**Database Management Term Project:**

**Final Report**

Kyle Galway

Sayaka Nakamura

Sheridan College

DBAS 32100: Database Management

Haxhi Dvorani

August 13th, 2023

Contents

[**Introduction** 3](#_Toc142835150)

[**Task One** 4](#_Toc142835151)

[**Task Two** 5](#_Toc142835152)

[**Task Three** 7](#_Toc142835153)

[**Task Four** 8](#_Toc142835154)

[**Task Five** 9](#_Toc142835155)

[**Task Six** 10](#_Toc142835156)

[**SQL Code** 13](#_Toc142835157)

[**Java Code** 14](#_Toc142835158)

# **Introduction**

The project completed during this semester and described within this report details a complete Data Integration process for a mock donations collection organization. The actions performed in this project include collecting data from a target directory containing donations information from Comma-separated Values (CSV) files, performing necessary format transformations on the data, uploading the data to the central donations repository Oracle database, and using this data to create a series of read-only Data Marts to make available the data to relevant subscribers and peoples of interest as per the Data Consolidation technique.

This project was divided into six separate tasks, each of which required a different approach as well as the use of various tools and techniques. The tools used to complete these tasks are briefly summarized below:

* Task One: Refresh the Central Repository Address Table. Completed with Talend Open Studio using the tMap component and Java Functions.
* Task Two: Upload Donations Data from CSV Source Files. Completed with Talend Open Studio using the tMap component and Java Functions.
* Task Three: Create the Star Schema to be used in the Data Mart Tables for Data Consolidation. Completed in SQL Developer using SQL Scripts.
* Task Four: Load Data into Star Schema. Completed in SQL Developer using SQL Scripts.
* Task Five: Create Views to Display Star Schema Data. Completed in SQL Developer using SQL Scripts.
* Task Six: Create Users to Transform Data and Read Views. Completed in SQL Developer using SQL Scripts.

In addition, the bonus task was completed, which required an online geolocation API to be used to insert missing postal codes into the Central Repository Address Table during Task One. This bonus task was completed using the Mapbox Geolocation API.

A list of all Talend Components and Code Scripts used in this project, as well as a listing of all functions and procedures, is provided at the end of this report.

# **Task One**

The first task in the project required the refreshing of the address table in the Central Oracle Database Repository. The addresses to be used to refresh this table were sourced from a Master Microsoft SQL Server. The fields for the addresses sourced from the master server are by default all uppercase, while it was decided that the data in the central repository would be stored in title case for all fields except for the province initials.

To accomplish this, a job was created in Talend Open Studio. This job used a tDBInput component linked to the master server to obtain the address fields, then loaded each address into a tMap component. Once inside the tMap component, the fields required to be in title case were ran through a custom Java function to perform the alteration.

Because this project also opted to complete the bonus postal code retrieval task, a second function was called to inside the tMap component to obtain the postal codes for each address where possible. This function ran each of the addresses through a web API and on a successful retrieval filled in the postal code field.

Though the API had extremely high success rate, eighty-six addresses either failed to retrieve a postal code or were found to be missing required fields for the central repository. These addresses were output to a CSV file for storing their invalid data. All other rows were successfully used to repopulate the address table in the central repository.

In addition, a trigger was created for the target address table which repeated the required transformations on the data. This provided a redundant check to ensure that the inserted data was transformed into the correct format.

**A diagram of a network

Description automatically generated**

1. Talend job layout for task one

# **Task Two**

The second task required that the donations data stored in the CSV files be uploaded into the donation table of the central repository. Like task one, this task was completed in Talend Open Studio.

To begin, the tFileList component was used to stream each of the CSV files located within the origin directory for the project. These file streams were then merged into a single unified data stream using the tUnion component. Afterwards, the combined files were loaded into a single combined file, which provided a backup of the combined data streams for later error checking.

The merged files streams were then passed through a tUniqRow component, which removed any duplicate data from the stream. The data was then passed through a tMap component. At the same time, the master repository was once again queried to retrieve a list of all valid street types and street directions. These types and directions were each denormalized into a single string containing all valid values for the field delimited by semi-colons.

These denormalized strings would later be used when the merged stream was passed through an additional tMap component. As the merged stream was passed through an the tMap component, Java functions were used to split the addresses into fields to match the central repository address schema. The type and address fields were identified using the normalized type and direction values to check for matching fields after splitting the addresses into individual words.

After, the data was passed through a final tMap component to check for any errors in the collected data. This component flagged for any missing fields required by the central address table and verified that the addresses in the data stream to ensure they belonged to addresses tracked in the central repository address table.

Where the data was found to be lacking required fields or containing an invalid address, the data was output into CSV files. Each output file was related to a leader within the donation organization by the volunteer ID, allowing for each leader to later investigate their team’s erroneous data. Data lacking a volunteer ID was instead output to a bulk CSV for all erroneous data which lacked an identifiable leader.

Once the data was properly transformed, formatted, and cleansed of errors, the remaining donations were uploaded to the central repository through a tDBOutput component. Using the Update or Insert action, all new rows were inserted into the central repository with sequentially generated IDs using a Talend function, while existing rows in the repository were updated with more recent data.

**A screenshot of a computer

Description automatically generated**

2. Talend Job layout for task two

# **Task Three**

The third task in the project required the creation of the star schema for the data mart tables as well as the development of the entity relation diagram to map the relations between the data mart entities. It was decided that the donation data would be best and most accurate served split into four dimensions tables tracking the volunteers, donation dates, donors, and address, and a single facts table to track the donations.

The dimensions tables included data used to identify the qualitative fields of each donation. These included the volunteer table, which contained the volunteer and leader names and IDs. The donor names and donations times were contained in the donor table. The donation date was tracked in the donation date table, which contained date, month, month name, year, and day of the week fields. Finally, the address and postal code for each donation was tracked in the address table.

The facts table included primarily metric data used to describe the quantitative information for each donation, such as the amount collected by the donation, and the payment type of the donation, which could be cash, check, or credit. Also contained in the donation table were the foreign keys used to link to the related facts tables through the unique dimension IDs for each table, allowing the donation table to be easily joined to the volunteer, donor, date, and address tables to display all donation data.

**A diagram of a computer program

Description automatically generated with medium confidence**

3. Entity Relation Diagram for Data mart tables

# **Task Four**

The fourth task required that data uploaded into the central repository in task two be loaded into the star schema as created in task three. For this purpose, a variety of objects were created for the ETLUser PRC, and Data Mart schemas.

To pass data between the later functions and procedures, data types and collections owned by the ETLUser were created for each star schema table. These included record types for containing donation, address, volunteer, donor, and date data, as well a collection type for containing instances of each record.

These data types were first used in the load functions owned by the PRC schema. These functions were responsible for loading data from the related PRC tables, creating the record instances to contain the data, and then adding each record to the related record collection. These collections were then returned and made available for insertion.

After the collections were returned, they were passed as arguments to the insert procedures owned by the Data Mart schema. These functions check the Data Mart dimensions tables and facts table for matching records. Where a match is found, the Data Mart table updates the data, and where a match is not found, a new record is inserted into the Data Mart table with a sequentially generated dimension or fact ID.

Where required, data was transformed using PL/SQL and SQL functions. Date data was transformed using the TO\_CHAR, TO\_NUMBER, and TO\_DATE functions, while a new function was written to combine address fields into a new field.

**A screenshot of a computer

Description automatically generated**

4. Sample contents of data mart donation table

# **Task Five**

The fifth task required the creation of several views to display data as retrieved from the data mart star schema. This task was completed using SQL to query the tables for the necessary data to be displayed.

**A screenshot of a computer

Description automatically generated**

5. sample selection from date donation view

# **Task Six**

In task six database security was implemented for two users: the ETLUser, and the Dashboard user.

The ETLUser was required to have permissions to read all repository data and to insert and upload data into the Data Mart star schema. A user role titled Extract\_Transform\_Load\_User was created to obtain all required permissions for the ETLUser, the most immediate of which is the CONNECT role to allow the user to connect to the database. After, two task roles were created, Read\_Repository\_Data, which was granted read permissions for the entire repository, and Update\_Insert\_Data\_Mart, which was granted write permissions to the star schema tables. These task roles were then granted to the user role, which was then granted to the ETLUser.

The Dashboard user was required to have permissions exclusively to read the views which present data contained in the star schema. For this, the Dashboard\_View\_User user role was created and granted the CONNECT role. The task role Read\_Datamart\_Views was created and granted read privileges on the Datamart donations views. This task role was then granted to the user role, which was then granted to the Dashboard user.

By following the above steps, the two users were granted the privileges necessary to complete their required tasks without granting the privileges directly to the users.

In addition to these dictated system and object privileges, the ETLUser, PRC, and Datamart schemas were granted additional privileges and roles required to operate on the objects owned by their schemas. The ETLUser was granted execute and read privileges on PRC’s load functions and on Datamart’s insert functions and sequential IDs. The PRC and Datamart schemas were granted access to the ETLUser’s record and table data objects, which were required to pass data between the load functions and insert procedures.

These additional privileges allowed the users to use the objects in their companion schemas but did not grant any additional privileges. The select, insert, update, and delete operations, and all system privileges remained unchanged, only allowing access to data objects in the specific schemas.

Some screen captures demonstrating the privileges (or lack thereof) of the ETL and Dashboard user are included below:

**A screenshot of a computer

Description automatically generated**

6. Successful selection of prc table by etluser

**A screenshot of a computer

Description automatically generated**

7. Successful (but invalid) demonstration of insertion into data mart by etluser

**A screenshot of a computer

Description automatically generated**

8. Successful (but invalid) Demonstration of update into data mart table by etluser

**A screenshot of a computer

Description automatically generated**

9. successful selection from data mart views by dashboard

**A screenshot of a computer

Description automatically generated**

10. Unsuccessful demonstration of insertion into data mart and prc tables by dashboard

**A screenshot of a computer program

Description automatically generated**

11. Unsuccessful demonstration of update into data mart and prc table by dashboard

# **SQL Code**

A listing of all SQL Scripts used in this project includes:

* Address\_Insertion\_Trigger.sql: Contains the SQL to create the trigger to repeating transformations on address data uploaded in Task One.
* Create\_Dashboard\_User.sql: Contains the SQL to create the Dashboard user in Task Six.
* Create\_Data\_Objects.sql: Contains the SQL to create the Data Objects for the ETLUser in Task Four.
* Create\_Datamart\_Views.sql: Contains the SQL to create the Data Mart views for Task Six.
* Create\_ETL\_User.sql: Contains the SQL to create the ETLUser in Task Six.
* Create\_Star\_Schema.sql: Contains the SQL to create the Star Schema in Task Three.
* Load\_Data\_Mart\_Data.sql: Contains the SQL to load the Data Mart data as the ETLUser in Task Four.
* Load\_Insert\_Addresses.sql: Contains the prc.load and datamart.insert functions to insert addresses in Task Four.
* Load\_Insert\_Dates.sql: Contains the prc.load and datamart.insert functions to insert dates in Task Four.
* Load\_Insert\_Donatons.sql: Contains the prc.load and datamart.insert functions to insert donations in Task Four.
* Load\_Insert\_Donors.sql: Contains the prc.load and datamart.insert functions to insert donors in Task Four.
* Load\_Insert\_Volunteers.sql: Contains the prc.load and datamart.insert functions to insert volunteers in Task Four.
* Select\_All\_Data.sql: Contains the select statements to read data from the entire central repository.

# **Java Code**

The two Java Scripts used to complete this project include:

* AddressDataTransformer.java: Talend routine used to perform transformations on address and donations date. Included the following functions:
  + isTypeValid(String strType): Boolean function to check if payment type is valid.
  + getLastName(String strName): String function to return donor last name.
  + getFirstName(String strName): String function to return donor last name.
  + getTitlecasedString(String strInput): String function returning title case String.
  + getAddressNumber(String strAddress): String function to return address number if available.
  + getStreetType(String strAddress, String strTypes): String function to return address street type.
  + getStreetDirection(String strAddress, String strAddresses): String function to return address street direction.
  + getAddressProvince(String strAddress): String function to return street address province.
  + getAddressCity(String strCity): String function to return street address city.
  + getStreetName(String strAddress, String strTypes, String strDirections): String function to return street address name.
  + combineAddressDetails (int number, String address, String type, String direction, String city, String province): String function to return address fields combined into single String.
  + getDateFromString(String strDate): Date function to parse a Date object from a provided String.
* PostalCodeApiRetriever.java: Used to run the API calls for retrieving postal codes data for the bonus task. Used the following function:
  + getPostalCode(String strAddress): String function to retrieve postal code from address using Mapbox Geolocation API.