**Group Project: Final Report**

# **DBAS32100 Database Management**

**Gurveer Singh, Student id: 991631940**

**Professor Amandeep Kaur**

**Sheridan College**

**April 18th, 2022**

**Task 1: Address Table**

This process is created using Talend Studio 7.3.1. In this process, The address table in the Oracle donation repository is refreshed with the addresses table in the master SQL server table. First of all, connection with the Sheridan VPN was so that we can connect with the Microsoft SQL server table. Further, entries of the input table were mapped with the output table using a tmap and the method to generate sequential address\_id was also defined here.

Some of the data types were also changed so that the address table can hold all the values. The addresses table did not have values for postal codes so a trigger was also created to supply default postal codes in the address table.

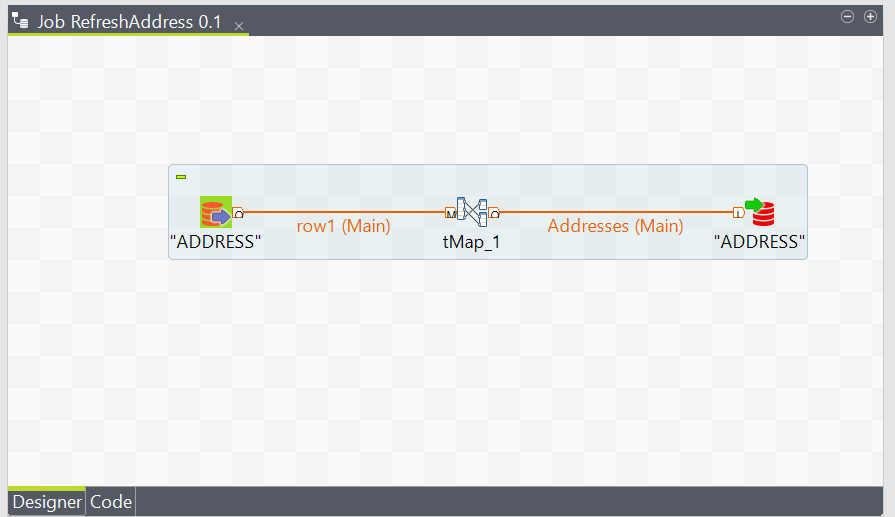


Figure 1: Job for refreshing address table with addresses table

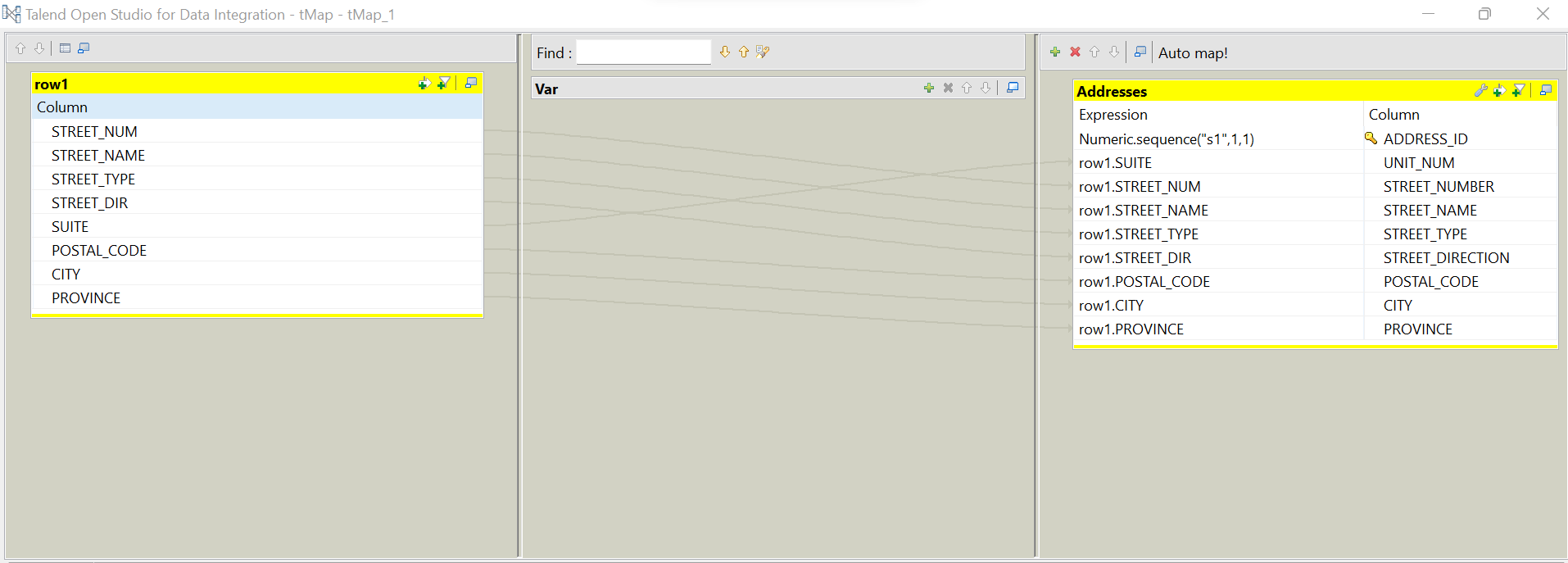


Figure 2: Mapping the two tables

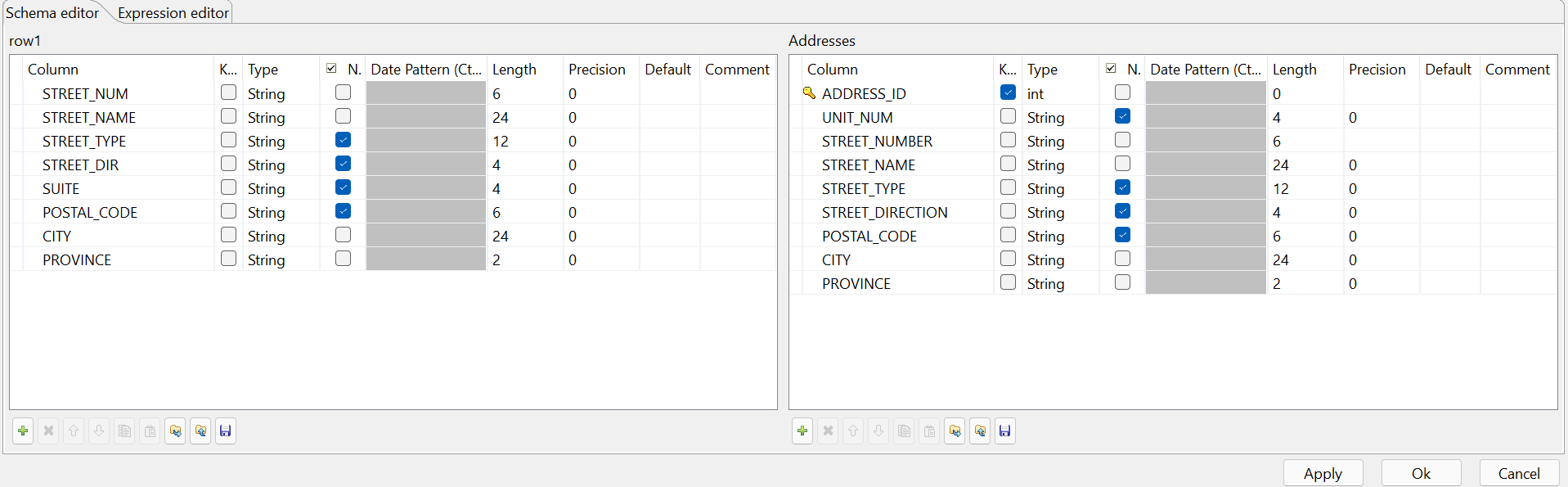


Figure 3: Schema for both tdb\_input and tdb\_output table.

**Task 2: Central Donations Table**

This process is created using Talend Studio 7.3.1. In this process, I have incorporated the entries for CSV files to the output tables called Addresses and Donations table. I had created a mapping technique where it can filter the valid addresses and valid donations entries. In the meantime, it rejects any entry that does not relate to valid entries of addresses or donations.

After I applied the CSV file, I saved the valid addresses into the database under a table that is called the address table. Also, I saved the valid donations into the database under another table that is named donations.

This process will be repeated for any new entries. The main purpose is to accept any entries that fall under a valid address format to be saved in the central donation repository; however, it rejects any other entries that have invalid address format or null value.

An illustration diagram has been attached below in figure 4 to describe the process above.

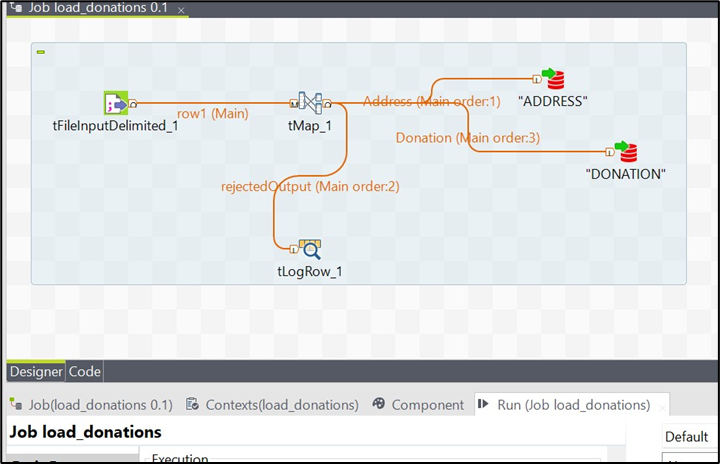
****

Figure 4. The process of CSV file entries loading into the Central Donation Repository

The schema that has been used to filter valid addresses is shown in figure 5.  
It shows how the valid address entries have been filtered and saved in the repository.

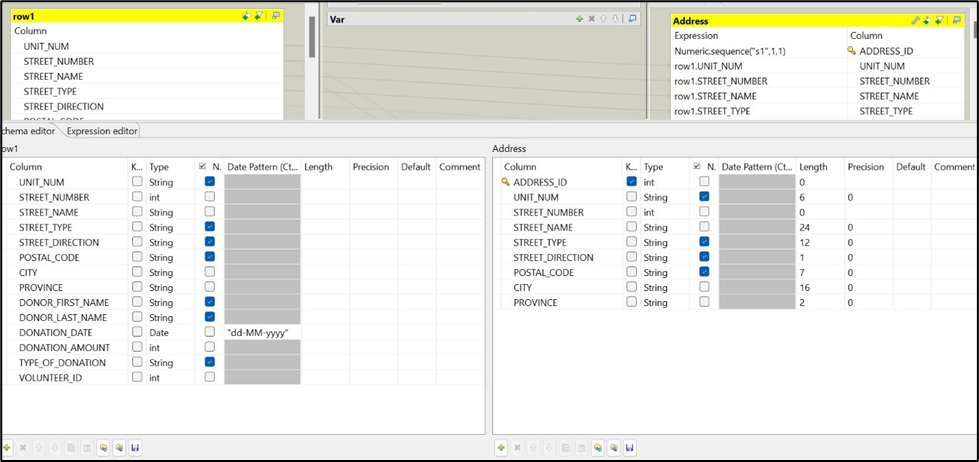


Figure 5. The schema used to filter valid addresses

In figure 6, it shows how the valid addresses have been saved in the repository. Also, it shows how the invalid or null addresses have been rejected after applying the CSV file.

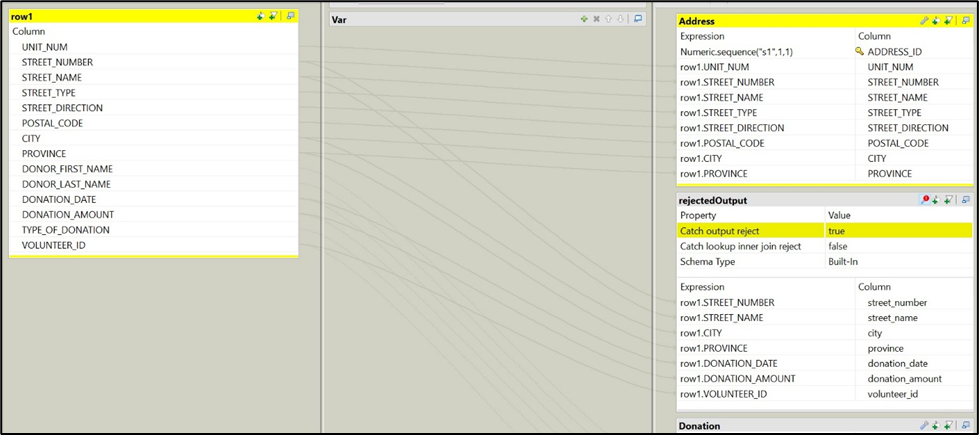


Figure 6. The valid addresses and rejected ones

**Task 3: Star Schema & ER Diagram**

**ER Diagram**

Below is the ER Diagram for the donation data mart. This process was done using SQL Developer in Relational Model Node, where I added the 3 tables (address, donation, volunteer) which then connected through objects with proper foregin key constraints and also you can see the relationships between each table.

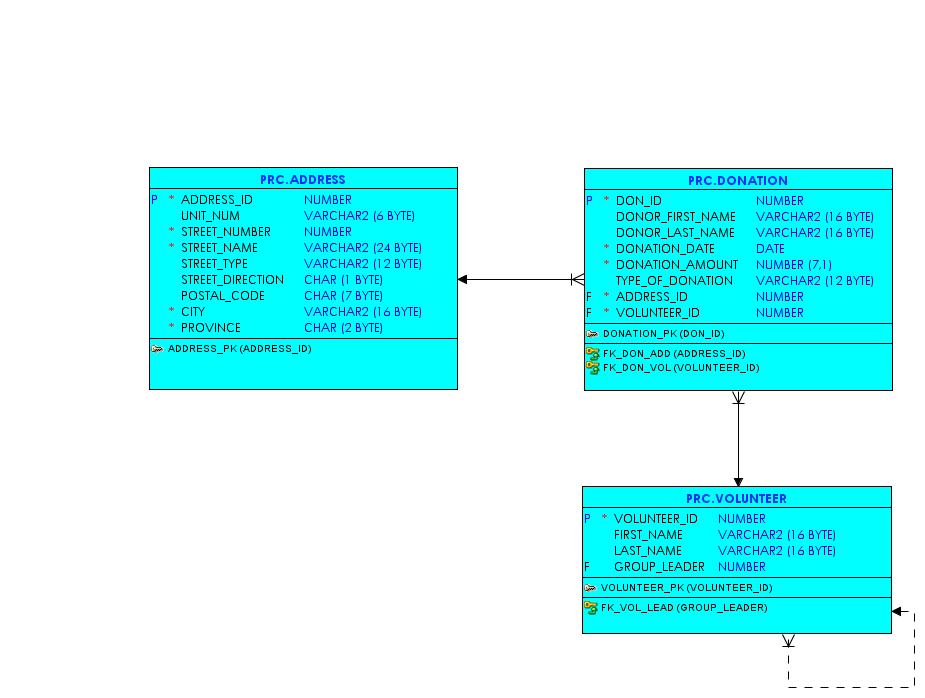


Figure 7. ER Diagram for the donation data mart

**Star Schema**

For this task, I started by understanding the 3 donation data mart tables which are address, donation and volunteer. Next, I started creating the appropriate dimension tables including dim\_donations, dim\_address, dim\_volunteer and dim\_date. I also created surrogate keys for dim\_donation, dim\_address and dim\_volunteer tables. The dimension tables also have their own primary keys such as donation\_key, address\_key, volunteer\_key and date\_key.

For the dim\_date table, I created the table with the appropriate date format. I also created a fact table named fact\_donation, which included primary keys from all 3 dimension tables as well as the number of donations and the total amount of donations. After carefully creating the dimension and fact tables, I started writing insert statements to insert appropriate data into each table.



Figure 8. Example of dim\_address table, here is the create, surrogate and insert statements for dim\_address table

Here are the results of extracting data from the address table and inserting it into dim\_address, here you can see the rows that are inserted and as well as the address\_key that is generated.

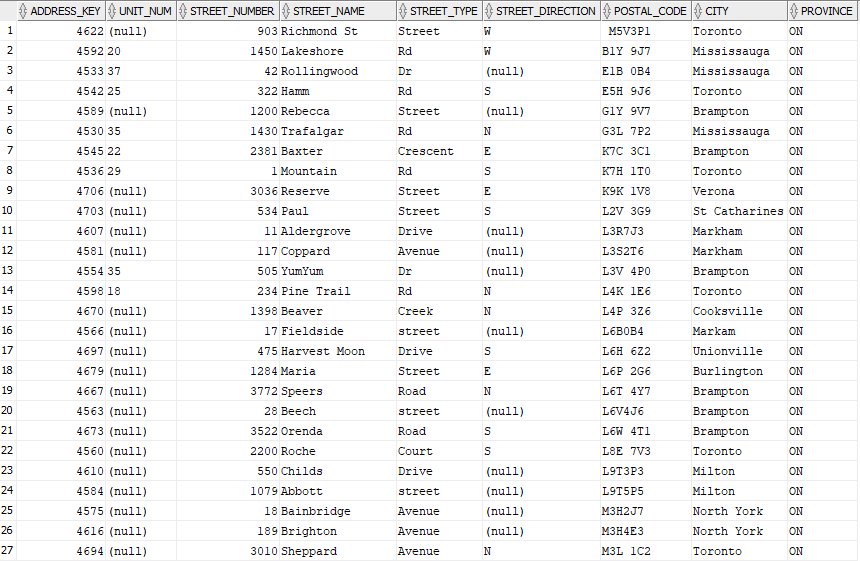


Figure 9. Results from dim\_address table

For the creation of the fact\_table, I used address\_key, donation\_key, volunteer and date\_key as number references for each dimension table.

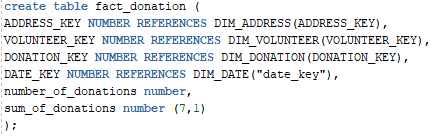
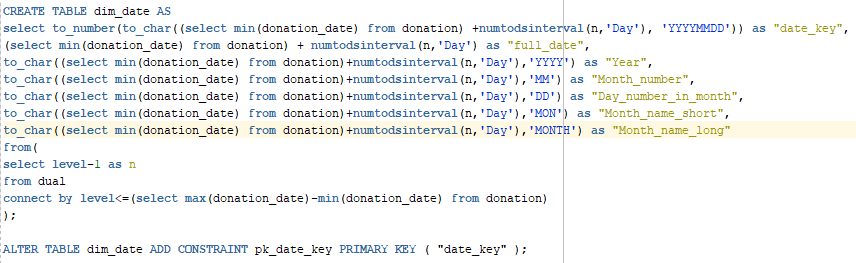
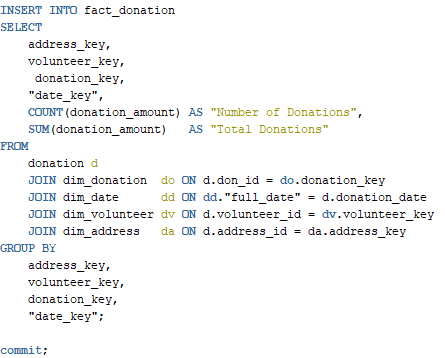


Figure 10. Create statement of fact\_donation table

For the dim\_date table, I started by reviewing the requirements and created a dim\_date table with and extracted the dates according to the requirements. I also did an alteration on the table to add “date\_key” as a primary key.

Figure 11. Creation of dim\_date table, with date extraction for specific formats

Finally, I added an insert statement to insert data in the fact\_donations table, therefore connecting all dimension tables to the fact table and with a commit statement at the end.

  
Figure 12. Fact\_donation table insert statement

**Task 5: Creating Views**

For the first view, the requirements were that the view should include the donation date, and from that date the year, month, and day were extracted. It also needed the month represented in name form, and including the number of donations and total donation amount. After extracting the month into number form, I created a case statement that takes the month number and provides the name of the month. This view displays the total amount of donations and count of donations and is grouped by each date.

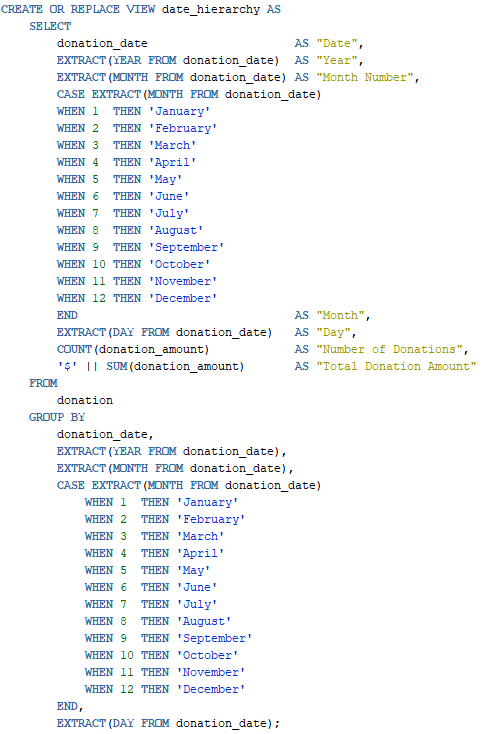


Figure 13. Creating a date\_hierarchy view with date extraction and case statements with number of donations and total donation amount.

For the second view, the requirements for a view was number of donations, sums and average donations with postal code and address hierarchy. First I wrote the create view statement that includes columns postal\_code and then a concatenated column of street\_number, street\_name, street\_type, street\_direction, city, and then province. Then, finally 3 other columns such as a count column with number of donations, a sum column with total donation amount and finally an average donation column. This data displays the total donations, amount and average donations for each postal code and address.



Figure 14. Creating a location\_hierarchy view that displays results by address order and grouped by address.

Finally, for the last view, the requirements was that it should include volunteer leaders’s name, volunteer’s name, number of donations, total donation amount and lastly average donation for each volunteer. First I checked who are the group leaders and their IDs then I wrote a create view statement with a case that assigns the volunteer’s name for specific group\_leader id, then I concatenated the volunteer’s first and last name and a count for the donation and sum of donation and finally the average of the donations, which is grouped by each volunteer and ordered by volunteer leader.

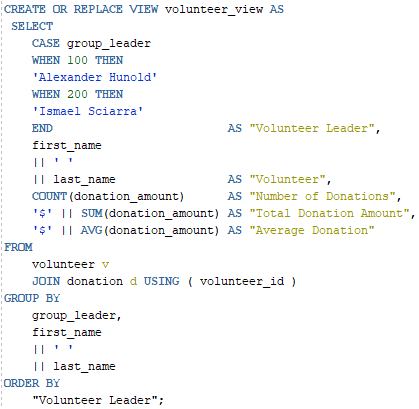


Figure 15. Creating a volunteer\_hierarchy view statement

**Task 6: Basic Security**A username with ETLUser name has been created. Permission to read the data has been granted to the ETLUser to read data from the Central Donation Repository. The screenshots added show that ETLUser has read permission only and rejects any other query if asked to use any other permission. This is shown in figure 15.

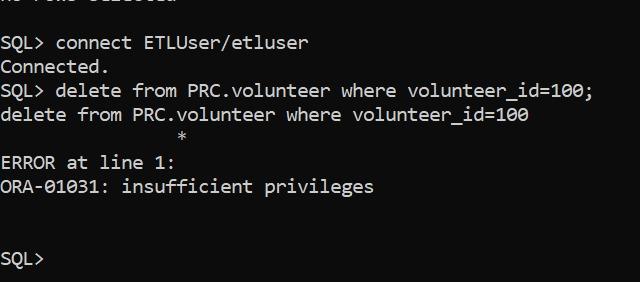
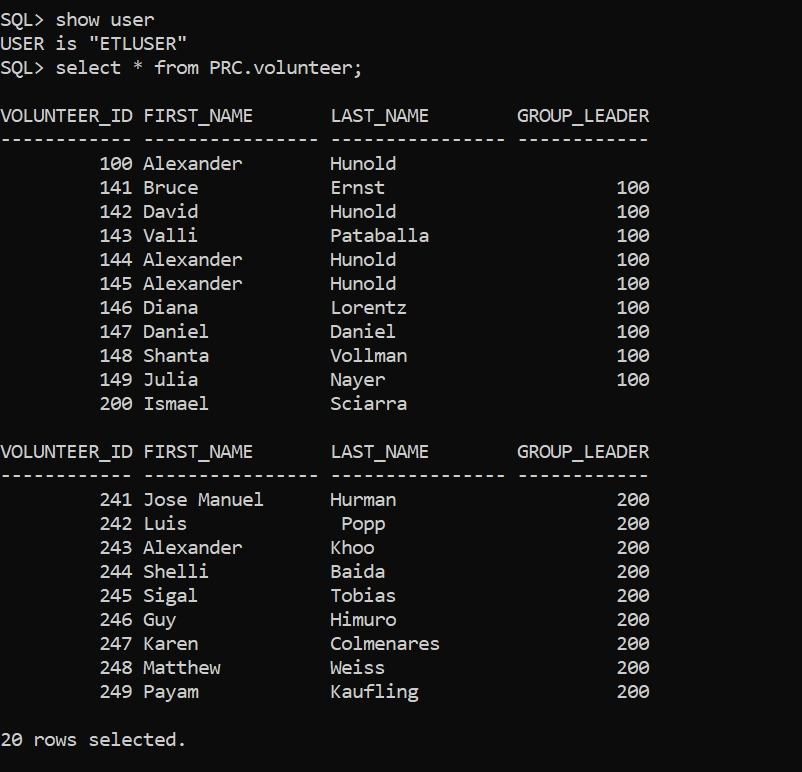
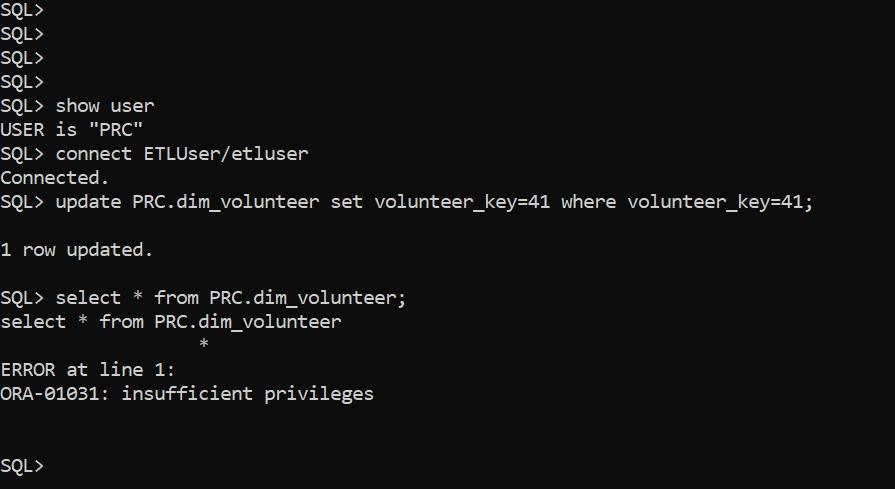


Figure 16: Read and Rejected Permissions for ETLUser to Central Donations Repository

Also, the user write and update permissions have been granted to ETLuser to the DataMart tables. Screenshots have been added that show how ETLUser rejects permissions other than write and update. This is shown in figure 16.

  
Figure 17: Write, update and rejected permission for ETLUser to the DataMart Tables

A username named Dashboard has been created. User read permission has been granted to view the data in the Central Donation Repository. That is shown in figure 17.

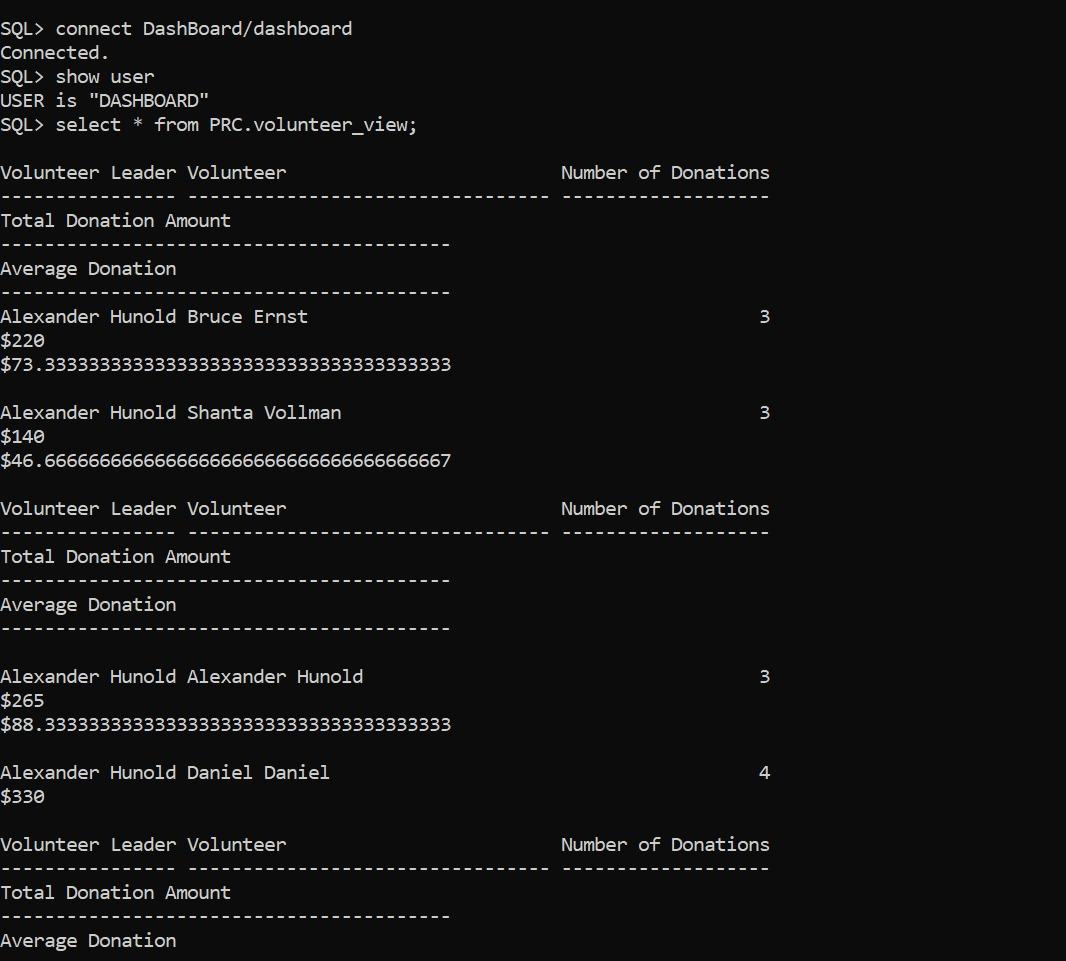


Figure 18: Read permission for Dashboard user to read the views