**DATABASE MANAGEMENT SYSTEMS**

**Name: A. GAYATRI**

**Roll.no : CB.SC.I5DAS210**

**Github URL of the project page:** [Gayatri-0507 (github.com)](https://github.com/Gayatri-0507)

**Kaggle URL of the dataset page(where the dataset is hosted):** [A. Gayatri | Novice | Kaggle](https://www.kaggle.com/agayatri/datasets)

**1.Section-1**

**Application Name : Myntra**

**Description: This application is to make online shopping easier. Availability of wide range of products like accessories, clothes, shoes etc. The searching process is easier with categories. The products suggested are based on the previous search or previous orders. Here comes the major role of data it should show only the customer’s choice what would the like to buy or what will be better for them. The number of items present of a choice. Discounts available on a particular product or discount on usage of any bank card are all present to make shopping easy with door step service in all brands present across the world.**

**[About the application]**

**Report:**

**List the report names with its purpose**

|  |  |
| --- | --- |
| **Report Name** | **Purpose** |
| Daily Sales Report | This report provides insights into daily sales performance, including revenue, the number of orders, bestselling products, and trends over time. It helps in monitoring the app's daily financial health. |
| User Activity Report | This report analyzes user interactions within the app, including page visits, click-through rates, time spent per session, and popular sections. It aids in understanding user behavior and preferences. |
| Inventory Status | Monitoring stock levels, product availability, and tracking popular items that might need restocking or promotion. It helps in managing inventory efficiently to avoid stockouts or overstocking. |
| Conversion Rate | Evaluating the rate at which app visits convert into actual purchases. It involves analyzing user flows, identifying bottlenecks, and optimizing the app's checkout process. |
| Customer Retention | Tracking returning customers, their purchasing patterns, and measuring the effectiveness of strategies aimed at retaining customers. It helps in fostering loyalty and improving user retention rates. |
| Marketing Campaign | Evaluating the success of different marketing initiatives, such as email campaigns, social media promotions, or discounts. It measures engagement and conversion generated by these campaigns. |
| Trend Analysis | :Identifying popular products, brands, or categories, and analyzing shopping trends based on customer preferences, seasonal variations, or emerging fashion trends. It aids in making informed decisions regarding product assortment and marketing strategies. |

**Technologies**

**SQL Based Application:**

|  |  |
| --- | --- |
| **Front End** | **Editor:** Visual Studio (versatile integrated development environment (IDE) used for front-end    **Language:** HTML, CSS used for front-end development.    **Framework:** HTML, CSS |
| **Back End** | **Editor:** Visual Studio    **Language:** JavaScript    **Framework:** Node.js |

**Why? What?**

**[[Write why this application is required]]**

Myntra is an indispensable application due to its convenience, offering a vast array of fashion products from numerous brands, personalized recommendations, intuitive user interface, exclusive deals, reliable delivery, and hassle-free returns. It's a one-stop destination that caters to diverse fashion needs, providing users with trends, reviews, and a seamless shopping experience from the comfort of their homes.

List of similar applications :

|  |  |
| --- | --- |
| Application Name | URL |
| Ajio | [Online Shopping for Women, Men, Kids – Clothing, Footwear | AJIO](https://www.ajio.com/) |
| Limeroad | [Online Shopping Site - Shop Men & Women Fashion Online in India (limeroad.com)](https://www.limeroad.com/) |
| Amazon Fashion | [Amazon Fashion: Clothing, Footwear and Accessories online for Men, Women and Kids](https://www.amazon.in/b?node=6648217031) |
| Snapdeal | [Shop Online for Men, Women & Kids Clothing, Shoes, Home Decor Items (snapdeal.com)](https://www.snapdeal.com/) |

**2.Section-2**

**DDL,DML,TCL operations**

**Table Details**

**Other than user Table there should be five master and six transaction tables**

**Master Table**

|  |  |
| --- | --- |
| **Table Name** | **Purpose** |
| User | Manages user accounts and facilitates login |
| Customer Details | Stores information of all customers who have shopped |
| Category Details | Serves as a master table for various product categories |
| Payment Methods | Serves as a master table for various payment methods |
| Brands | Serves as a master table for different product brands |
| Promotion | Stores promotional campaign details |
| Order Details | Records transactions and details of customer orders |
| Refund Transaction | Records transactions related to refunds |
| Reviews | Stores customer reviews and ratings |
| Cart | Manages items added to the shopping cart |
| Address | Stores shipping and billing addresses |
| Product | Contains details about various products available |

**For Each master table:<<Five table other than user table>>**

**Table Name: Customer Details**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| customer\_id | NUMBER (10) | Primary key |
| customer\_name | VARCHAR (20) | Not null |
| customer\_address | VARCHAR (15) | Not null |
| Email\_id | VARCHAR (15) | Not null |
| Mobile\_no | NUMBER(10) | Not null |
| Gender | VARCHAR(10) | Not null |
| DOB | date(10) | - |

**Table Name:** category details

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| Categoryid | NUMBER (10) | Primary key |
| categoryname | VARCHAR (20) | Not null |
| description | VARCHAR (100) | Not null |

**Table Name: promotion details**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| promotionid | NUMBER (10) | Primary key |
| promotioncode | VARCHAR (20) | Not null |
| Description | VARCHAR (15) | Not null |
| discountpercentage | VARCHAR (15) | Not null |
| startdate | NUMBER(10) | Not null |
| enddate | VARCHAR(10) | Not null |

**Table Name:** payment method details

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| paymentmethodid | NUMBER (10) | Primary key |
| paymentmethodname | VARCHAR (20) | Not null |
| description | VARCHAR (100) | Not null |

**Table Name:** brand details

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| **Brand-id** | **INTEGER** | **PRIMARY KEY** |
| **Brand-name** | **VARCHAR(100)** | **NOT NULL** |
| **description** | **VARCHAR(50)** | **-** |
| **website** | **VARCHAR(100)** | **UNIQUE** |

**For Each transaction table:<<Six table other than user table>>**

**Table Name:** address details

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| Addressid | NUMBER (10) | Primary key |
| customerid | NUMBER (10) | Foreign Key |
| streetaddress | VARCHAR (100) | Not null |
| city | VARCHAR (15) | Not null |
| state | NUMBER (10) | Primary key |
| zipcode | NUMBER (10) | Not null |
| country | VARCHAR (100) | Not null |

**Table Name:** order details

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| order\_id | INTEGER | Primary key |
| customer\_id | INTEGER | Foreign key |
| order\_date | DATE | Not null |
| total\_amount | DECIMAL(10, 2) | Not null |
| shipping\_address | Varchar(100) | Not null |
| payment\_method | VARCHAR(50) | Not null |
| status | VARCHAR(20) | Not Null |

**Table Name:** refundtransaction details

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| Refund\_id | INTEGER | Primary key |
| Order\_id | INTEGER | Foreign key |
| Refund\_date | DATE | Not null |
| total\_amount | DECIMAL(10, 2) | Not null |
| Refund\_amount | DECIMAL(10, 2) | Not null |
| Reason | VARCHAR(50) | Not null |
| status | VARCHAR(20) | Not Null |

**Table Name:** paymenttransaction details

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| transaction\_id | INTEGER | Primary key |
| order\_id | INTEGER | Foreign key |
| Paymentmethod\_id | INTEGER | Foreign key |
| transaction\_date | timestamp | Not null |
| Amount | DECIMAL(10, 2) | Not null |
| status | VARCHAR(20) | Not Null |

**Table Name:** product details

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| Product\_id | INTEGER | Primary key |
| Brand\_id | INTEGER | Foreign key |
| Category\_id | INTEGER | Foreign key |
| Refund\_date | DATE | Not null |
| Price | DECIMAL(10, 2) | Not null |
| Stock\_quantity | Integer | Not null |
| Description | VARCHAR(100) | Not null |
| Imageurl | VARCHAR(200) | Not Null |

**Table Name:** cart details

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| Cart\_id | INTEGER | Primary key |
| Customer\_id | INTEGER | Foreign key |
| Product\_id | INTEGER | Foreign key |
| Dateadded | timestamp | Not null |
| Quantity | Integer | Not null |

**Operations (DDL,DML,TCL):**

**1.Create**

**2.insert**

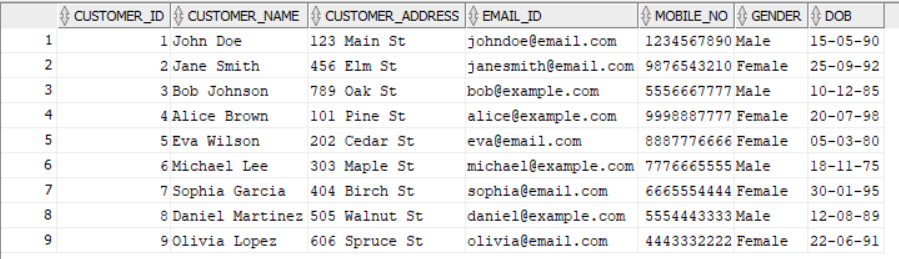
**3.Delete**

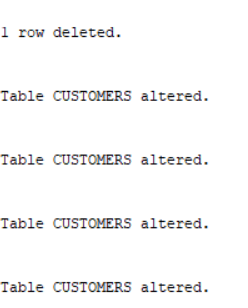
**4.Update**

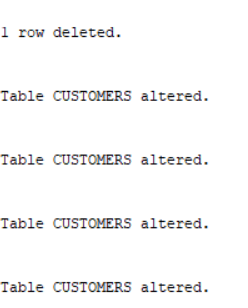
**Delete a row from the table**

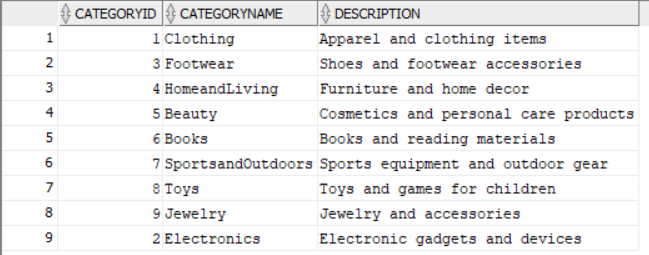
**-- Sql query file name<<**[**basic\_queries.sql**](file:///C:\Users\gayatri\basic_queries.sql) **>>**

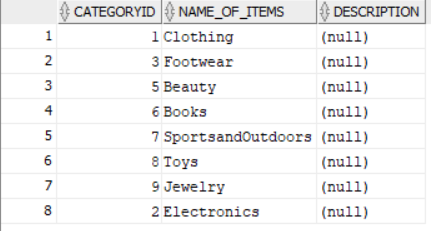
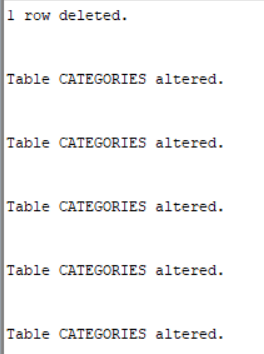
**Screenshot:**

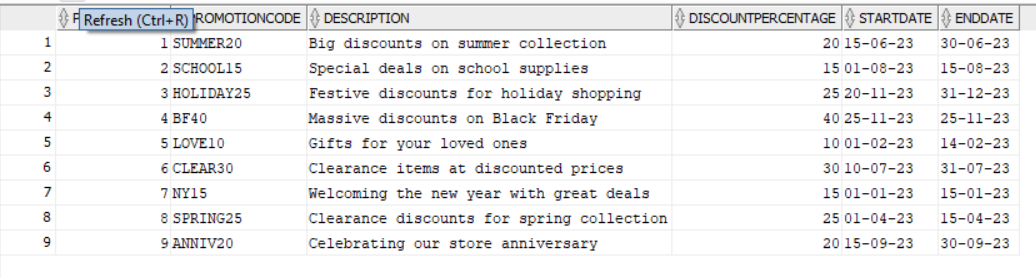
****

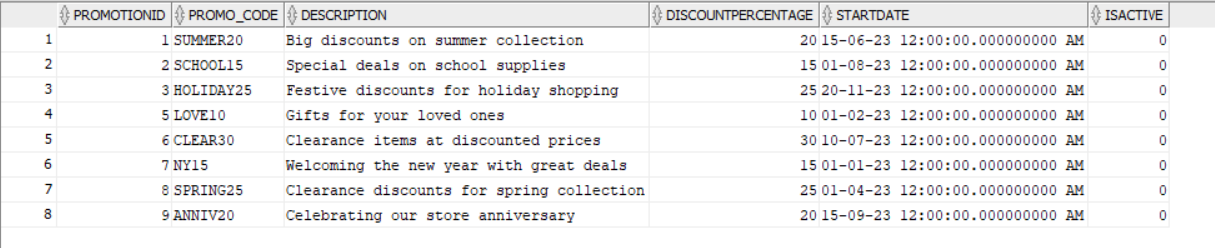
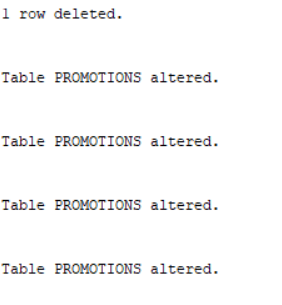
****

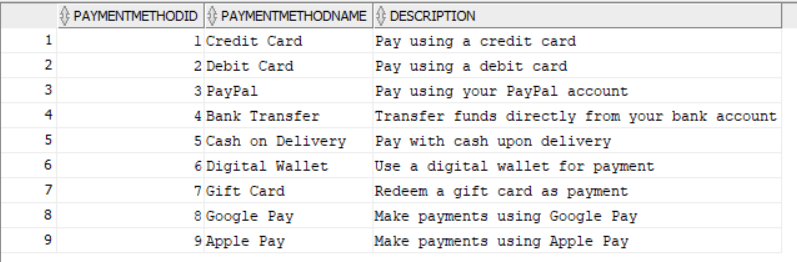
****

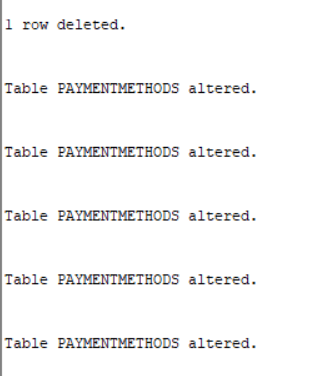
****

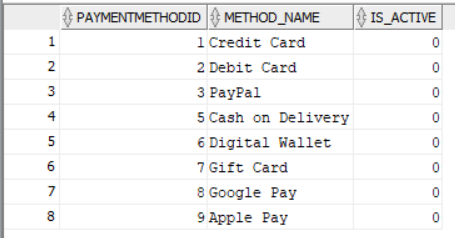
****

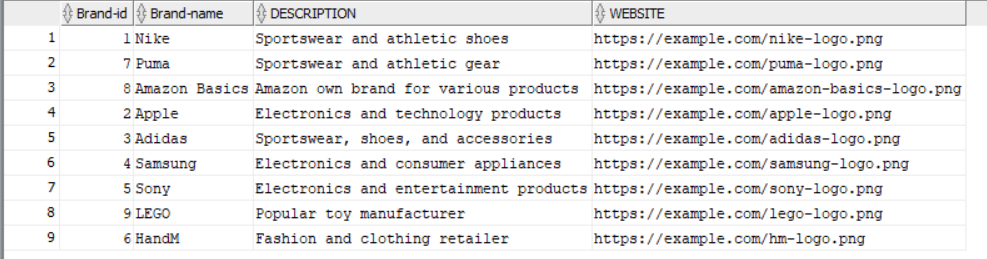
****

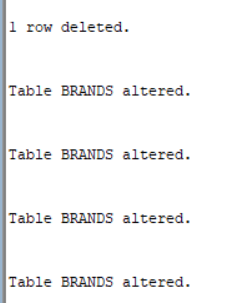
****

****

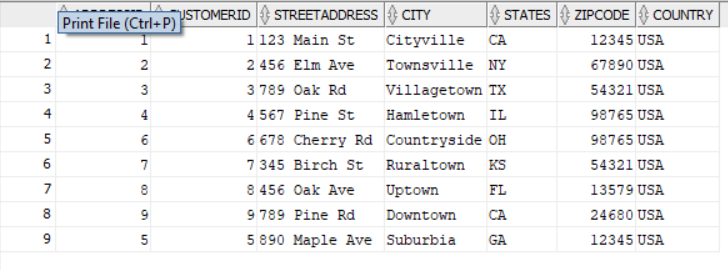


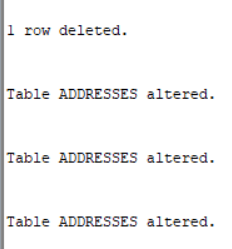




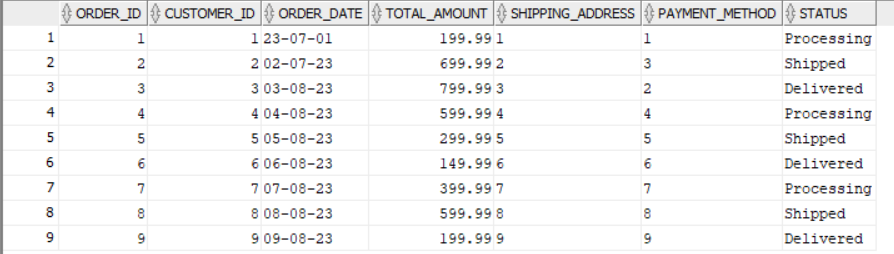
****

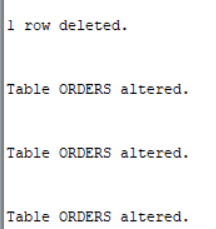
****

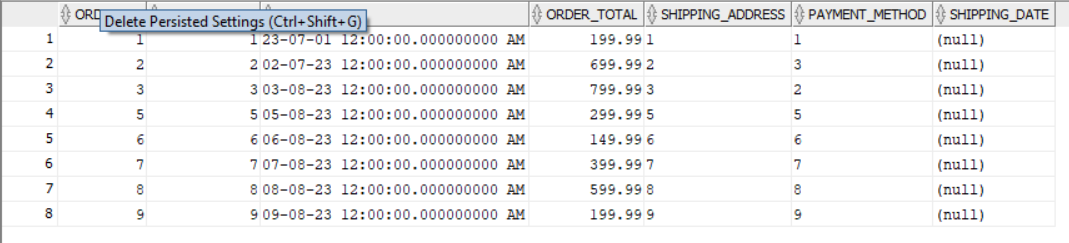


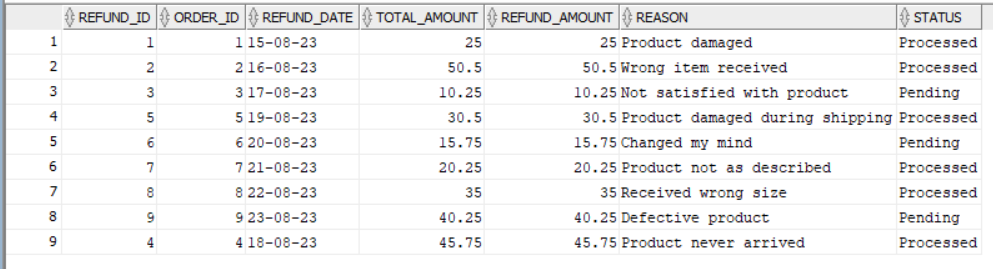


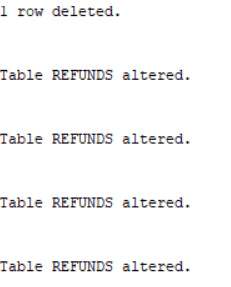
****

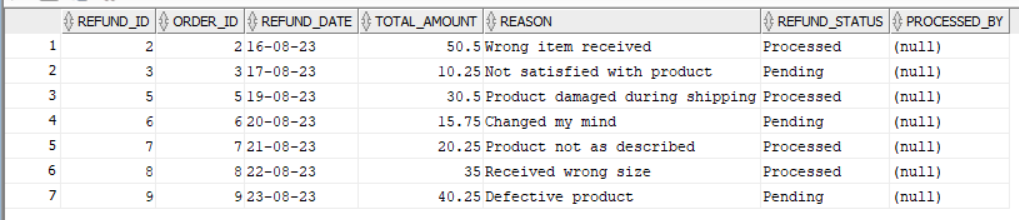
****

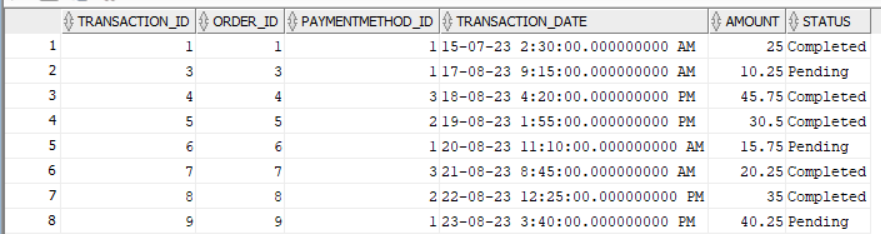
****

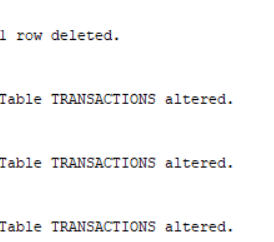
****

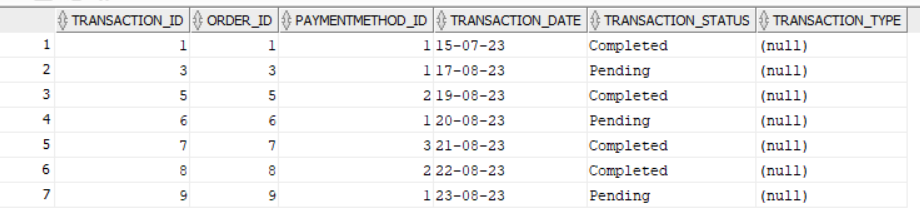
****

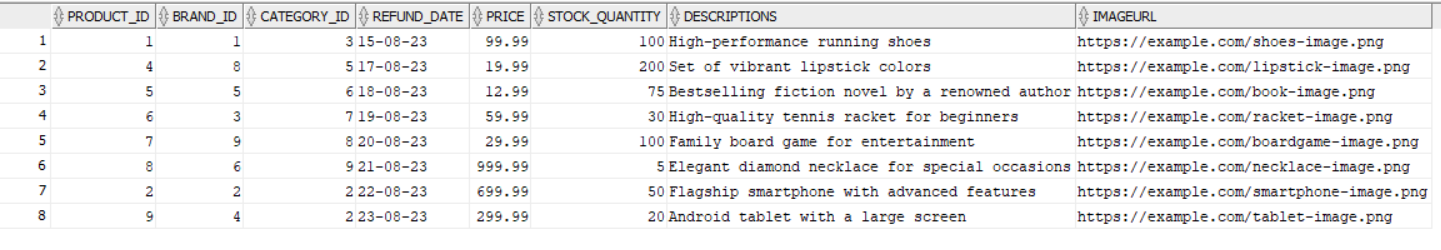
****

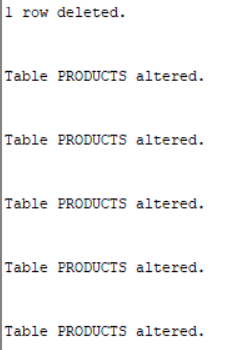
****

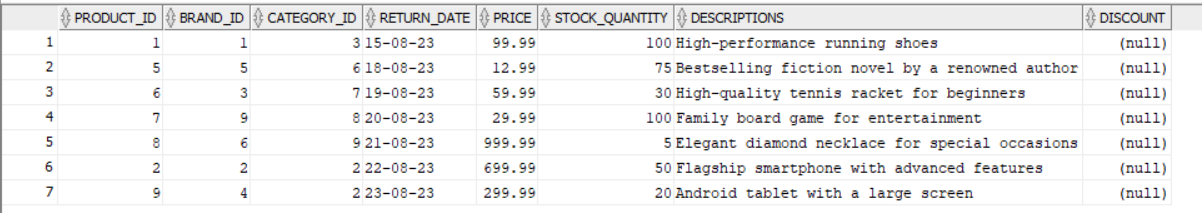
****

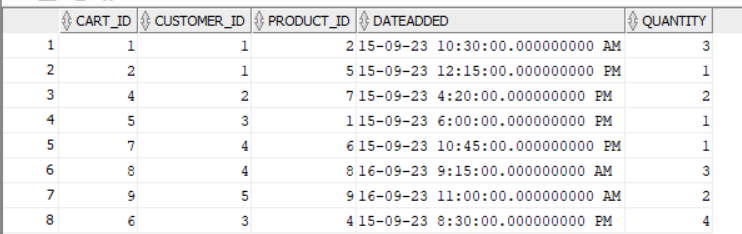


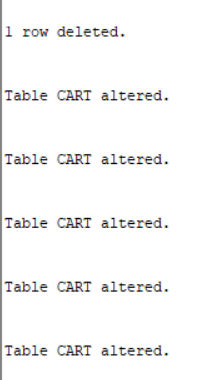


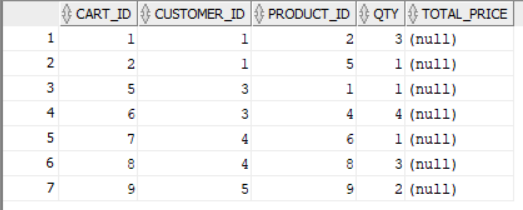
****

****

****

****

****

****

**<<Place outputs of execution>>**

**3.Section-3(Join)**

**-- Sql query file name<<**[**joins.sql**](file:///C:\Users\gayatri\joins.sql) **>>**

**Screenshot:**

**<<Place outputs of execution>> Left join**

**SELECT \***

**FROM Brands**

**LEFT JOIN Promotions ON Brands.Brand-id = Promotions.Brand-id;**

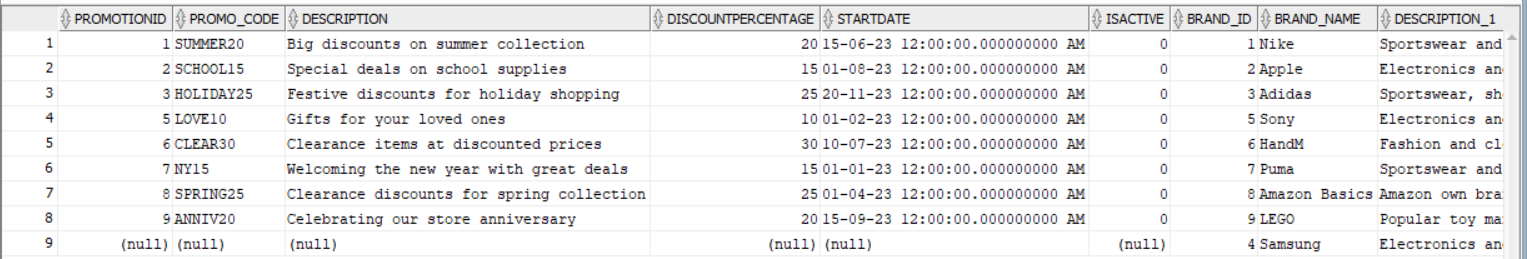
****

**Right join**

**SELECT \***

**FROM Promotions**

**RIGHT JOIN Brands ON Promotions.Brand-id = Brands.Brand-id;**

****

**Full outer join**

**SELECT \***

**FROM Brands**

**FULL OUTER JOIN Promotions ON Brands.Brand-id = Promotions.Brand-id;**

****

**Inference:**

**<<Explain what you would like to explain about the output>>**

LEFT JOIN, indicating that all records from the Refunds table will be included in the result set. The join is based on the order\_id column, linking records between the two tables where the order\_id values match.

Includes all Refunds records.

Shows related Transactions details where order\_id matches.

Displays NULL values for Transactions columns where no match is found

RIGHT JOIN, indicating that all records from the Brands table will be included in the result set.The join is based on the Brand-id column, linking records between the two tables where the Brand-id values match.

The output will contain all records from the Brands table.

For each record in the Brands table, it will display matching records from the Promotions table based on the Brand-id. If there's no matching Brand-id in the Promotions table for a record in Brands, columns from the Promotions table will have NULL values in the output.

FULL OUTER JOIN, which includes all records from both the Brands and Promotions tables, joining them based on the Brand-id column.The join condition is the equality between the Brand-id column in both tables.

The result will contain all records from both the Brands and Promotions tables.

If a Brand-id exists in both tables, the output will display combined data from both tables for that Brand-id.

If a Brand-id exists in only one table, the output will display data from that table and NULL values for the columns from the other table.

Rows where Brand-id matches will be merged, and rows where it doesn't match will still appear in the result with NULL values for non-matching columns from the opposite table.

**4.Section-4(Nested Queries)**

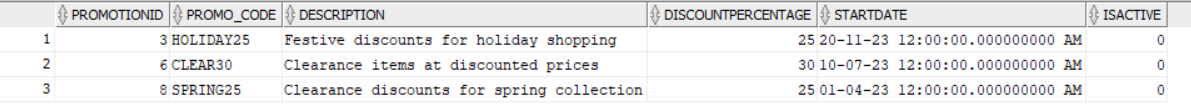
**-- Sql query file name<<**[**nested queries.sql**](file:///C:\Users\gayatri\nested%20queries.sql) **>>**

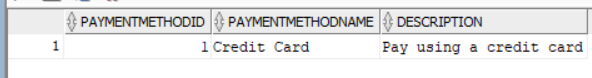
**Screenshot:**

**<<Place outputs of execution>>**

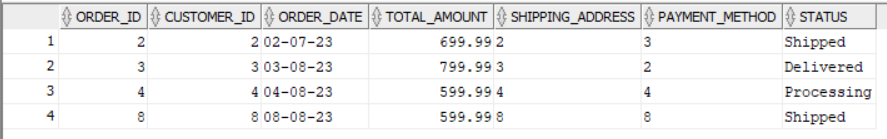


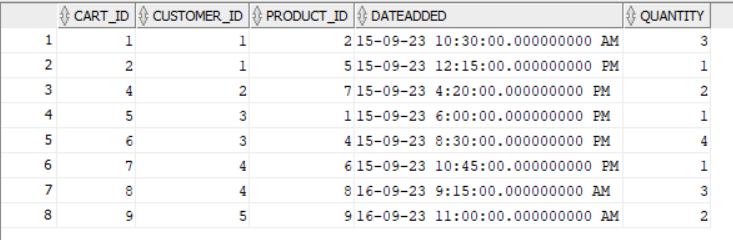




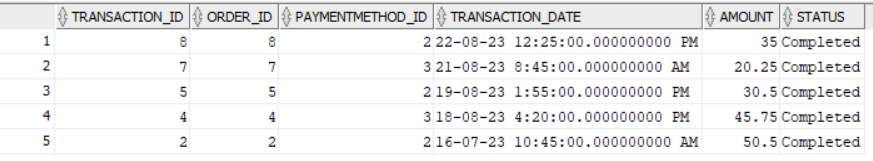










****

**Inference:**

Customers Table:

Retrieves all customers who have placed orders.

Categories Table:

Fetches categories that have products with a price greater than 0.

Promotions Table:

Retrieves promotions with a discount percentage higher than 20.

PaymentMethods Table:

Fetches payment methods used for orders paid by credit card.

Brands Table:

Retrieves products associated with the brand 'Nike'.

Addresses Table:

Retrieves customer details for those residing in 'Cityville'.

Orders Table:

Fetches orders with a total amount greater than the average total amount across all orders.

Cart Table:

Retrieves cart details for customers.Refunds Table:

Retrieves refund details for cases where the reason for refund is 'Received wrong size'.

Transactions Table:

Fetches transactions that have a status of 'Completed'.

Products Table:

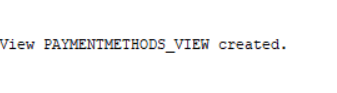
Retrieves products from a category based on a specific product ID (in this case, product\_id=3).

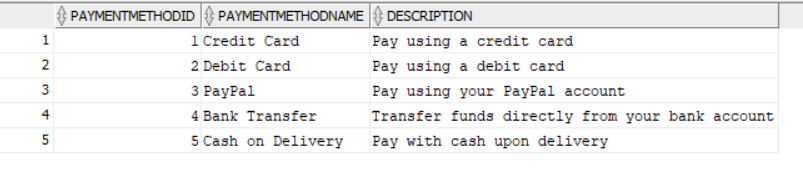
**5.Section-5(view)**

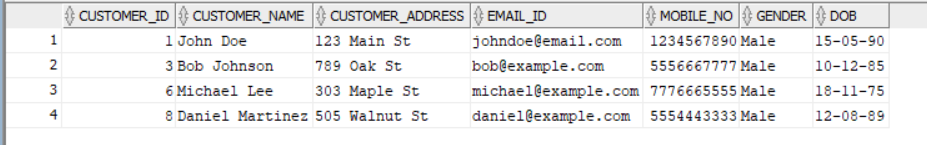
**-- Sql query file name<<**[**view.sql**](file:///C:\Users\gayatri\view%20query.sql) **>>**

**Screenshot:**

**<<Place outputs of execution>>**



**Inference:**

This creates a view named PaymentMethods\_View that selects all columns from the paymentmethods table but filters the records based on a WHERE condition. In this case, it includes only those payment methods with an paymentmethodid less than or equal to 5.

This view will contain all columns from the customers table but will only include records where the Gender column is set to 'Male'

The view are created to display subset of a given table.

**6. Section-6(PLSQL file)**

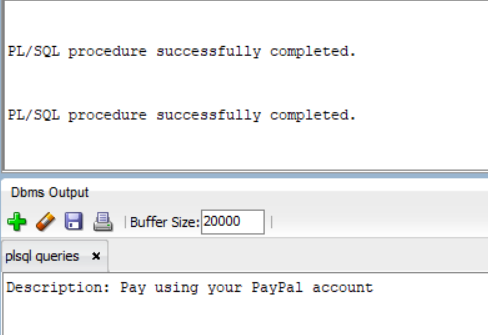
**<<Can be more than one PL/SQL but should have function>>**

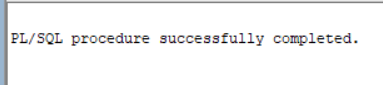
|  |  |
| --- | --- |
| **Function Name** | **GetPaymentMethodDescription**  **Table operated: paymentmethod**  **Expected output: paymentmethodid 3 is printed** |
| **Procedure Name** | **Count\_Male\_Customers**  **Table operated: customers**  **Expected Output: 4 male customers** |

**-- Sql query file name<<plsql >>**

**Screenshot:**

**<<Place outputs of execution>>**

****





**Inference:**

this function is designed to retrieve payment method descriptions based on the provided ID and handle scenarios where the payment method ID is not found or errors occur during the retrieval process

This procedure Count\_Male\_Customers counts the number of male customers in the customers table and displays the count using DBMS\_OUTPUT.PUT\_LINE. Import thing to remember if the output is not given then turn on your server with this code ‘ **set SERVEROUTPUT on;**’

**<<Explain what you would like to explain about the output>>**

**7. Section-7(Cursor file)**

**<<Can be more than one cursor file>>**

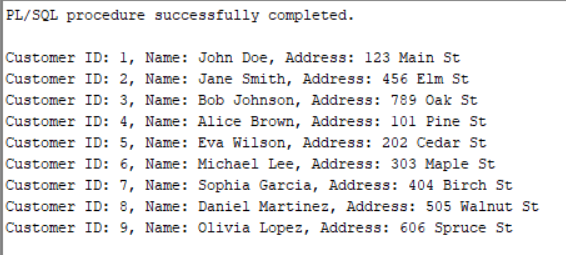
|  |  |
| --- | --- |
| **Cursor Name** | **Explanation of the cursor** |
| **C1** | 1. Declares a cursor named "c1" that selects the customer\_id, customer\_name, and customer\_address from the "customers" table.  2. Declares a record variable "rec" of the same type as the cursor's row.  3. Opens the cursor.  4. Starts a loop to fetch each row from the cursor into the "rec" variable.  5. Exits the loop when there are no more rows to fetch.  6. Prints the customer ID, name, and address using the DBMS\_OUTPUT.PUT\_LINE() function.  7. Closes the cursor.  This code will iterate through each row in the "customers" table and display the customer ID, name, and address for each customer. |
| **Expected Output** | Each line represents a customer record with their corresponding ID, name, and address. The code fetches each row from the cursor and prints this information using the DBMS\_OUTPUT.PUT\_LINE() function. |

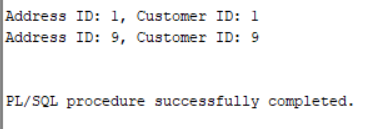
|  |  |
| --- | --- |
| **Cursor Name** | **Explanation of the cursor** |
| c\_addresses | 1. The `DECLARE` keyword is used to begin the declaration section of the PL/SQL block. This is where you declare variables, cursors, and other program elements.  2. Next, you declare a cursor named `c\_addresses` using the `CURSOR` keyword. This cursor is defined to select all rows from the `addresses` table where the `states` column is equal to 'CA'. It acts as a pointer to the result set of the query.  3. After the `BEGIN` keyword, the actual execution of the PL/SQL block begins.  4. The `FOR` loop is used to iterate over each row fetched by the cursor. In this case, the loop variable `address` represents each row of data.  5. Within the loop, the `DBMS\_OUTPUT.PUT\_LINE` statement is used to display the address ID and customer ID for each address. The `||` operator is used for concatenation, combining the text with the values from the `address` record.  6. The `END LOOP` statement marks the end of the loop.  7. Finally, the `/` symbol is used to execute the PL/SQL block.  This code allows you to fetch and process the addresses associated with the state of California from the `addresses` table. |
| **Expected Output** | In your specific code, the cursor output is displaying the `Address ID` and `Customer ID` for each address retrieved from the database. These values can be used for further processing or analysis, depending on the requirements of your application. |

**-- Sql query file name<<**[**cursor queries.sql**](file:///C:\Users\gayatri\cursor.sql) **>>**

**Screenshot:**

**<<Place outputs of execution>>**





**Inference:**

**<<Explain what you would like to explain about the output>>**

**8. Section-8(Trigger file)**

**<<Can be more than one trigger operation>>**

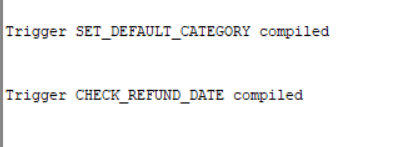
|  |  |
| --- | --- |
| **Cursor Name** | **1.Explanation of the trigger operation**  **2.Type of Trigger** |
| Set\_Default\_Category | **1. The trigger is named "Set\_Default\_Category" and it is associated with the categories table.**  **2. The trigger is defined as a BEFORE INSERT trigger, which means it will be executed before a new row is inserted into the categories table.**  **3. The trigger is set to execute FOR EACH ROW, meaning it will be triggered for each individual row that is being inserted.**  **4. Inside the trigger, there is an IF statement that checks if the value of the description column for the new row, represented by :NEW.description, is NULL.**  **5. If the description value is indeed NULL, the trigger will assign the value 'Default description' to the :NEW.description column for that particular row.**  **6. This ensures that if a description is not provided during the insertion of a new row, the trigger will automatically set a default description of 'Default description'.**  **Type:** **BEFORE INSERT trigger.** |
| **Expected Output** | **First trigger created then It will give error if any change made** |

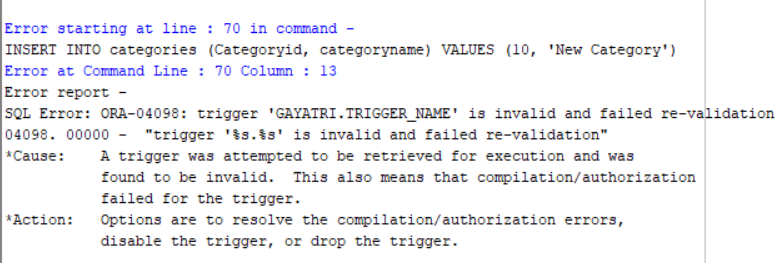
|  |  |
| --- | --- |
| **Cursor Name** | **1.Explanation of the trigger operation**  **2.Type of Trigger** |
| Check\_Refund\_Date | **1. The trigger is created or replaced using the "CREATE OR REPLACE TRIGGER" statement. It is associated with the products table.**  **2. The trigger is defined as a BEFORE INSERT OR UPDATE trigger, meaning it will be executed before a new row is inserted or an existing row is updated in the products table.**  **3. The trigger is set to execute FOR EACH ROW, indicating that it will be triggered for each individual row affected by the insert or update operation.**  **4. Inside the trigger, there is an IF statement that checks if the value of the Refund\_date column for the new or updated row, represented by :NEW.Refund\_date, is less than the current system date (SYSDATE).**  **5. If the Refund\_date value is indeed in the past, the trigger raises an application error using the RAISE\_APPLICATION\_ERROR function. The error code -20001 is specified, and the error message 'Refund date cannot be in the past' is provided.**  **6. This ensures that any attempt to insert or update a row with a Refund\_date in the past will result in an application error being raised.**  **In summary, the trigger "Check\_Refund\_Date" checks if the Refund\_date value for a new or updated row in the products table is in the past, and raises an application error if it is.**  **Type:** **BEFORE INSERT trigger** |
| **Expected Output** | **First trigger create then It will give error if any change made** |

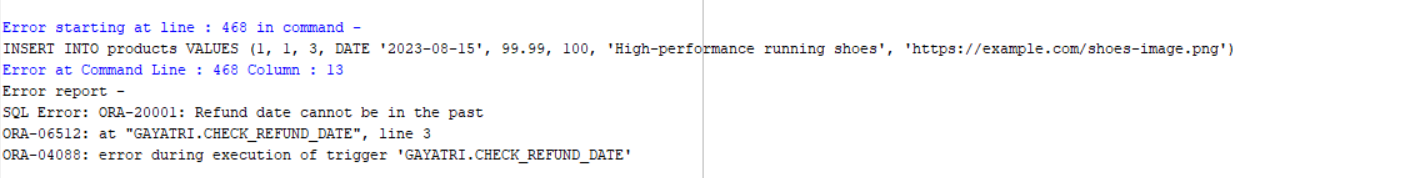
**-- Sql query file name<<**[**triggers query.sql**](file:///C:\Users\gayatri\triggers.sql) **>>**

**Screenshot:**

**<<Place outputs of execution>>**







**Inference:**

**<<Explain what you would like to explain about the output>>**

**9.Section -9**

**<< Web Application>>**

* **5 Master Tables**
* **6 Transaction table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Form Name** | **Category(Master**  **/Transaction)** | **Table Name associated with the form** | **Form Name /css**  **For ex: Emp\_master.html**  **Emp,m\_master.css** | **File Name for reference in the shared drive** | **Type of operations**  **(Insert/Update/Delete**  **/Search/Display)** |
| Customer details form | **Master** | customer details | customer details.html  customer.css | customer details.html  customer.css |  |
| Category details form | **Master** | category details | **Category.html**  **Category.css** | **Category.html**  **Category.css** |  |
| Promotion details form | **Master** | **promotion details** | **Promotions.html**  **Promotion.css** | **Promotions.html**  **Promotion.css** |  |
| Payment method details form | **Master** | payment method details | **Payment method.html**  **Payment.css** | **Payment method.html**  **Payment.css** |  |
| Brand details form | **Master** | brand details | **Brand details.html**  **Brand.css** | **Brand details.html**  **Brand.css** |  |
| Payment transaction form | **Transaction** | Payment transaction details | **Payment transaction.html**  **Payment transaction.css** | **Payment transaction.html**  **Payment transaction.css** |  |
| Refund transaction form | **Transaction** | Refund transaction details | **Refundtransaction.html**  **Refund.css** | **Refundtransaction.html**  **Refund.css** |  |
| Address form | **Transaction** | address details | **Address.html**  **Address.css** | **Address.html**  **Address.css** |  |
| Order details form | **Transaction** | order details | **Order details.html**  **Order.css** | **Order details.html**  **Order.css** |  |
| Cart form | **Transaction** | cart details | **Cart.html**  **Cart.css** | **Cart.html**  **Cart.css** |  |
| Product detail form | **Transaction** | product details | **product details.html**  **product.css** | **product details.html**  **product.css** |  |

**Operations :**

**<<Repeat the 1.insert for all the operations delete,update,search,display>>**

**1.Insert**

**<<keep the screenshot for the form operations for all the forms>>**

**Inference**

**Inserion operation is performed in the form which is reflected in database using js code and the result store for long run.**

**--Write 1-2 sentence explaining the operation shown**

**-- record the video of the working output and keep in the drive**

**10.Section-10(No-SQL Application)**

**10.Section-10(No-SQL Application)** [Nosql tutorial assignment.docx](https://amritavishwavidyapeetham-my.sharepoint.com/:w:/g/personal/cb_sc_i5das21010_cb_students_amrita_edu/EZ34xrlC365Aope5ZzSTin4B-jlQnh3nMI8-2xM_YFt3gw?e=XP2yO8)

**-- Can show for one Table alone**

**Technologies:**

|  |  |
| --- | --- |
| **Front End** | **Back End** |
|  |  |
| **Editor:**Html , Css | **Editor:**Mongodb |
| **Language:**JavaScript | **Language:**Javascript |
| **Framework:** | **Framework:** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Form Name** | **Category(Master**  **/Transaction)** | **Table Name associated with the form** | **Form Name /css**  **For ex: Emp\_master.html**  **Emp\_master.css** | **File Name for reference in the shared drive** | **Type of operations**  **(Insert/Update/Delete**  **/Search/Display)** |
|  |  |  |  |  |  |

**-- record the video of the working output and keep in the drive**

|  |  |  |
| --- | --- | --- |
| **Section No** | **Mark** | **Marks Awarded** |
| **1** | **5** |  |
| **2** | **10** |  |
| **3** | **5** |  |
| **4** | **5** |  |
| **5** | **5** |  |
| **6** | **10** |  |
| **7** | **10** |  |
| **8** | **10** |  |
| **9** | **30** |  |
| **10** | **10** |  |