**DATABASE MANAGEMENT SYSTEMS**

**Name: S JAYANTH SURYA**

**Roll.no : CB.SC.I5DAS21078**

**Github URL of the project page:** [jayanthsurya/Trading\_View-Database: Data of the trading view database and required sources to present it as a webpage (github.com)](https://github.com/jayanthsurya/Trading_View-Database)

1. **Section-1**

**Application Name:**

**TradingView.com -** [ **a platform to view and trade at the prices of assets like stocks, commodity, currencies, etc]**

**Description:**

‘Tradeview.com’ is an application that could be used for trading and financial analysis. It provides a platform for users to view real-time market data and analyse assets, execute trades, and manage their investment portfolios. The application likely includes features like charting tools, technical indicators, watchlists, trade execution, and account management which is an intense model of database management system.

**Report:**

**List of reports with their purposes:**

|  |  |
| --- | --- |
| **Report Name** | **Purpose** |
| User portfolio report | Summary of assets held and value of portfolio for each user |
| Order activity status | Daily, weekly or monthly reports on number of orders and order status |
| Trading analytics | Analyze profit/loss, best/worst performers, trading frequency etc. per user |
| Account asset allocation | Breakdown of asset types in each account and their percentage allocation |
| Geographic portfolio trends | Portfolio value and composition analysed across user locations |
| Account status | Active, inactive and locked user accounts and related account details |
| Asset price volatility | Historical volatility reports and risk analysis for each asset |

**Technologies:**

**SQL-Based Application:**

|  |  |
| --- | --- |
| **Front End** | HTML, CSS |
| **Back End** | Oracle(11.2.0.1) |
| **Editor** | Visual Studio Code |
| **Language** | JavaScript |
| **Framework** | Node.js |
| **Web Server** | Google Web Server |

**How would this application be efficient to the users:**

For ordinary investors wishing to efficiently track, manage, and expand their investment portfolios, this stock trading app meets a vital requirement. Through the centralization of portfolio data, account information, market research tools, order execution platforms, and other essential elements, users are enhanced with confidence and control to execute trading strategies that are in line with their financial objectives. Its user-friendly interfaces, which are accessible on both mobile and web platforms, reduce obstacles to entry and encourage more active participation from investors of various backgrounds. Robust back testing features promote experimentation without putting actual money at risk. Through the maintenance of an extensive database covering users, assets, orders, transactions, and interfaces with market data streams, this program can provide comprehensive reporting that provides actionable insights.

By using data specific to their interests, users may keep an eye on their portfolios, evaluate their trading performance, and adjust their methods. With its affordable prices and emphasis on automation and ease of use, it hopes to draw in new investor demographics and hold on to seasoned ones with its premium solutions. Ultimately, the goal of this application is to support the values of openness, accessibility, and analytical rigor as cornerstones that empower more people with the information and resources they need to safely unleash the potential for wealth creation through stock investment.

In conclusion, this stock trading platform seeks to level the playing field for individual investors by offering a user-friendly, insights-driven interface for portfolio construction, risk management, skill development, and, eventually, the achievement of increased financial prosperity.

List of similar applications :

|  |  |
| --- | --- |
| Application Name | URL |
| E\*Trade | https://us.etrade.com/ |
| TD Ameritrade | https://www.tdameritrade.com/ |
| Charles Schwab | https://www.schwab.com/ |
| Merrill Edge | https://www.merrilledge.com/ |

1. **Section-2**

**DDL,DML,TCL operations**

**Table Details:**

**Master Tables:**

|  |  |
| --- | --- |
| **Table Name** | **Purpose** |
| userdetails | Stores registered user information such as name, contact info, username, password etc. |
| assets | Holds details about each asset or security such as stocks, bonds, ETFs etc. |
| account | Maintains details of accounts created per user to manage transactions |
| location | Captures location information like country, state and city for each user |
| bank | Contains list of banking details required for deposits & withdrawals |
| exchange | Defines the stock exchanges on which assets are traded |

**For Each master table:**

**Table Details:** Master Table

**Table Name:** Userdetails

**Purpose:** This table stores information about registered users of the Tradeview.com application.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| user\_id | NUMBER (10) | Primary key |
| user\_name | VARCHAR (20) | Unique, Not null |
| email\_address | VARCHAR (15) | Unique, Not null |
| password | VARCHAR(10) | Not null |
| contact | NUMBER(10) | Unique, Not null |
| Account\_type | VARCHAR(10) | Not null |

**Table Details:** Master Table

**Table Name:** Assets\_details

**Purpose:** It provides information about tradable assets

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| asset\_id | INTEGER | Primary key |
| asset\_name | VARCHAR (20) | Unique, Not null |
| asset\_type | VARCHAR(20) | Not null |

**Table Details:** Master Table

**Table Name:** Account\_details

**Purpose:** It provides information about details of user accounts

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| account\_id | INTEGER | Primary key |
| user\_id | INTEGER | Foreign key, Unique,  Not null |
| account\_type | VARCHAR(20) | Not null |
| Account\_status | VARCHAR(20) | Not null |

**Table Details:** Master Table

**Table Name:** Location

**Purpose:** It provides information about location of users

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| location\_id | INTEGER | Primary key |
| country | VARCHAR(20) | Not null |
| state | VARCHAR(20) | Not null |
| city | VARCHAR(20) | Not null |

**Table Details:** Master Table

**Table Name:** Bank

**Purpose:** It provides information about bank details of users

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| bank\_id | INTEGER | Primary key |
| bank\_name | VARCHAR(20) | Not null |
| bank\_address | VARCHAR(20) | Not null |

**Table Details:** Master Table

**Table Name:** Exchange

**Purpose:** It provides information about stock exchanges.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| exchange\_id | INTEGER | Primary key |
| exchange\_name | VARCHAR(20) | Not null |
| exchange\_country | VARCHAR(20) | Not null |

**Transaction Tables:**

|  |  |
| --- | --- |
| **Table Name** | **Purpose** |
| portfolio\_details | Tracks portfolio holdings per user with asset, quantity and price |
| orders | Stores details about buy/sell orders placed for assets by users |
| trades | Records trade transactions resulting from order executions |
| deposits | Logs money deposit transactions funded into user accounts |
| withdrawals | Tracks money withdrawals transactions made from user accounts |
| fees | Maintains any fees deducted from the user accounts |

**For each Transaction Table:**

**Table Details:** Transaction Table

**Table Name: Portfolio** details

**Purpose:** It provides the detailed holdings

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| portfolio\_id | INTEGER | Primary key |
| User\_id | INTEGER | Foreign key, Not null |
| Asset\_id | INTEGER | Foreign key, Not null |
| quantity | INTEGER | Not null |
| price | DECIMAL | Not null |

**Table Details:** Transaction Table

**Table Name:** Orders

**Purpose:** It provides information about order details placed by users.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| order\_id | INTEGER | Primary key |
| User\_id | INTEGER | Foreign key, Not null |
| Asset\_id | INTEGER | Foreign key, Not null |
| Order\_type | VARCHAR | Not null |
| Order\_status | VARCHAR | Not null |

**Table Details:** Transaction Table

**Table Name:** Trades

**Purpose:** It provides information about trade execution details against orders.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| trade\_id | INTEGER | Primary key |
| order\_id | INTEGER | Foreign key, Not null |
| Trade\_price | DECIMAL | Not null |
| Trade\_volume | INTEGER | Not null |

**Table Details:** Transaction Table

**Table Name:** Deposits

**Purpose:** It provides information about deposit transactions made by users.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| deposit\_id | INTEGER | Primary key |
| user\_id | INTEGER | Foreign key, Not null |
| Bank\_id | INTEGER | Foreign key, Not null |
| Deposit\_amount | DECIMAL | Not null |
| Deposit\_date | DATE | Not null |

**Table Details:** Transaction Table

**Table Name:** Withdrawls

**Purpose:** It provides information about withdrawal transactions made by users.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| withdrawal\_id | INTEGER | Primary key |
| user\_id | INTEGER | Foreign key, Not null |
| Bank\_id | INTEGER | Foreign key, Not null |
| withdrawal\_amount | DECIMAL | Not null |
| withdrawal\_date | DATE | Not null |

**Table Details:** Transaction Table

**Table Name:** Fees

**Purpose:** It provides information about fee transactions charged to users.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** |
| fee\_id | INTEGER | Primary key |
| user\_id | INTEGER | Foreign key, Not null |
| Fee\_amount | DECIMAL | Foreign key, Not null |
| Fee\_description | VARCHAR(100) | Not null |

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

**1.Create**

**2.insert**

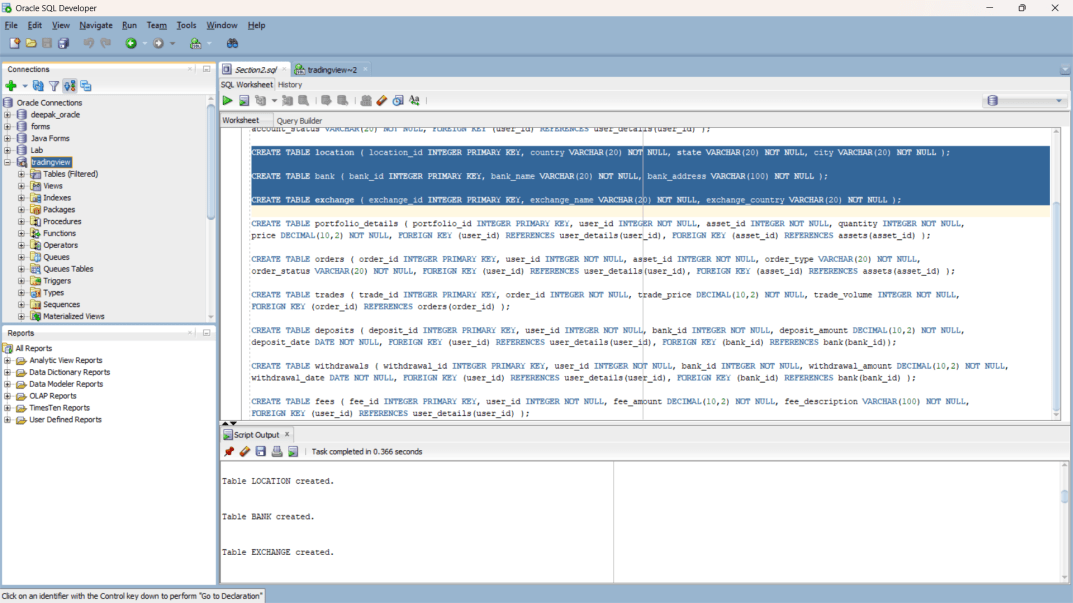
**3.delete**

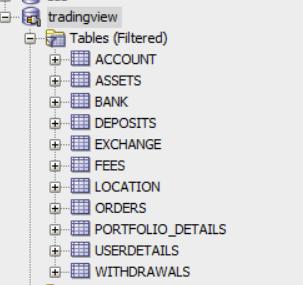
**4.update**

**Sql query file named: “Section2.sql”**

1. **create:**

**12 tables including 1 - user table, 5 - master and 6 - transaction tables has been created as mentioned in above descriptions.**

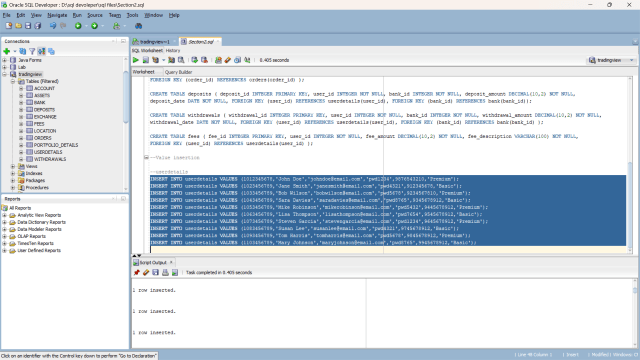


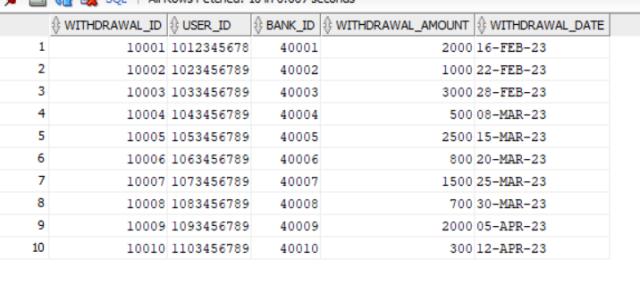


1. **insert:**

**10 rows of data have been inserted into each of the 12 tables.**

**(code in sql file)**

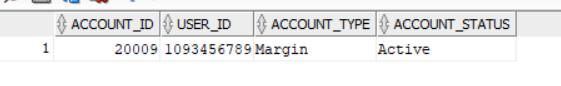




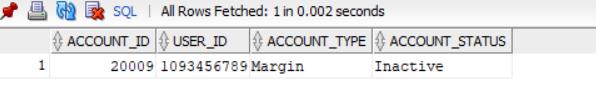
1. **update:**

**A row in the account table has been updated from active state to inactive. (sql code in sql file)**

**Before updation:**



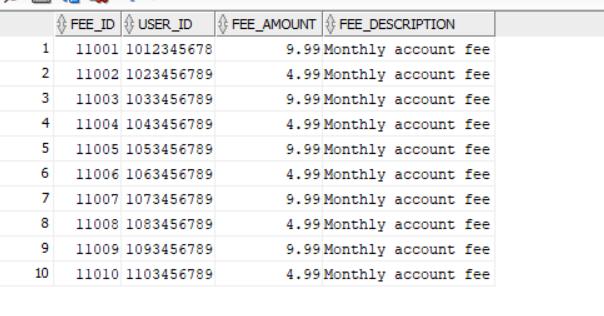
**After updation:**



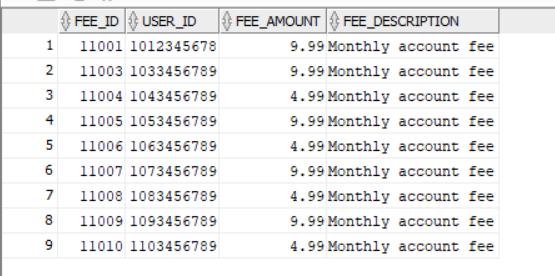
**4.delete:**

**A row from the fees table has been deleted. (code in sql file)**

**Before deletion:**



**After deletion: (row with fee id 11002 is deleted)**



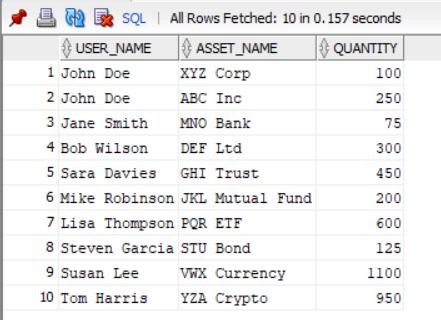
**3.Section-3(Join)**

**Sql query file name Section3.sql**

**TYPES OF JOINS:**

* **INNER JOIN:**

**Output:**

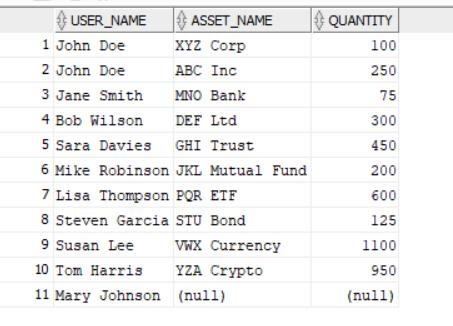


**Inference:**

Retrieves user names, asset names, and quantities from the **portfolio\_details** table, connecting users to their portfolios and assets where user IDs and asset IDs match.

* **LEFT JOIN:**

**Output:**

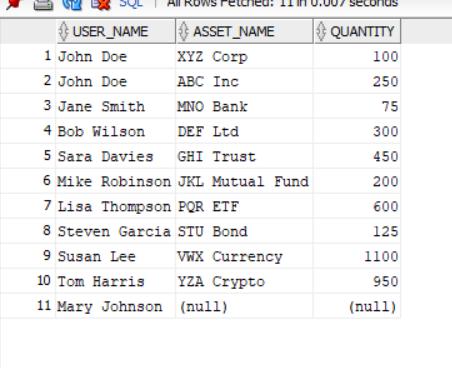


**Inference:**

Returns user names, asset names, and quantities from the **portfolio\_details** table, including unmatched user details. If there are no matching portfolio details, columns from the **portfolio\_details** and **assets** tables will contain NULL values.

* **RIGHT JOIN:**

**output:**

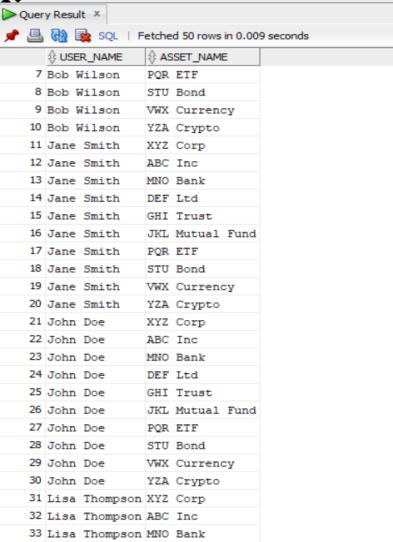


**Inference:**

Fetches user names, asset names, and quantities, prioritizing details from the **assets** table. Includes all asset details even if there are no matching portfolio details. If there are no matching user details, columns from the **userdetails** table will contain NULL values.

* **CROSS JOIN:**

**OUTPUT:**

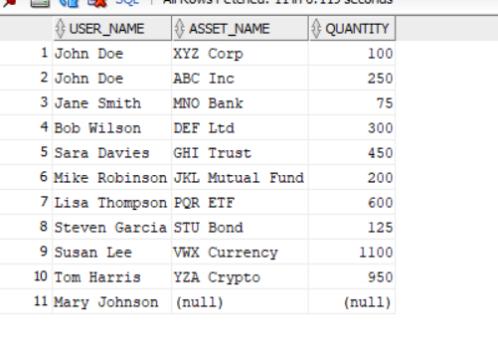


**Inference:**

Produces a Cartesian product of user details and assets, pairing each user with every asset, resulting in all possible combinations of user names and asset names.

* **FULL OUTER JOIN:**

**Output:**



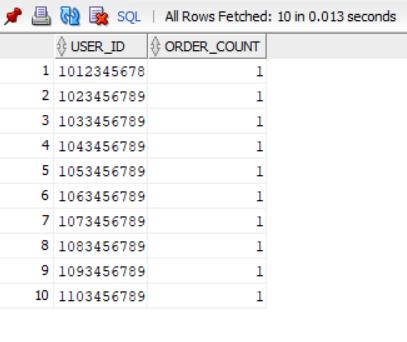
**Inference:**

Retrieves user names, asset names, and quantities, combining details from both **userdetails** and **assets**. Includes unmatched user or asset details, filling in NULL values where there are no matches in either the **userdetails** or **assets** tables.

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

**Sql query file name Section4.sql**

**Output:**



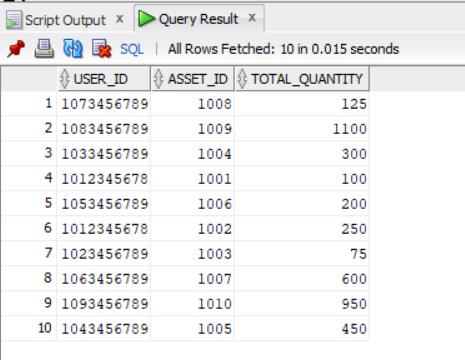
**Inference:**

This SQL query retrieves a list of **user\_id** values along with the corresponding count of orders placed by each user. The subquery **(SELECT Count(\*) FROM orders WHERE user\_id = o.user\_id)** is used to calculate the order count for each user by correlating the outer query's **user\_id** with the **user\_id** in the subquery. The result set includes one row for each unique **user\_id** in the **orders** table, with an additional column named **order\_count** indicating the total number of orders placed by each user.

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

**Sql query file name Section5.sql**

**Output:**



**Inference:**

This SQL code creates a view named **user\_portfolio** by summarizing the total quantity of assets held by each user in their portfolio. The view is constructed from the **portfolio\_details** table, grouping the data by **user\_id** and **asset\_id**. The **SUM(quantity)** aggregates the quantity of each asset for each user, resulting in the total quantity of each asset held by each user. The view will present a simplified and aggregated perspective of user portfolios, making it easier to analyze and query the total quantity of assets for each user.

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

|  |  |
| --- | --- |
| **Function Name** | This is an anonymous PL/SQL block and does not have a specific function or procedure name. |
| **Procedure Name** | This is an anonymous PL/SQL block and does not have a specific function or procedure name. |
| **Expected Output** | The expected output is the total number of users in the **userdetails** table, printed to the output console using **dbms\_output.put\_line**. |
| **Tables operated** | The code interacts with the **userdetails** table to retrieve the count of users. |

**Sql query file name Section6.sql**

**Code to display:**

**DECLARE**

**user\_count INTEGER;**

**BEGIN**

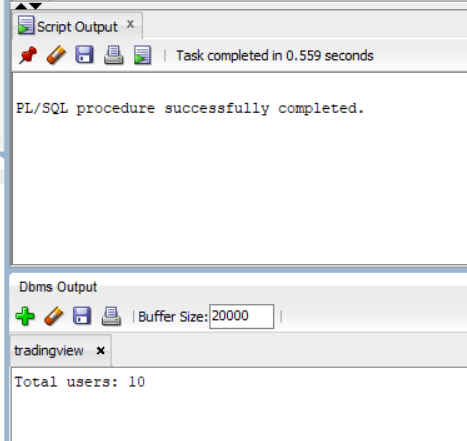
**SELECT Count(\*) INTO user\_count**

**FROM userdetails;**

**dbms\_output.put\_line('Total users: ' || user\_count);**

**END;**

**Output:**



**Inference:**

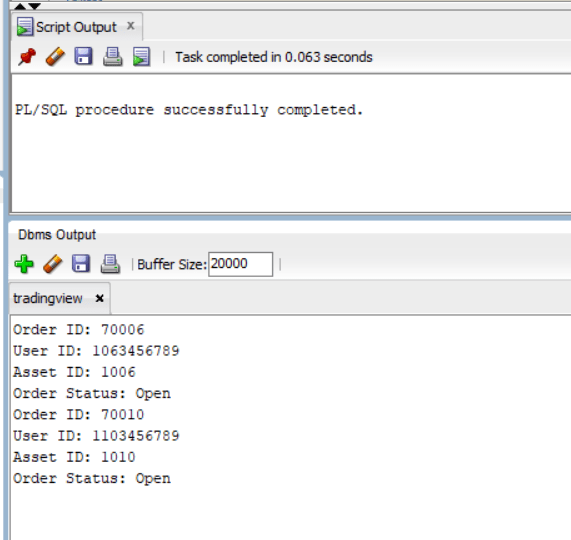
This PL/SQL block counts the number of rows in the **userdetails** table, representing the total number of users. The count is stored in the **user\_count** variable. The **dbms\_output.put\_line** statement is used to display the total number of users to the output console. The block is designed for informational purposes, providing a quick overview of the total user count in the database.

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

|  |  |
| --- | --- |
| **Cursor Name** | **Explanation of the cursor** |
| **c1** | This PL/SQL cursor retrieves and processes details of orders with a 'Open' status. It iterates through the result set, displaying information such as order ID, user ID, asset ID and order status for each pending order using DBMS\_OUTPUT.PUT\_LINE. |
| **Expected Output** | The cursor will display information about orders with a 'Open’ status, including Order ID, User ID, Asset ID and Order status. |

**Sql query file name Section7.sql**

**Output:**



**Inference:**

This PL/SQL code uses a cursor to dynamically retrieve and display details of orders with a 'Open' status.

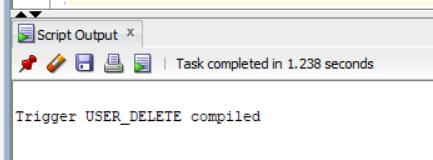
**8. Section-8(Trigger file)**

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

|  |  |
| --- | --- |
| **Cursor Name** | **1.Explanation of the trigger operation**  **2.Type of Trigger** |
| **customer\_insert\_trigger** | 1. This trigger fires before a delete on the user\_details table, capturing and logging changes in orders and portfolio\_details table.  2. BEFORE DELETE trigger. |
| **Expected Output** | Deletes row of same user id in the tables orders and portfolio\_details when a row is deleted in user\_details. |

**-- Sql query file name Section8.sql**

**Screenshot:**



**Inference:**

This is a row-level before delete trigger defined on the userdetails table. When a delete occurs on a specific user\_details record, this trigger gets invoked before that user row is deleted. It then propagates the delete operation to the associated orders and portfolio\_details records for that particular user, by matching the :old.user\_id field.

**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\_master.css** | **File Name for reference in the shared drive** | **Type of operations**  **(Insert/Update/Delete**  **/Search/Display)** |
| User\_details | Master Table | User\_details.html | User\_details.css | User\_details | Insert ,search ,display ,reset |
| Assets\_details | Master Table | Assets\_details.html | Assets\_details.css | Assets\_details.html | Insert ,search ,display ,reset |
| Account\_details | Master Table | Account\_details.html | Account\_details.css | Account\_details.html | Insert ,search ,display ,reset |
| Location | Master Table | Location.html | Location.css | Location.html | Insert ,search ,display ,reset |
| Bank | Master Table | Bank.html | Bank.css | Bank.html | Insert ,search , reset |
| Exchange | Master Table | Exchange.html | Exchange.css | Exchange.html | Insert ,search ,display ,reset |
| Portfolio\_details | Transaction Table | Portfolio\_details.html | Portfolio\_details.css | Portfolio \_details.html | Insert ,Delete ,search ,reset ,display |
| Orders | Transaction Table | Orders.html | Orders.css | Orders.html | Insert ,search ,display ,reset |
| Trades | Transaction Table | Trades.html | Trades.css | Trades.html | Insert ,search ,display ,reset |
| Deposits | Transaction Table | Deposits.html | Deposits.css | Deposits.html | Insert ,delete ,search ,reset ,display |
| Withdrawls | Transaction Table | Withdrawls.html | Withdrawls.css | Withdrawls.html | Insert ,Delete ,search ,reset ,display |
| Fees | Transaction Table | Fees.html | Fees.css | Fees.html | Insert ,search , reset, display |

**Database connectivity info:**

**Form name: user\_details.html**

**Java script: user\_details.js**

**Sql query: Section9.sql**

**Operations :**

**1.Insert**

**Updated:** --entered data are successfully registered to the database.

**Display:** --reference for the data been insert into the table named ‘userdetails’

**Inference:**

I have inserted user details into the user\_details.html form with required fields and the values has been registered in the backend sql and the values stored has been displayed in the last screenshot.

**-- the video of the working output has been kept in the drive.**

**FOR REFERENCE video filename: (SQL Application)**

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

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

**Technologies:**

|  |  |
| --- | --- |
| **Front End** | **Html version 5, css** |
| **Back End** | **Mongodb** |
| **Editor** | **Visual studio code** |
| **Language** | **Java script** |
| **Framework** | **Node js** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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)** |
| **User\_details** | **master** | **Userdetails** | **User\_details.html**  **User\_details.js** | **PROJECT FILES - NO SQL Application/ user\_details.html** | **Insert** |

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

**For reference video filename:(NOSQL Application)**

|  |  |  |
| --- | --- | --- |
| **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** |  |