



## ASSIGNMENT COVER SHEET

Course Code and Description	UECS 3203 ADVANCED DATABASE SYSTEMS		
Lecturer Name	Dr. Sugumaran a/l Nallusamy		
Assignment Title	Assignment 1		
<b>DECLARATION</b>			
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

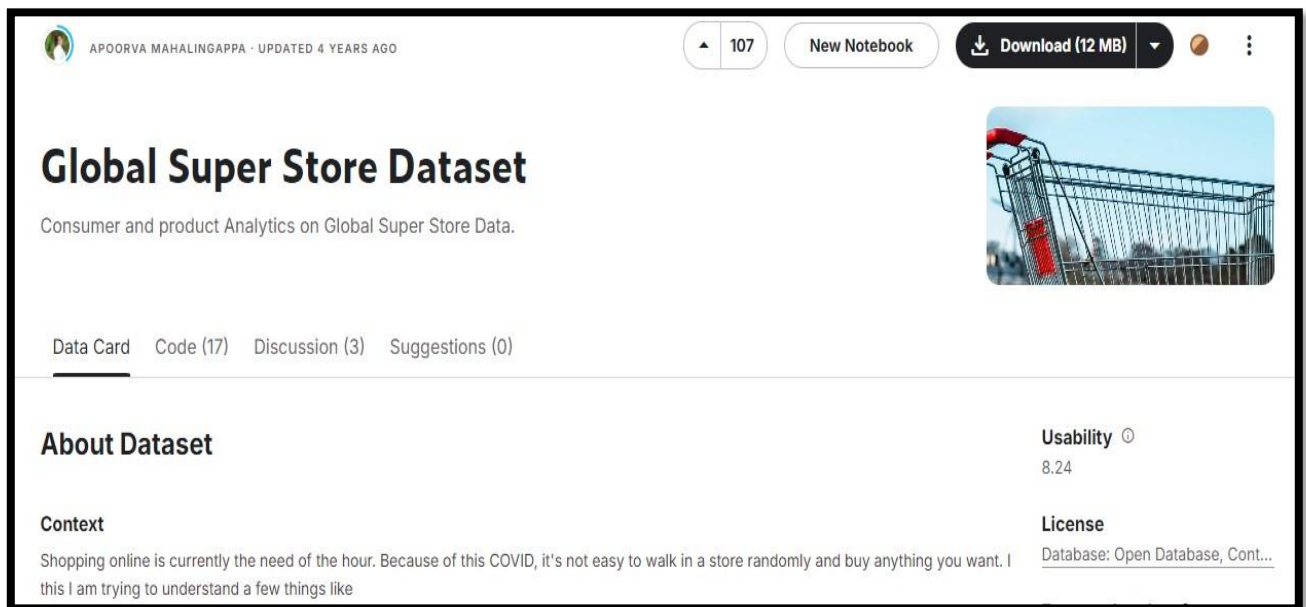
CHAPTER 1	DATA SELECTION .....	4
CHAPTER 2	DATA IMPORT .....	5
2.1	Request the path and filename .....	5
2.2	Validate File .....	6
2.3	Read & Import Data from File .....	7
CHAPTER 3	TABLE CREATION & INDEXES .....	8
3.1	Table Structure .....	8
3.2	Indexes Suggestion .....	10
CHAPTER 4	CRUD OPERATIONS .....	12
4.1	Create .....	12
4.2	Read .....	20
4.3	Update .....	22
4.4	Delete .....	26
CHAPTER 5	TRANSACTION MANAGEMENT .....	28
CHAPTER 6	TRIGGERS .....	29
6.1	Shipping_Audit_Table .....	29
6.2	Shipping_Audit_Sequence .....	29
6.3	Shipping_Audit_ID_Trigger .....	29
6.4	Shipping_Audit_Sequence .....	30
6.5	Shipping_After_Insert_Update .....	30
6.6	Shipping_Before_Input_Update .....	31
6.7	OrderItem_Before_Input_Update .....	32
6.8	Shipping_Before_Delete .....	32
6.9	Output: Shipping_Audit .....	33
CHAPTER 7	DATA ANALYSIS .....	34
7.1	Sales Report for Monthly or Yearly .....	34
7.1.1	Sales Report for Monthly or Yearly .....	34
7.1.2	Procedure to Generate Report .....	35
7.1.3	Sample Outputs .....	36
7.2	Sales by Market Report .....	37
7.2.1	Execution Way .....	37
7.2.2	Procedure .....	37

7.2.3	Sample Outputs .....	38
7.3	Top 5 Sales Products in Each Market .....	39
7.3.1	User Input Interface .....	39
7.3.2	Procedure .....	39
7.3.3	Sample Outputs .....	42
7.4	Total Sales in Each Shipping Mode Report .....	43
7.4.1	Way to Execute .....	43
7.4.2	Procedure .....	43
7.4.3	Sample Outputs .....	44
7.5	End of Report .....	44
7.5.1	User Input Interface .....	45
7.5.2	Procedure .....	45
7.5.3	Sample Outputs .....	48
CHAPTER 8	REFERENCES .....	50

## CHAPTER 1 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.)*

## CHAPTER 2 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.

### 2.1 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;
```

```
/
```

## 2.2 Validate File

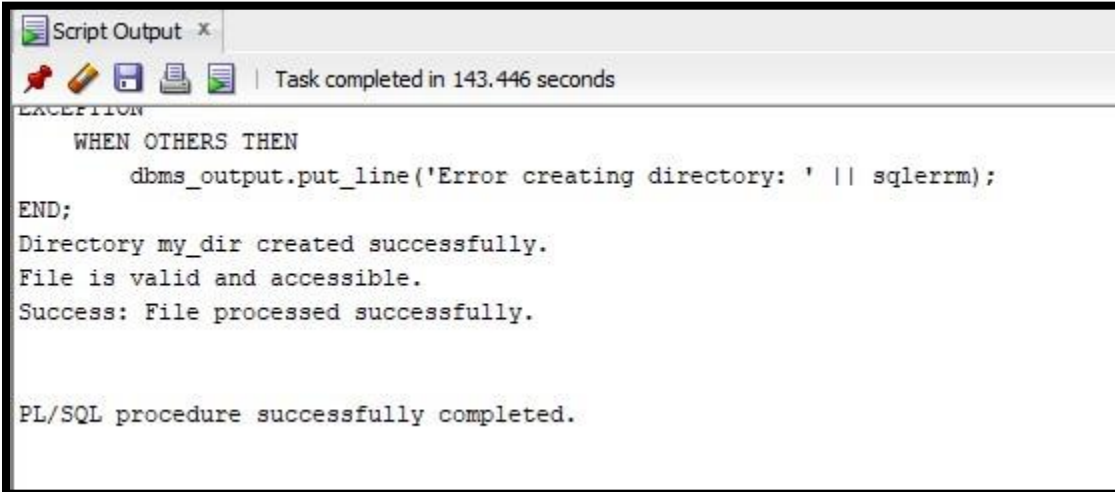
The filename will be passed to a `validate_file_path` function to verify that the file exists and can be accessed before importing the data form. By running the `validate_file_path` function, the existence and accessibility of a CSV file from the input of the user will be verified. By reference to Oracle `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;
```

### 2.3 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.



```
Script Output x
Task completed in 143.446 seconds

EXCEPTION
  WHEN OTHERS THEN
    dbms_output.put_line('Error creating directory: ' || sqlerrm);
END;
Directory my_dir created successfully.
File is valid and accessible.
Success: File processed successfully.

PL/SQL procedure successfully completed.
```

Figure 2.1: Data Import to Tables

## CHAPTER 3 TABLE CREATION & INDEXES

### 3.1 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.

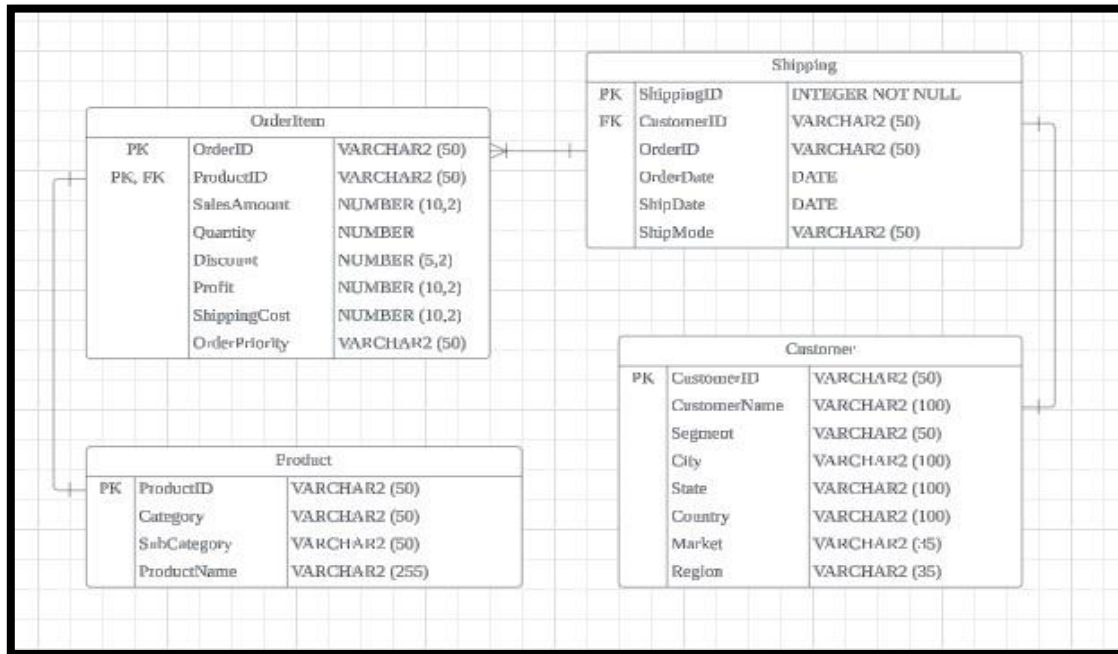


Figure 3.1: Tables Structure

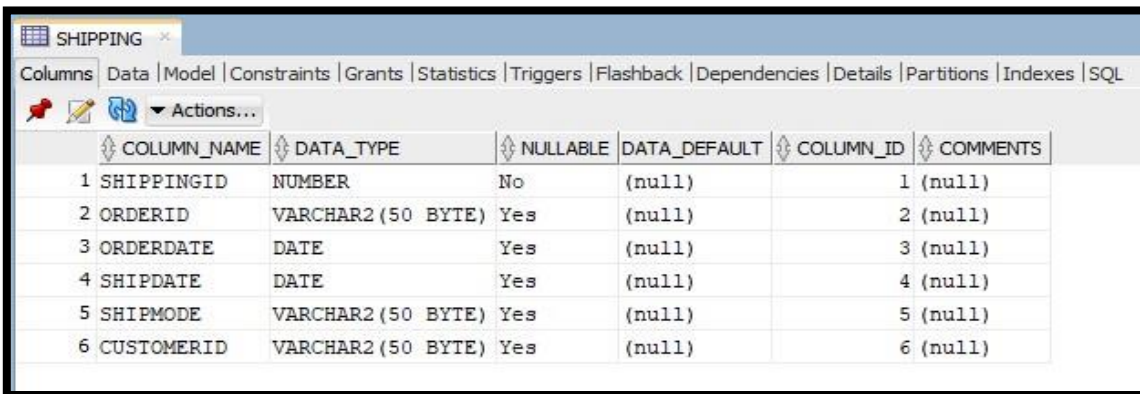
The Customer table contains customer-related information, including CustomerID as the primary key, and fields such as CustomerName, Segment, City, State, Country, Market, and Region. This table stores all the necessary details about each customer, ensuring that customer data is organized and easily accessible.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CUSTOMERID	VARCHAR2 (50 BYTE)	No	(null)	1 (null)	
2	CUSTOMERNAME	VARCHAR2 (100 BYTE)	Yes	(null)	2 (null)	
3	SEGMENT	VARCHAR2 (50 BYTE)	Yes	(null)	3 (null)	
4	CITY	VARCHAR2 (100 BYTE)	Yes	(null)	4 (null)	
5	STATE	VARCHAR2 (100 BYTE)	Yes	(null)	5 (null)	
6	COUNTRY	VARCHAR2 (100 BYTE)	Yes	(null)	6 (null)	
7	MARKET	VARCHAR2 (35 BYTE)	Yes	(null)	7 (null)	
8	REGION	VARCHAR2 (35 BYTE)	Yes	(null)	8 (null)	

Figure 3.2: Customer Table



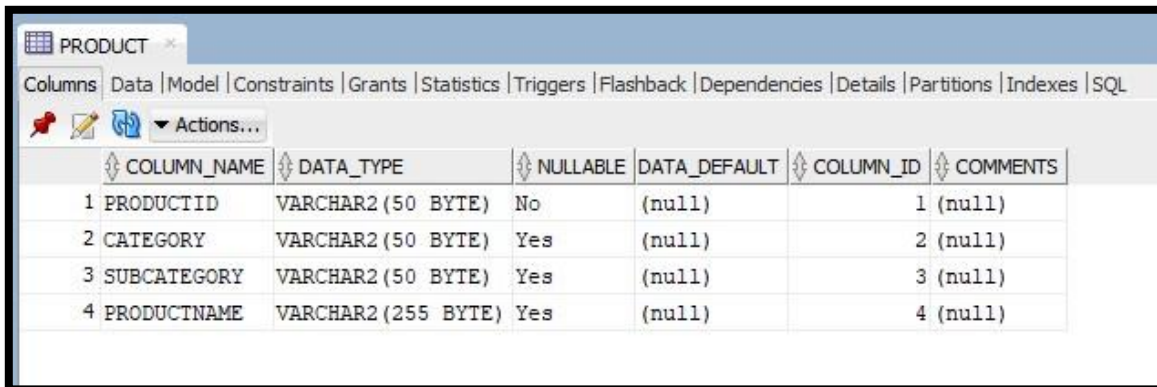
Details about each order's shipment are recorded in the Shipping table. It establishes a parent-child relationship between the two tables using ShippingID as the primary key and foreign key references to the CustomerID from the Customer table. It is simple to follow the logistics of each order with the 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.



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	SHIPPINGID	NUMBER	No	{null}	1	{null}
2	ORDERID	VARCHAR2 (50 BYTE)	Yes	{null}	2	{null}
3	ORDERDATE	DATE	Yes	{null}	3	{null}
4	SHIPDATE	DATE	Yes	{null}	4	{null}
5	SHIPMODE	VARCHAR2 (50 BYTE)	Yes	{null}	5	{null}
6	CUSTOMERID	VARCHAR2 (50 BYTE)	Yes	{null}	6	{null}

*Figure 3.3: Shipping Table*

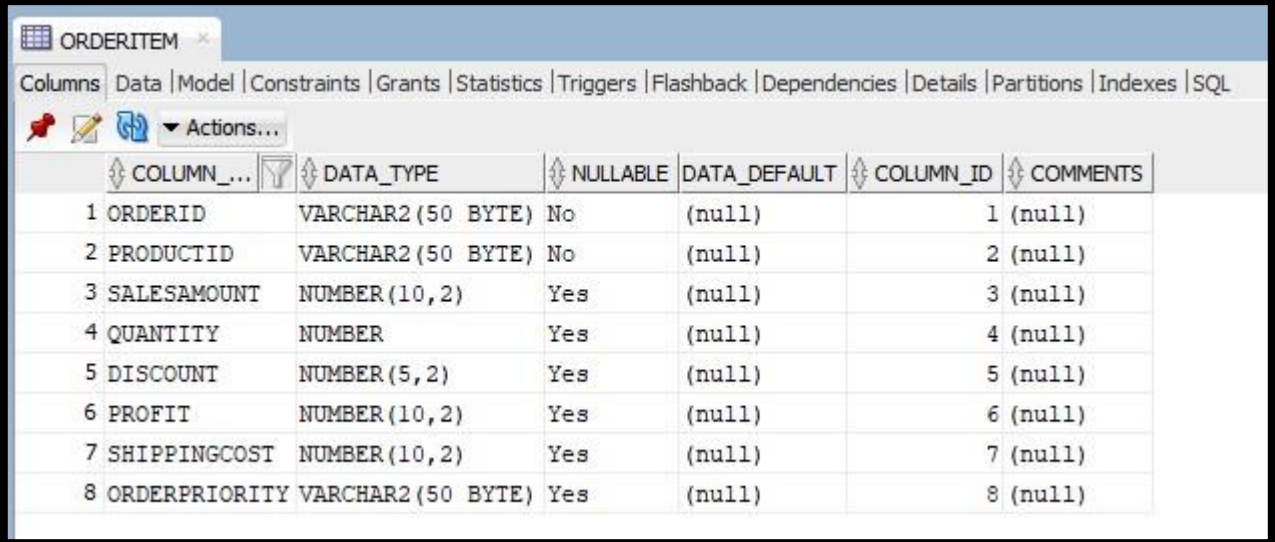
The Product table stores data about the products, with ProductID as the primary key. It categorizes products into categories and subcategories through fields like Category and SubCategory, including the ProductName. This table provides a structure for managing product information independently from customer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



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PRODUCTID	VARCHAR2 (50 BYTE)	No	{null}	1	{null}
2	CATEGORY	VARCHAR2 (50 BYTE)	Yes	{null}	2	{null}
3	SUBCATEGORY	VARCHAR2 (50 BYTE)	Yes	{null}	3	{null}
4	PRODUCTNAME	VARCHAR2 (255 BYTE)	Yes	{null}	4	{null}

*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.



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ORDERID	VARCHAR2(50 BYTE)	No	(null)	1	(null)
2	PRODUCTID	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	SALESAMOUNT	NUMBER(10,2)	Yes	(null)	3	(null)
4	QUANTITY	NUMBER	Yes	(null)	4	(null)
5	DISCOUNT	NUMBER(5,2)	Yes	(null)	5	(null)
6	PROFIT	NUMBER(10,2)	Yes	(null)	6	(null)
7	SHIPPINGCOST	NUMBER(10,2)	Yes	(null)	7	(null)
8	ORDERPRIORITY	VARCHAR2(50 BYTE)	Yes	(null)	8	(null)

*Figure 3.5: OrderItem Table*

### 3.2 Indexes Suggestion

- i. 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.

- ii. 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.

- iii. 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.

iv. 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.

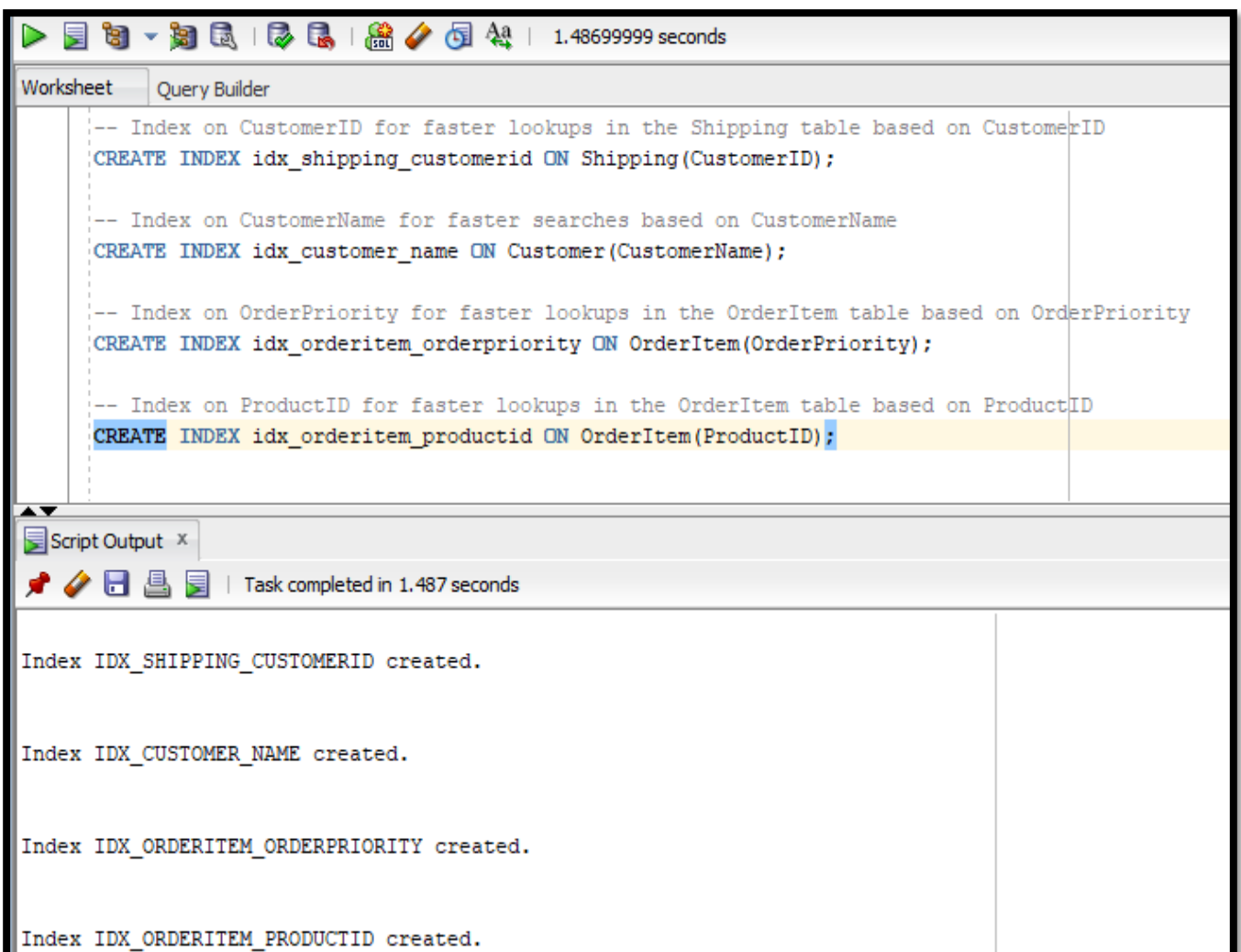
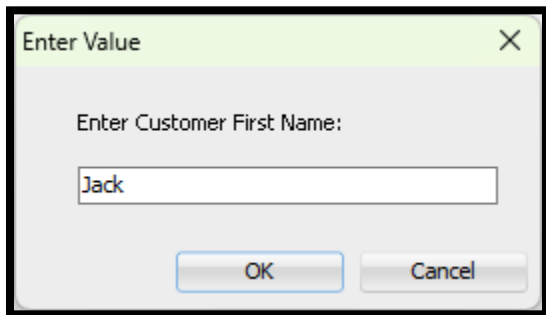


Figure 3.6: Indexes Creation

## CHAPTER 4 CRUD OPERATIONS

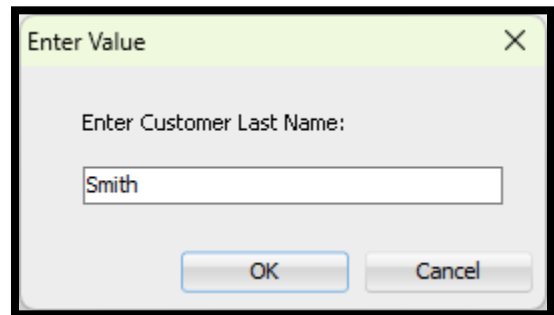
### 4.1 Create

This PL/SQL script allow user to enter new value into customer, shipping, product and orderItem table. The script will start prompt user to enter user's First Name, Last Name, Segment, City, State, Country, Market, and Region. The figures below show the scripts prompt user to enter the relative values.



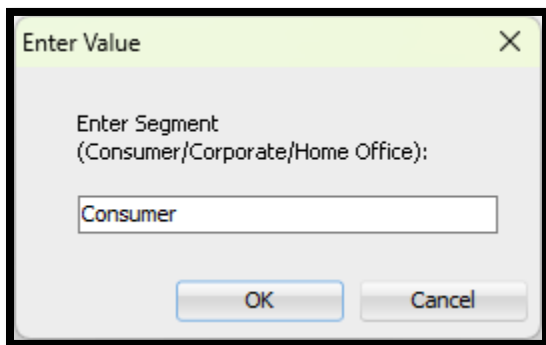
A screenshot of a 'Enter Value' dialog box. The title bar is green with a close button. The main area is light gray. The text 'Enter Customer First Name:' is displayed above a text input field containing the text 'Jack'. At the bottom, there are two buttons: 'OK' and 'Cancel'.

*Figure 4.1: Request User's First Name*



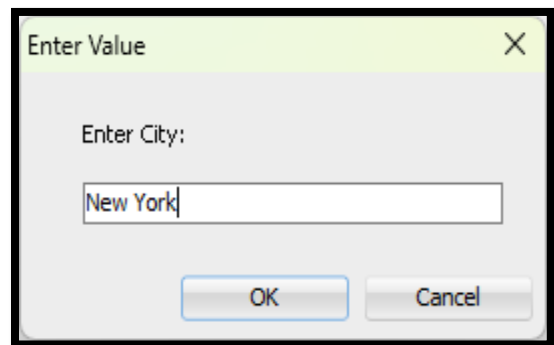
A screenshot of a 'Enter Value' dialog box. The title bar is green with a close button. The main area is light gray. The text 'Enter Customer Last Name:' is displayed above a text input field containing the text 'Smith'. At the bottom, there are two buttons: 'OK' and 'Cancel'.

*Figure 4.2: Request User's Last Name*



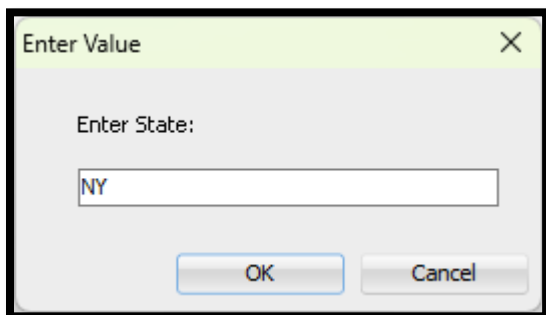
A screenshot of a 'Enter Value' dialog box. The title bar is green with a close button. The main area is light gray. The text 'Enter Segment (Consumer/Corporate/Home Office):' is displayed above a text input field containing the text 'Consumer'. At the bottom, there are two buttons: 'OK' and 'Cancel'.

*Figure 4.3: Request Segment*



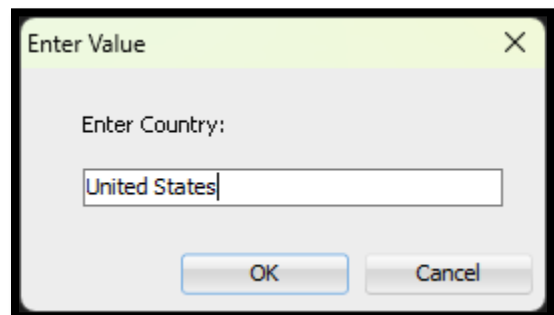
A screenshot of a 'Enter Value' dialog box. The title bar is green with a close button. The main area is light gray. The text 'Enter City:' is displayed above a text input field containing the text 'New York'. At the bottom, there are two buttons: 'OK' and 'Cancel'.

*Figure 4.4: Request City*



A screenshot of a 'Enter Value' dialog box. The title bar is green with a close button. The main area is light gray. The text 'Enter State:' is displayed above a text input field containing the text 'NY'. At the bottom, there are two buttons: 'OK' and 'Cancel'.

*Figure 4.5: Request State*



A screenshot of a 'Enter Value' dialog box. The title bar is green with a close button. The main area is light gray. The text 'Enter Country:' is displayed above a text input field containing the text 'United States'. At the bottom, there are two buttons: 'OK' and 'Cancel'.

*Figure 4.6: Request Country*

A dialog box titled "Enter Value" with a close button (X) in the top right corner. It contains a label "Enter Market:" followed by a text input field containing the text "US". At the bottom, there are two buttons: "OK" and "Cancel".

*Figure 4.7: Request Market*

 A dialog box titled "Enter Value" with a close button (X) in the top right corner. It contains a label "Enter Region:" followed by a text input field containing the text "Central". At the bottom, there are two buttons: "OK" and "Cancel".

*Figure 4.8: Request Region*

After user successfully enter the values, the script will auto generate the customerID by selecting the first alphabet of user's first and last name and follow by 5 unique sequence number. The figure below shows the example outputs.

```
Generated Customer ID: JS-16032
Customer Details
First Name: Jack
Last Name: Smith
Segment: Consumer
City: New York
State: NY
Country: United States
Market: US
Region: Central
```

*Figure 4.9: Sample output of Customer Table*

Secondly, the script will prompt users to enter the ship date and mode. The figures below show that the scripts prompt the user to enter the relative values.

 A dialog box titled "Enter Value" with a close button (X) in the top right corner. It contains a label "Enter Ship Date (YYYY-MM-DD):" followed by a text input field containing the text "2024-08-25". At the bottom, there are two buttons: "OK" and "Cancel".

*Figure 4.10: Request Ship Date*

 A dialog box titled "Enter Value" with a close button (X) in the top right corner. It contains a label "Enter Ship Mode (Same Day/Standard Class/First Class/Second Class):" followed by a text input field containing the text "Standard Class". At the bottom, there are two buttons: "OK" and "Cancel".

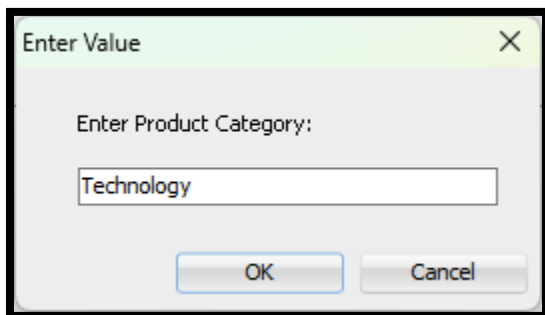
*Figure 4.11: Request Ship Mode*

After the user successfully enters the values, the script will auto-generate the shippingID using the sequence number. The figure below shows the example outputs.

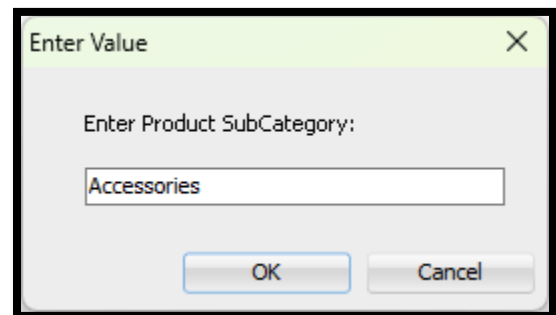
```
Generated Shipping ID: 51323
Shipping Details
Ship Date: 2024-08-25
Ship Mode: Standard Class
```

*Figure 4.12: Sample output of Shipping Table*

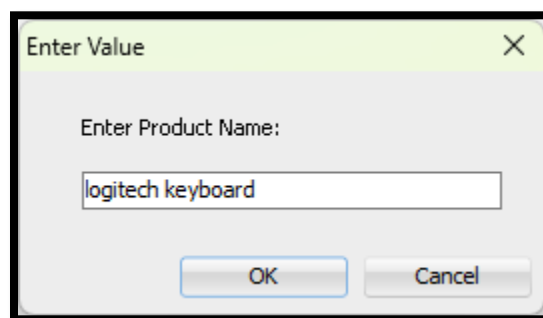
Thirdly, the script will prompt user to input the product category, product subcategory, and product name. The figures below show that the scripts prompt the user to enter the relative values.

A dialog box titled "Enter Value" with a close button (X) in the top right corner. It contains the text "Enter Product Category:" followed by a text input field containing the word "Technology". At the bottom, there are two buttons: "OK" and "Cancel".

*Figure 4.13: Request Product Category*

A dialog box titled "Enter Value" with a close button (X) in the top right corner. It contains the text "Enter Product SubCategory:" followed by a text input field containing the word "Accessories". At the bottom, there are two buttons: "OK" and "Cancel".

*Figure 4.14: Request Product Subcategory*

A dialog box titled "Enter Value" with a close button (X) in the top right corner. It contains the text "Enter Product Name:" followed by a text input field containing the text "logitech keyboard". At the bottom, there are two buttons: "OK" and "Cancel".

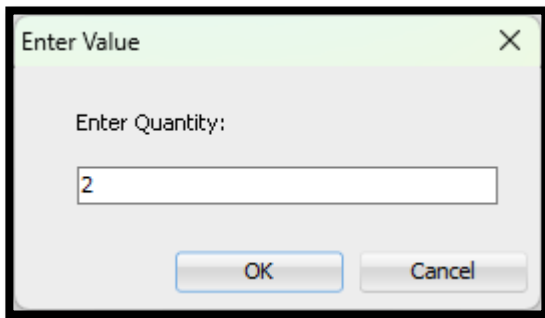
*Figure 4.15: Request Product Name*

After the user successfully enters the values, the script will auto-generate the productID based on the first 3 alphabets of the product category and product subcategory followed by an 8-digit sequence number. The figure below shows the example outputs.

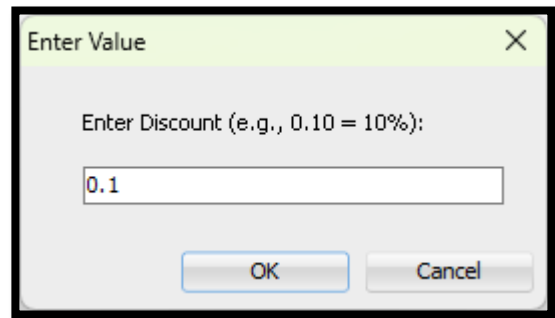
```
Generated Product ID: TEC-LOG-00000043
Product Details
Category: Technology
SubCategory: Accessories
Product Name: logitech keyboard
```

*Figure 4.16: Sample Output of Product Table*

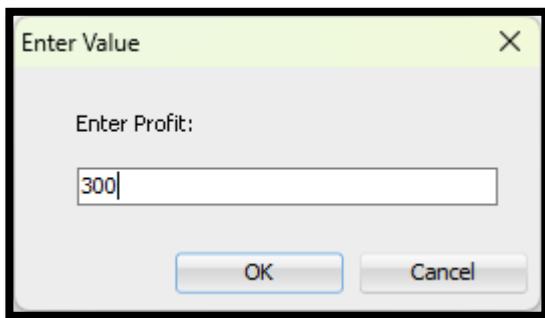
Fourthly, the script will prompt the user to enter the product quantity, discount value, profit, shipping cost, and order priority. The figures below show the scripts prompt the user to enter the relative values.

A dialog box titled "Enter Value" with a close button (X) in the top right corner. The text "Enter Quantity:" is displayed above a text input field containing the number "2". At the bottom, there are two buttons: "OK" and "Cancel".

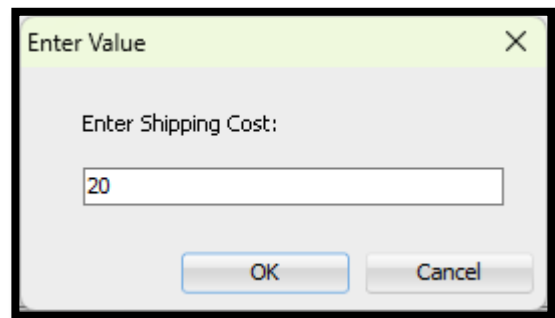
*Figure 4.17: Request Product Quantity*

A dialog box titled "Enter Value" with a close button (X) in the top right corner. The text "Enter Discount (e.g., 0.10 = 10%):" is displayed above a text input field containing the number "0.1". At the bottom, there are two buttons: "OK" and "Cancel".

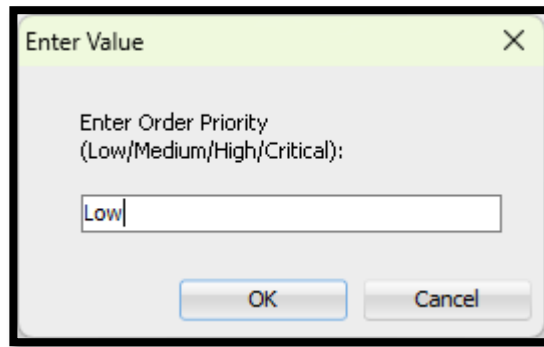
*Figure 4.18: Request Discount Rate*

A dialog box titled "Enter Value" with a close button (X) in the top right corner. The text "Enter Profit:" is displayed above a text input field containing the number "300". At the bottom, there are two buttons: "OK" and "Cancel".

*Figure 4.19: Request Profit*

A dialog box titled "Enter Value" with a close button (X) in the top right corner. The text "Enter Shipping Cost:" is displayed above a text input field containing the number "20". At the bottom, there are two buttons: "OK" and "Cancel".

*Figure 4.20: Request Shipping Cost*



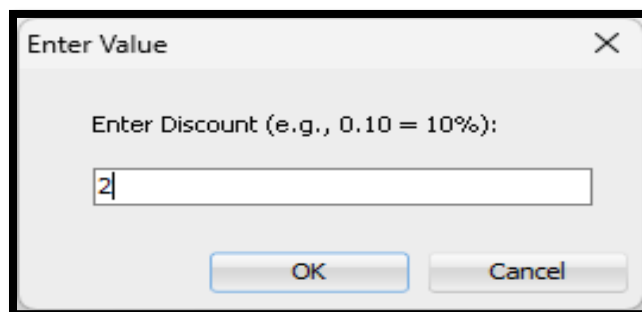
*Figure 4.21: Request Order Priority*

After user successfully enter the values, the script will auto generate the orderID based on the first 2 alphabets of user's market and the year of user's order data and followed by 4 digits sequence number. The figure below shows the example outputs.

```
Generated Order ID: UN-2024-0001
OrderItem Details
Quantity: 2
Discount: .1%
Profit: 300
Shipping Cost: 20
Order Priority: Low
```

*Figure 4.22: Sample Output of Order Table*

Finally, the script will display any error when the user input any false or incorrect value into the system. For example, the value for discount should only read 0.10 as 10%, 0.25 as 25% and so on. If the user input value larger than 1, it will prompt error due to the maximum value for discount is 100%. Next example is if user enter invalid shipping mode to the system such as "Air", it will display error message due to shipping mode does not have "Air" in the script.



*Figure 4.23: Invalid value to Discount*

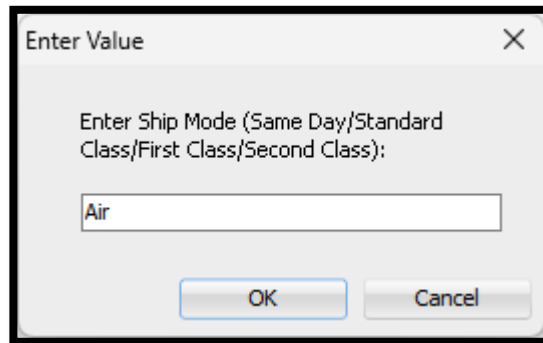


```

OrderItem Insert Error: ORA-20008: Invalid discount. Discount must be not exceed 100%.
ORA-06512: at "USER3.ORDERITEM_BEFORE_INPUT_UPDATE", line 9
ORA-04088: error during execution of trigger 'USER3.ORDERITEM_BEFORE_INPUT_UPDATE'
Transaction Error: ORA-20008: Invalid discount. Discount must be not exceed 100%.
ORA-06512: at "USER3.ORDERITEM_BEFORE_INPUT_UPDATE", line 9
ORA-04088: error during execution of trigger 'USER3.ORDERITEM_BEFORE_INPUT_UPDATE'

```

*Figure 4.24: Error Handling for Discount*



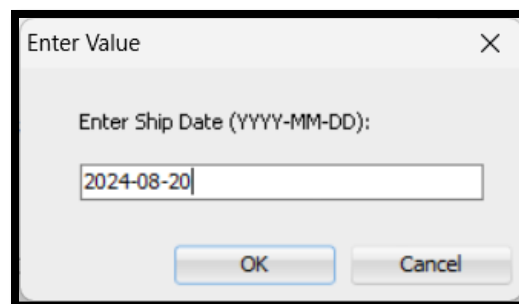
*Figure 4.25: Invalid value to Ship Mode*

```

Invalid shipping mode. Allowed values are Same Day, Standard Class, First Class, Second Class.
Shipping Insert Error: ORA-20003: Invalid shipping mode. Allowed values are Same Day, Standard Class, First Class, Second Class.
ORA-06512: at "USER3.SHIPPING_BEFORE_INPUT_UPDATE", line 29
ORA-04088: error during execution of trigger 'USER3.SHIPPING_BEFORE_INPUT_UPDATE'
Transaction Error: ORA-20003: Invalid shipping mode. Allowed values are Same Day, Standard Class, First Class, Second Class.
ORA-06512: at "USER3.SHIPPING_BEFORE_INPUT_UPDATE", line 29
ORA-04088: error during execution of trigger 'USER3.SHIPPING_BEFORE_INPUT_UPDATE'

```

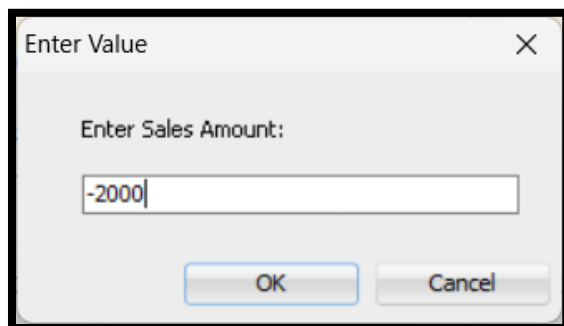
*Figure 4.26: Error Handling for Ship Mode*



*Figure 4.27: Invalid Date to Ship Date*

```
Shipping Insert Error: ORA-20002: Invalid shipping date. Shipping need to be after order date
ORA-06512: at "USER3.SHIPPING_BEFORE_INPUT_UPDATE", line 5
ORA-04088: error during execution of trigger 'USER3.SHIPPING_BEFORE_INPUT_UPDATE'
Transaction Error: ORA-20002: Invalid shipping date. Shipping need to be after order date
ORA-06512: at "USER3.SHIPPING_BEFORE_INPUT_UPDATE", line 5
ORA-04088: error during execution of trigger 'USER3.SHIPPING_BEFORE_INPUT_UPDATE'
```

*Figure 4.25: Error Handling for Ship Date*

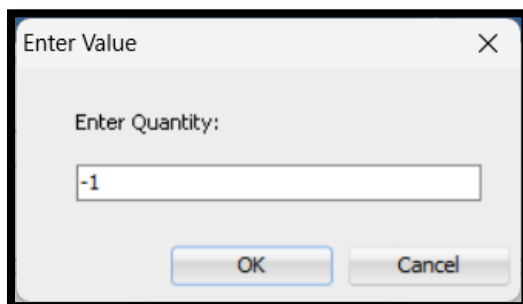


A screenshot of a Windows-style dialog box titled "Enter Value" with a close button (X) in the top right corner. The dialog contains the text "Enter Sales Amount:" followed by a text input field. The input field contains the value "-2000". At the bottom of the dialog are two buttons: "OK" and "Cancel".

*Figure 4.28: Invalid Value to Sales Amount*

```
OrderItem Insert Error: ORA-20007: Invalid sales amount. Sales amount cannot be negative.
ORA-06512: at "USER3.ORDERITEM_BEFORE_INPUT_UPDATE", line 7
ORA-04088: error during execution of trigger 'USER3.ORDERITEM_BEFORE_INPUT_UPDATE'
Transaction Error: ORA-20007: Invalid sales amount. Sales amount cannot be negative.
ORA-06512: at "USER3.ORDERITEM_BEFORE_INPUT_UPDATE", line 7
ORA-04088: error during execution of trigger 'USER3.ORDERITEM_BEFORE_INPUT_UPDATE'
```

*Figure 4.29: Error Handling for Sales Amount*



A screenshot of a Windows-style dialog box titled "Enter Value" with a close button (X) in the top right corner. The dialog contains the text "Enter Quantity:" followed by a text input field. The input field contains the value "-1". At the bottom of the dialog are two buttons: "OK" and "Cancel".

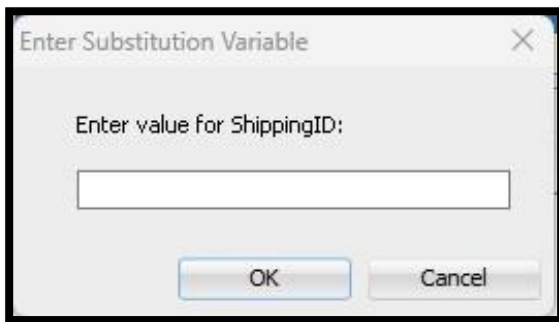
*Figure 4.30: Invalid Value to Quantity*

```
OrderItem Insert Error: ORA-20006: Invalid quantity. Quantity cannot be negative.  
ORA-06512: at "USER3.ORDERITEM_BEFORE_INPUT_UPDATE", line 5  
ORA-04088: error during execution of trigger 'USER3.ORDERITEM_BEFORE_INPUT_UPDATE'  
Transaction Error: ORA-20006: Invalid quantity. Quantity cannot be negative.  
ORA-06512: at "USER3.ORDERITEM_BEFORE_INPUT_UPDATE", line 5  
ORA-04088: error during execution of trigger 'USER3.ORDERITEM_BEFORE_INPUT_UPDATE'
```

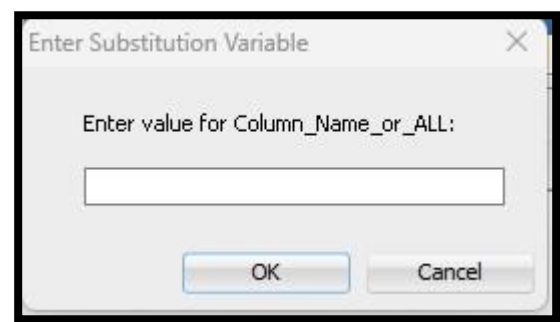
*Figure 4.31: Error Handling for Quantity*

## 4.2 Read

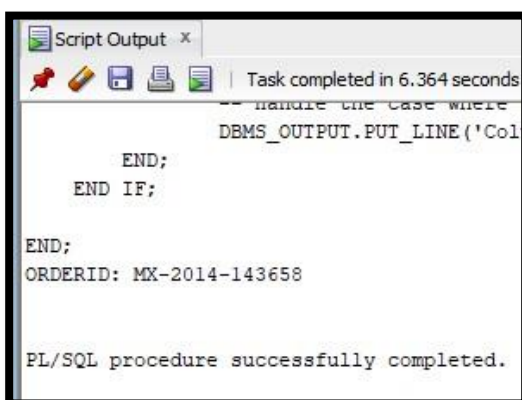
This PL/SQL script enables users to obtain the value of a single column or an entire row by providing the ShippingID. The script begins by verifying the ShippingID entered to make sure it exists in the table. The user has the option to obtain a single column or the complete row of data if the ShippingID 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.



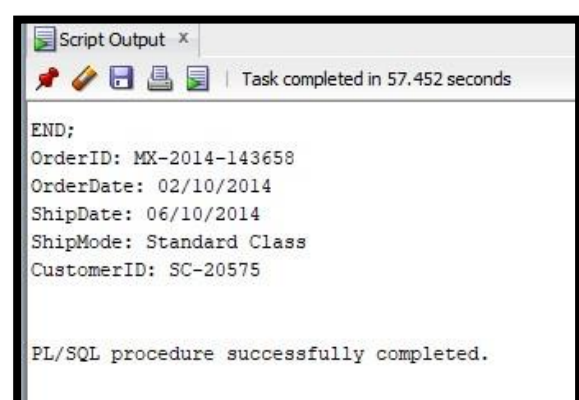
*Figure 4.32: Request Shipping ID*



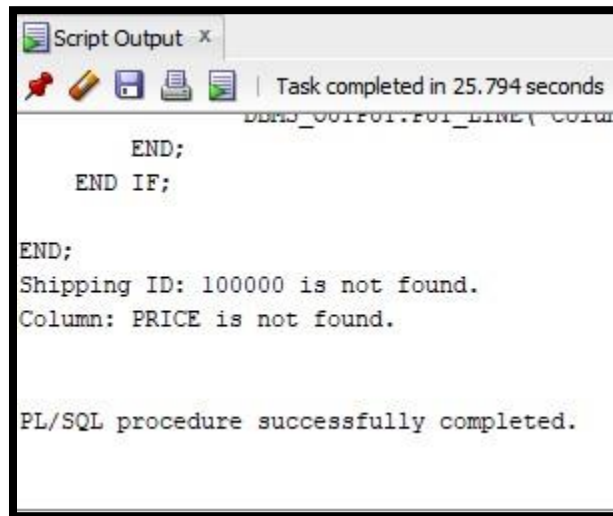
*Figure 4.33: Request Column name*



*Figure 4.34: Retrieve Specific Column*



*Figure 4.35: Retrieve All Column*



The screenshot shows a 'Script Output' window with a toolbar containing icons for a pushpin, a pencil, a save icon, a print icon, and a document icon. The status bar at the top right indicates 'Task completed in 25.794 seconds'. The main text area contains the following PL/SQL code and output:

```
END;  
END IF;  
  
END;  
Shipping ID: 100000 is not found.  
Column: PRICE is not found.  
  
PL/SQL procedure successfully completed.
```

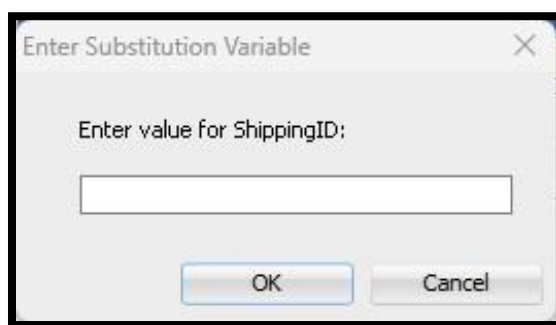
*Figure 4.36: Error Handling for Read*

### 4.3 Update

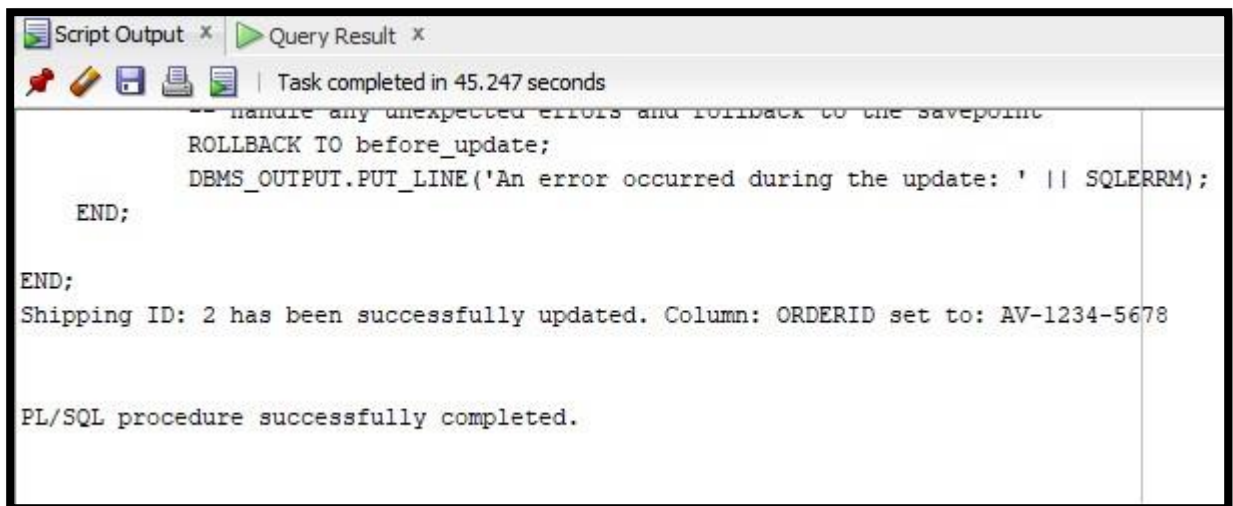
This PL/SQL script is designed to update a specific column in a record within the Shipping table using user input. Its error handling and validation capabilities are robust. Firstly, the script requests that the user enter the ShippingID. It then checks to see if the ShippingID is present in the Shipping 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 4.37: Request Shipping ID*



Script Output x Query Result x

Task completed in 45.247 seconds

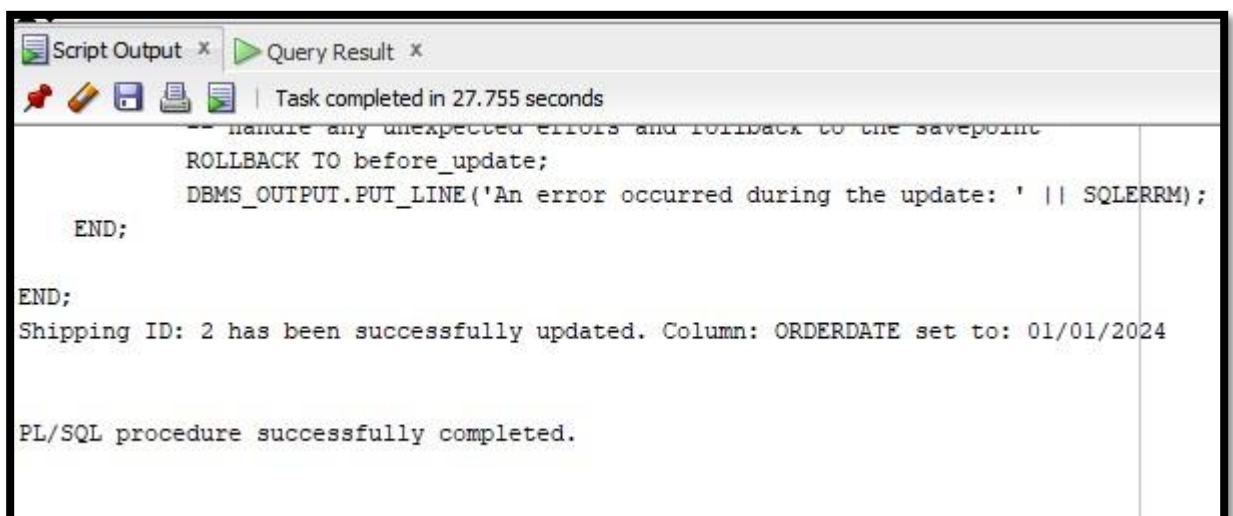
```
-- handle any unexpected errors and rollback to the savepoint
ROLLBACK TO before_update;
DBMS_OUTPUT.PUT_LINE('An error occurred during the update: ' || SQLERRM);

END;

END;
Shipping ID: 2 has been successfully updated. Column: ORDERID set to: AV-1234-5678

PL/SQL procedure successfully completed.
```

*Figure 4.38: Successfully Updated OrderID*



Script Output x Query Result x

Task completed in 27.755 seconds

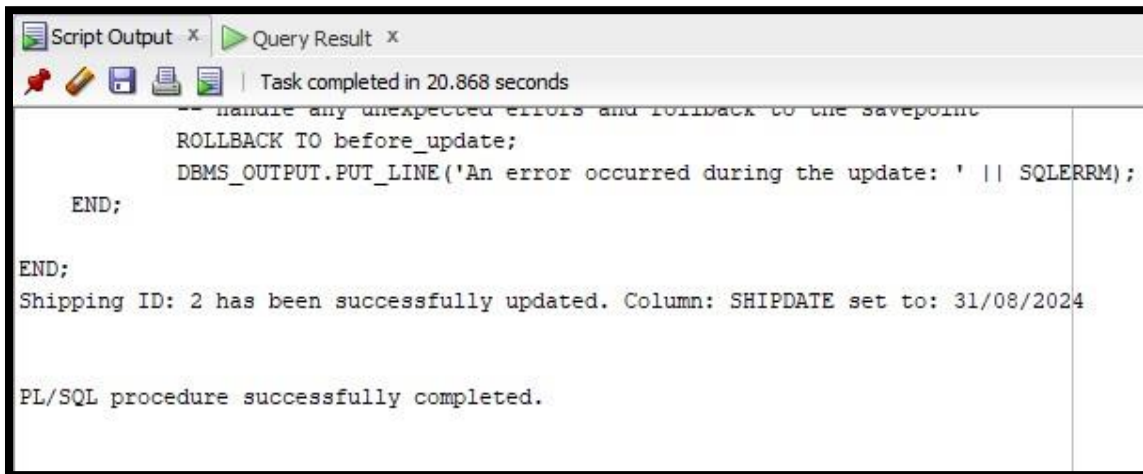
```
-- handle any unexpected errors and rollback to the savepoint
ROLLBACK TO before_update;
DBMS_OUTPUT.PUT_LINE('An error occurred during the update: ' || SQLERRM);

END;

END;
Shipping ID: 2 has been successfully updated. Column: ORDERDATE set to: 01/01/2024

PL/SQL procedure successfully completed.
```

*Figure 4.39: Successfully Updated OrderDate*



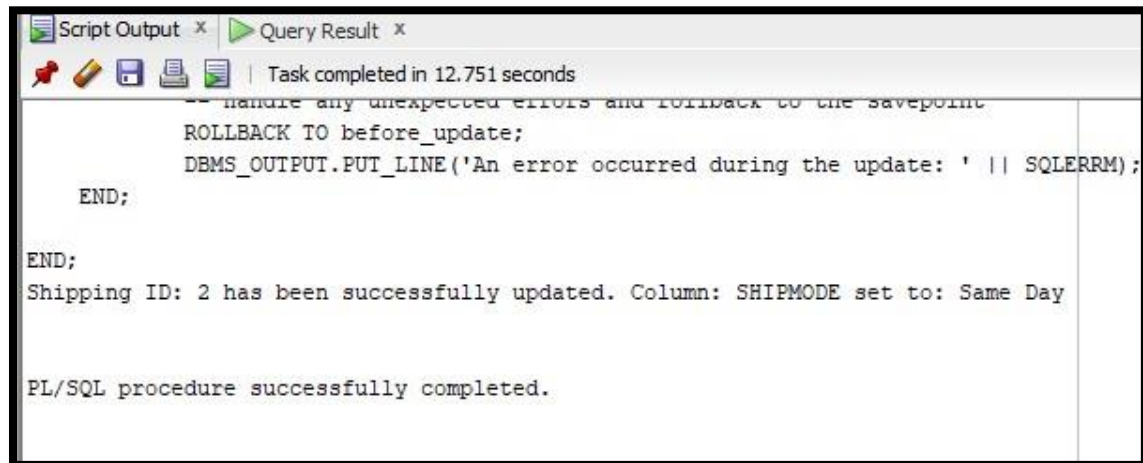
```
Script Output x Query Result x
Task completed in 20.868 seconds
-- handle any unexpected errors and rollback to the savepoint
ROLLBACK TO before_update;
DBMS_OUTPUT.PUT_LINE('An error occurred during the update: ' || SQLERRM);

END;

END;
Shipping ID: 2 has been successfully updated. Column: SHIPDATE set to: 31/08/2024

PL/SQL procedure successfully completed.
```

*Figure 4.40: Successfully Updated ShipID*



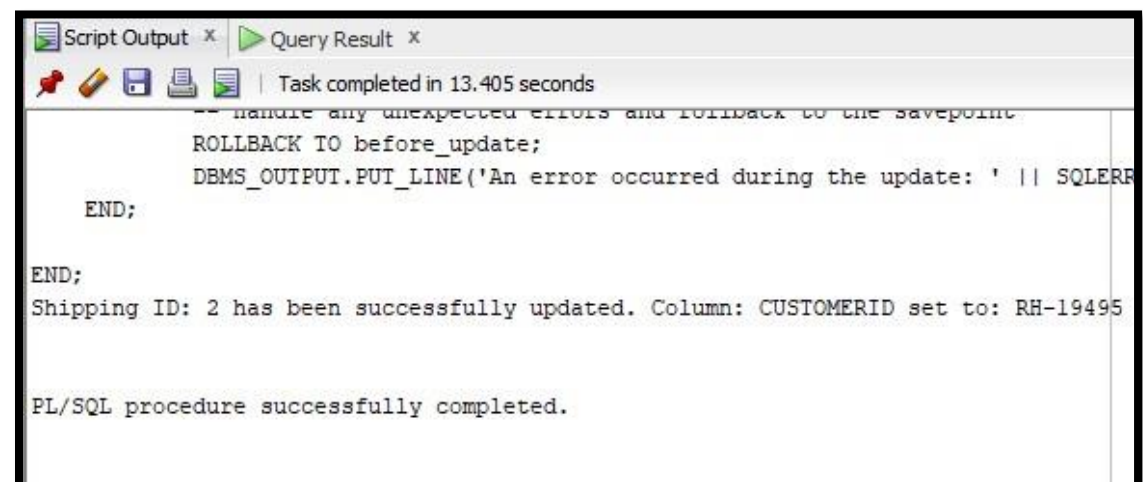
```
Script Output x Query Result x
Task completed in 12.751 seconds
-- handle any unexpected errors and rollback to the savepoint
ROLLBACK TO before_update;
DBMS_OUTPUT.PUT_LINE('An error occurred during the update: ' || SQLERRM);

END;

END;
Shipping ID: 2 has been successfully updated. Column: SHIPMODE set to: Same Day

PL/SQL procedure successfully completed.
```

*Figure 4.41: Successfully Updated ShipMode*



```
Script Output x Query Result x
Task completed in 13.405 seconds
-- handle any unexpected errors and rollback to the savepoint
ROLLBACK TO before_update;
DBMS_OUTPUT.PUT_LINE('An error occurred during the update: ' || SQLERRM);

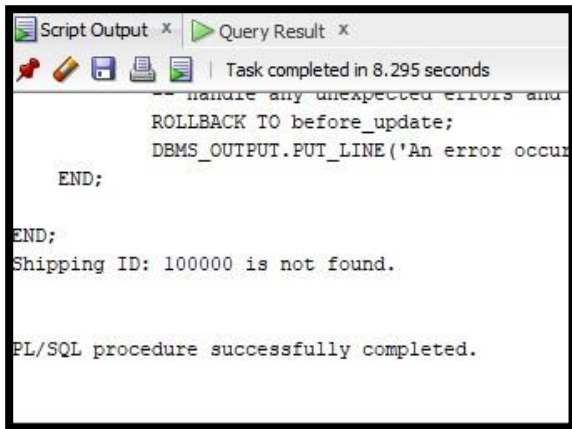
END;

END;
Shipping ID: 2 has been successfully updated. Column: CUSTOMERID set to: RH-19495

PL/SQL procedure successfully completed.
```

*Figure 4.42: Successfully Updated CustomerID*



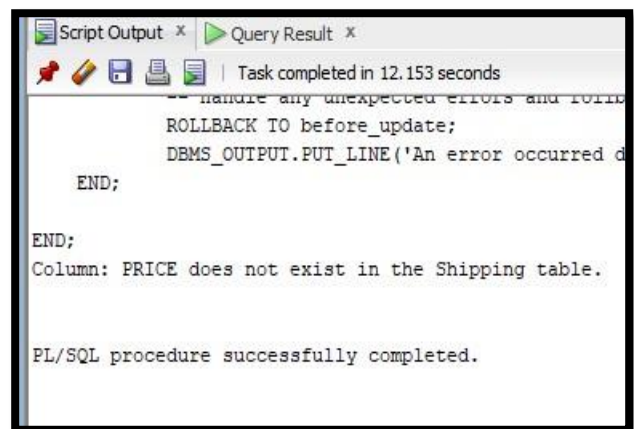


```
Script Output x Query Result x
Task completed in 8.295 seconds
-- handle any unexpected errors and
ROLLBACK TO before_update;
DBMS_OUTPUT.PUT_LINE('An error occur
END;

END;
Shipping ID: 100000 is not found.

PL/SQL procedure successfully completed.
```

*Figure 4.43: ShippingID Not Found*



```
Script Output x Query Result x
Task completed in 12.153 seconds
-- handle any unexpected errors and roll
ROLLBACK TO before_update;
DBMS_OUTPUT.PUT_LINE('An error occurred d
END;

END;
Column: PRICE does not exist in the Shipping table.

PL/SQL procedure successfully completed.
```

*Figure 4.44: Column Not Found*

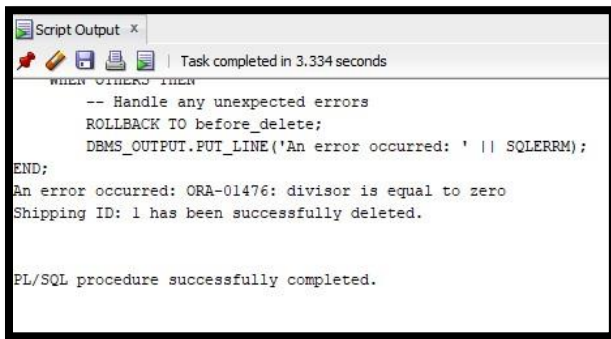
#### 4.4 Delete

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 4.45: Request Shipping ID*

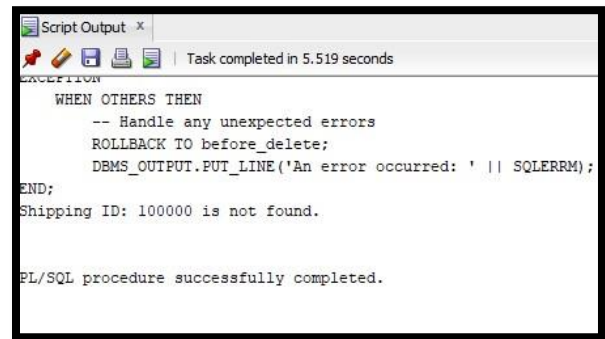


Script Output x | Task completed in 3.334 seconds

```
WHEN OTHERS THEN
    -- Handle any unexpected errors
    ROLLBACK TO before_delete;
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
An error occurred: ORA-01476: divisor is equal to zero
Shipping ID: 1 has been successfully deleted.

PL/SQL procedure successfully completed.
```

*Figure 4.46: Successfully Deleted*



Script Output x | Task completed in 5.519 seconds

```
EXCEPTION
WHEN OTHERS THEN
    -- Handle any unexpected errors
    ROLLBACK TO before_delete;
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
Shipping ID: 100000 is not found.

PL/SQL procedure successfully completed.
```

*Figure 4.47: Error Handling for Delete*

## **CHAPTER 5 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.

## CHAPTER 6 TRIGGERS

### 6.1 Shipping\_Audit\_Table

The code snippet below will be regarding After & Before Triggers which contains the purpose of the code. We will be mainly applying the triggers to “shipping” and “orderItem”. As well as displaying the output for Shipping\_Audit.

```
CREATE TABLE shipping_audit (  
    auditID NUMBER PRIMARY KEY,  
    shippingID NUMBER,  
    shipDate DATE,  
    shipMode VARCHAR2(35),  
    action VARCHAR2(10),  
    actionTimestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    customerID VARCHAR2(200),  
    orderID VARCHAR2(35),  
    orderDate DATE,  
);
```

**Purpose:** This table is used to log actions (INSERT, UPDATE, DELETE) performed on the wx.shipping table.

### 6.2 Shipping\_Audit\_Sequence

```
-- Create Sequence for Shipping Audit  
CREATE SEQUENCE shipping_audit_seq  
START WITH 1  
INCREMENT BY 1  
NOCACHE  
NOCYCLE;
```

**Purpose:** The sequence above is used to auto-generate unique “auditID” values for the “wx.Shipping\_audit” table.

### 6.3 Shipping\_Audit\_ID\_Trigger

```
-- Create Trigger for Shipping Audit  
CREATE OR REPLACE TRIGGER shipping_audit_id_trigger  
BEFORE INSERT ON shipping_audit  
FOR EACH ROW  
BEGIN  
    :NEW.auditID := shipping_audit_seq.NEXTVAL;  
END;
```

**Purpose:** This trigger ensures that every new record inserted into the “wx.shipping\_audit” table gets a unique “auditID” from the sequence “wx.shipping\_audit\_seq”.

## 6.4 Shipping\_Audit\_Sequence

```
-- SHIPPING_AFTER_DELETE
create or replace TRIGGER shipping_after_delete
AFTER INSERT OR UPDATE OR DELETE ON shipping
FOR EACH ROW
BEGIN

    IF DELETING THEN
        INSERT INTO shipping_audit (auditID, shippingID, orderID, orderDate, shipDate,
shipMode, customerID, action)
        VALUES
(shipshipping_audit_seq.NEXTVAL, :OLD.shippingID, :OLD.orderID, :OLD.orderDate, :OLD.shipD
ate, :OLD.shipMode, :OLD.customerID, 'DELETE');
    END IF;
END;
```

**Purpose:** This is after trigger logs deletions from the wx“shipping” table into the “wx.shipping\_audit” table.

## 6.5 Shipping\_After\_Insert\_Update

```
-- SHIPPING_AFTER_INSERT_UPDATE
create or replace TRIGGER shipping_after_Insert_Update
AFTER INSERT OR UPDATE on shipping
FOR EACH ROW
BEGIN
    --- Will insert into the audit table
    IF INSERTING THEN
        INSERT INTO shipping_audit (auditID, shippingID, orderID, orderDate, shipDate,
shipMode, customerID, action)
        VALUES
(shipshipping_audit_seq.NEXTVAL, :NEW.shippingID, :NEW.orderID, :NEW.orderDate, :NEW.shipD
ate, :NEW.shipMode, :NEW.customerID, 'INSERT');
    ELSIF UPDATING THEN
        INSERT INTO shipping_audit (auditID, shippingID, orderID, orderDate, shipDate,
shipMode, customerID, action)
        VALUES
(shipshipping_audit_seq.NEXTVAL, :NEW.shippingID, :NEW.orderID, :NEW.orderDate, :NEW.shipD
ate, :NEW.shipMode, :NEW.customerID, 'UPDATE');
    END IF;
END;
```

**Purpose:** This after trigger logs insertions and updates on the “wx.shipping” table into the “wx.shipping\_audit” table.

## 6.6 Shipping\_Before\_Input\_Update

```
-- For Shipping Table
create or replace TRIGGER shipping_before_Input_Update
BEFORE INSERT OR UPDATE ON shipping
FOR EACH ROW
BEGIN
    -- Validate ShipDate and ShipMode
    IF INSERTING THEN
        IF :NEW.shipDate IS NULL OR :NEW.shipDate < :NEW.orderDate THEN
            RAISE_APPLICATION_ERROR(-20002, 'Invalid shipping date. Shipping needs to
be after order date: ' || TO_CHAR(:NEW.shipDate, 'DD-MON-YYYY HH24:MI:SS') || ' - ' ||
TO_CHAR(:NEW.orderDate, 'DD-MON-YYYY HH24:MI:SS'));

            ELSIF :NEW.shipDate < TO_DATE('1900-01-01', 'YYYY-MM-DD') THEN
                RAISE_APPLICATION_ERROR(-20004, 'Invalid shipping date. The date is too
far in the past.');
```

```
            ELSIF :NEW.shipMode NOT IN ('Same Day', 'Standard Class', 'First Class',
'Second Class') THEN
                RAISE_APPLICATION_ERROR(-20003, 'Invalid shipping mode. Allowed values are
Same Day, Standard Class, First Class, Second Class.');
```

```
            END IF;

        ELSIF UPDATING THEN
            IF :NEW.shipDate IS NULL OR :NEW.shipDate < :NEW.orderDate OR :NEW.shipDate
< :OLD.orderDate OR :OLD.shipDate < :NEW.orderDate THEN
                RAISE_APPLICATION_ERROR(-20002, 'Invalid shipping date. Shipping need to
be after order date');
```

```
            ELSIF :NEW.shipDate < TO_DATE('1900-01-01', 'YYYY-MM-DD') THEN
                RAISE_APPLICATION_ERROR(-20004, 'Invalid shipping date. The date is too
far in the past.');
```

```
            END IF;

            IF :NEW.shipMode IS NULL THEN
                RAISE_APPLICATION_ERROR(-20003, 'Invalid shipping mode. Shipping mode
cannot be NULL.');
```

```
            ELSIF :NEW.shipMode NOT IN ('Same Day', 'Standard Class', 'First Class',
'Second Class') THEN
                RAISE_APPLICATION_ERROR(-20003, 'Invalid shipping mode. Allowed values are
Same Day, Standard Class, First Class, Second Class.');
```

```
            END IF;
        END IF; -- End of INSERTING/UPDATING block
    END;
```

**Purpose:** This before trigger validates the “shipDate” and “shipMode” fields before inserting or updating records in the “wx.shipping” table, ensuring dates are within valid ranges and shipping modes are within allowed values. As well as Order Date cannot be after Shipping Date.

## 6.7 OrderItem\_Before\_Input\_Update

```
-- For OrderItem Table
create or replace TRIGGER orderItem_before_Input_Update
BEFORE INSERT OR UPDATE ON orderItem
FOR EACH ROW
BEGIN
    -- Validate Quantity, SalesAmount, and Discount for Inserts
    IF INSERTING THEN
        IF :NEW.quantity < 0 THEN
            RAISE_APPLICATION_ERROR(-20006, 'Invalid quantity. Quantity cannot be
negative.');
```

```
        ELSIF :NEW.salesAmount < 0 THEN
            RAISE_APPLICATION_ERROR(-20007, 'Invalid sales amount. Sales amount cannot
be negative.');
```

```
        ELSIF :NEW.discount > 1.0 THEN
            RAISE_APPLICATION_ERROR(-20008, 'Invalid discount. Discount must be not
exceed 100%.');
```

```
        END IF;

    -- Validate Quantity, SalesAmount, and Discount for Updates
    ELSIF UPDATING THEN
        IF :NEW.quantity < 0 THEN
            RAISE_APPLICATION_ERROR(-20006, 'Invalid quantity. Quantity cannot be
negative.');
```

```
        ELSIF :NEW.salesAmount < 0 THEN
            RAISE_APPLICATION_ERROR(-20007, 'Invalid sales amount. Sales amount cannot
be negative.');
```

```
        ELSIF :NEW.discount > 1.0 THEN
            RAISE_APPLICATION_ERROR(-20008, 'Invalid discount. Discount must be not
exceed 100%.');
```

```
        END IF;
    END IF; -- End of INSERTING/UPDATING block
END;
```

**Purpose:** This before trigger validates the “quantity”, “salesAmount”, and “discount” fields before inserting or updating records in the “wx.orderItem” table, ensuring values are within allowed ranges.

## 6.8 Shipping\_Before\_Delete

```
-- SHIPPING_BEFORE_DELETE
create or replace TRIGGER shipping_before_delete
BEFORE DELETE ON shipping
FOR EACH ROW

DECLARE
    order_count NUMBER;
```



```

BEGIN
  -- change to user admin username
  IF USER != 'wx' THEN
    RAISE_APPLICATION_ERROR(-20006, 'Only admin can delete orders.');
```

```

  END IF;
END;
```

**Purpose:** This trigger restricts deletion of records from the “wx.shipping” table to users with the username wx, ensuring that only admins can perform delete operations.

## 6.9 Output: Shipping\_Audit

AUDITID	SHIPPINGID	SHIPDATE	SHIPMODE	ACTION	ACTIONTIMESTAMP	CUSTOMERID
7	38362	19/08/2024	Ground	INSERT	18/08/2024 22:27:39.087000000	RS-19765
AUDITID	SHIPPINGID	SHIPDATE	SHIPMODE	ACTION	ACTIONTIMESTAMP	CUSTOMERID
22	14070	19/02/2011	Second Class	DELETE	19/08/2024 11:02:33.924000000	ON-18715
AUDITID	SHIPPINGID	SHIPDATE	SHIPMODE	ACTION	ACTIONTIMESTAMP	CUSTOMERID
9	38362	19/08/2024	First Class	UPDATE	18/08/2024 22:30:08.363000000	RS-19765

Figure 6.1 Shipping\_Audit Output

## CHAPTER 7 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.

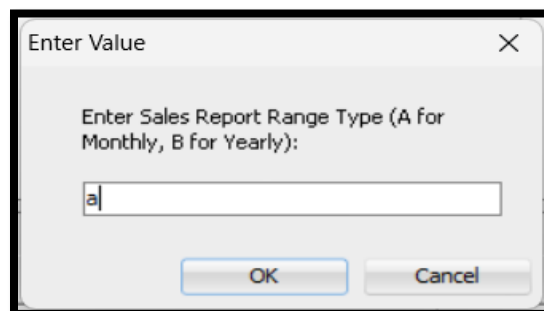
### 7.1 Sales Report for Monthly or Yearly

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.

#### 7.1.1 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."

```
ACCEPT range_type char PROMPT "Enter Sales Report Range Type (A for Monthly, B for Yearly): ";  
DECLARE  
    range_type VARCHAR2(10);  
BEGIN  
    range_type := UPPER('&range_type');  
    range_sales_report(range_type);  
END;  
/
```



*Figure 7.1: Request Sales Report Range Type*

### 7.1.2 Procedure to Generate Report

After the user inputs the type, it will run the *range\_sales\_report* procedure with the type as a parameter. The *range\_type* will be checked with an if else statement to ensure the valid input. If the input is invalid, the invalid message will show back to the user. Otherwise, it will continue to query the selected range type to produce the sales report. To optimize the query, operations like projection, and selection have been done in the joining operation.

```
create or replace PROCEDURE range_sales_report (
    range_type IN VARCHAR2
) IS

    CURSOR monthly_cursor IS
    SELECT
        TO_CHAR(s.ORDERDATE, 'YYYY-MM') AS month,
        SUM(oi.SALESAMOUNT) AS totalsales
    FROM
        SHIPPING s
        JOIN (SELECT o.ORDERID, o.SALESAMOUNT FROM ORDERITEM o) oi ON s.ORDERID =
oi.ORDERID
    GROUP BY
        TO_CHAR(s.ORDERDATE, 'YYYY-MM')
    ORDER BY
        month DESC;

    CURSOR yearly_cursor IS
    SELECT
        TO_CHAR(s.ORDERDATE, 'YYYY') AS year,
        SUM(oi.SALESAMOUNT) AS totalsales
    FROM
        SHIPPING s
        JOIN (SELECT o.ORDERID, o.SALESAMOUNT FROM ORDERITEM o) oi ON s.ORDERID =
oi.ORDERID
    GROUP BY
        TO_CHAR(s.ORDERDATE, 'YYYY')
    ORDER BY
        year DESC;

    monthly_record monthly_cursor%ROWTYPE;
    yearly_record yearly_cursor%ROWTYPE;
BEGIN
    IF range_type = 'MONTHLY' OR range_type = 'A' THEN
        OPEN monthly_cursor;
        DBMS_OUTPUT.PUT_LINE('Month' || CHR(9) || 'Total Sales');
        DBMS_OUTPUT.PUT_LINE('-----');
```

```

        LOOP
            FETCH monthly_cursor INTO monthly_record;
            EXIT WHEN monthly_cursor%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE(monthly_record.month || CHR(9) || 'RM ' ||
monthly_record.totalsales);
        END LOOP;
        CLOSE monthly_cursor;

    ELSIF range_type = 'YEARLY' OR range_type = 'B' THEN
        OPEN yearly_cursor;
        DBMS_OUTPUT.PUT_LINE('Year' || CHR(9) || 'Total Sales');
        DBMS_OUTPUT.PUT_LINE('-----');
        LOOP
            FETCH yearly_cursor INTO yearly_record;
            EXIT WHEN yearly_cursor%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE(yearly_record.year || CHR(9) || 'RM ' ||
yearly_record.totalsales);
        END LOOP;
        CLOSE yearly_cursor;

    ELSE
        DBMS_OUTPUT.PUT_LINE('Invalid range type. Please use "MONTHLY" or "YEARLY".');
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/

```

### 7.1.3 Sample Outputs

#### i. Monthly Sales Report

Month	Total Sales
-----	
2014-12	RM 1528355.05
2014-11	RM 1856869.87
2014-10	RM 1334558.15
2014-09	RM 1557185.6
2014-08	RM 1477867.72
2014-07	RM 825864.98
2014-06	RM 1208500.06
2014-05	RM 849302.19

Figure 7.2: Sample Output for Monthly Sales Report

ii. Yearly Sales Report

Year	Total Sales
2014	RM 13561707.05
2013	RM 10476745.39
2012	RM 8225355.2
2011	RM 7050874.23

Figure 7.3: Sample Output for Yearly Sales Report

iii. Invalid Input Message

```
END;  
Invalid range type. Please use "MONTHLY" or "YEARLY".
```

Figure 7.4 Invalid Range Type

## 7.2 Sales by Market Report

The purpose of the *sales\_by\_market\_report* procedure is to show the total sales from each market.

### 7.2.1 Execution Way

```
execute sales_by_market_report();
```

### 7.2.2 Procedure

In this procedure, there will be a cursor to loop all the results from the selection, so that we can log out the result to the user. The query involves three tables which are customer table for the market, orderitem table for sales amount and shipping table that links two tables. The query is optimised by using the projection and selection before joining the table to reduce the cost of the query.

```
create or replace PROCEDURE sales_by_market_report IS  
  
    CURSOR market_cursor IS  
    SELECT  
        c.MARKET,  
        SUM(oi.SALESAMOUNT) AS total_sales  
    FROM
```

```

        SHIPPING s
        JOIN (SELECT o.SALESAMOUNT, o.ORDERID FROM ORDERITEM o) oi ON s.ORDERID =
oi.ORDERID
        JOIN (SELECT c.CUSTOMERID, c.MARKET FROM CUSTOMER c) c ON s.CUSTOMERID =
c.CUSTOMERID
        GROUP BY
            c.MARKET
        ORDER BY
            total_sales DESC;

market_record market_cursor%ROWTYPE;
BEGIN
    OPEN market_cursor;
    DBMS_OUTPUT.PUT_LINE(RPAD('MARKET', 20) || 'Total Sales');
    DBMS_OUTPUT.PUT_LINE('-----');
    LOOP
        FETCH market_cursor INTO market_record;
        EXIT WHEN market_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(RPAD(market_record.MARKET, 20) || 'RM ' ||
market_record.total_sales);
    END LOOP;
    CLOSE market_cursor;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/

```

### 7.2.3 Sample Outputs

MARKET	Total Sales
-----	
APAC	RM 12229769.38
EU	RM 8635337.65
US	RM 8014078.65
LATAM	RM 5297521.96
EMEA	RM 2471574.86
Africa	RM 2466833.53
Canada	RM 199565.84

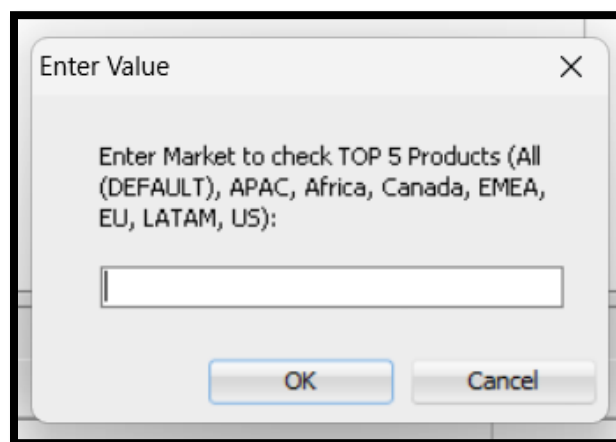
Figure 7.5: Sample Output for Market

### 7.3 Top 5 Sales Products in Each Market

The purpose is for users to get the top 5 highest sales in each market or across all the markets.

#### 7.3.1 User Input Interface

```
ACCEPT market char DEFAULT "ALL" PROMPT "Enter Market to check TOP 5 Products  
(All(DEFAULT), APAC, Africa, Canada, EMEA, EU, LATAM, US): ";  
DECLARE  
    market VARCHAR2(10);  
BEGIN  
market := UPPER('&market');  
TOP_5_PRODUCTS_BY_MARKET(market);  
END;  
/
```



*Figure 7.6: Request Market*

#### 7.3.2 Procedure

The input will be validated by the procedure to ensure it is a valid market or “ALL”. If the input value is invalid it will prompt a message to alert the user. Otherwise, the query will be run based on the market. For a specific market, the query will involve a customer table, product table, shipping table, and orderitem table. For checking across all markets, it will not involve a customer table to get the marketplace. The result will be ordered by the total sales quantity of product descending and get the first five products.

```

create or replace PROCEDURE top_5_products_by_market (p_market VARCHAR2 DEFAULT 'ALL')
IS

    market_count INTEGER;

    -- Cursor to check if the market exists
    CURSOR market_check_cursor IS
    SELECT COUNT(*)
    FROM CUSTOMER c
    WHERE UPPER(c.MARKET) = p_market;

    -- Cursor to retrieve the top 5 products for a specific market
    CURSOR product_cursor (market_name VARCHAR2) IS
    SELECT *
    FROM (
        SELECT
            p.PRODUCTID,
            p.PRODUCTNAME,
            SUM(oi.quantity) AS total_sales
        FROM
            ORDERITEM oi
            JOIN (SELECT p.productID,p.PRODUCTNAME from PRODUCT p) p ON oi.PRODUCTID =
p.PRODUCTID
            JOIN (SELECT s.orderID, s.customerID from SHIPPING s) s ON oi.ORDERID =
s.ORDERID
            JOIN (SELECT c.customerID, c.market FROM CUSTOMER c) c ON s.CUSTOMERID =
c.CUSTOMERID
        WHERE
            UPPER(c.MARKET) = UPPER(market_name)
        GROUP BY
            p.PRODUCTID, p.PRODUCTNAME
        ORDER BY
            total_sales DESC
    )
    WHERE ROWNUM <= 5;

    -- Cursor to retrieve the top 5 products accross all markets
    CURSOR product_cursor_all IS
    SELECT * FROM
    (
        SELECT
            p.productid,
            p.productname,
            SUM(oi.quantity) AS total_sales
        FROM
            orderitem oi

```



```

        JOIN (SELECT p.productid, p.productname FROM product p) p ON oi.productid =
p.productid
        JOIN (SELECT s.orderid FROM shipping s) s ON oi.orderid = s.orderid
        GROUP BY
            p.productid, p.productname
        ORDER BY
            total_sales DESC
    )
    WHERE ROWNUM <= 5;

all_product_record product_cursor_all%ROWTYPE;
product_record product_cursor%ROWTYPE;
BEGIN
    IF p_market = 'ALL' THEN

        DBMS_OUTPUT.PUT_LINE('Market: ALL MARKET');
        DBMS_OUTPUT.PUT_LINE(RPAD('Product ID',18) || RPAD('Product Name',60) ||
'Total Sales');
        DBMS_OUTPUT.PUT_LINE('-----
-----');

        OPEN product_cursor_all;
        LOOP
            FETCH product_cursor_all INTO all_product_record;
            EXIT WHEN product_cursor_all%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE(RPAD(all_product_record.PRODUCTID,18) ||
RPAD(all_product_record.PRODUCTNAME,60) || all_product_record.total_sales);
        END LOOP;
        CLOSE product_cursor_all;
    ELSE
        -- Check if the specified market exists
        OPEN market_check_cursor;
        FETCH market_check_cursor INTO market_count;
        CLOSE market_check_cursor;

        IF market_count > 0 THEN
            -- Process the specific market
            DBMS_OUTPUT.PUT_LINE('Market: ' || p_market);
            DBMS_OUTPUT.PUT_LINE(RPAD('Product ID',18) || RPAD('Product Name',60) ||
'Total Sales');
            DBMS_OUTPUT.PUT_LINE('-----
-----');

            OPEN product_cursor(p_market);
            LOOP
                FETCH product_cursor INTO product_record;

```

```

        EXIT WHEN product_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(RPAD(product_record.PRODUCTID,18) ||
RPAD(product_record.PRODUCTNAME,60) || product_record.total_sales);
    END LOOP;
    CLOSE product_cursor;

    DBMS_OUTPUT.PUT_LINE('');
ELSE
    -- Market does not exist
    DBMS_OUTPUT.PUT_LINE('Market "' || p_market || '" does not exist.');
```

```

END IF;
END IF;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/
```

### 7.3.3 Sample Outputs

#### i. All Market

Market: ALL MARKET		
Product ID	Product Name	Total Sales
OFF-AR-10003651	Sanford Pencil Sharpener  Easy-Erase	551
OFF-BI-10002570	Cardinal 3-Hole Punch  Clear	414
OFF-BI-10001808	Cardinal Binding Machine  Clear	406
OFF-BI-10002799	Cardinal Binder Covers  Clear	354
OFF-AR-10003829	Stanley Markers  Fluorescent	348

Figure 7.7: Sample Output for All Market

#### ii. Specific Market

Market: EU		
Product ID	Product Name	Total Sales
OFF-BI-10003012	Wilson Jones Hole Reinforcements  Economy	191
OFF-BI-10003917	Wilson Jones 3-Hole Punch  Recycled	170
OFF-AR-10003829	Stanley Markers  Fluorescent	159
TEC-AC-10004568	Maxell?LTO Ultrium - 800 GB	159
OFF-BI-10002570	Cardinal 3-Hole Punch  Clear	149

Figure 7.8: Sample Output for Specific Market

### iii. Invalid Input

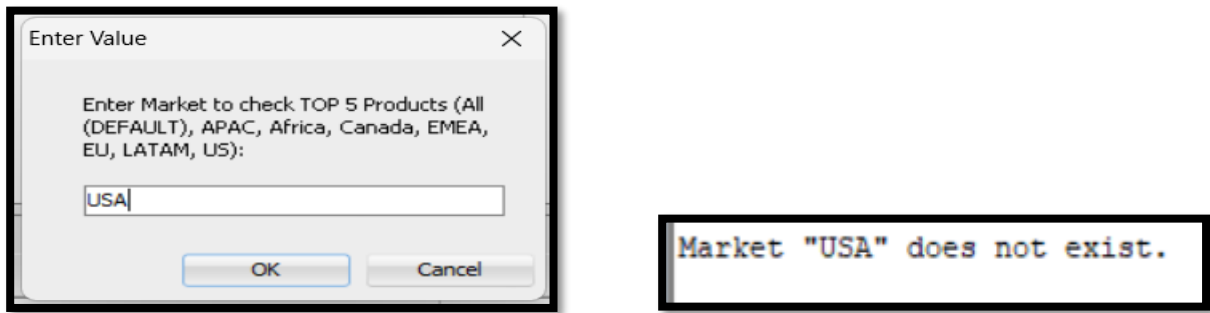


Figure 7.9: Invalid Value for Market

## 7.4 Total Sales in Each Shipping Mode Report

This report will show users the total orders and profit that earn in each Shipping Mode.

### 7.4.1 Way to Execute

```
execute shipping_mode_report();
```

### 7.4.2 Procedure

The `shipping\_mode\_report` procedure, it will generate a report on shipping modes by concluding the amount of orders and the profit for each mode. It selects and aggregates data from the orderitem and shipping tables. After joining the tables, the data will be sorted descending by the profit gained by each shipping mode. Besides that, the procedure also counts the amount of the order for each shipping mode.

```
create or replace PROCEDURE shipping_mode_report IS

    CURSOR quantity_cursor IS
    SELECT
        s.SHIPMODE,
        COUNT(DISTINCT oi.ORDERID) AS order_count,
        SUM(oi.PROFIT) AS total_profit
    FROM
        ORDERITEM oi
        JOIN (SELECT SHIPPING.SHIPMODE, SHIPPING.ORDERID FROM SHIPPING) s ON
oi.ORDERID = s.ORDERID
    GROUP BY
        s.SHIPMODE
    ORDER BY
        total_profit DESC;
```

```

    quantity_record quantity_cursor%ROWTYPE;
BEGIN
    OPEN quantity_cursor;
    DBMS_OUTPUT.PUT_LINE(RPAD('Shipping Mode',20) || RPAD('Total Order',16) || 'Total
Profit');
    DBMS_OUTPUT.PUT_LINE('-----');

    LOOP
        FETCH quantity_cursor INTO quantity_record;
        EXIT WHEN quantity_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(RPAD(quantity_record.SHIPMODE, 20) ||
RPAD(quantity_record.order_count,16) || 'RM ' || quantity_record.total_profit);
    END LOOP;
    CLOSE quantity_cursor;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/

```

### 7.4.3 Sample Outputs

Shipping Mode	Total Order	Total Profit
-----		
Standard Class	15154	RM 2757399.19
Second Class	5119	RM 905110.81
First Class	3821	RM 586510.62
Same Day	1347	RM 264402.73

*Figure 7.10: Sample Output for Shipping Mode*

### 7.5 End of Report

This report can show the analysis report by the end of day, month or year. It has included the amount of order, total of sales, profit and the total order for each shipping mode.

### 7.5.1 User Input Interface

It prompts the user to input a date range type with the options which are year, month and day. After that, user is required to enter a specific date in the appropriate date format. The inputs then passed to the *end\_of\_report* procedure.

```
ACCEPT i_type CHAR PROMPT 'Enter Range Type (YEAR, MONTH, DAY): ';
ACCEPT i_date CHAR PROMPT 'Enter Date with Format (YEAR = YYYY, MONTH = YYYY-MM, DAY =
YYYY-MM-DD): ';

DECLARE
    v_type VARCHAR2(10);
    v_date VARCHAR2(15);
BEGIN
    v_type := UPPER('&i_type');
    v_date := '&i_date';

    end_of_report(v_type, v_date);
END;
/
```

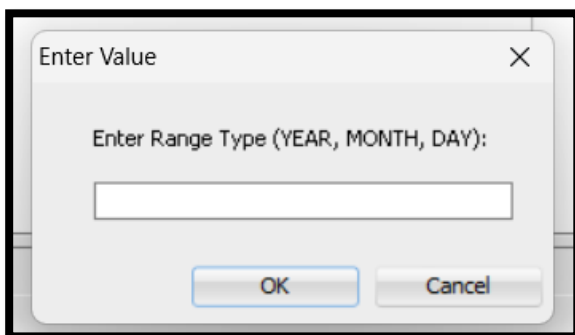


Figure 7.11: Request Range Type

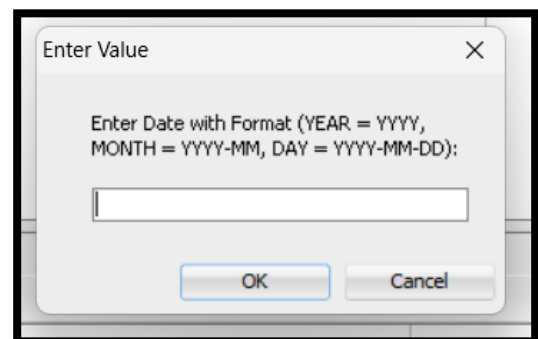


Figure 7.12: Request Date

### 7.5.2 Procedure

The *end\_of\_report* procedure generates a report based on a specified date range (YEAR, MONTH, or DAY). It first validates the input date type and format by using the REGEXP\_LIKE (Oracle Live SQL - Script: REGEXP\_LIKE- Examples, n.d.). Then, it calculates and outputs the total order count, sales amount, and profit for the given date. The procedure also includes different shipping modes, counting the number of orders for each mode within the specified date range and displaying the results in

descending order by count. The procedure handles exceptions, including cases where no data is found, or other errors occur.

```
CREATE OR REPLACE PROCEDURE end_of_report (  
    p_date_type IN VARCHAR2,  
    p_date_value IN VARCHAR2  
) IS  
    v_order_count    NUMBER;  
    v_total_sales    NUMBER;  
    v_total_profit   NUMBER;  
    v_mode_count     NUMBER;  
BEGIN  
  
    -- Validate input date type and format  
    IF p_date_type NOT IN ('MONTH', 'DAY', 'YEAR') THEN  
        RAISE_APPLICATION_ERROR(-20001, 'Invalid date type. Use "MONTH", "DAY", or  
"YEAR".');  
    END IF;  
  
    IF p_date_type = 'MONTH' AND (NOT regexp_like(p_date_value, '^[0-9]{4}-[0-  
9]{2}$')) THEN  
        RAISE_APPLICATION_ERROR(-20001, 'Invalid date value. Use YYYY-MM for MONTH.');
```

```
    END IF;  
  
    IF p_date_type = 'DAY' AND (NOT regexp_like(p_date_value, '^[0-9]{4}-[0-9]{2}-[0-  
9]{2}$')) THEN  
        RAISE_APPLICATION_ERROR(-20001, 'Invalid date value. Use YYYY-MM-DD for DAY.');
```

```
    END IF;  
  
    IF p_date_type = 'YEAR' AND (NOT regexp_like(p_date_value, '^[0-9]{4}$')) THEN  
        RAISE_APPLICATION_ERROR(-20001, 'Invalid date value. Use YYYY for YEAR.');
```

```
    END IF;  
  
    -- Fetch the total order count, sales, and profit  
    SELECT  
        COUNT(DISTINCT oi.ORDERID),  
        SUM(oi.SALESAMOUNT),  
        SUM(oi.PROFIT)  
    INTO  
        v_order_count,  
        v_total_sales,  
        v_total_profit  
    FROM  
        ORDERITEM oi  
    JOIN SHIPPING s ON oi.ORDERID = s.ORDERID
```

```

WHERE
    (p_date_type = 'MONTH' AND TO_CHAR(s.ORDERDATE, 'YYYY-MM') = p_date_value)
    OR
    (p_date_type = 'DAY' AND TO_CHAR(s.ORDERDATE, 'YYYY-MM-DD') = p_date_value)
    OR
    (p_date_type = 'YEAR' AND TO_CHAR(s.ORDERDATE, 'YYYY') = p_date_value);

IF v_order_count = 0 THEN
    DBMS_OUTPUT.PUT_LINE('No data found for the specified date. ');
    RETURN;
END IF;

-- Output the totals
DBMS_OUTPUT.PUT_LINE('Report for end of ' || LOWER(p_date_type) || ' ' ||
p_date_value);
DBMS_OUTPUT.PUT_LINE(' ');
DBMS_OUTPUT.PUT_LINE('Order Count: ' || v_order_count);
DBMS_OUTPUT.PUT_LINE('Total Sales: RM ' || v_total_sales);
DBMS_OUTPUT.PUT_LINE('Total Profit: RM ' || v_total_profit);
DBMS_OUTPUT.PUT_LINE(' '); -- New line for formatting

-- Cursor to iterate through the ship modes and their counts
DBMS_OUTPUT.PUT_LINE( RPAD('Shipping Mode', 18) || 'Order Count');
FOR r IN (
    SELECT
        s.SHIPMODE,
        COUNT(*) AS mode_count
    FROM
        SHIPPING s
    WHERE
        (p_date_type = 'MONTH' AND TO_CHAR(s.ORDERDATE, 'YYYY-MM') = p_date_value)
        OR
        (p_date_type = 'DAY' AND TO_CHAR(s.ORDERDATE, 'YYYY-MM-DD') =
p_date_value)
        OR
        (p_date_type = 'YEAR' AND TO_CHAR(s.ORDERDATE, 'YYYY') = p_date_value)
    GROUP BY
        s.SHIPMODE
    ORDER BY
        mode_count DESC
) LOOP
    DBMS_OUTPUT.PUT_LINE( RPAD(r.SHIPMODE, 18) || r.mode_count);
END LOOP;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No data found for the specified date. ');
    WHEN OTHERS THEN

```

```
DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);  
END;  
/
```

### 7.5.3 Sample Outputs

#### i. EOY Report

```
Report for end of year 2013  
  
Order Count: 6721  
Total Sales: RM 10476745.39  
Total Profit: RM 1226442.82  
  
Shipping Mode      Order Count  
Standard Class     8254  
Second Class       2838  
First Class        1958  
Same Day           749
```

*Figure 7.13: EOY Report*

#### ii. EOM Report

```
Report for end of month 2011-12  
  
Order Count: 620  
Total Sales: RM 1056592.31  
Total Profit: RM 142788.29  
  
Shipping Mode      Order Count  
Standard Class     689  
Second Class       317  
First Class        209  
Same Day           48
```

*Figure 7.14: EOM Report*



iii. EOD Report

```
Report for end of day 2011-11-11

Order Count: 23
Total Sales: RM 96534.91
Total Profit: RM 7574.84

Shipping Mode      Order Count
Standard Class     35
First Class        14
Second Class       11
Same Day           1
```

*Figure 7.15: EOD Report*

iv. No Data Handling

```
new:DECLARE
    v_type VARCHAR2(10);
    v_date VARCHAR2(15);
BEGIN
    v_type := UPPER('year');
    v_date := '2022';

    end_of_report(v_type, v_date);
END;
No data found for the specified date.
```

*Figure 7.16: Empty Data*

v. Invalid Input Handling

```
An error occurred: ORA-20001: Invalid date type. Use "MONTH", "DAY", or "YEAR".

An error occurred: ORA-20001: Invalid date value. Use YYYY for YEAR.

An error occurred: ORA-20001: Invalid date value. Use YYYY-MM for MONTH.

An error occurred: ORA-20001: Invalid date value. Use YYYY-MM-DD for DAY.
```

*Figure 7.17: Invalid Input*

## CHAPTER 8 REFERENCES

- Global Super Store Dataset. (n.d.). Wwww.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](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](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](https://livesql.oracle.com/apex/livesql/file/content_BCIYBOLU3HSIRATHWNFKWHXIM.html)