

ASSIGNMENT COVER SHEET

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Course Code and Description | UECS 3203 ADVANCED DATABASE SYSTEMS | | | |
| Lecturer Name | Dr. Sugumaran a/l Nallusamy | | | |
| Assignment Title | Assignment 1 | | | |
| **DECLARATION** | | | | |
| We declare that this is a group assignment and that no part of this submission has been copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part been written for us by another person. | | | | |
| Programme | | Student ID Numbers | Student Names | Practical  Group |
| SE | | 2002759 | AVEN DING XIAN KAI | 1 |
| SE | | 2400056 | CHOO WEI XIANG | 1 |
| SE | | 2301374 | SIA LI ZE | 1 |
| SE | | 2003632 | TH’NG ZI QIN | 1 |

Table of Contents

[CHAPTER 1 DATA SELECTION 3](#_Toc175157247)

[CHAPTER 2 DATA IMPORT 4](#_Toc175157248)

[CHAPTER 3 TABLE CREATION & INDEXES 5](#_Toc175157249)

[3.1 Table Structure 5](#_Toc175157250)

[3.2 Indexes Suggestion 6](#_Toc175157251)

[CHAPTER 4 CRUD OPERATIONS 7](#_Toc175157252)

[4.1 Create 7](#_Toc175157253)

[4.2 Read 7](#_Toc175157254)

[4.3 Update 7](#_Toc175157255)

[4.4 Delete 7](#_Toc175157256)

[CHAPTER 5 TRANSACTION MANAGEMENT 8](#_Toc175157257)

[CHAPTER 6 TRIGGERS 9](#_Toc175157258)

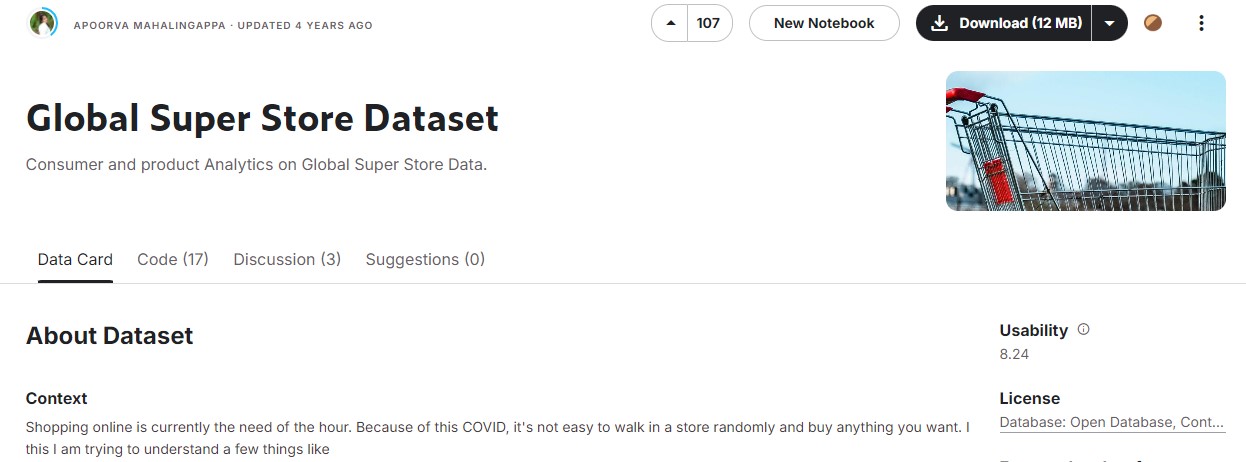
[CHAPTER 7 DATA ANALYSIS 10](#_Toc175157259)

[CHAPTER 8 REFERENCES 11](#_Toc175157260)

# DATA SELECTION

The Global Superstore Dataset was chosen by our group from Kaggle for this assignment. The dataset includes information on detailed order records, shipping details, customer profiles, market research, and product details. The data is contained in a single CSV file and is organized into 51,291 rows, each with 23 attributes.

The reason we decided on this dataset was that every single one of the columns had complete and rich data. The dataset is also appropriate from a variety of analytical angles because it provides a broad range of attributes. This versatility helps us achieve our goals for data analysis by enabling us to extract valuable insights from a variety of perspectives. This dataset is a great place for novices to start learning about e-commerce analytics in the field of data science. It offers lots of chances to extract substantial.



*Figure 1.1: Global Super Store Dataset* (*Global Super Store Dataset*, n.d.)

# DATA IMPORT

In this task, we have implemented PL/SQL programs to fulfill the needs of importing data from a CSV file into our Oracle database. These programs include a few steps which are to accept path and filename from a user, validate the existence and format of the CSV file, and import data with error handling.

## Request the path and filename

The user is first prompted to enter the path and filename of the CSV file that needs to be imported in this step. The file\_path and file\_name will be assigned to users once they have entered the path and name. To enable the database to access the path, a directory called my\_dir will be created in the database using the file\_path. Invokes the function validate\_file\_path(file\_name) to verify the validity and accessibility of the file.

SET SERVEROUTPUT ON

    ACCEPT file\_path char PROMPT "Enter File Path: ";

    ACCEPT file\_name char PROMPT "Enter Filename: ";

DECLARE

    file\_path VARCHAR2(255);

    file\_name VARCHAR2(255);

    result\_import VARCHAR2(100);

BEGIN

    file\_path := '&file\_path';  -- Prompt for input

    file\_name := '&file\_name';  -- Prompt for input

    -- Create the directory

    EXECUTE IMMEDIATE 'CREATE OR REPLACE DIRECTORY my\_dir AS '''

                      || file\_path

                      || '''';

    dbms\_output.put\_line('Directory my\_dir created successfully.');

    -- Validate the file

    IF validate\_file\_path(file\_name) THEN

        dbms\_output.put\_line('File is valid and accessible.');

        result\_import:= process\_csv\_file(file\_name);

        dbms\_output.put\_line(result\_import);

    ELSE

        dbms\_output.put\_line('File is invalid or inaccessible.');

    END IF;

EXCEPTION

    WHEN OTHERS THEN

        dbms\_output.put\_line('Error creating directory: ' || sqlerrm);

END;

/

## Validate File

The filename will be passed to a validate\_file\_path function to verify that the file exists and can be accessed before importing the data form. By running the *validate\_file\_path* function, the existence and accessibility of a CSV file from the input of the user will be verified. By reference to Oracle *UTL\_FILE* documentation (Kannan et al., 2024), this validation can be done by *UTL\_FILE.FOPEN* method. If the CSV file opens successfully, it will return true back. However, if any errors occur during this process such as an invalid path or operation, the exception function will log out to the user and return false. This can ensure that any issues that happen during validation can be caught by the system and feedback to the user.

create or replace FUNCTION validate\_file\_path(p\_file\_name IN VARCHAR2)

RETURN BOOLEAN

IS

    csv\_file    UTL\_FILE.FILE\_TYPE;

BEGIN

--    DBMS\_OUTPUT.PUT\_LINE(p\_file\_path);

    csv\_file := UTL\_FILE.FOPEN('MY\_DIR', p\_file\_name, 'R');

    UTL\_FILE.FCLOSE(csv\_file);

    RETURN TRUE;

EXCEPTION

    WHEN UTL\_FILE.INVALID\_PATH THEN

        DBMS\_OUTPUT.PUT\_LINE('Invalid directory path: MY\_DIR');

        RETURN FALSE;

    WHEN UTL\_FILE.INVALID\_MODE THEN

        DBMS\_OUTPUT.PUT\_LINE('Invalid file open mode.');

        RETURN FALSE;

    WHEN UTL\_FILE.INVALID\_OPERATION THEN

        DBMS\_OUTPUT.PUT\_LINE('Invalid operation.');

        RETURN FALSE;

    WHEN UTL\_FILE.READ\_ERROR THEN

        DBMS\_OUTPUT.PUT\_LINE('Read error.');

        RETURN FALSE;

    WHEN UTL\_FILE.ACCESS\_DENIED THEN

        DBMS\_OUTPUT.PUT\_LINE('Access denied.');

        RETURN FALSE;

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

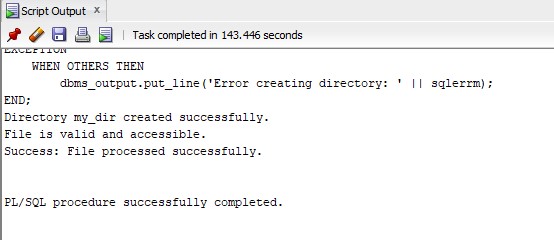
        RETURN FALSE;

END validate\_file\_path;

## Read & Import Data from File

Import the data using the process\_csv\_file function into the database if the file is valid. Data processing and import into database tables from a CSV file is the purpose of this function. The CSV file input by the user will be opened from the directory *my\_dir* and skip the header line. It will read each line from the CSV file and split the data by using the Oracle regular expression (12 Using Regular Expressions With Oracle Database, n.d.). One row of data will be split and inserted into four tables which are customer, shipping, product, and *orderitem* tables. Before inserting the data into the table, it will check the existing data with each id, *orderitem* will check with the order id and product id because one order may have multiple products.

During the operation to get file data and insert data into the table, a save point *savepoint\_before\_line* will be recorded and the errors will be handled by the exception function. It will give feedback to the user by logging out the error message for the user and roll back to the *savepoint\_before\_line*. Once the error for NOT\_DATA\_FOUND happens in the get line from the CSV file, the loop will end since it means the CSV file completes reading by function. It will return a success message to the user.

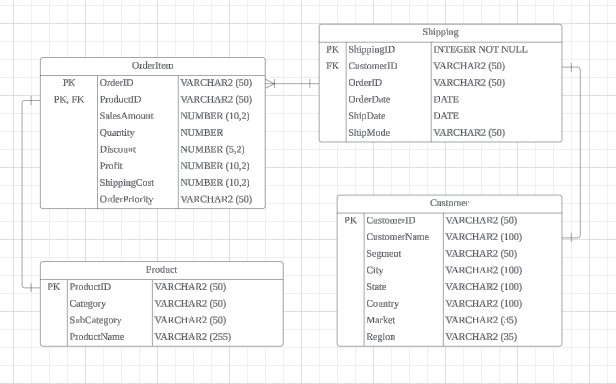


*Figure 2.1: Data Import to Tables*

# TABLE CREATION & INDEXES

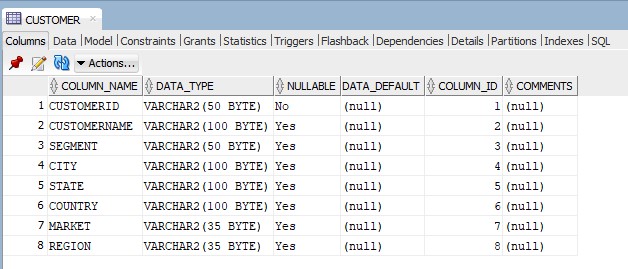
## Table Structure

The subsequent step after importing the dataset is to normalize the data into several tables to minimize redundancy and improve the efficiency of the analysis. The dataset was divided into four important tables, which we designated Customer, Shipping, Product, and OrderItem.



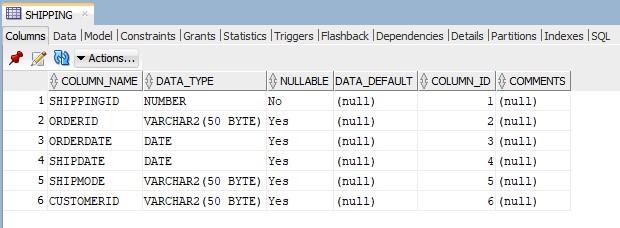
*Figure 3.1: Tables Structure*

The Customer table contains customer-related information, including CustomerID as the primary key, and fields such as CustomerName, Segment, City, State, Country, Market, and Region. This table stores all the necessary details about each customer, ensuring that customer data is organized and easily accessible.



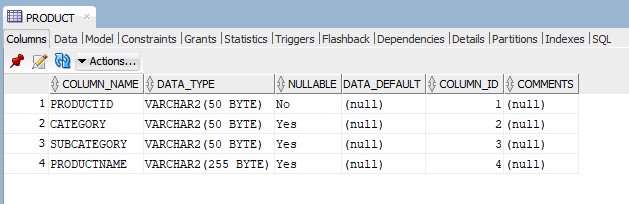
*Figure 3.2: Customer Table*

Details about each order's shipment are recorded in the Shipping table. It establishes a parent-child relationship between the two tables using ShippingID as the primary key and foreign key references to the CustomerID from the Customer table. It is simple to follow the logistics of each order with the help of this table, which also includes information like the OrderID, OrderDate, ShipDate, and ShipMode. Since a single shipment can cover multiple orders, OrderID is inappropriate for use as a foreign key in the Shipping table. It would be misleading to indicate that each shipment is associated with a single order if OrderID were a foreign key. In the OrderItem table, orders are tracked individually, but ShippingID continues to be the primary key in the Shipping table. By avoiding redundancy and keeping a flexible system that accurately reflects the many-to-one relationship between shipments and orders, this structure enables us to track shipments independently of specific orders.



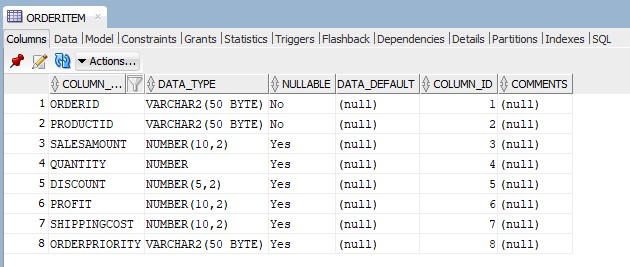
*Figure 3.3: Shipping Table*

The Product table stores data about the products, with ProductID as the primary key. It categorizes products into categories and subcategories through fields like Category and SubCategory, including the ProductName. This table provides a structure for managing product information independently from customer and order data.



*Figure 3.4: Product Table*

Last but not least, every single order's specific products are stored in the OrderItem table. OrderID and ProductID make up its composite primary key, which guarantees the uniqueness of every order and product combination. Important metrics like SalesAmount, Quantity, Discount, Profit, ShippingCost, and OrderPriority are included in this table. Each order item is linked to its corresponding product through a foreign key reference from the OrderItem table to the Product table via ProductID.



*Figure 3.5: OrderItem Table*

## Indexes Suggestion

1. Index on CustomerID in the Shipping table (idx\_shipping\_customerid)

CustomerID column in the Shipping table act as a foreign key from the Customer table that connects each shipping record to a customer. The efficiency of queries that filter or join the Shipping table based on CustomerID can be increased by indexing on CustomerID. This is especially helpful for tasks that include merging shipment records with customer information or searching through all shipments for a particular customer. These operations would need a full table scan, which can be time-consuming, particularly for large datasets, if there was no index.

1. Index on CustomerName in the Customer table (idx\_customer\_name)

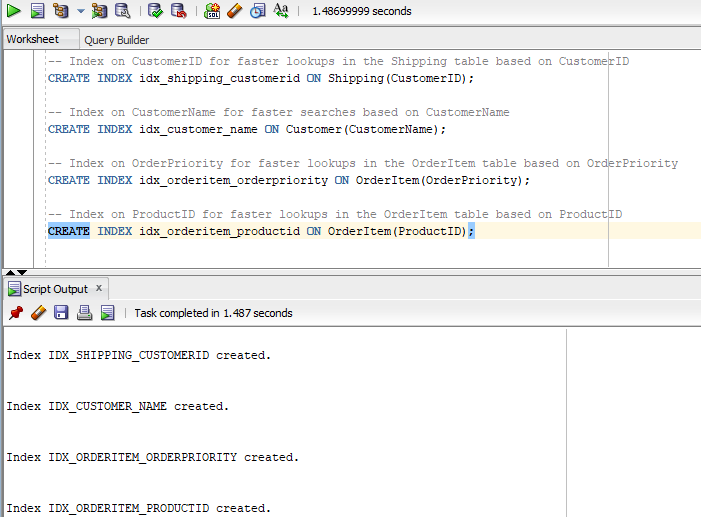
The CustomerName column in the Customer table is often used in queries for searching, filtering, or reporting purposes. Operations that involve looking up customers by name perform better when CustomerName is indexed. This index, as opposed to a full table scan, guarantees that operations such as finding a customer by name or generating reports that filter customers by name are completed more quickly if you regularly need to do so.

1. Index on OrderPriority in the OrderItem table (idx\_orderitem\_orderpriority)

Orders can be filtered or sorted according to their priority level using the OrderPriority column in the OrderItem table. The performance of queries involving filtering or sorting according to order priority is enhanced by creating an index in this column. This index is useful for operations that need to quickly retrieve orders with a certain priority level or for creating reports or analytics that classify orders according to their priority.

1. Index on ProductID in the OrderItem table (idx\_orderitem\_productid)

Each order item in the OrderItem table is associated with a unique product in the Product table via a foreign key found in the ProductID column. Queries that filter order items based on the product or join OrderItem with Product are sped up by indexing ProductID. When you have to look up every order item connected to a specific product or carry out operations involving product details, this is essential for performance.



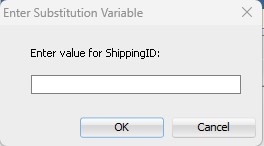
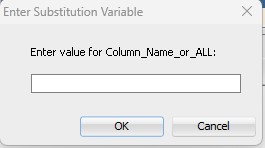
*Figure 3.6: Indexes Creation*

# CRUD OPERATIONS

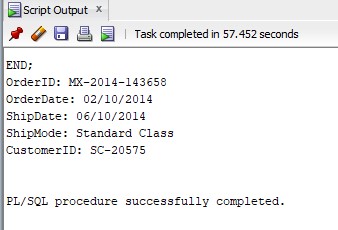
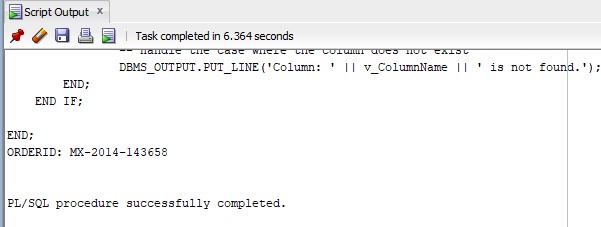
## Create

## Read

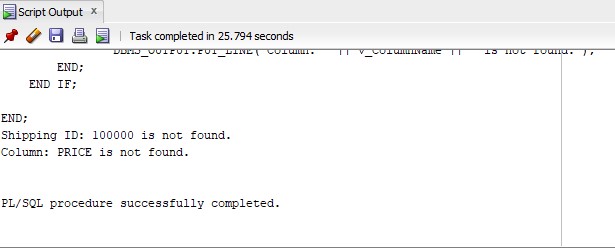
This PL/SQL script enables users to obtain the value of a single column or an entire row by providing the ShippingID. The script begins by verifying the ShippingID entered to make sure it exists in the table. The user has the option to obtain a single column or the complete row of data if the ShippingID is valid. The script uses dynamic SQL to select and display the value of a specified column when that column is requested. The script retrieves and displays all relevant columns for the specified ShippingID, such as OrderID, OrderDate, ShipDate, ShipMode, and CustomerID, if the user chooses to retrieve the entire row. To handle situations in which the ShippingID is invalid or the specified column name is incorrect, the script also includes error handling. When this happens, relevant error messages are shown to users so they are aware of any problems that may have arisen while retrieving data.

*Figure 3.7: Request Shipping ID Figure 3.8: Request Column name*



*Figure 3.9: Retrieve Specific Column Figure 3.10: Retrieve All Column*



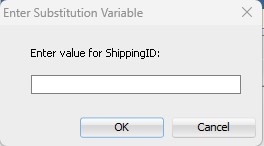
*Figure 3.11: Error Handling for Read*

## Update

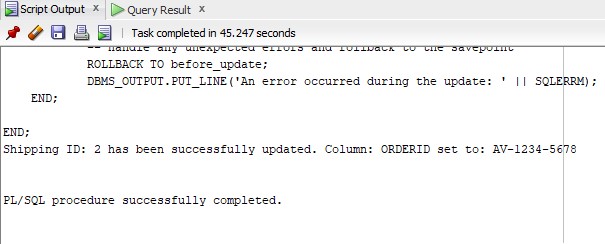
This PL/SQL script is designed to update a specific column in a record within the Shipping table using user input. Its error handling and validation capabilities are robust. Firstly, the script requests that the user enter the ShippingID. It then checks to see if the ShippingID is present in the Shipping table. If the ShippingID cannot be found, the processing is halted, rolling back to a savepoint and displaying the appropriate error message.

Following that, the user is prompted to enter the column name that requires updating. The script checks the Shipping table to see if this column exists using an ALL\_TAB\_COLUMNS view query. The script rolls back to the savepoint and notifies the user if there are any unexpected errors during this validation if the column is missing.

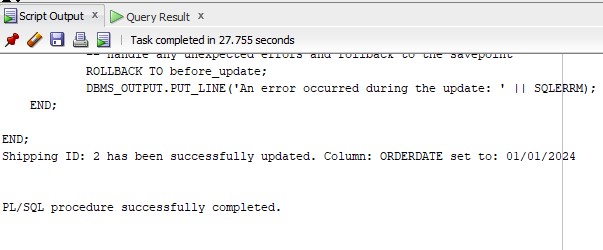
The script asks the user to enter the new value after validating the column name and the ShippingID. It formats the input in date-type columns to the correct format. The script uses dynamic SQL to update the designated column for the record that the ShippingID identifies with the new value. After verifying that the update was successful, it commits the transaction to save the modifications. If there are any problems with the update, the script rolls back to the last savepoint and displays a clear error message. By allowing errors to be handled gracefully and keeping the state constant throughout execution, this method guarantees data integrity.



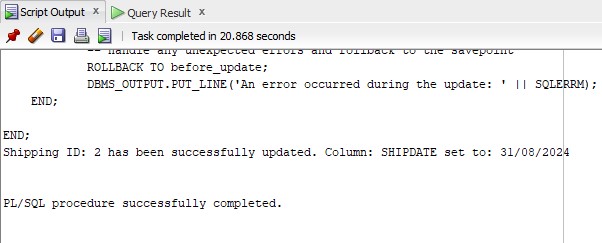
*Figure 3.11: Request Shipping ID*



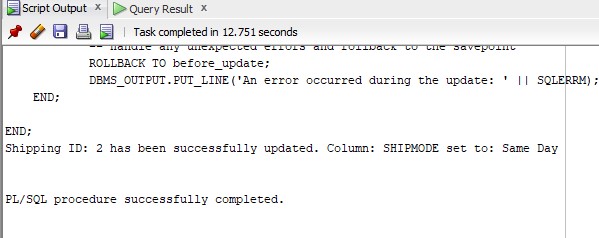
*Figure 3.11: Successfully Updated OrderID*



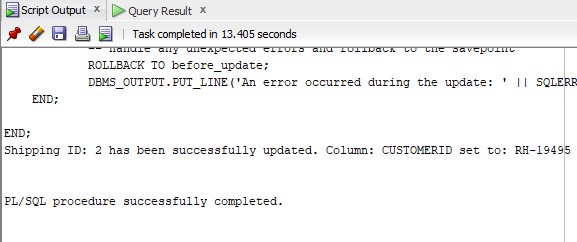
*Figure 3.11: Successfully Updated OrderDate*



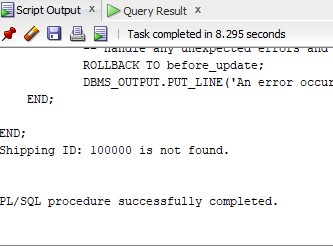
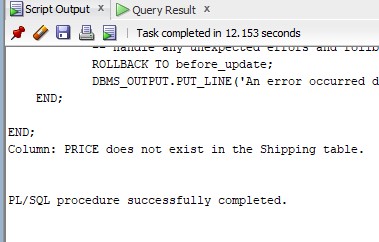
*Figure 3.11: Successfully Updated ShipID*



*Figure 3.11: Successfully Updated ShipMode*



*Figure 3.11: Successfully Updated CustomerID*

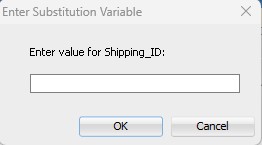
 

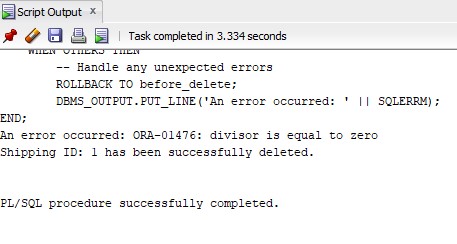
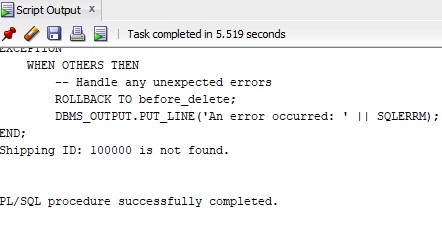
*Figure 3.11: ShippingID Not Found Figure 3.11: Column Not Found*

## Delete

This PL/SQL script is designed to safely delete a record from the Shipping table based on a given ShippingID. Firstly, the script counts the number of records that have the specified ShippingID to determine if it is present in the Shipping table. The script rolls back to a predetermined savepoint and ends, displaying a message stating that the ShippingID was not found, if no records are found (v\_Count = 0).

The script creates a savepoint called before\_delete if the ShippingID is found, allowing any changes to be undone if needed. If there isn't a problem, the script removes the record from the Shipping table that has the given ShippingID. Once the record has been successfully deleted, the transaction is committed, making the change permanent and confirming the deletion with a message.



*Figure 3.12: Request Shipping ID*  

*Figure 3.13: Successfully Deleted Figure 3.14: Error Handling for Delete*

# TRANSACTION MANAGEMENT

In this assignment, we have included strong error-handling capabilities for insert, update, and delete operations by utilizing the SAVEPOINT and ROLLBACK functions. Every operation starts with the setting of the SAVEPOINT, which creates a point at which, in the event of an error, the transaction can be rolled back.

If something goes wrong, the system will quickly perform ROLLBACK to the SAVEPOINT that was previously set and will produce an error message. Statements to insert, update, or delete are performed; the outcome is only committed to permanent storage when it is error-free.

By doing so, any changes made during the transaction up to the SAVEPOINT are undone and consistency in the database is maintained. This method allows recovery from unforeseen problems that may arise when making database modifications, safeguards against incorrect or incomplete updates, and preserves data integrity.

# TRIGGERS

# DATA ANALYSIS

In data analysis, we use the PL/SQL queries to do aggregations, filter, sort, and join the database tables to carry out a report for users. There are 5 procedures we made for analyzing the data in database tables which are sales report for monthly or yearly, sales by market report, top 5 sales products in each market, total sales in each shipping mode report, and end of today report.

## Sales Report for Monthly or Yearly

The purpose of this procedure is to show users the total sales with the options for each month or year. This enables users to analyze sales performance over different periods and make informed business decisions.

### Sales Report for Monthly or Yearly

The type of range for the sales report must be entered by the user when the SQL query is launched. For a month-by-month sales report, the user can enter "A" or "Monthly," and for a year-by-year sales report, "B" or "Yearly."

# REFERENCES

* *Global Super Store Dataset*. (n.d.). Www.kaggle.com. <https://www.kaggle.com/datasets/apoorvaappz/global-super-store-dataset>
* Kannan, P., Morin, L., Raphaely, D., Ashdown, L., Baker, D., Carver, D., Chaliha, M., Cheng, B., Day, R., Fogel, S., Llewellyn, B., Lane, P., McDermid, D., Morales, T., Murphy, A., Murray, C., Pelski, S., Rich, K., Romero, A., . . . Chaudhry, A. (2024, July 29). UTL\_FILE. Oracle Help Center. <https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/UTL_FILE.html>
* 12 using regular expressions with Oracle database. (n.d.). <https://docs.oracle.com/cd/B12037_01/appdev.101/b10795/adfns_re.htm>
* *Oracle Live SQL - Script: REGEXP\_LIKE- examples*. (n.d.). <https://livesql.oracle.com/apex/livesql/file/content_BCIYBOLU3HSIRATHWNFKWHXIM.html>

‌