the Sqoop and Hive files, no more detail is available. The solutions proposed in this case study are not the only and most efficient from the number of characters written in documents and IDE screens. Yet, they provide a baseline process to produce the required results.**

**If you have any questions, or need more information; please send an email message to** [**rb2017houser@gmail.com**](mailto:rb2017houser@gmail.com)**.**

**Sincerely,**

**Robert B. Houser**