**Database Group Project Report**

Sienna Truong

Camila Esguerra

Dong Nguyen

Sheridan College

Software Development and Networking Engineering

Sami Ahmed

Database management

DBAS32100

November 27 of 2024

**Description**

For the sake of simplicity, henceforth, whenever Leader1 is mentioned, it refers to the account S16\_TRUONCHI in Oracle. Similiarly for Leader2 and Leader3, the accounts S16\_ESGUERCA and S16\_NGUYE617 are chosen. We opted to describe these roles as Leader1, Leader2, and Leader3 to shorten the names and to make it more relevant to the original project description.

For the project, the team successfully completed the project tasks based on the outline in the group project document provided in class, and all results were documented with screenshots and straightforward descriptions in this report. In Part A, using Leader1's login, the team created the necessary tables in **Oracle SQL** based on the provided Excel data, ensuring proper table structures and relationships. Then, based on the Oracle tables, they created CSV files that contain the data provided by the original task, and modified them to fit the new schema. After that, they used Talend Open Studio to export data from CSV files into Oracle tables, verifying the data import with SELECT statements in SQL Developer. Additional jobs were created in Talend to display data and print customer rental details.

In Part B, using Leader2's account, the team developed PL/SQL scripts for inserting, updating, and deleting customer data, making the scripts flexible with substitution variables. They also wrote SQL queries for analytical tasks, such as listing customer names and DVD rental costs, and created PL/SQL procedures and functions for specific tasks.

In Part C, using Leader3's account, the team analyzed the DVD table to identify dimensions and facts, created a Star Schema and ER diagram, and populated the fact and dimension tables using insert and select queries. They ensured data accuracy and consistency.

In some sections of the following Parts, you will notice a discrepancy between the file name in the screenshot, namely `create-tables.sql`, versus the final version of the file `main.sql`. This was an intentional rename as the name `create-tables` was no longer suitable for the task of CREATE and INSERT which are expected to fit into only 1 SQL file.

**Part A – Sienna**

**Task 1**

Prompt: “Create the tables in Oracle as per data given in the excel file (GP - DBAS32100). Feel free to add any fields you think may be necessary, and feel free to change any existing fields to appropriate data (i.e. ID fields/ data types). Make sure to mark Primary key, foreign key relationships, unique constraints and check constraints wherever required, for this task make sure to observe data present in csv files. Once you have the database/tables with appropriate tables/fields/relationships ready. Create appropriate “.cvs” files for the given tables in the excel sheet if there is any change introduced by you in tables. Once ready take screenshots of table structure that you have created.”

**Execution**

All the Enums where created first to preserve data integrity across the tables and ensure that the foreign key constraints can be properly established later with other tables.

First, these three tables where created to store the possible ratings for DVDs, to store the possible statuses for customers, and to store the possible actions to be taken when a DVD is returned.

A screenshot of a computer

Description automatically generated

In the following images, we can observe how the code went through and created the Enum tables successfully.

A screenshot of a computer program

Description automatically generated

After creating the Enum tables, the creation of the rest of the following tables was done as well. Including proper primary and foreign keys, as well as the same table columns from excel file.  
A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

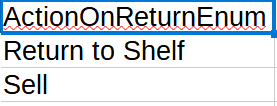
Description automatically generated

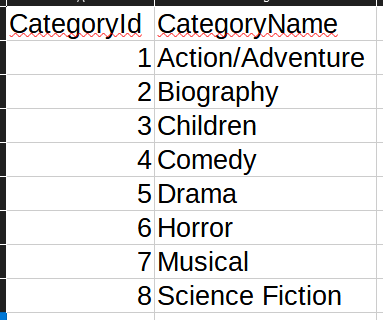
A screenshot of a computer

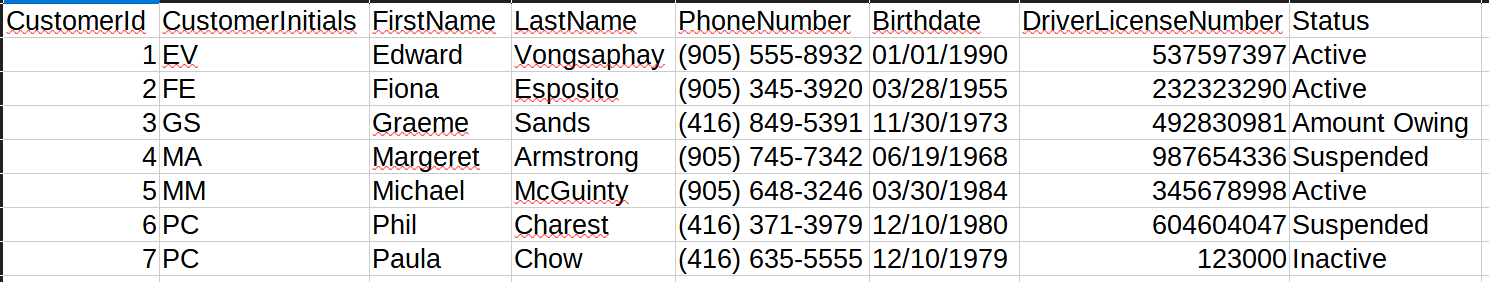
Description automatically generated

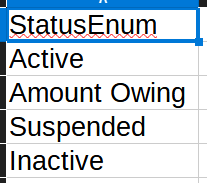
Next is the task of creating CSV files. To do so, we will split the data accordingly from the original Excel file into their appropriate CSV files. This means some data might need to be added, and some data will need to be moved around. The end result is as follows:

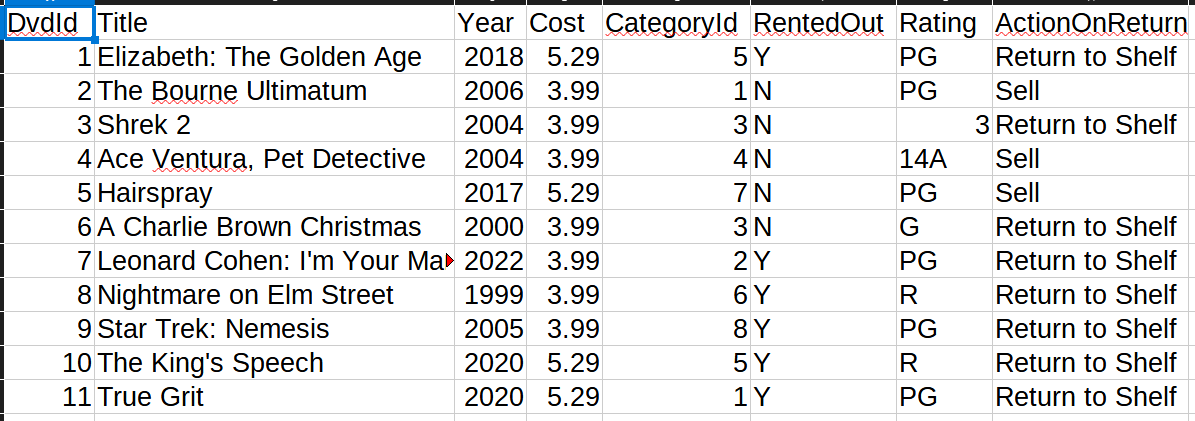
**ActionOnReturn**

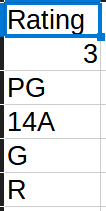
**Category**

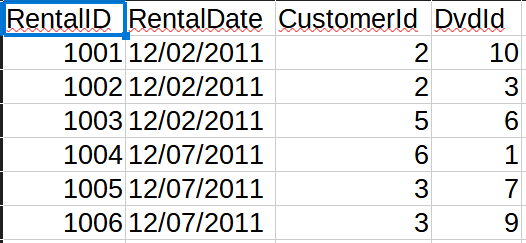
**Customer**

**CustomerStatus**

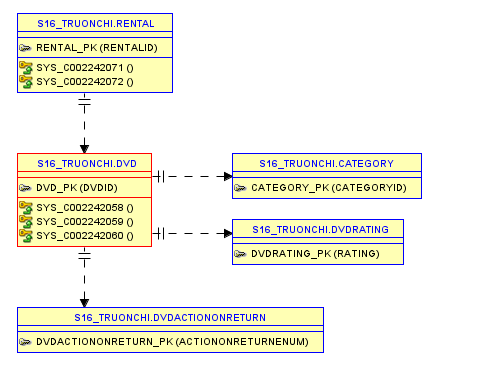
**Dvd**

**Rating**

**Rental**

After this, all that’s left is populating all of the tables with the correct data in Task 2 and run a select script to make sure it made effect, thus ensuring it matches the value for the table and the information on the excel.

**Final Table Structure**

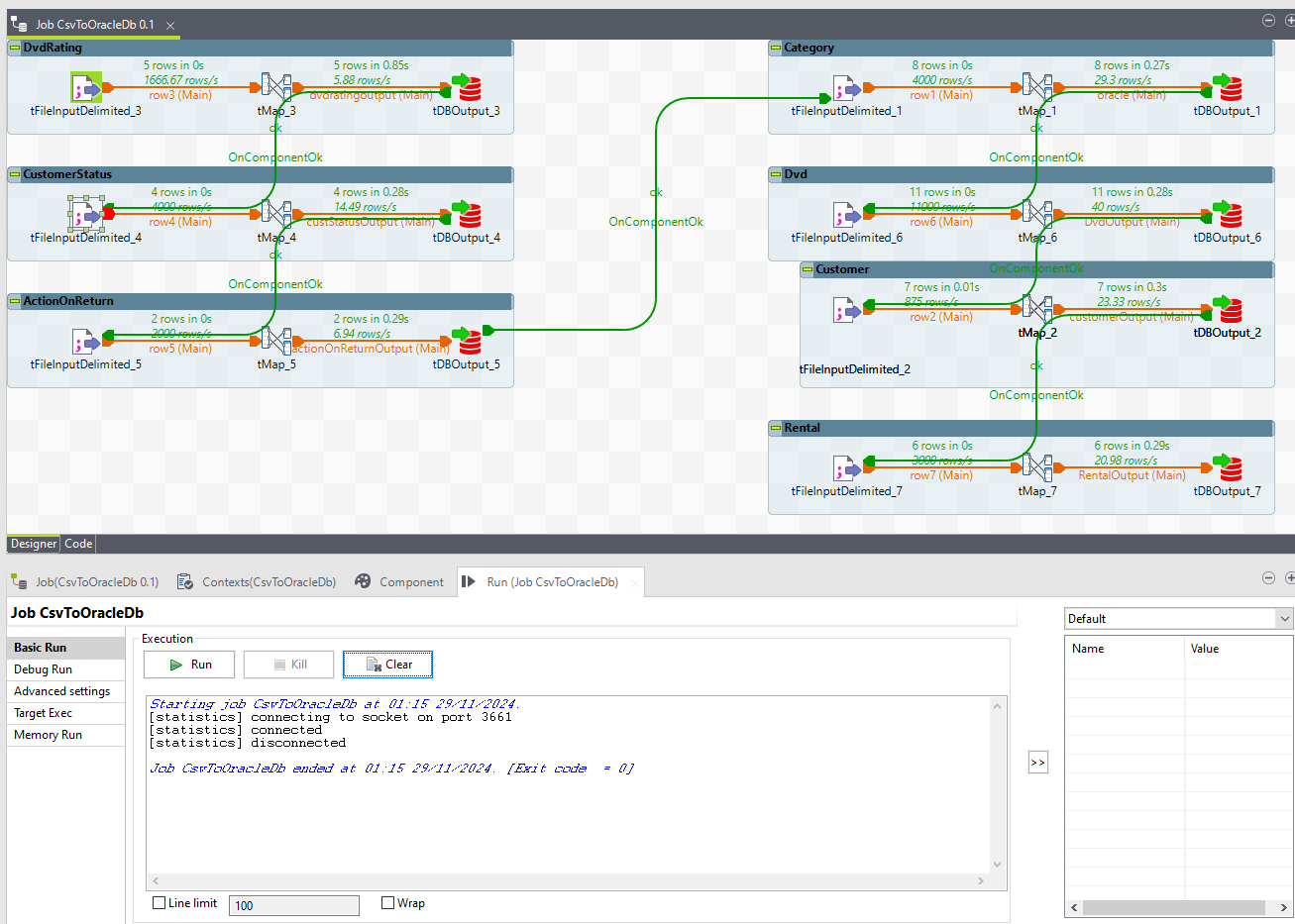


**Task 2**

**Prompt:** “Using Talend Open Studio, export the data from the CSV file one by one into the respective oracle tables that you have created in the SQL developer, and later show the data from the tables in SQL developer using select statements. Take screenshots of all tables and jobs showing data is successfully populated.”

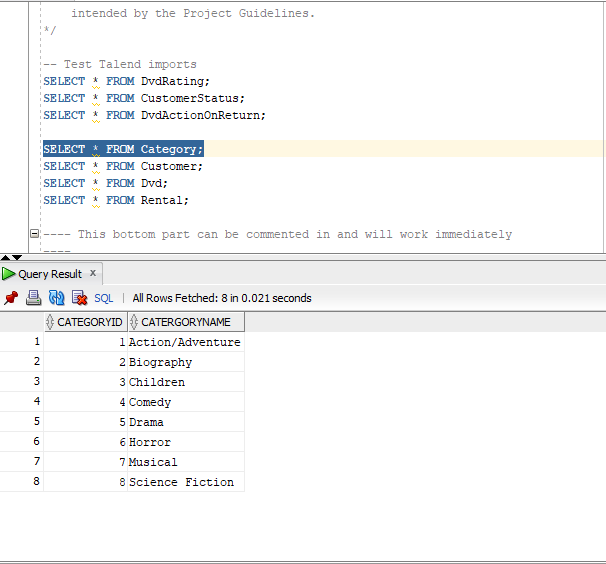
**Execution:** Within Talend Open Studio, we will create a job with 7 subjobs, each of them will only run after the last one succeeds. Furthermore, we will create individual schemas for each object and modify the data types accordingly using Java.

After the creation of all these schemas and subjobs, we are left with a complete job to run.

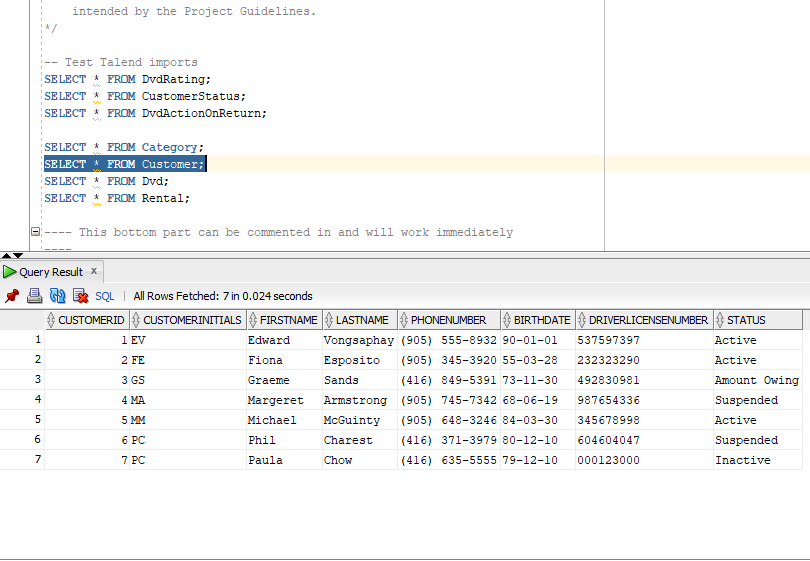
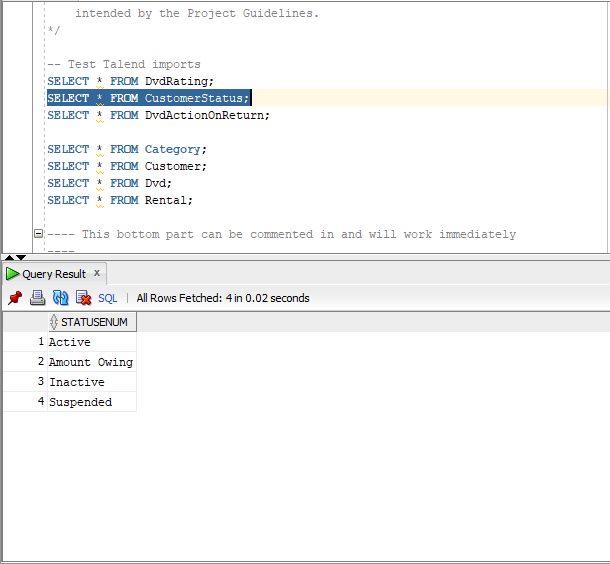
As we can see, this job was executed successfully, meaning all the data provided in `tFileInputDelimited` are successfully mapped into the correct types by `tMap` and are inserted successfully into our DB. To test this, we will run select statements to double check.

**ActionOnReturn**

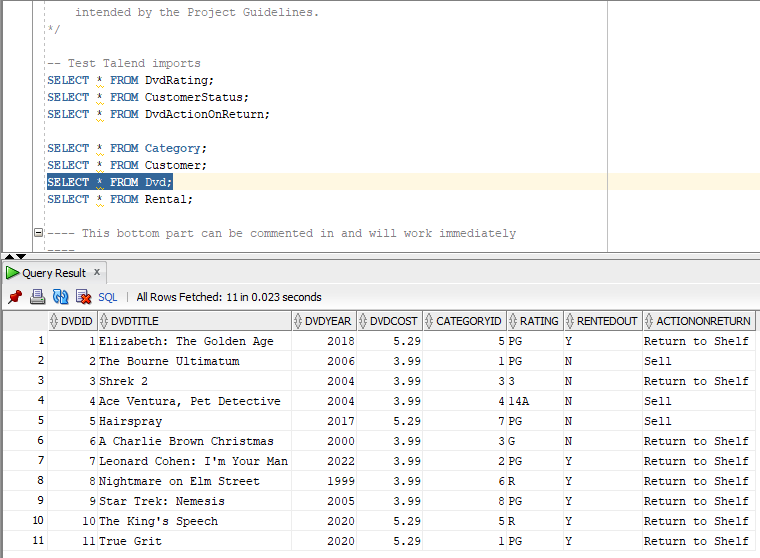
**Category**

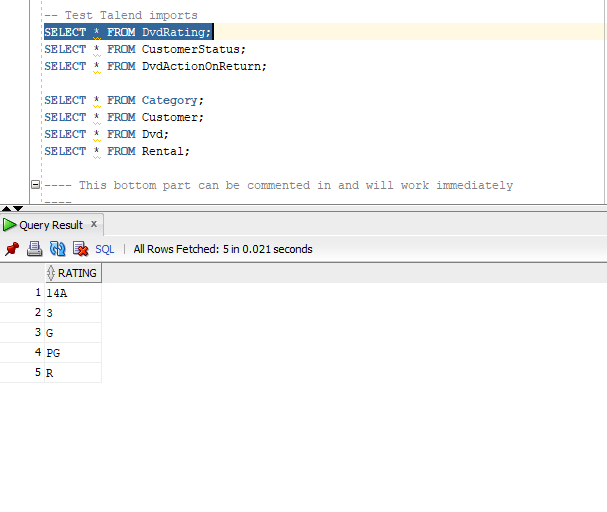
****

**Customer**

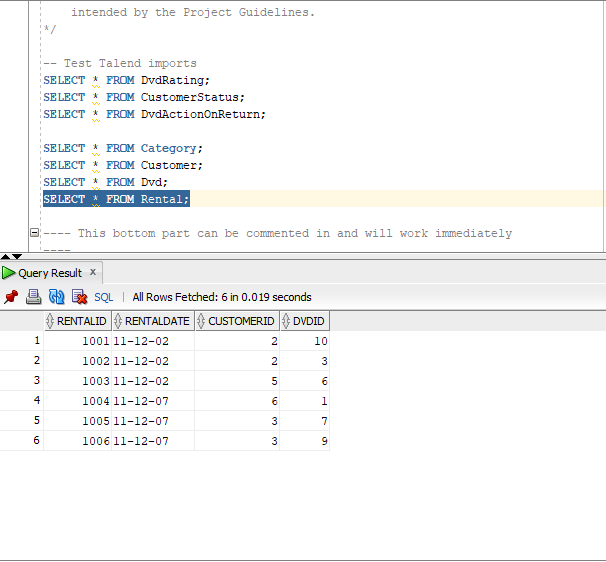
**CustomerS****tatus**

**Dvd**

**Rating**

****

**Rental**

As we can see, all the CSV are inserted successfully into Oracle! Moving on to Task 3.

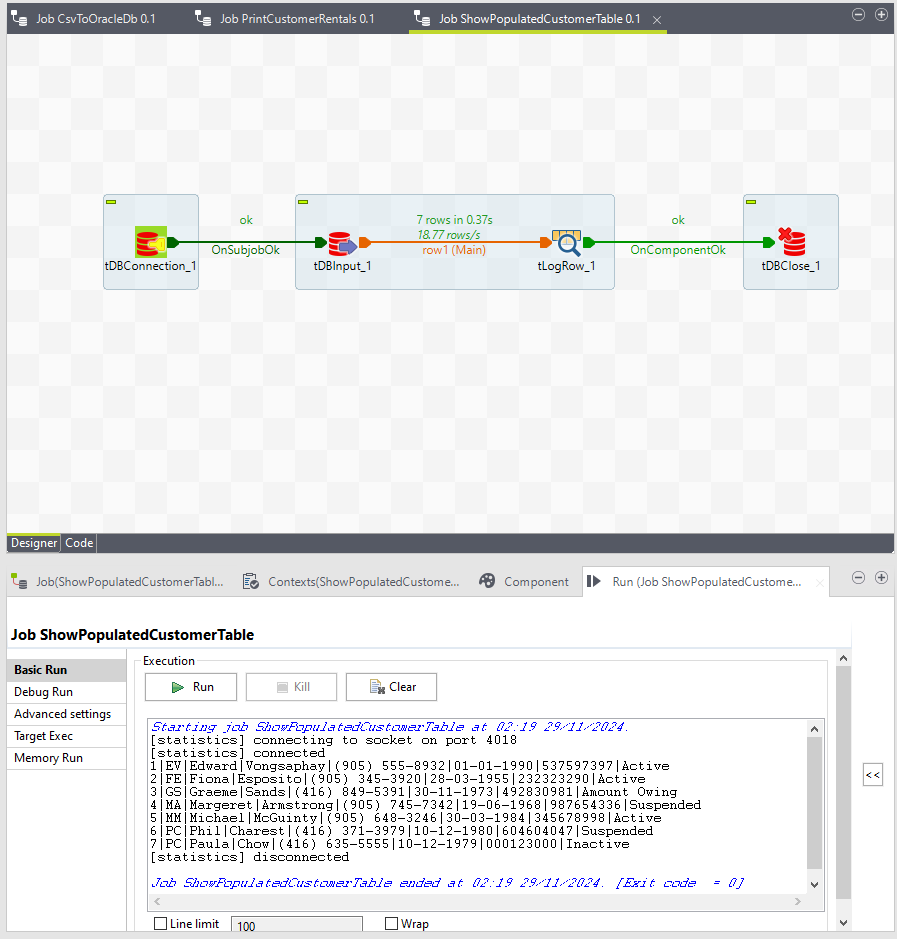
**Task 3**

**Prompt**: “Using Talend Open Studio, Create a new Job. Select any 1 table from Oracle that you have populated in last step and In that job, display the data from the oracle tables to log and take screenshots.”

**Execution**: For this task I chose the Customer table because it has the most Foreign Keys and it would prove the integrity of our database the most. For this, we make a new job in Talend, add a `tDBConnection` block and connect it to our DB. Once successful, execute the next block which is `t\_DBInput`. This block allows us to input a SQL statement into the DB and get a return value. In our case, we are inputting a SELECT statement (SELECT \* FROM Customer).

After the Input was successful, it will return us all the rows of the SELECT statement. Then we need to log each row into our Log down the execution button. For this we use `tLogRow` to output each row into the console.

Finally, once we’re done, we close the connection with `tDBClose`. Thus, our job will look like this.



**Task 4**

**Prompt**: “Using Talend Open Studio, Create a new Job to print customer names, dvd’s that they have rented and rental cost using TMap and Joins. Make sure to display customers who have not rented any movie yet as well.”

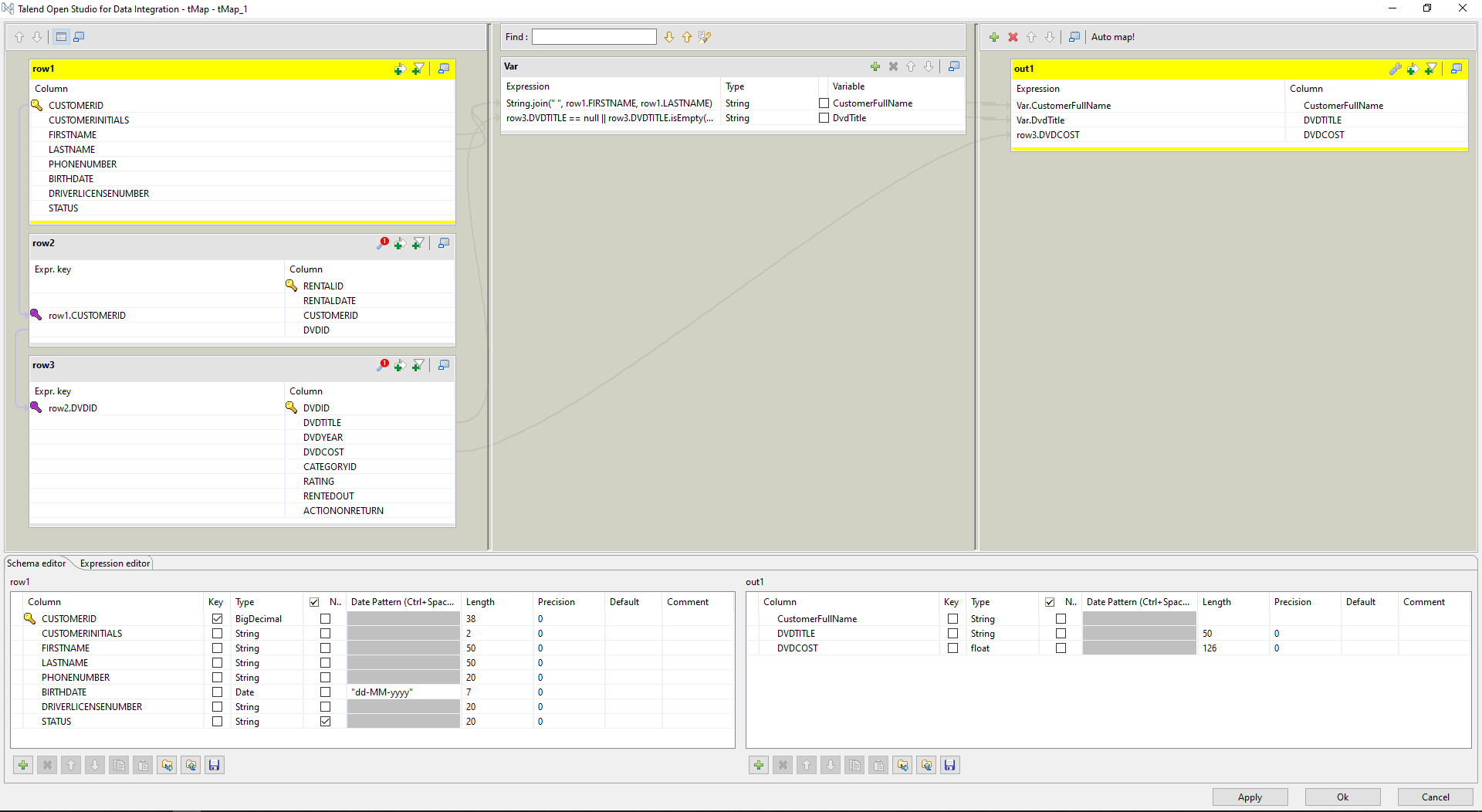
**Execution**: This task is quite similar to the previous one. Start by creating a new job and set up a connection to the DB with `tOracleConnection`. Then we retrieve our necessary data. Since the task requires a list of customer names and the DVDs they rented, and their cost, we will need data from 3 different tables: Customer, Dvd, and Rental. Start with the Customer table, we will do the same as Task 3: we use `tOracleInput` to run a query (SELECT \* FROM Customer). Then we do the same for Rental and Dvd, each in their separate `tOracleInput`. We should end up with 3 inputs.

To join this data, we use `tMap`. We will link the inputs into this block and this will create a new subjob. Now we configure tMap to know what data we are mapping.

* We map the CustomerId from the Customer table to the CustomerId in the Rental table. This joins Rental into Customer.
* Then we map the DvdId from the Rental table to the DvdId in the Dvd table. This joins DVD into Rental.

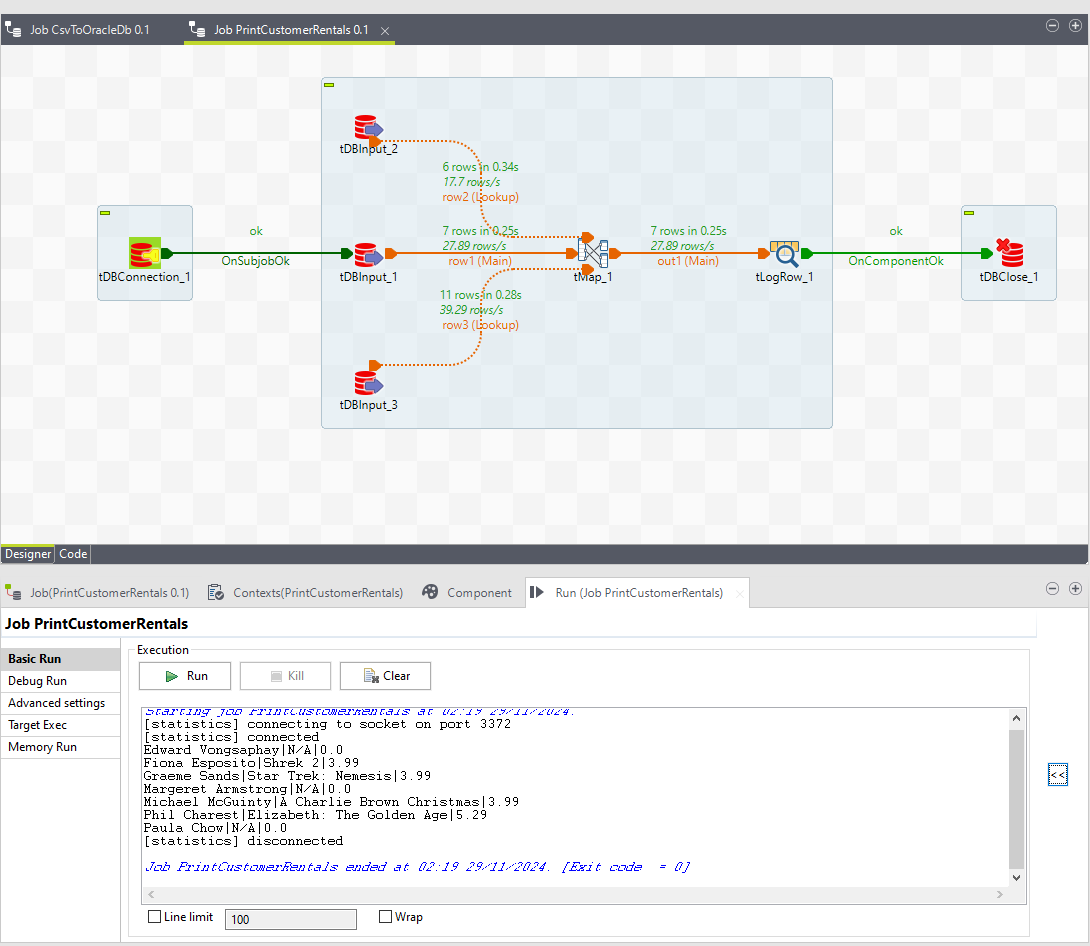
Now that all the tables are joined, we map the columns we want into the correct types. Then we map those into the output.

**tMap**

****

Finally, we connect `tMap` to `tLogRow` to show the rows in the console, and kill the connection after with `tDBClose`.

**Final job**



**Part B – Camila**

**Task 5**

Prompt: “Create PL/SQL scripts(you can use PL/SQL procedures or functions as per your choice) to support the following operations for customer table. Once done paste code for all three scripts and their testing screenshots. Remember PL/SQL scripts required not just SQL Queries.

* Insert Data
* Update Data
* Delete Data

[Make sure to use substitution variables to take input from user and test your script to insert data by inserting one record for each group member where customer name should be name of the group members in your group, then test update script by updating data for any one of you and then test delete data script by deleting any one-of yours’ record. Script should be executable as many numbers of times end user wants without any change.]”

This procedure inserts a new customer record into the Customer table. It takes various customer details as input parameters and inserts them into the table.

A screenshot of a computer

Description automatically generated

Testing the procedure after running

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer error

Description automatically generated

A screenshot of a computer error

Description automatically generated

A screenshot of a phone number

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer error

Description automatically generated

A screenshot of a phone number

Description automatically generated

This is the update data procedure for the Customer table, where the user should be able to update some values like phone number or status.

A screenshot of a computer

Description automatically generated

A screenshot of a computer error

Description automatically generated

A screenshot of a computer error

Description automatically generated

A screenshot of a computer

Description automatically generated

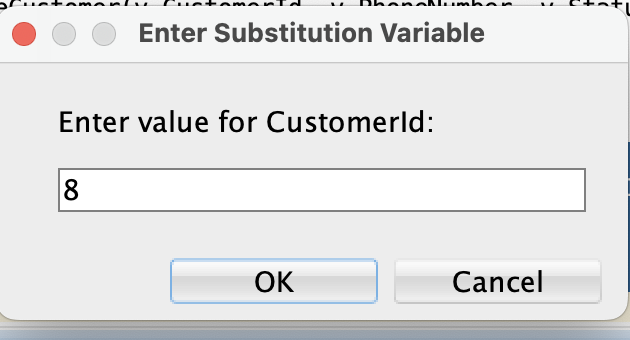
A screenshot of a computer

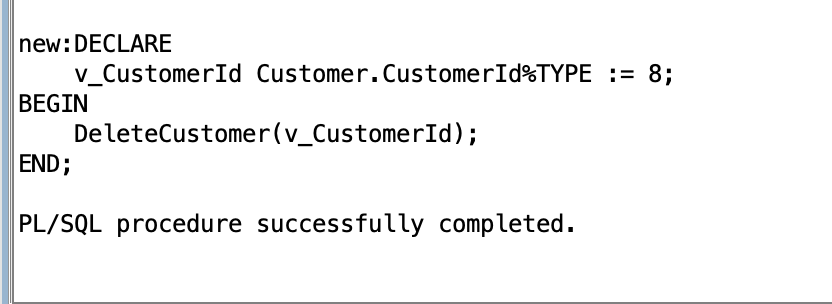
Description automatically generated

This is the delete data procedure for the Customer table, where the user should be able to delete a whole customer based on their id.

A screenshot of a computer code

Description automatically generated





A screenshot of a computer

Description automatically generated

**Task 6**

Prompt: “Write SQL Queries for following tasks as per instructions given below;-

* 1. Show the customer name and DVD name along with rental cost in list starting from which has least renting cost to highest renting cost using analytical functions?
  2. Show the titles and cost of DVDs who are with third and fourth highest price using Analytical Functions?”

**Part a**

This query lists customer names, DVD titles, and rental costs, sorted from the least to the highest rental cost using the RANK() analytical function.

A screenshot of a computer

Description automatically generated

**Part b**

This query shows the titles and costs of DVDs with the third and fourth highest prices using the ROW\_NUMBER() analytical function.

A screenshot of a computer program

Description automatically generated

**Task 7**

1. Prompt: “Pick any 2 questions from below of your choice and create procedures/functions in PL/SQL . You can choose any argument (of your choice) for the function/procedure.
2. Show the names of the customers, movie name and the number of movies rented by them. Make sure to show subtotals as per customer (number of movies rented) and subtotal for each movie showing how many times that movie was rented.
3. What is the difference between DVD cost and the average cost of all the DVDs of each year.”

For the first point of task 7, I divided the tasks into two procedures so it’s easier for the coding and output.

**First procedure (a)**

This procedure displays the names of customers, the movies they rented, and the number of times each movie was rented.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

**First procedure (b)**

This procedure displays subtotals of rentals per customer and per movie and per customer.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Second procedure**

This function calculates the difference between the cost of each DVD and the average cost of all DVDs for each year.

A screenshot of a computer

Description automatically generatedThe code for the entirety of PartB:

```

-- Insert data

CREATE OR REPLACE PROCEDURE InsertCustomer (

p\_CustomerId IN Customer.CustomerId%TYPE,

p\_CustomerInitials IN Customer.CustomerInitials%TYPE,

p\_FirstName IN Customer.FirstName%TYPE,

p\_LastName IN Customer.LastName%TYPE,

p\_PhoneNumber IN Customer.PhoneNumber%TYPE,

p\_Birthdate IN Customer.Birthdate%TYPE,

p\_DriverLicenseNumber IN Customer.DriverLicenseNumber%TYPE,

p\_Status IN Customer.Status%TYPE

) AS

BEGIN

INSERT INTO Customer (CustomerId, CustomerInitials, FirstName, LastName, PhoneNumber, Birthdate, DriverLicenseNumber, Status)

VALUES (p\_CustomerId, p\_CustomerInitials, p\_FirstName, p\_LastName, p\_PhoneNumber, p\_Birthdate, p\_DriverLicenseNumber, p\_Status);

COMMIT;

END;

/

--Update data

CREATE OR REPLACE PROCEDURE UpdateCustomer (

p\_CustomerId IN Customer.CustomerId%TYPE,

p\_PhoneNumber IN Customer.PhoneNumber%TYPE,

p\_Status IN Customer.Status%TYPE

) AS

BEGIN

UPDATE Customer

SET PhoneNumber = p\_PhoneNumber,

Status = p\_Status

WHERE CustomerId = p\_CustomerId;

COMMIT;

END;

/

-- Delete data

CREATE OR REPLACE PROCEDURE DeleteCustomer (

p\_CustomerId IN Customer.CustomerId%TYPE

) AS

BEGIN

DELETE FROM Customer

WHERE CustomerId = p\_CustomerId;

COMMIT;

END;

/

---

--Task a: Show the customer name and DVD name along with rental cost in a list starting

--from the least renting cost to the highest renting cost using analytical functions.

SELECT

c.FirstName || ' ' || c.LastName AS CustomerName,

d.DvdTitle,

d.DvdCost,

RANK() OVER (ORDER BY d.DvdCost ASC) AS CostRank

FROM

Rental r

JOIN

Customer c ON r.CustomerId = c.CustomerId

JOIN

Dvd d ON r.DvdId = d.DvdId

ORDER BY

d.DvdCost ASC;

----------------

-- Task b: Show the titles and cost of DVDs with the third and

-- fourth highest price using Analytical Functions.

SELECT

DvdTitle,

DvdCost

FROM(

SELECT

DvdTitle,

DvdCost,

ROW\_NUMBER() OVER (ORDER BY DvdCost DESC) AS PriceRank

FROM

Dvd

)

WHERE

PriceRank IN (3, 4);

---

--Task 1: Show the names of the customers, movie name, and the number of

--movies rented by them. Make sure to show subtotals as per customer

--(number of movies rented) and subtotal for each movie showing how many times that movie was rented.

CREATE OR REPLACE PROCEDURE ShowCustomerMovieRentals AS

BEGIN

-- Show the names of the customers, movie name, and the number of movies rented by them

DBMS\_OUTPUT.PUT\_LINE('Customer Name | Movie Name | Number of Rentals');

FOR rec IN (

SELECT

c.FirstName || ' ' || c.LastName AS CustomerName,

d.DvdTitle,

COUNT(r.RentalID) AS NumberOfRentals

FROM

Rental r

JOIN

Customer c ON r.CustomerId = c.CustomerId

JOIN

Dvd d ON r.DvdId = d.DvdId

GROUP BY

c.FirstName || ' ' || c.LastName, d.DvdTitle

ORDER BY

c.FirstName || ' ' || c.LastName, d.DvdTitle

) LOOP

DBMS\_OUTPUT.PUT\_LINE(rec.CustomerName || ' | ' || rec.DvdTitle || ' | ' || rec.NumberOfRentals);

END LOOP;

-- Show subtotals per customer

DBMS\_OUTPUT.PUT\_LINE('Customer Name | Total Rentals');

FOR rec IN (

SELECT

c.FirstName || ' ' || c.LastName AS CustomerName,

COUNT(r.RentalID) AS TotalRentals

FROM

Rental r

JOIN

Customer c ON r.CustomerId = c.CustomerId

GROUP BY

c.FirstName || ' ' || c.LastName

ORDER BY

c.FirstName || ' ' || c.LastName

) LOOP

DBMS\_OUTPUT.PUT\_LINE(rec.CustomerName || ' | ' || rec.TotalRentals);

END LOOP;

-- Show subtotals per movie

DBMS\_OUTPUT.PUT\_LINE('Movie Name | Total Rentals');

FOR rec IN (

SELECT

d.DvdTitle,

COUNT(r.RentalID) AS TotalRentals

FROM

Rental r

JOIN

Dvd d ON r.DvdId = d.DvdId

GROUP BY

d.DvdTitle

ORDER BY

d.DvdTitle

) LOOP

DBMS\_OUTPUT.PUT\_LINE(rec.DvdTitle || ' | ' || rec.TotalRentals);

END LOOP;

END;

/

---

----Task 2: What is the difference between DVD cost and the average cost of all the DVDs of each year.

CREATE OR REPLACE FUNCTION DvdCostDifference

RETURN SYS\_REFCURSOR AS

dvd\_cursor SYS\_REFCURSOR;

BEGIN

OPEN dvd\_cursor FOR

SELECT

DvdTitle,

DvdCost,

DvdYear,

DvdCost - AVG(DvdCost) OVER (PARTITION BY DvdYear) AS CostDifference

FROM

Dvd;

RETURN dvd\_cursor;

END;

/

----

testing the show customer movie rentals procedure:

BEGIN

ShowCustomerMovieRentals;

END;

testing the dvd cost difference function:

SET SERVEROUTPUT ON;

DECLARE

dvd\_rec SYS\_REFCURSOR;

dvd\_title Dvd.DvdTitle%TYPE;

dvd\_cost Dvd.DvdCost%TYPE;

dvd\_year Dvd.DvdYear%TYPE;

cost\_difference NUMBER;

BEGIN

dvd\_rec := DvdCostDifference;

LOOP

FETCH dvd\_rec INTO dvd\_title, dvd\_cost, dvd\_year, cost\_difference;

EXIT WHEN dvd\_rec%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Title: ' || dvd\_title || ', Year: ' || dvd\_year || ', Cost: ' || dvd\_cost || ', Difference: ' || cost\_difference);

END LOOP;

CLOSE dvd\_rec;

END;

--testing for the insert:

DECLARE

v\_CustomerId Customer.CustomerId%TYPE := &CustomerId;

v\_CustomerInitials Customer.CustomerInitials%TYPE := '&CustomerInitials';

v\_FirstName Customer.FirstName%TYPE := '&FirstName';

v\_LastName Customer.LastName%TYPE := '&LastName';

v\_PhoneNumber Customer.PhoneNumber%TYPE := '&PhoneNumber';

v\_Birthdate Customer.Birthdate%TYPE := TO\_DATE('&Birthdate', 'MM/DD/YYYY');

v\_DriverLicenseNumber Customer.DriverLicenseNumber%TYPE := '&DriverLicenseNumber';

v\_Status Customer.Status%TYPE := '&Status';

BEGIN

InsertCustomer(v\_CustomerId, v\_CustomerInitials, v\_FirstName, v\_LastName, v\_PhoneNumber, v\_Birthdate, v\_DriverLicenseNumber, v\_Status);

END;

/

SELECT \* FROM Customer;

--

```

This code is tested to run fine in SQL Developer. Any errors that stem from copying the Word document is from the auto formatting done by Word. Please use the SQL file if you can.

**Part C - Dong**

**Task 8**

Prompt: “Consider DVD table from the excel sheet. What can be the possible dimensions and facts in this table? Explain.”

Recall that dimensions and facts are terms used in Data Warehousing. Dimensions provide contextual data to the facts, and facts are quantitative data that can be measured. In short: dimensions are descriptive, facts are calculative. In the DVD table, the possible dimensions are:

* CategoryID: Identifies the category of the DVD (e.g., Action/Adventure, Biography).
* Title: The name of the DVD.
* Year: The release year of the DVD.
* Rating: The rating of the DVD (e.g., PG, R).
* ActionOnReturn: What to do when DVD is returned (e.g., Return to Shelf, Sell).

While the facts are:

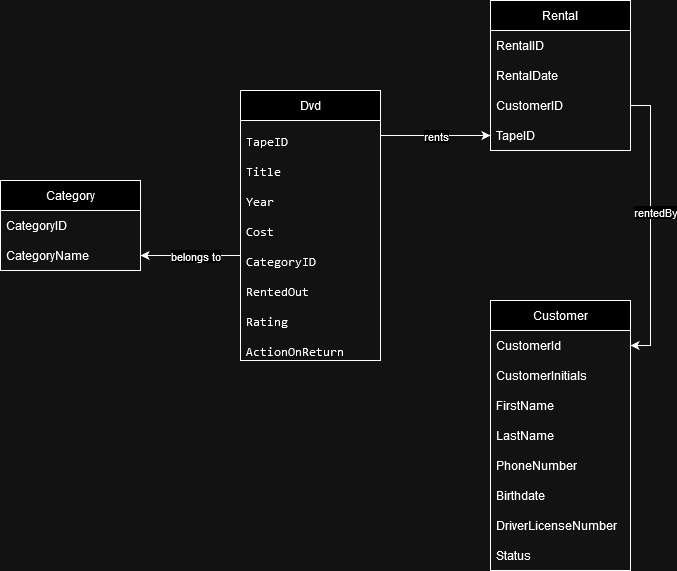
* Cost: The cost of the DVD.
* RentedOut: Indicates whether the DVD is rented out (Yes/No).

**Task 9**

Prompt: “Create a Star Schema and an ER diagram (using draw.io) out of DVD table showing Star Schema, to find out the total amount of cost for each category, each year and Rating.”

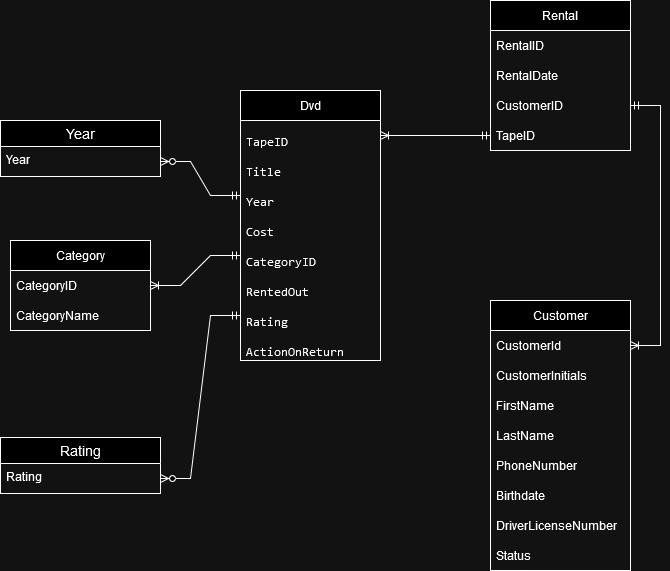
Assuming this task is referring to the Excel table rather than the SQL table, this will be the diagrams.

**Star Schema**

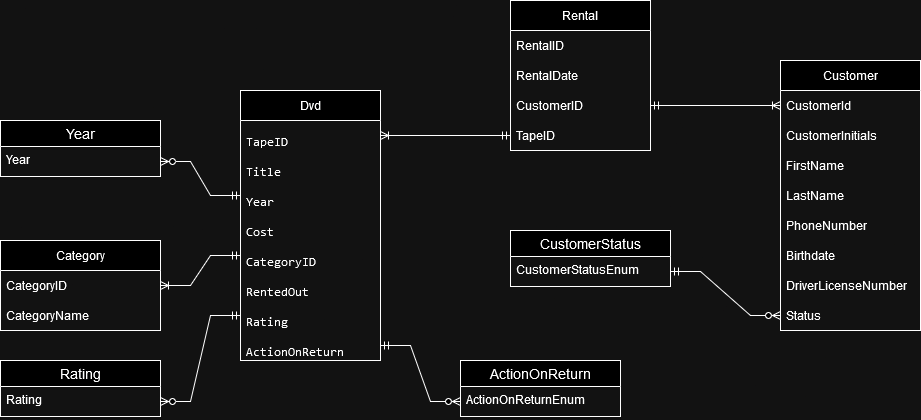


**ER diagram**

To make this ER diagram that can find out the total amount of cost for each category, each year and Rating, we split the DVD table into 3 more tables: Category, Year, Rating. Since we already have Category and Rating, we just need to draw 1 more table.



Here is the ER Diagram that includes all the tables we made (assuming we are using the modified table structure), including the enums.



**Task 10**

Prompt: “Populate each of the tables (fact and dimension) with appropriate values from the DVD table using insert and select query, manual insertion of data will not get you any marks. Note that you need to use surrogate keys for the dimension tables and use cursor to populate the values in the fact table without losing any information.”

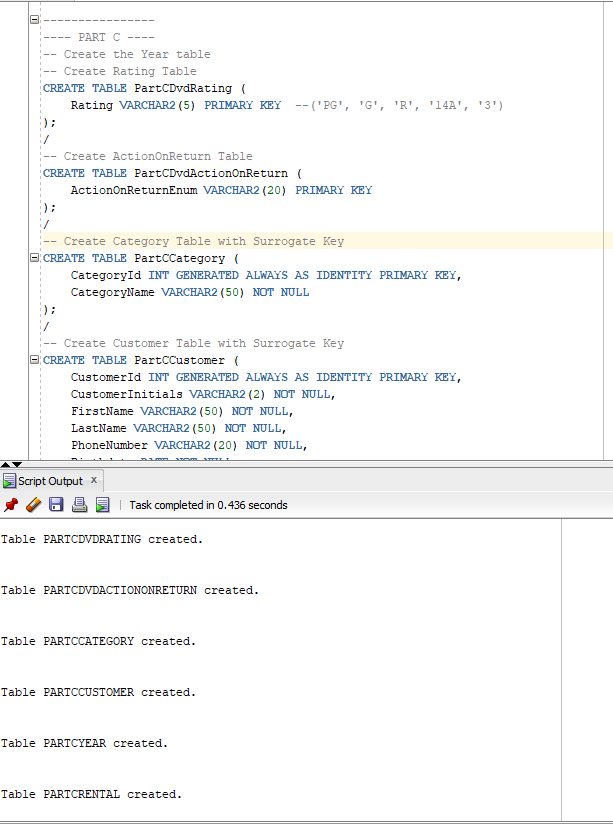
Assuming the Dvd table was inserted first, and we have all the information, we will begin as such.

1. Create new tables, in our case, we make the corresponding PartC tables
2. Insert data into these tables using queries and insert statements, no manual entry needed.
3. Use cursor to populate values in the fact table (Rental in this case).

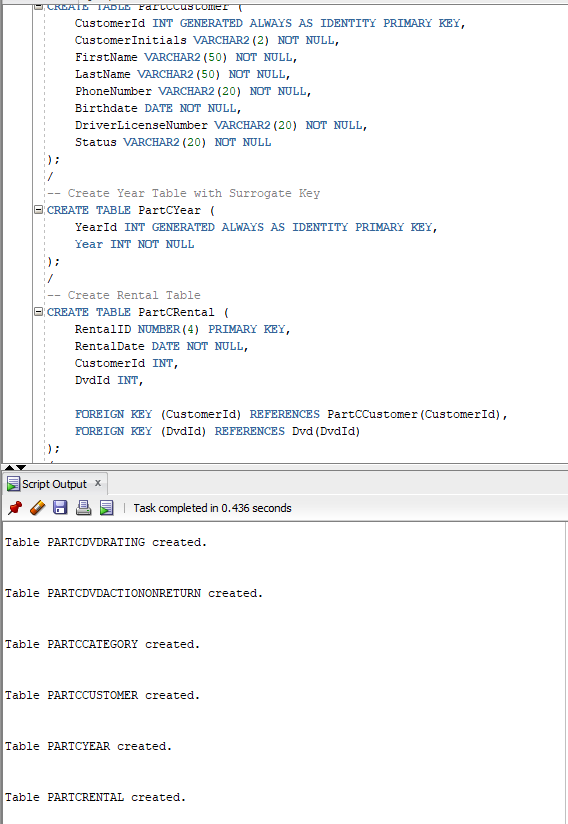
Step 1:

First 3 tables

We create these tables based on the original tables, but for the ones we split up, we will add Surrogate keys to them.

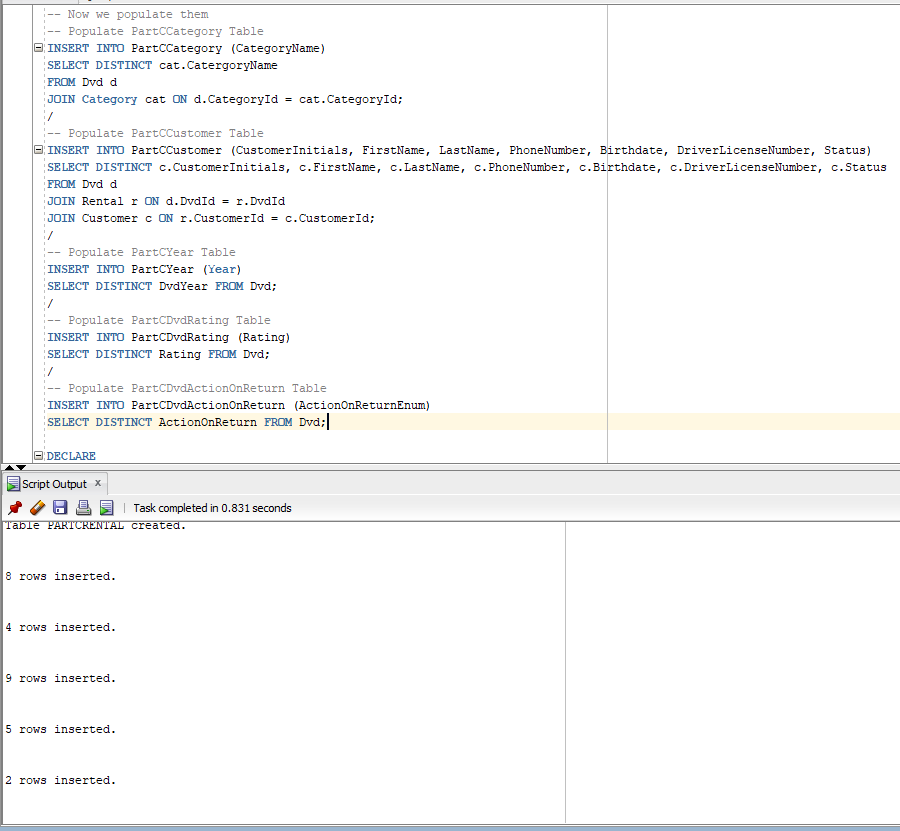


Next 3 tables

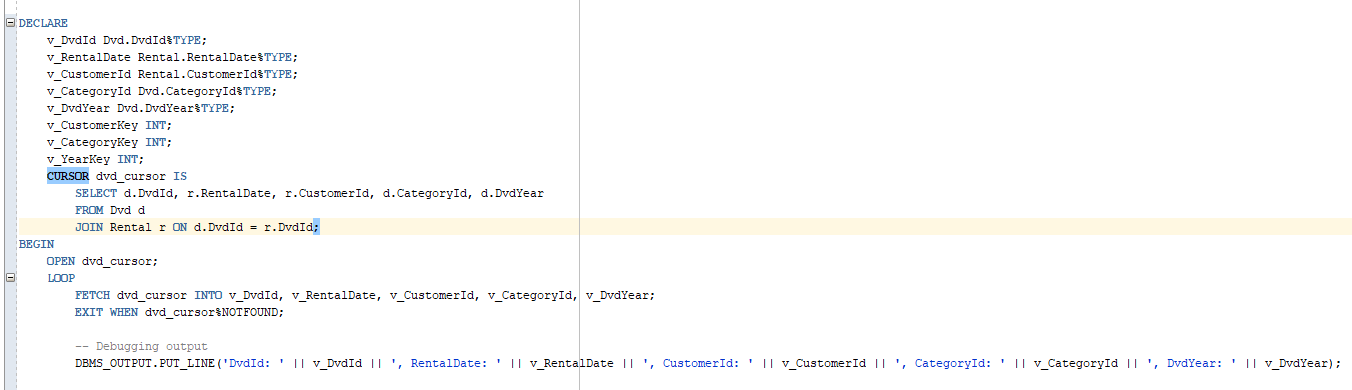
Step 2: Populate the tables

Dvd contains incomplete data. It only has a few values from other tables and for the values it does have, they are all keys. Thus, we must join tables in order to get the right data to populate our fact table. Starting with Customer.

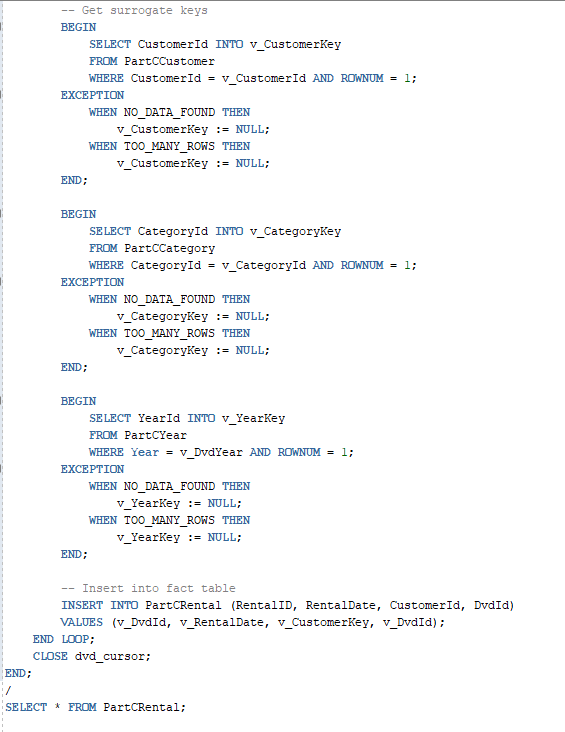
Customer table contains customer id and name. Rental contains customer id and dvd id. Thus if we want to get data from customer, we must join Dvd to Rental to Customer.

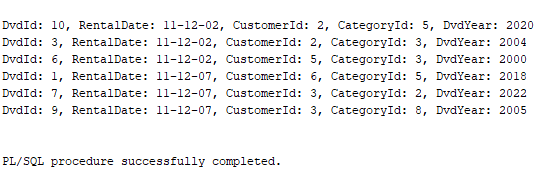
As we can see, similarly, we can do the same things to the other tables, though only Category and Customer actually have such nature. So the rest are straightforward: just insert distinct values into their respective tables.

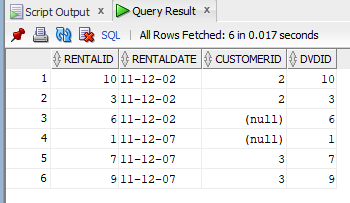
Step 3: Populate fact table using Cursor

First we declare our variables and set up a cursor. This cursor will contain all the surrogate keys within dvd joined with rentals. We loop through this cursor to get the keys for each rental. Then we insert the values we could find into the new rentals table.

This is not the most comprehensive way since the end result still returns some null values where we don’t want to.

Execution Results:

As we can see these customer IDs are null because the data we withdrew from populating the tables earlier actually don’t contain these customers within PartCCustomer. This is a potential oversight on the methodology we have chosen, or it might have been an unintentional side effect of our table creation method.

In whichever case, the procedure does work (though a bit wrong), and the population of the tables are also working.