**DATABASE MANAGEMENT SYSTEMS**

**Name: Uthayarahavan M S**

**Roll.no : CB.SC.I5DAS21065**

**Github URL of the project page:**

**Kaggle URL of the dataset page(where the dataset is hosted):**

**1.Section-1**

**Application Name : MyStore**

**Description:**

The MyStore – Welcome to MyStore, your one-stop destination for all your shopping needs! Discover a world of books, movies, products, courses, restaurants, and more, all in one convenient app. Whether you're a bookworm, a movie enthusiast, a tech geek, a foodie, or someone looking to enhance your skills, MyStore has something for everyone.

**Report:**

**List the report names with its purpose**

|  |  |
| --- | --- |
| **Report Name** | **Purpose** |
| **Wide Variety of Products** | The app hosts an extensive collection of books, movies, products, courses, and restaurants, ensuring there's something for every user. |
| **Entertainment and Learning** | Users can indulge in their favorite books and movies, as well as educational applications. |
| **Seamless Shopping Experience** | MyStore provides an easy and secure checkout process for purchasing products, ensuring a hassle-free shopping experience. |
| **Personalized Recommendations** | The app offers personalized recommendations based on user preferences, helping users discover new content and products. |
| **Transaction Tracking** | Users can track their transactions, making it convenient to manage and review past purchases. |

**Technologies**

**SQL Based Application:**

|  |  |
| --- | --- |
| **Front End** | HTML 5.5, CSS |
| **Back End** | ORACLE SQLDEVELOPER |
| **Editor** | VS CODE |
| **Language** | JAVASCRIPT |
| **Framework** | NODE JS |

**Why?**

**Convenience and Variety:** The primary goal of MyStore is to provide users with a convenient platform where they can access a wide variety of products and services without having to visit multiple stores or platforms.

1. **Catering to Diverse Interests:** MyStore caters to diverse interests, targeting audiences with different preferences, such as bookworms, movie enthusiasts, tech geeks, foodies, and individuals seeking educational courses.
2. **Centralized Shopping Hub:** The application aims to serve as a centralized hub for users to fulfill their various shopping needs, making it easier for them to explore and purchase different types of items.

**What?**

1. **Product and Service Categories:**
   * **Books:** Offering a collection of books with details like title, author, publication date, genre, and price.
   * **Movies:** Providing information about movies, including title, director, release date, genre, and price.
   * **Products:** Showcasing various products with details like name, brand, manufacture date, category, and price.
   * **Courses:** Featuring educational courses with information on the course name, instructor, start date, category, and price.
   * **Restaurants:** Listing restaurants with details like name, cuisine, opening date, rating, and price range.
2. **User Transactions:**
   * Implementing tables like **transactions\_books**, **transactions\_movies**, etc., to track user transactions, including the purchased items, transaction date, and purchase price.
3. **User Registration and Authentication:**
   * Enabling users to register and log in securely to access personalized features and track their transactions.
4. **Dynamic Web Interface:**
   * Creating dynamic web interfaces for each category to allow users to browse, search, and purchase items conveniently.
5. **Database Schema:**
   * Designing a relational database with tables like **users**, **master\_books**, **master\_movies**, etc., to store and manage data efficiently.

In summary, MyStore is designed to offer a centralized and diverse shopping experience, catering to users with different interests, and providing a seamless platform for exploring and purchasing a wide range of products and services.

List of similar applications :

My application is the integration of all the other application.

|  |  |
| --- | --- |
| Application Name | URL |
| Amazon | [Online Shopping site in India: Shop Online for Mobiles, Books, Watches, Shoes and More - Amazon.in](https://www.amazon.in/) |
| Netflix | [Netflix India – Watch TV Shows Online, Watch Movies Online](https://www.netflix.com/in/) |
| Coursera | [Coursera | Online Courses & Credentials From Top Educators. Join for Free](https://www.coursera.org/) |
| Zomato | [Best Restaurants in India - Zomato](https://www.zomato.com/india) |

**2.Section-2**

**DDL,DML,TCL operations**

**Table Details**

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

**Users Table:**

| Column Name | Datatype | Constraint |
| --- | --- | --- |
| username | VARCHAR2(50) | PRIMARY KEY NOT NULL |
| email | VARCHAR2(100) | UNIQUE |
| password | VARCHAR2(100) | NOT NULL |
| first\_name | VARCHAR2(50) |  |
| last\_name | VARCHAR2(50) |  |
| gender | VARCHAR2(10) |  |
| registration\_date | DATE |  |
| last\_login | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

**Master Table**

|  |  |
| --- | --- |
| **Table Name** | **Purpose** |
| **master\_books** | **This is the table where all the details of the books are stored.** |
| **master\_movies** |  |
| **master\_products** |  |
| **master\_courses** |  |
| **master\_restaurants** |  |

**Each master table:**

**Master Books Table:**

| **Column Name** | **Datatype** | **Constraint** |
| --- | --- | --- |
| book\_id | NUMBER | PRIMARY KEY |
| book\_title | VARCHAR2(100) | NOT NULL |
| author | VARCHAR2(100) |  |
| publication\_date | DATE |  |
| genre | VARCHAR2(50) |  |
| price | NUMBER(10, 2) | CHECK (price >= 0) |

**Master Movies Table:**

| **Column Name** | **Datatype** | **Constraint** |
| --- | --- | --- |
| movie\_id | NUMBER | PRIMARY KEY |
| movie\_title | VARCHAR2(100) | NOT NULL |
| director | VARCHAR2(100) |  |
| release\_date | DATE |  |
| genre | VARCHAR2(50) |  |
| price | NUMBER(10, 2) | CHECK (price >= 0) |

**Master Products Table:**

| **Column Name** | **Datatype** | **Constraint** |
| --- | --- | --- |
| product\_id | NUMBER | PRIMARY KEY |
| product\_name | VARCHAR2(100) | NOT NULL |
| brand | VARCHAR2(100) |  |
| manufacture\_date | DATE |  |
| category | VARCHAR2(50) |  |
| price | NUMBER(10, 2) | CHECK (price >= 0) |

**Master Courses Table:**

| **Column Name** | **Datatype** | **Constraint** |
| --- | --- | --- |
| course\_id | NUMBER | PRIMARY KEY |
| course\_name | VARCHAR2(100) | NOT NULL |
| instructor | VARCHAR2(100) |  |
| start\_date | DATE |  |
| category | VARCHAR2(50) |  |
| price | NUMBER(10, 2) | CHECK (price >= 0) |

**Master Restaurants Table:**

| **Column Name** | **Datatype** | **Constraint** |
| --- | --- | --- |
| restaurant\_id | NUMBER | PRIMARY KEY |
| restaurant\_name | VARCHAR2(100) | NOT NULL |
| cuisine | VARCHAR2(100) |  |
| opening\_date | DATE |  |
| rating | NUMBER(3, 2) | CHECK (rating >= 0 AND <= 5) |
| price\_range | VARCHAR2(20) |  |

**Each transaction table:**

**Transactions Books Table:**

| Column Name | Datatype | Constraint |
| --- | --- | --- |
| transaction\_id | NUMBER | PRIMARY KEY |
| username | VARCHAR2(200) | FOREIGN KEY (username) REFERENCES users(username) |
| book\_id | NUMBER | FOREIGN KEY (book\_id) REFERENCES master\_books(book\_id) |
| transaction\_date | DATE |  |
| purchase\_price | NUMBER(10, 2) | CHECK (purchase\_price >= 0) |
| timestamp | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

**Transactions Movies Table:**

| **Column Name** | **Datatype** | **Constraint** |
| --- | --- | --- |
| transaction\_id | NUMBER | PRIMARY KEY |
| username | VARCHAR2(200) | FOREIGN KEY (username) REFERENCES users(username) |
| movie\_id | NUMBER | FOREIGN KEY (movie\_id) REFERENCES master\_movies(movie\_id) |
| transaction\_date | DATE |  |
| purchase\_price | NUMBER(10, 2) | CHECK (purchase\_price >= 0) |
| timestamp | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

**Transactions Products Table:**

| **Column Name** | **Datatype** | **Constraint** |
| --- | --- | --- |
| transaction\_id | NUMBER | PRIMARY KEY |
| username | VARCHAR2(200) | FOREIGN KEY (username) REFERENCES users(username) |
| product\_id | NUMBER | FOREIGN KEY (product\_id) REFERENCES master\_products(product\_id) |
| transaction\_date | DATE |  |
| purchase\_price | NUMBER(10, 2) | CHECK (purchase\_price >= 0) |
| timestamp | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

**Transactions Courses Table:**

| **Column Name** | **Datatype** | **Constraint** |
| --- | --- | --- |
| transaction\_id | NUMBER | PRIMARY KEY |
| username | VARCHAR2(200) | FOREIGN KEY (username) REFERENCES users(username) |
| course\_id | NUMBER | FOREIGN KEY (course\_id) REFERENCES master\_courses(course\_id) |
| transaction\_date | DATE |  |
| purchase\_price | NUMBER(10, 2) | CHECK (purchase\_price >= 0) |
| timestamp | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

**Transactions Restaurants Table:**

| **Column Name** | **Datatype** | **Constraint** |
| --- | --- | --- |
| transaction\_id | NUMBER | PRIMARY KEY |
| username | VARCHAR2(200) | FOREIGN KEY (username) REFERENCES users(username) |
| restaurant\_id | NUMBER | FOREIGN KEY (restaurant\_id) REFERENCES master\_restaurants(restaurant\_id) |
| transaction\_date | DATE |  |
| total\_bill | NUMBER(10, 2) | CHECK (total\_bill >= 0) |
| timestamp | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

**Transactions Miscellaneous Table:**

| **Column Name** | **Datatype** | **Constraint** |
| --- | --- | --- |
| transaction\_id | NUMBER | PRIMARY KEY |
| username | VARCHAR2(200) | FOREIGN KEY (username) REFERENCES users(username) |
| description | VARCHAR2(4000) |  |
| transaction\_date | DATE |  |
| amount | NUMBER(10, 2) | CHECK (amount >= 0) |
| timestamp | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

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

**1.Create**

CREATE TABLE master\_books (

book\_id NUMBER PRIMARY KEY,

book\_title VARCHAR2(100) NOT NULL,

author VARCHAR2(100),

publication\_date DATE,

genre VARCHAR2(50),

price NUMBER(10, 2),

CONSTRAINT chk\_price\_books CHECK (price >= 0)

);

**2.insert**

INSERT INTO master\_books (book\_id, book\_title, author, publication\_date, genre, price)

VALUES

(1, 'Book Title 1', 'Author A', '15-05-2022', 'Fiction', 19.99);

INSERT INTO master\_books (book\_id, book\_title, author, publication\_date, genre, price)

VALUES

(2, 'Book Title 2', 'Author B', '20-10-2021', 'Mystery', 12.99);

INSERT INTO master\_books (book\_id, book\_title, author, publication\_date, genre, price)

VALUES

(3, 'Book Title 3', 'Author C', '25-08-2022', 'Sci-Fi', 24.99);

INSERT INTO master\_books (book\_id, book\_title, author, publication\_date, genre, price)

VALUES

(4, 'Book Title 4', 'Author D', '20-01-2023', 'Thriller', 15.99);

INSERT INTO master\_books (book\_id, book\_title, author, publication\_date, genre, price)

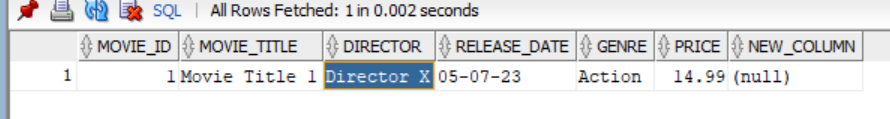
VALUES

(5, 'Book Title 5', 'Author E', '05-03-2023', 'Non-Fiction', 29.99);

Select:

SELECT \* FROM master\_movies

WHERE genre = 'Action';



**3.Delete**

DELETE FROM Transaction\_books

WHERE book\_id = 5;

DELETE FROM master\_books

WHERE book\_id = 5;

DELETE FROM transactions\_movies

WHERE movie\_id = 5;

DELETE FROM master\_movies

WHERE movie\_id = 5;

**4.Update**

UPDATE master\_books

SET price = 24.99

WHERE book\_title = 'Book Title 4';

UPDATE master\_movies

SET price = 29.99

WHERE movie\_id = 5;

**5. Alter**

ALTER TABLE master\_books

ADD isbn VARCHAR2(20);

ALTER TABLE master\_movies

ADD (new\_column VARCHAR2(50));

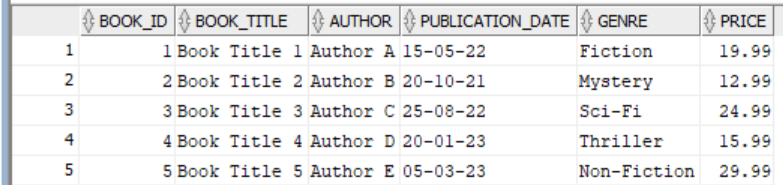
**Transaction Control Language(TCS):**

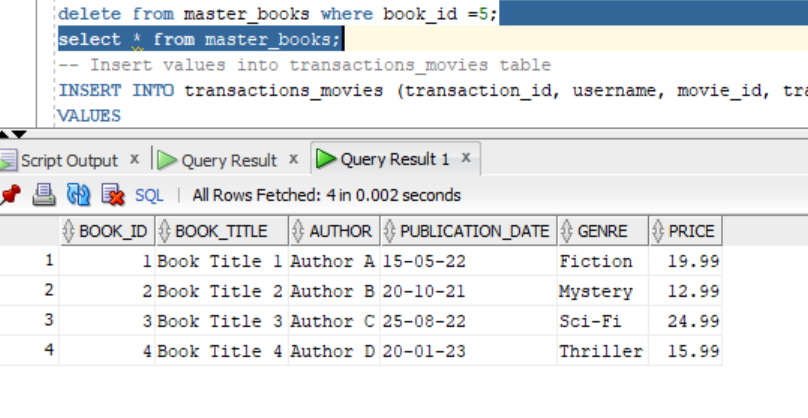
**COMMIT;**

**ROLLBACK;**

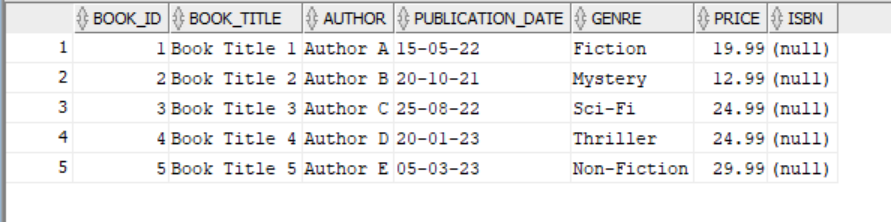
**-- Sql query file name<<Section 2.sql>>**

**Screenshot:**

****

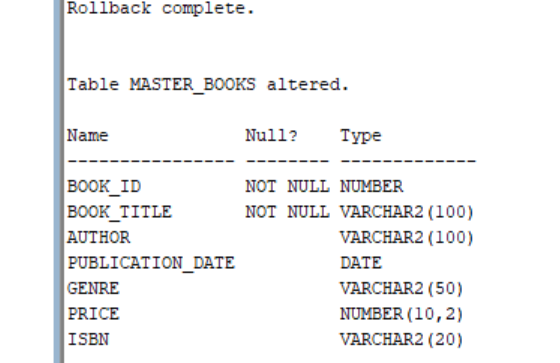
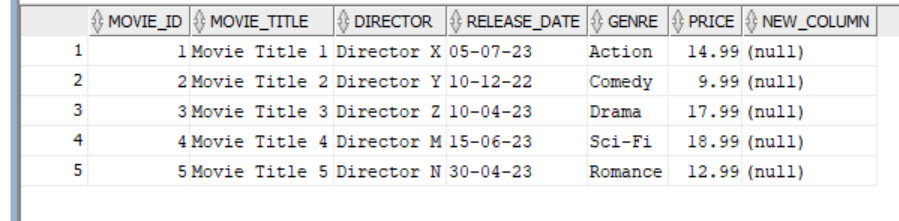
**After deleting a row** **** 

Updating





Altering Master\_books:

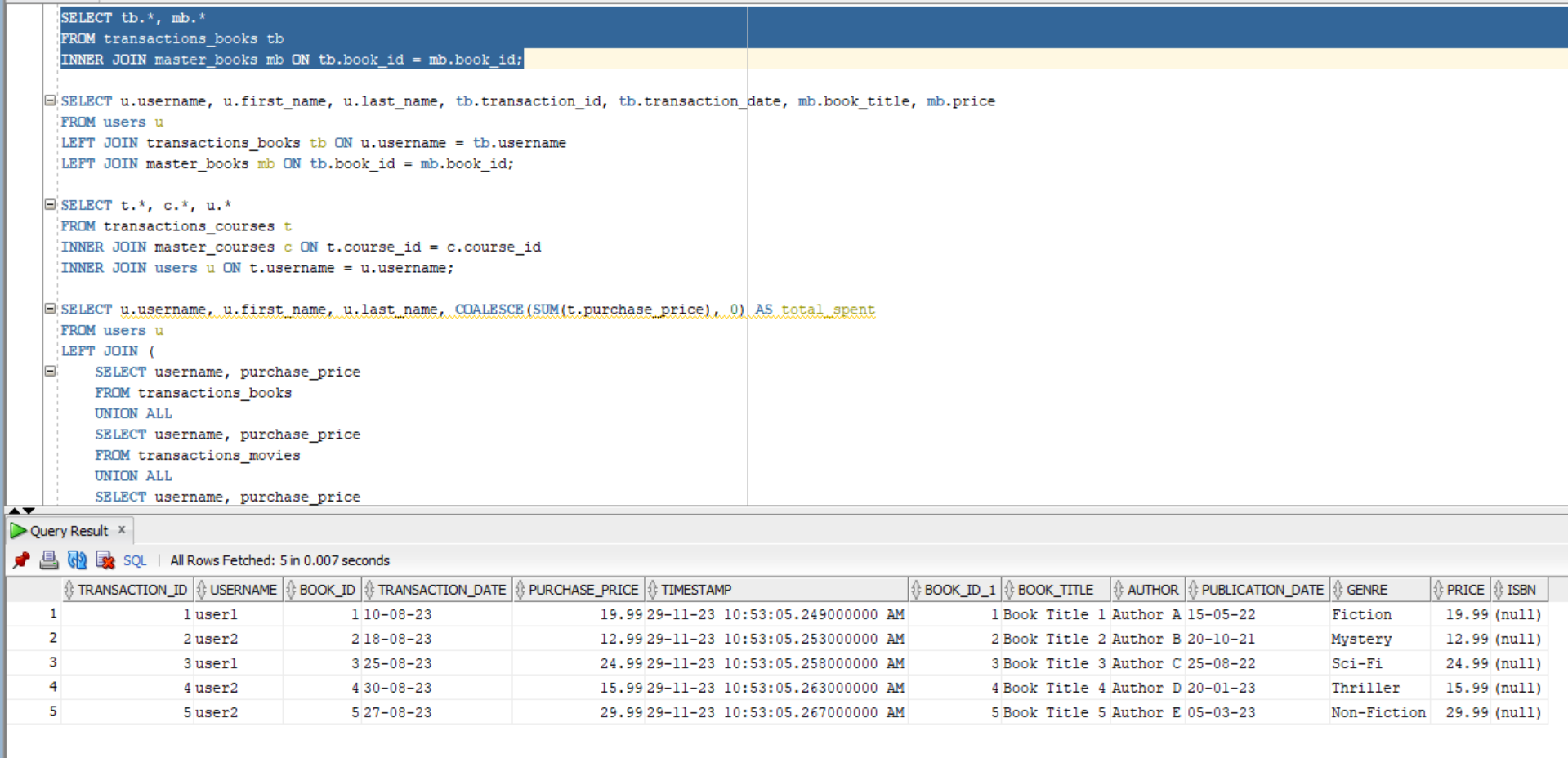
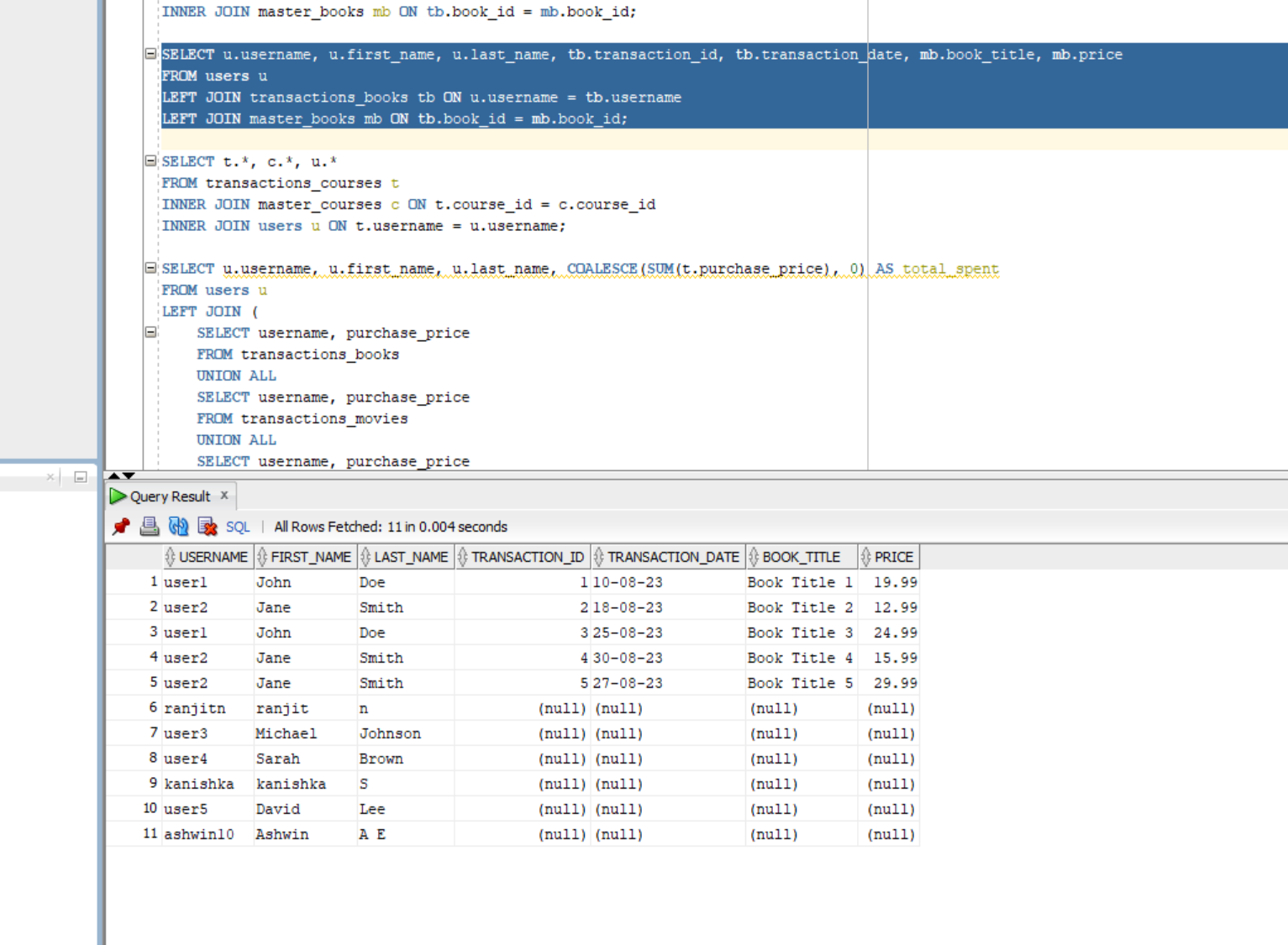
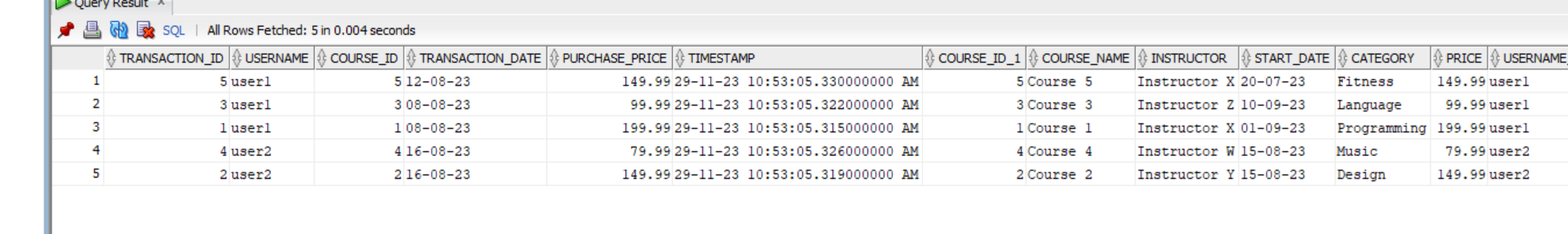
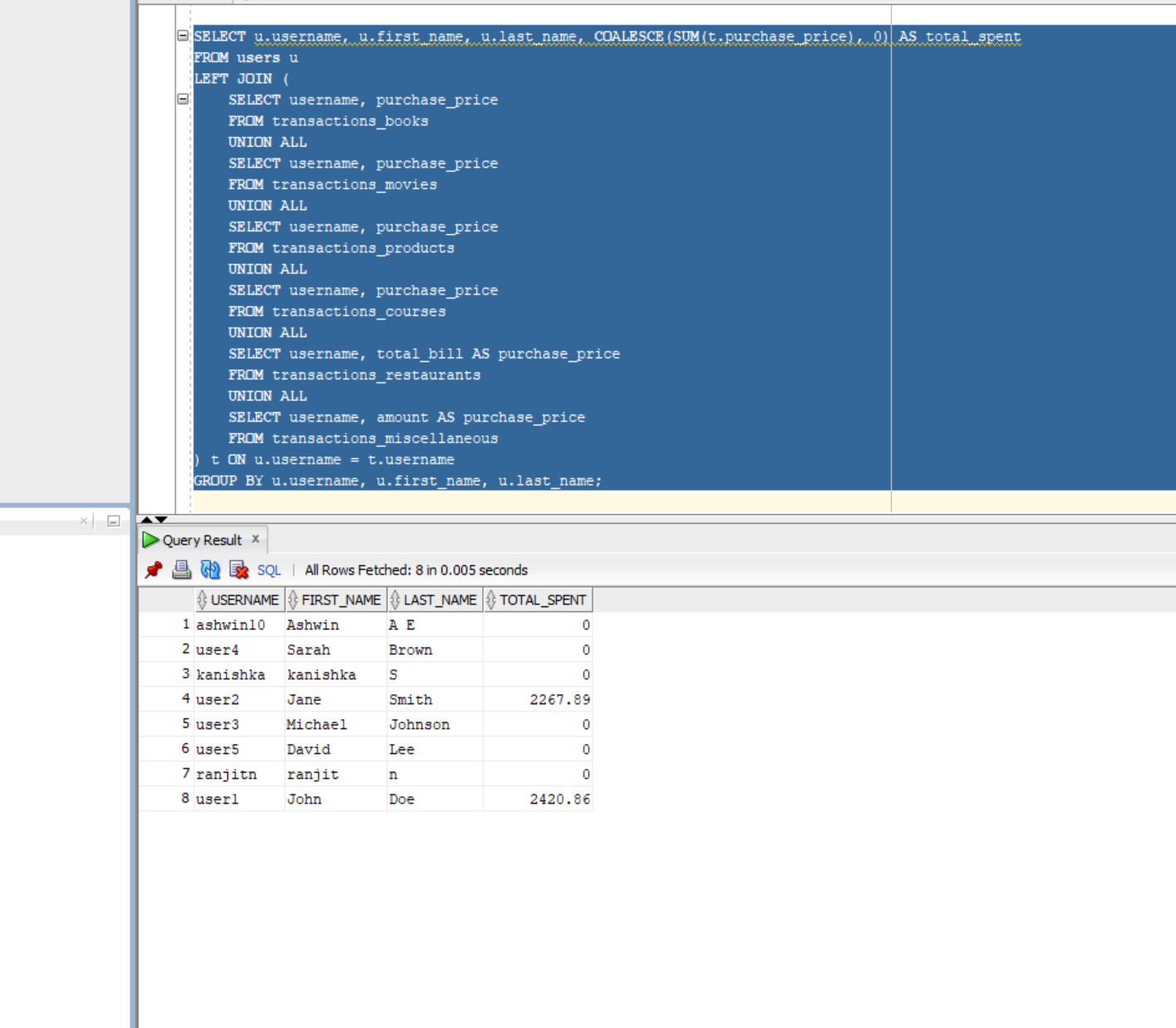
**<<Place outputs of execution>>**

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

**-- Sql query file name<<Section 3.sql>>**

**Screenshot:**

1

****2 ****3  4

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

**Inference:**

**1**. This query retrieves information about transactions related to books along with details about the corresponding books.

The 'INNER JOIN' ensures that only rows with matching values in both 'transactions\_books' and 'master\_books' are included.

The result includes columns from both tables ('tb. \* and`mb. \*\* ).

**2**. This query retrieves information about users, their book transactions, and details about the books (if available).

It uses LEFT JOIN to include all users, even if they haven't made book transactions.

The result includes columns for username, first\_name, last\_name, transaction\_id, transaction\_date, book\_title, and price.

If a user has made a book transaction, the corresponding book details will be included; otherwise, the columns related to books will be filled with null values.

3. This query combines information from three tables: transactions\_courses, master\_courses, and users.

Each row in the result represents a transaction for a course, including details about the course itself and the user who made the transaction.

The INNER JOIN ensures that only rows with matching values in all three tables are included in the result.

The result includes columns from all three tables (t.\*, c.\*, u.\*)

4. This query retrieves information about users and the total amount they spent across various transaction categories.

It uses a LEFT JOIN to include all users, even if they haven't made any transactions.

The subquery (t) combines information from multiple transaction tables using UNION ALL.

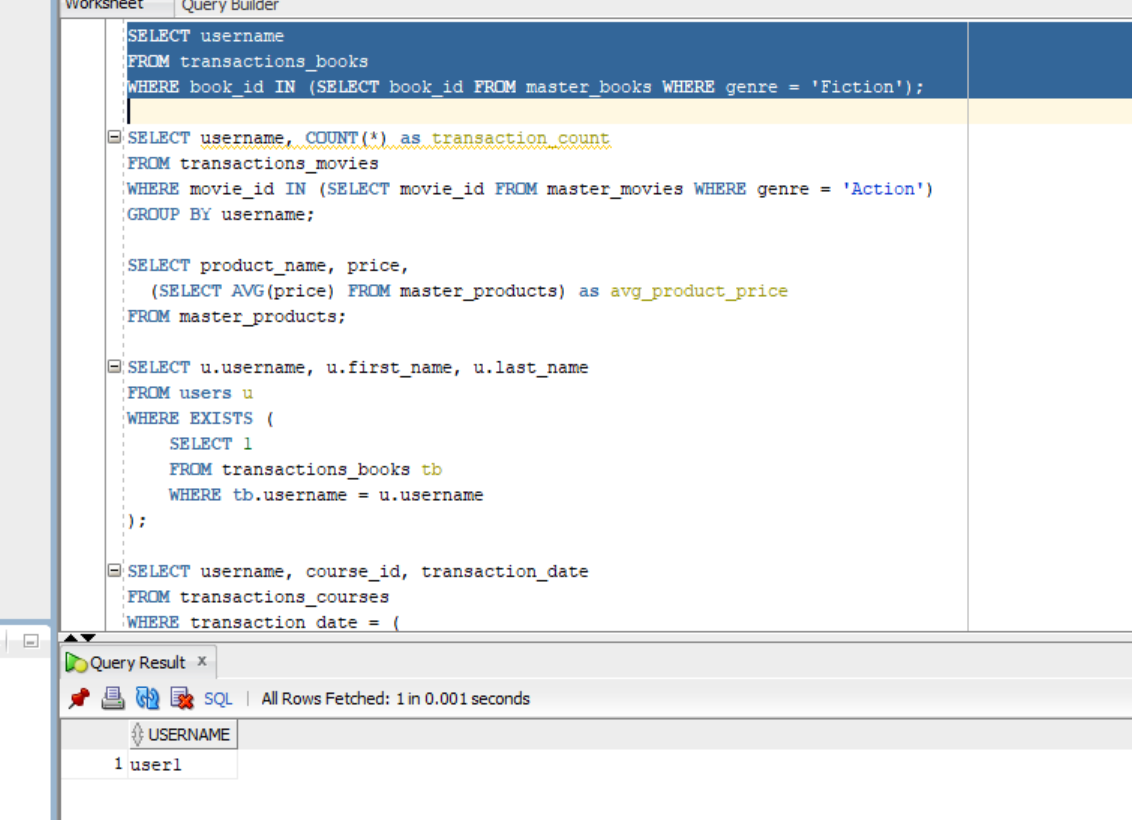
The result includes columns for username, first\_name, last\_name, and total\_spent.

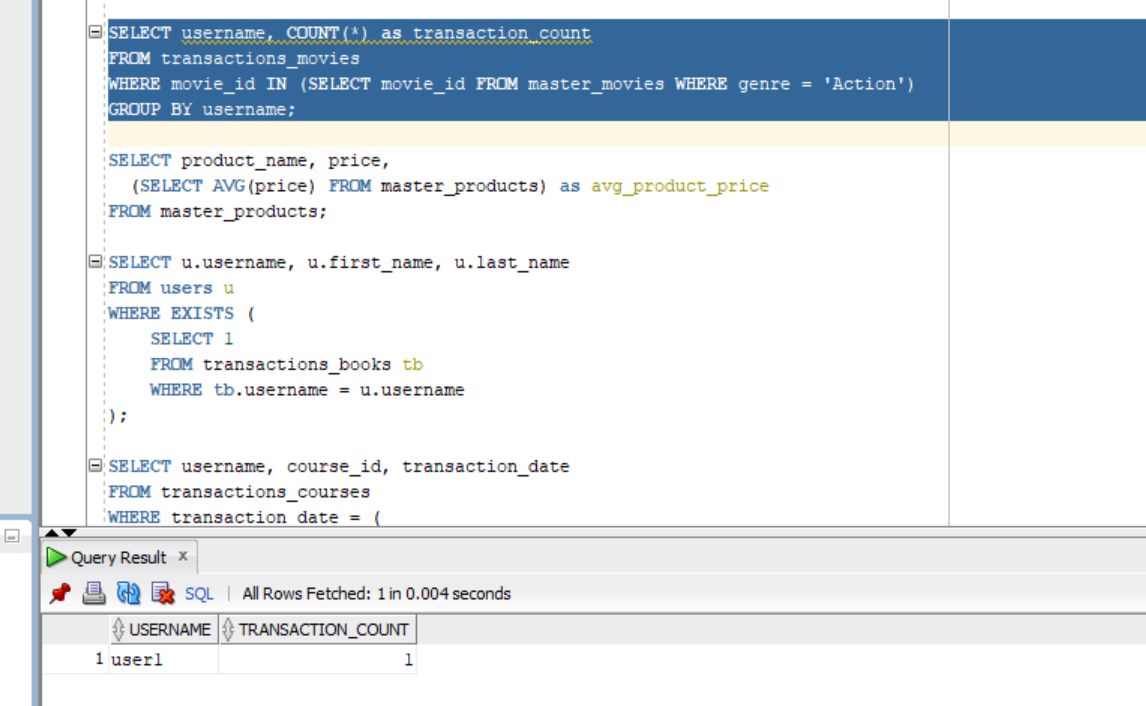
The COALESCE function is used to handle cases where a user has no transactions (the total spent will be shown as 0).

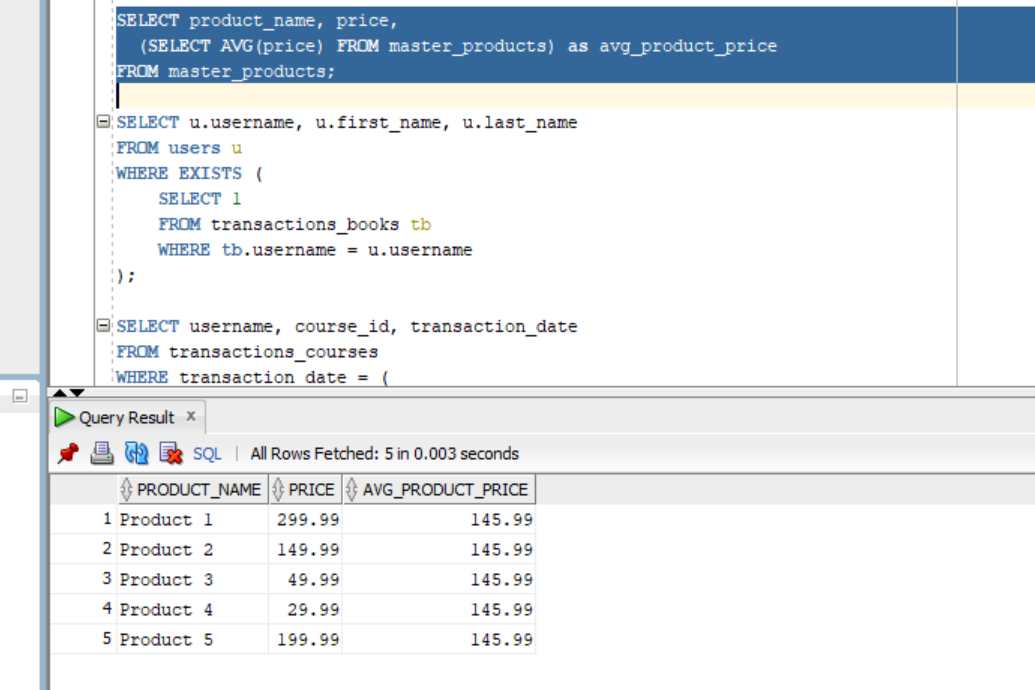
**4.Section-4(Nested Queries)**

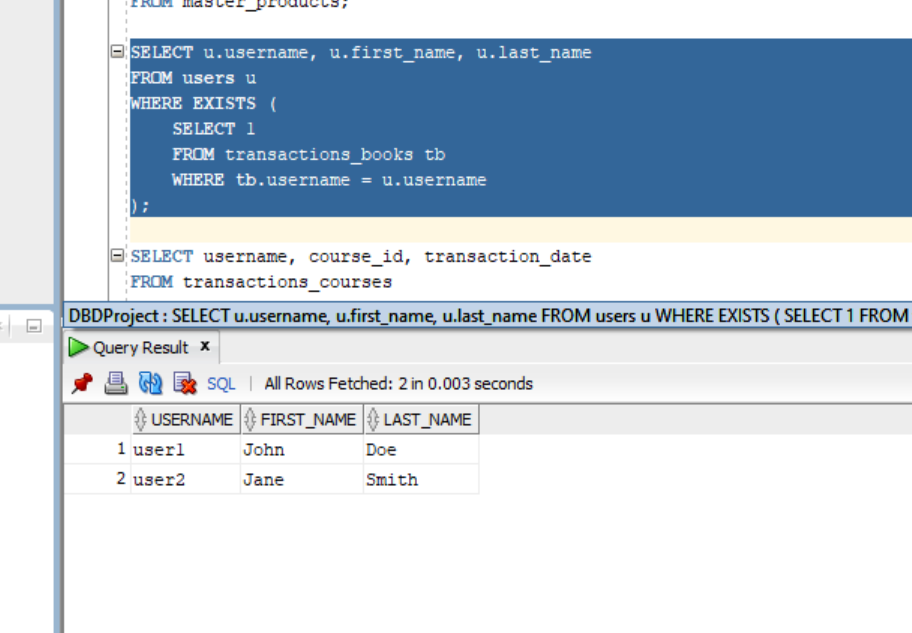
**-- Sql query file name<<Section 4.sql>>**

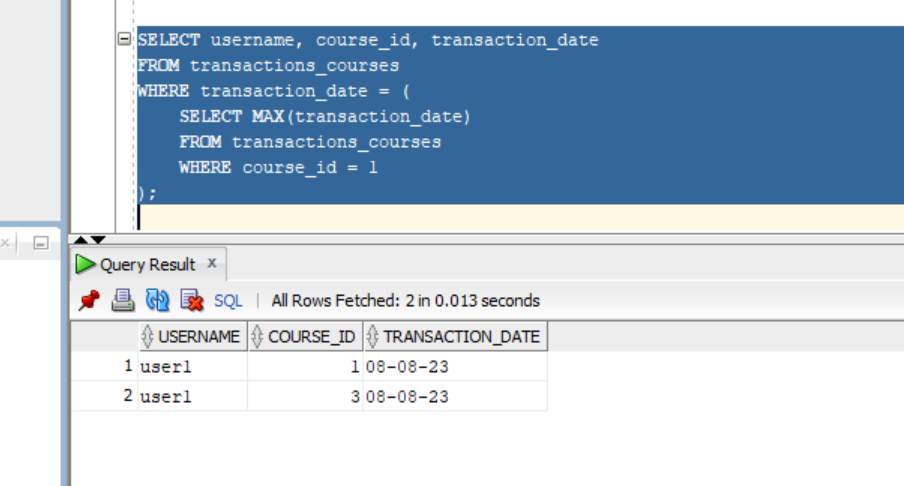
**Screenshot:**

**1. **

**2.** 

**3. **

**4.** 

**5.** 

**Inference:**

1. This query retrieves usernames of users who have made transactions for books in the 'Fiction' genre.

The inner query (SELECT book\_id FROM master\_books WHERE genre = 'Fiction') retrieves book IDs of books in the 'Fiction' genre.

The outer query then selects usernames from transactions\_books where the book ID is in the list obtained from the inner query.

2. This query counts the number of transactions for movies in the 'Action' genre for each user.

The inner query retrieves movie IDs of movies in the 'Action' genre.

The outer query then counts the number of transactions for each user, filtering only those related to 'Action' genre movies.

3. This query retrieves product names, prices, and the average price of all products.

The correlated subquery (SELECT AVG(price) FROM master\_products) calculates the average price of all products for each row in the outer query.

4. This query retrieves the usernames, first names, and last names of users who have made transactions for books.

The nested query checks for the existence of transactions in the transactions\_books table for each user in the users table.

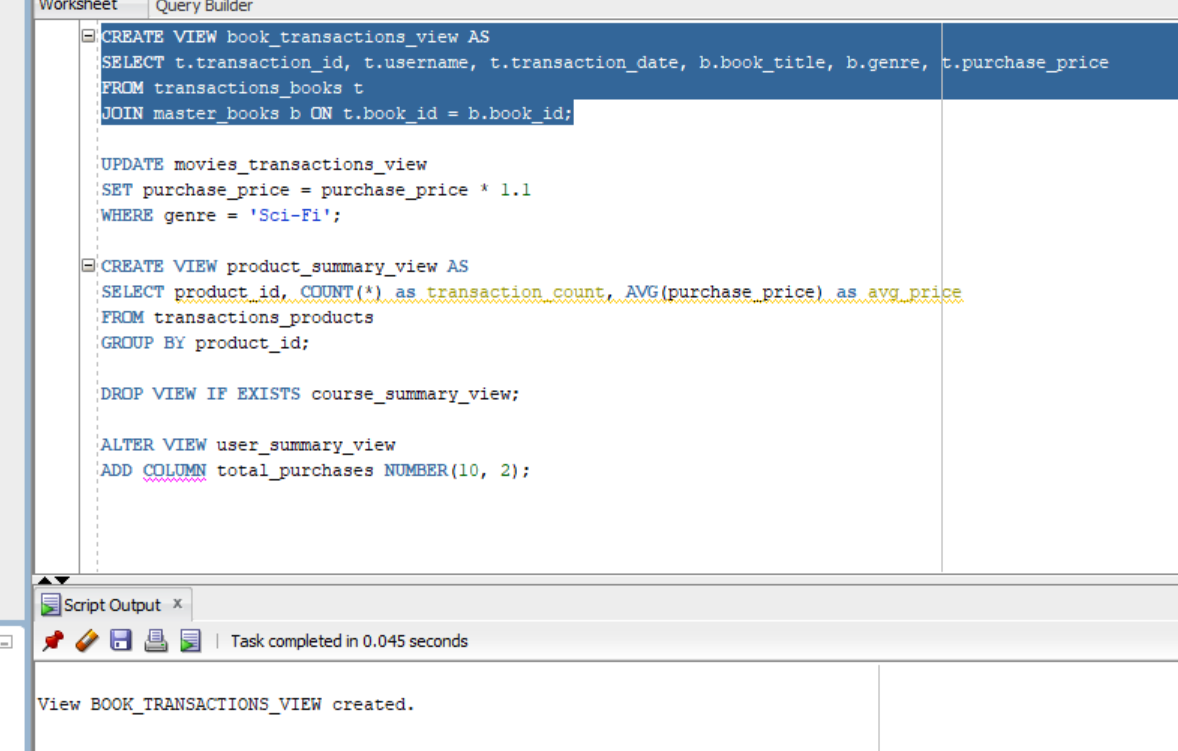
5. This query retrieves usernames, course IDs, and transaction dates for transactions with the latest date for a specific course.

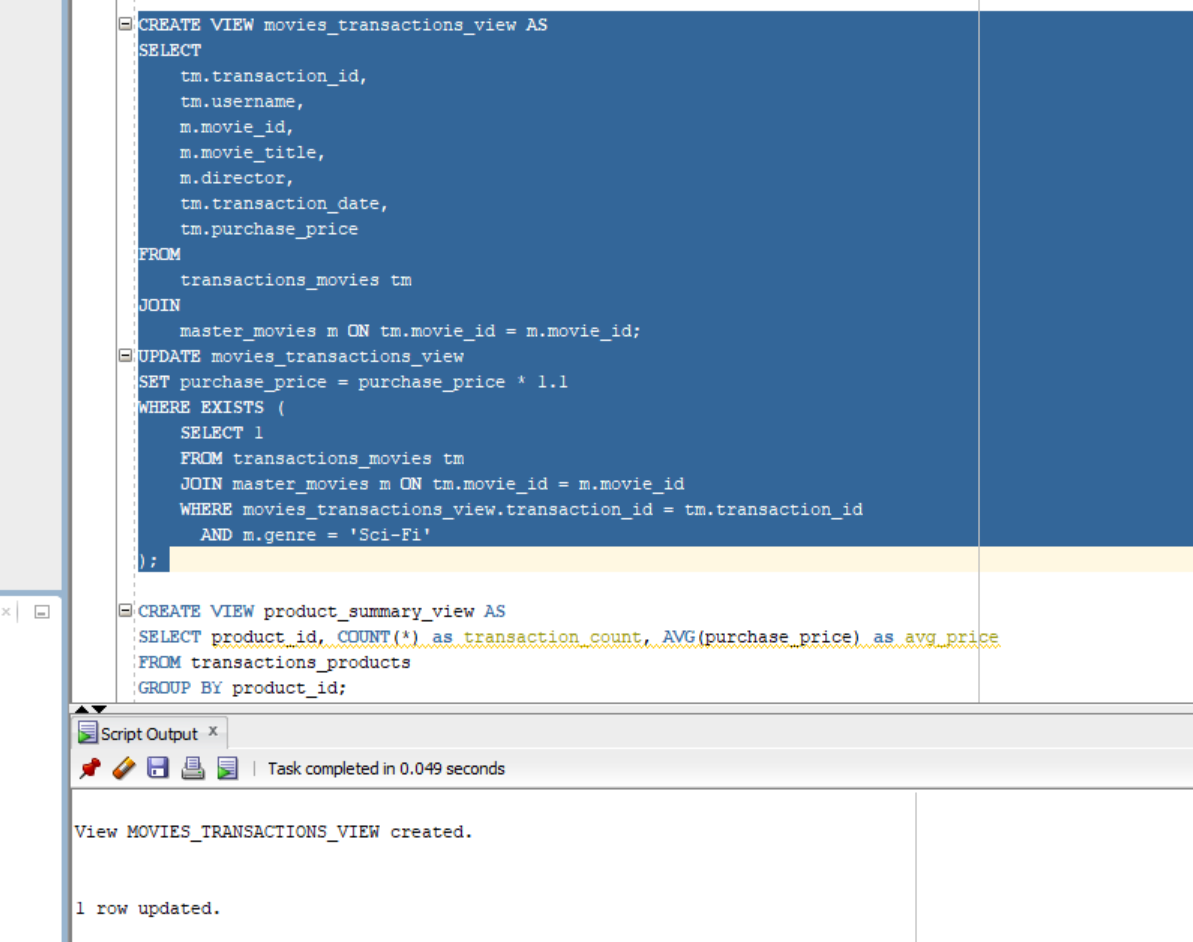
The nested query finds the maximum transaction date for transactions related to course\_id = 1, and the outer query selects transactions with that date.

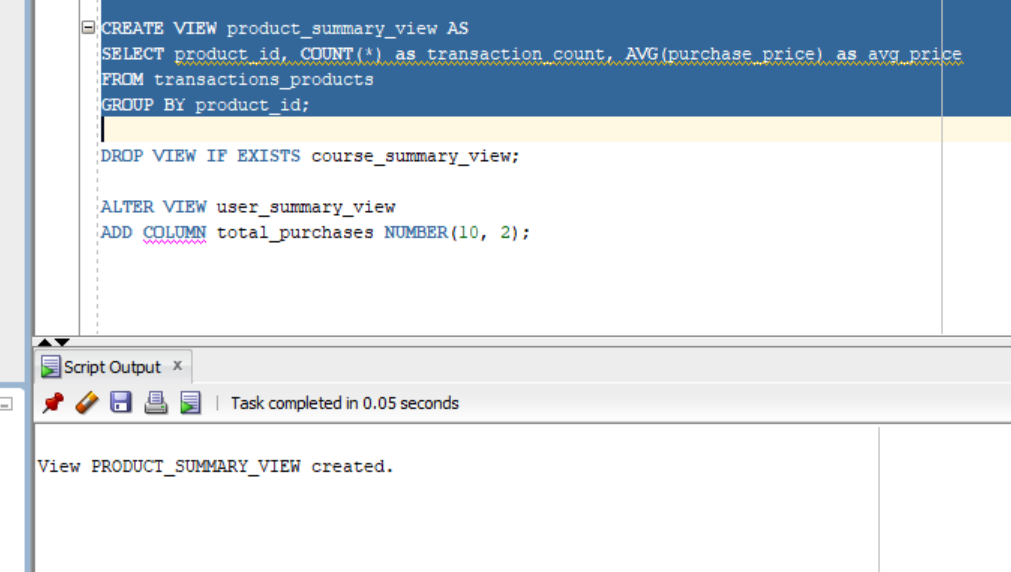
**5.Section-5(view)**

**-- Sql query file name<<section 5.sql>>**

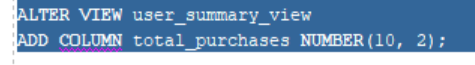
**Screenshot:**

**1. **

**2.** ****

**3.** ****

**4.** ****

**5.** ****

**Inference:**

1. This operation creates a view named book\_transactions\_view that combines data from transactions\_books and master\_books.

The view includes columns such as transaction\_id, username, transaction\_date, book\_title, genre, and purchase\_price.

The JOIN condition links transactions to the corresponding book details using the book\_id.

2. The view movies\_transactions\_view provides a virtual table that combines data from transactions\_movies and master\_movies.

The UPDATE statement modifies the purchase\_price in the view based on a condition involving the existence of certain rows in the underlying tables.

3. This operation creates a view named product\_summary\_view that summarizes transaction data for products.

The view includes columns like product\_id, transaction\_count (number of transactions for each product), and avg\_price (average purchase price).

4. This operation drops the view named course\_summary\_view if it exists.

DROP VIEW is used to remove a previously created view from the database.

5. This operation modifies the structure of the user\_summary\_view by adding a new column named total\_purchases.

The ALTER VIEW statement allows you to make structural changes to an existing view.

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

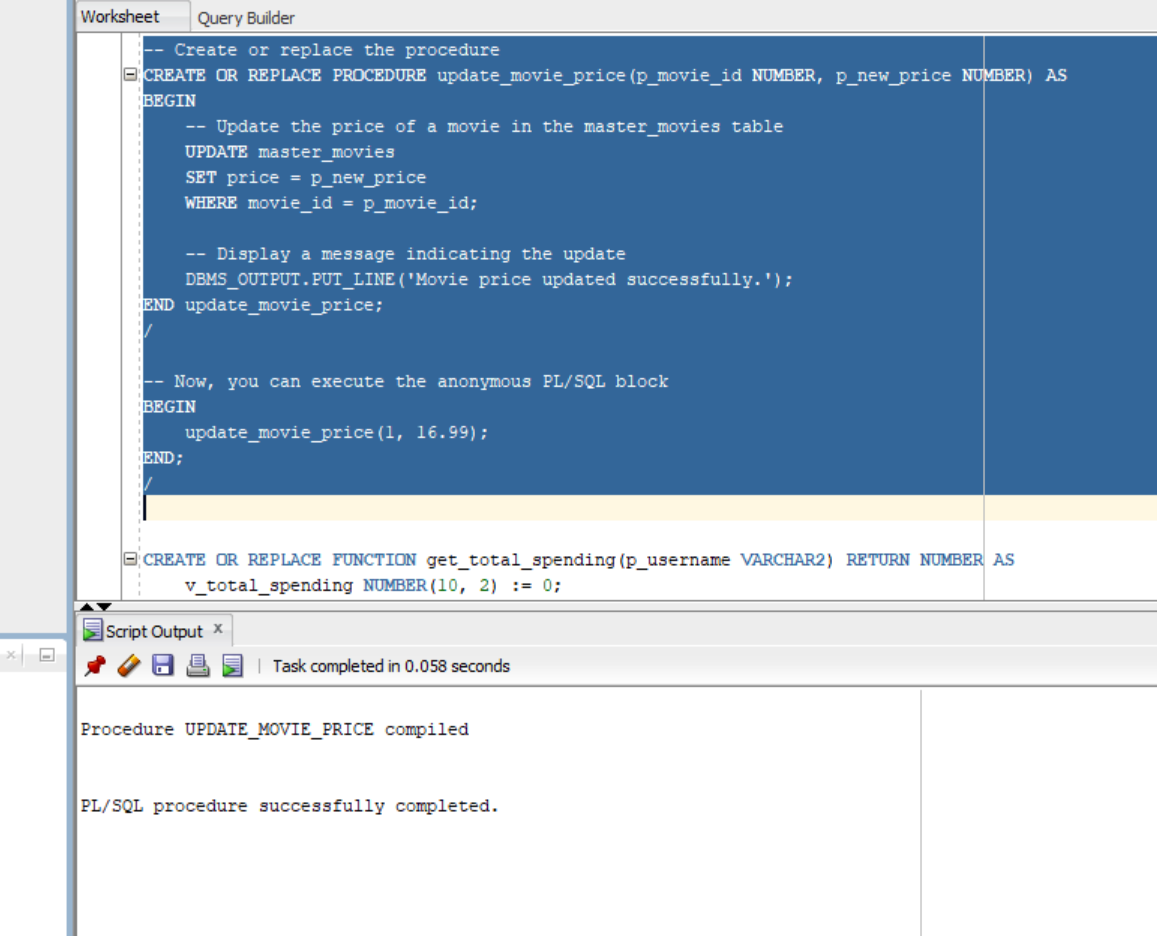
**<<Can be more than one PL/SQL but should have function>>**

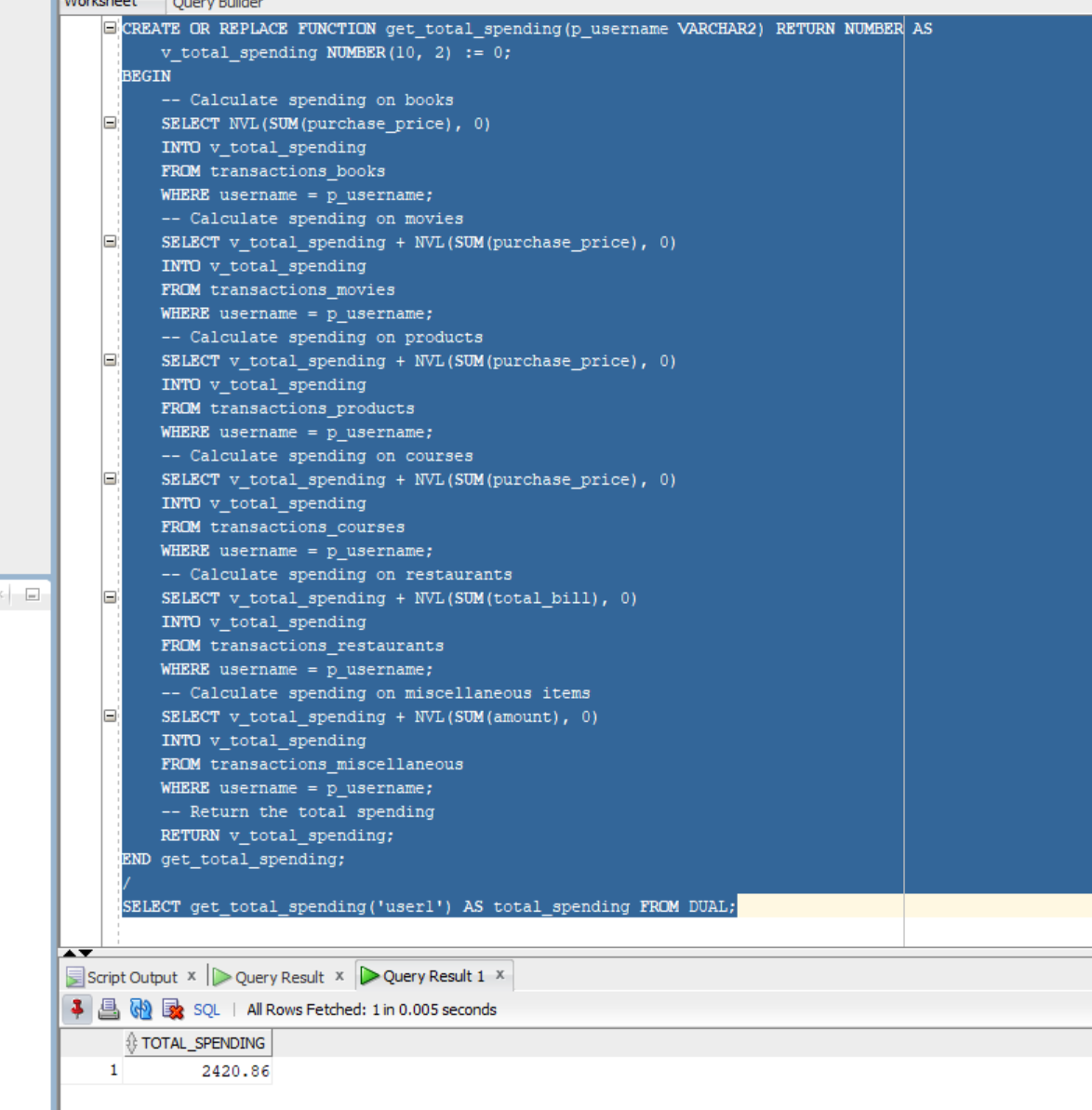
|  |  |
| --- | --- |
| **Function Name** |  |
| **Procedure Name** | **update\_movie\_price** |
| **Expected Output** | **The updation of the movie price** |
| **Tables operated** | **master\_movies** |

|  |  |
| --- | --- |
| **Function Name** |  |
| **Procedure Name** |  |
| **Expected Output** | **….** |
| **Tables operated** |  |

**-- Sql query file name<<section 6.sql>>**

**Screenshot:**

**1.** ****

**2.** ****

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

**Inference:**

1. The code is designed to update the price of a movie in the master\_movies table, specifically for the movie with movie\_id equal to 1.

The new price is set to 16.99.

After the update, a message is printed to the output using DBMS\_OUTPUT.PUT\_LINE.

2. The function provides a centralized way to calculate the total spending for a user across various transaction types.

The NVL function is used to handle potential null values in the sum calculations.

The function is executed in a SQL environment using the SELECT statement.

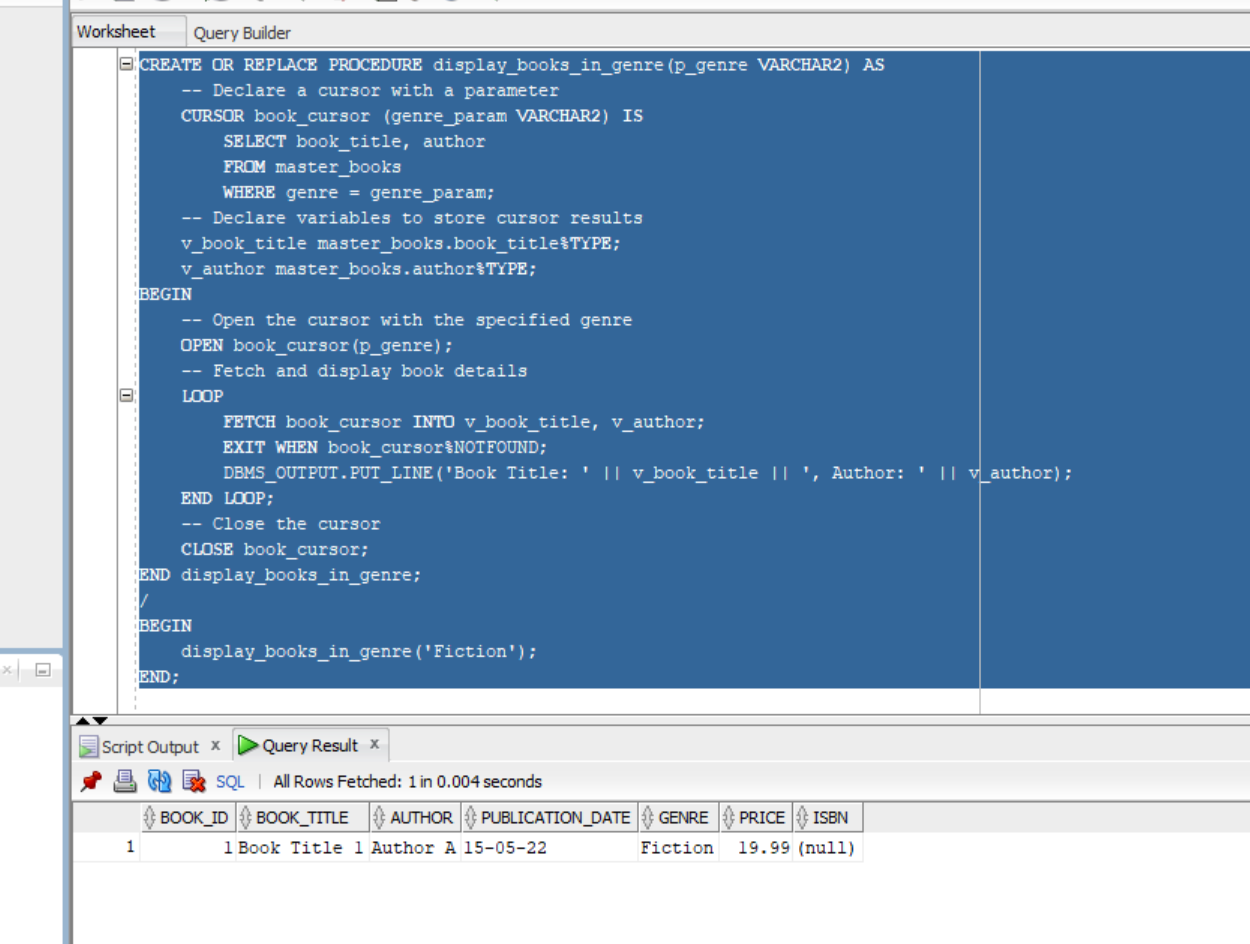
**7. Section-7(Cursor file)**

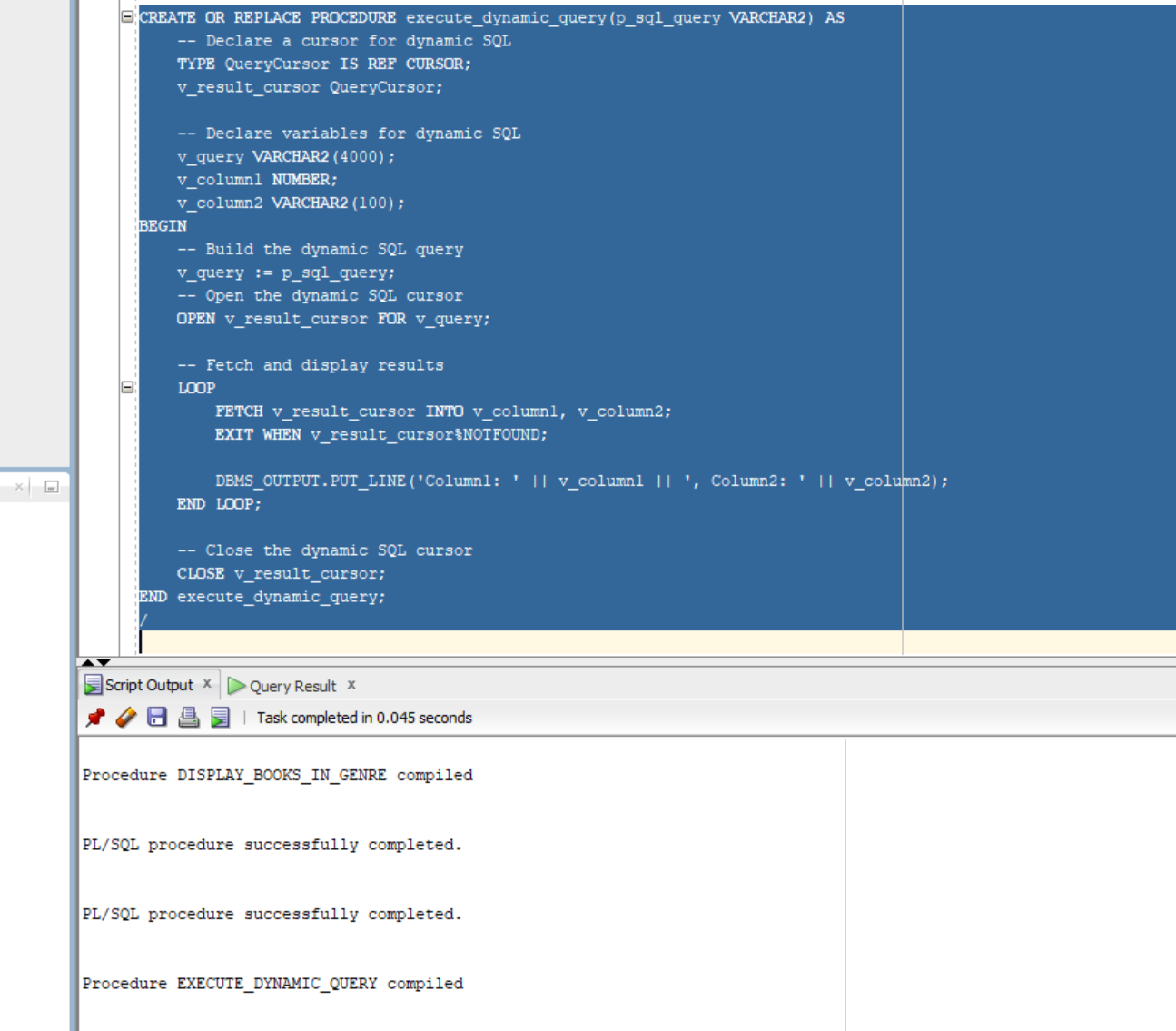
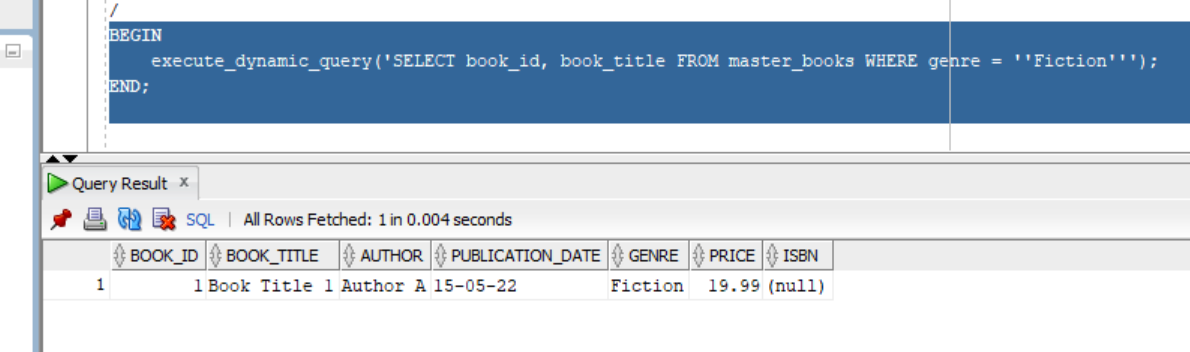
|  |  |
| --- | --- |
| Cursor Name: display\_books\_in\_genre | Explanation of the cursor |
| Expected Output (display\_books\_in\_genre procedure):  The procedure is designed to display the book titles and authors for a specified genre.  For the provided example where the genre is 'Fiction,' the output would be the book titles and authors in the 'Fiction' genre from the master\_books table. | * Declare a cursor named book\_cursor with a parameter genre\_param of type VARCHAR2 * Define the cursor to select book\_title and author from the master\_books table based on the genre\_param * Declare variables v\_book\_title and v\_author to store the results fetched from the cursor * Open the cursor with the desired genre passed as an argument to p\_genre * Inside a loop, fetch the values from the cursor into v\_book\_title and v\_author and display them using DBMS\_OUTPUT.PUT\_LINE * Exit the loop when there are no more rows to fetch from the cursor * Close the cursor |

|  |  |
| --- | --- |
| Cursor Name: execute\_dynamic\_query | Explanation of the cursor |
| Expected Output (execute\_dynamic\_query procedure):  The procedure is designed to execute a dynamically provided SQL query and display the results.  For the provided example where the query selects book\_id and book\_title for books in the 'Fiction' genre, the output would be the book details meeting the criteria specified in the dynamic query. | * Declare a cursor type named QueryCursor for dynamic SQL * Declare a variable v\_result\_cursor of the cursor type * Declare a variable v\_query to store the dynamic SQL query and assign the provided query to it * Open the cursor v\_result\_cursor for the dynamic SQL query v\_query * Inside a loop, fetch the values from the cursor into v\_column1 and v\_column2 and display them using DBMS\_OUTPUT.PUT\_LINE * Exit the loop when there are no more rows to fetch from the cursor * Close the cursor v\_result\_cursor |

**-- Sql query file name<<section 7.sql>>**

**Screenshot:**

**1.** ****

**2.** ****

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

**Inference:**

**1**. **Procedure Definition (display\_books\_in\_genre):**

It declares a cursor named book\_cursor with a parameter genre\_param.

The cursor retrieves book\_title and author from the master\_books table based on the specified genre.

It declares variables v\_book\_title and v\_author to store the cursor results.

The procedure opens the cursor with the specified genre, fetches and displays book details using a loop, and finally closes the cursor.

**Procedure Invocation (BEGIN display\_books\_in\_genre('Fiction'); END;):**

It calls the display\_books\_in\_genre procedure with the argument 'Fiction'.

The procedure opens the cursor for the 'Fiction' genre, fetches and displays book details, and then closes the cursor.

**Output:**

The procedure outputs book titles and authors for books in the 'Fiction' genre using the DBMS\_OUTPUT.PUT\_LINE statements.

**DBMS\_OUTPUT:**

To see the output, you need to enable DBMS\_OUTPUT in your SQL client. In Oracle SQL Developer, you can go to "View" -> "DBMS Output" to open the DBMS Output panel.

**Error Handling:**

The code doesn't include explicit error handling. If there are no books found for the specified genre, the loop will exit when book\_cursor%NOTFOUND.

Overall, this code is designed to display book details for a specific genre and should work as expected if there are books in the 'Fiction' genre in the master\_books table.

2. procedure named execute\_dynamic\_query that accepts a SQL query as a parameter, dynamically executes the query, and fetches the results using a cursor. The procedure uses a reference cursor (QueryCursor) to handle the dynamic SQL execution. The variables v\_column1 and v\_column2 are declared to store the results of the dynamic query, representing a numeric and a string column, respectively.

In the main block, the procedure is invoked with a specific dynamic SQL query, which selects the book\_id and book\_title columns from the master\_books table where the genre is 'Fiction'. The dynamic query is built using the input parameter p\_sql\_query, and the results are then fetched and displayed using a loop.

The use case here is to provide flexibility in executing different SQL queries dynamically. This could be particularly useful in scenarios where the structure of the query or the columns being selected may vary.

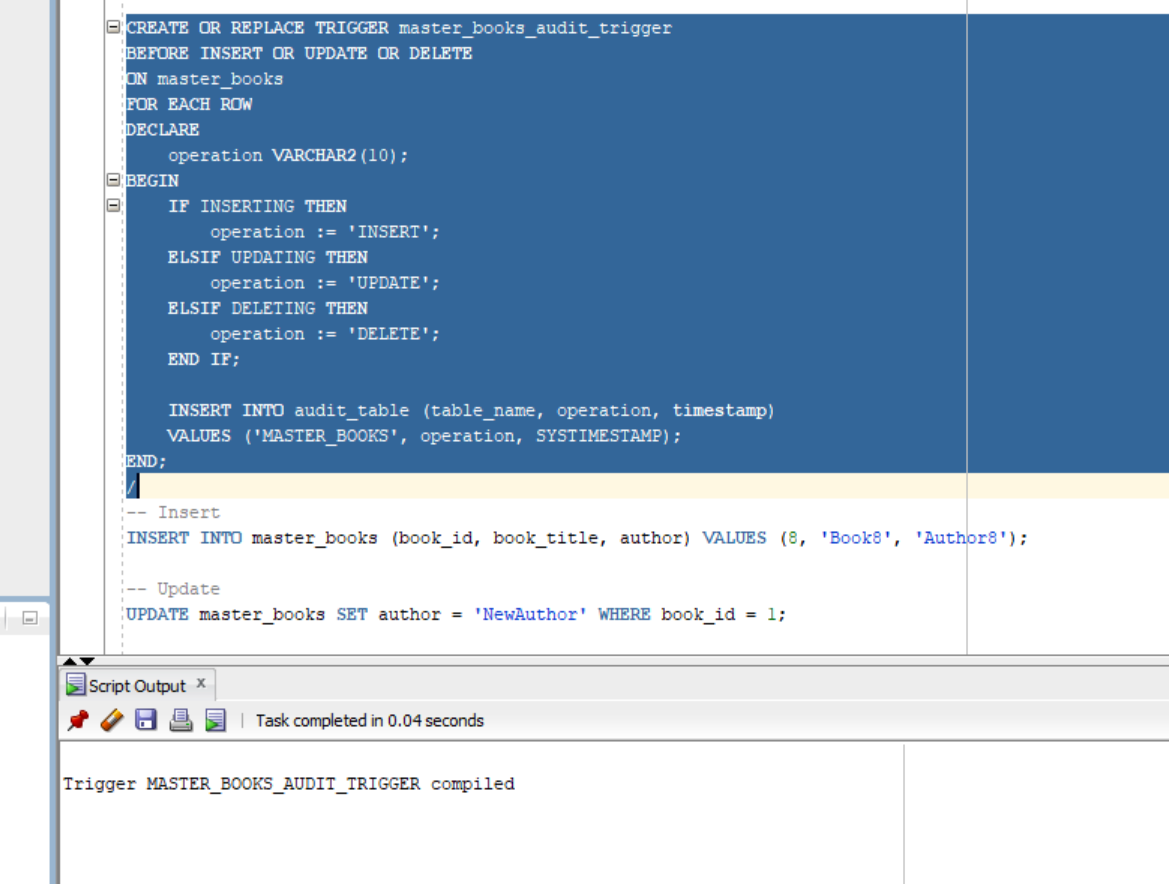
**8. Section-8(Trigger file)**

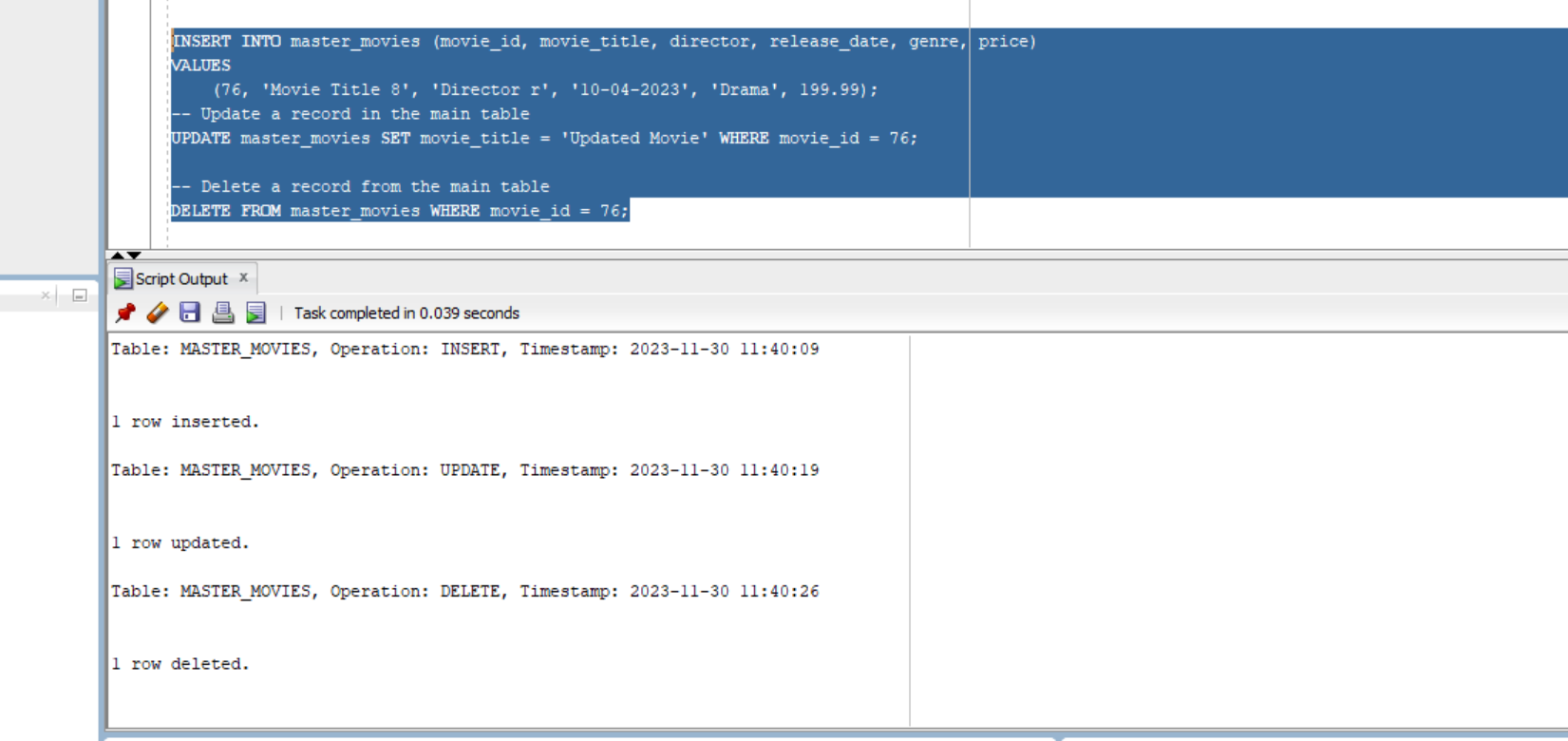
|  |  |
| --- | --- |
| **Trigger Name: master\_books\_audit\_trigger**: | 1.Explanation of the trigger operation  2.Type of Trigger |
| **Expected Output:** For each operation (INSERT, UPDATE, DELETE) on the "master\_books" table, a corresponding record will be inserted into the "audit\_table" with details such as the table name, operation type, and timestamp. | 1. This trigger is defined to execute before each INSERT, UPDATE, or DELETE operation on the "master\_books" table.  It determines the operation type (INSERT, UPDATE, or DELETE) and inserts a record into the "audit\_table" with information about the operation, table name ("MASTER\_BOOKS"), and the timestamp of the operation. |
|  | 2. This is a BEFORE ROW-level trigger because it fires before each row is processed in the main table ("master\_books"). |

|  |  |
| --- | --- |
| **Trigger Name: master\_movies\_output\_trigger:** | 1.Explanation of the trigger operation  2.Type of Trigger |
| **Expected Output:** For each operation (INSERT, UPDATE, DELETE) on the "master\_movies" table, a corresponding message will be displayed in the console, showing details such as the table name, operation type, and timestamp. | 1. This trigger is defined to execute before each INSERT, UPDATE, or DELETE operation on the "master\_movies" table.  It determines the operation type (INSERT, UPDATE, or DELETE) and uses DBMS\_OUTPUT.PUT\_LINE to display information about the operation, table name ("MASTER\_MOVIES"), and the timestamp of the operation in the console. |
|  | 2. This is a BEFORE ROW-level trigger because it fires before each row is processed in the main table ("master\_movies"). |

**-- Sql query file name<<section 8.sql>>**

**Screenshot:**

**1.** ****

**2.** ****

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

**Inference:**

1. audit\_table Table:

Purpose:

The audit\_table is designed to store audit trail information related to changes in other tables.

Structure:

audit\_id: Primary key for the audit table.

table\_name: Indicates the name of the table on which an operation (INSERT, UPDATE, DELETE) occurred.

operation: Represents the type of operation performed on the table (INSERT, UPDATE, DELETE).

timestamp: Captures the date and time when the operation took place.

master\_books\_audit\_trigger Trigger:

Purpose:

The master\_books\_audit\_trigger trigger is specifically designed for the master\_books table.

It captures INSERT, UPDATE, and DELETE operations on the master\_books table and records them in the audit\_table.

Functionality:

Timing:

The trigger fires BEFORE each row is inserted, updated, or deleted in the master\_books table.

Operations:

Determines the type of operation (INSERT, UPDATE, DELETE) being performed.

Inserts a corresponding record into the audit\_table with details such as the table name ('MASTER\_BOOKS'), the operation type, and the timestamp of the operation.

Usage:

This trigger is part of an auditing system to track changes in the master\_books table.

2. The trigger master\_movies\_output\_trigger is defined to fire before INSERT, UPDATE, or DELETE operations on the "master\_movies" table.

The trigger uses DBMS\_OUTPUT.PUT\_LINE to print information about the operation, table, and timestamp to the output.

You can run INSERT, UPDATE, and DELETE statements on the "master\_movies" table to see the trigger output in the console.

Top of Form

**9.Section -9**

**<< Web Application>>**

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

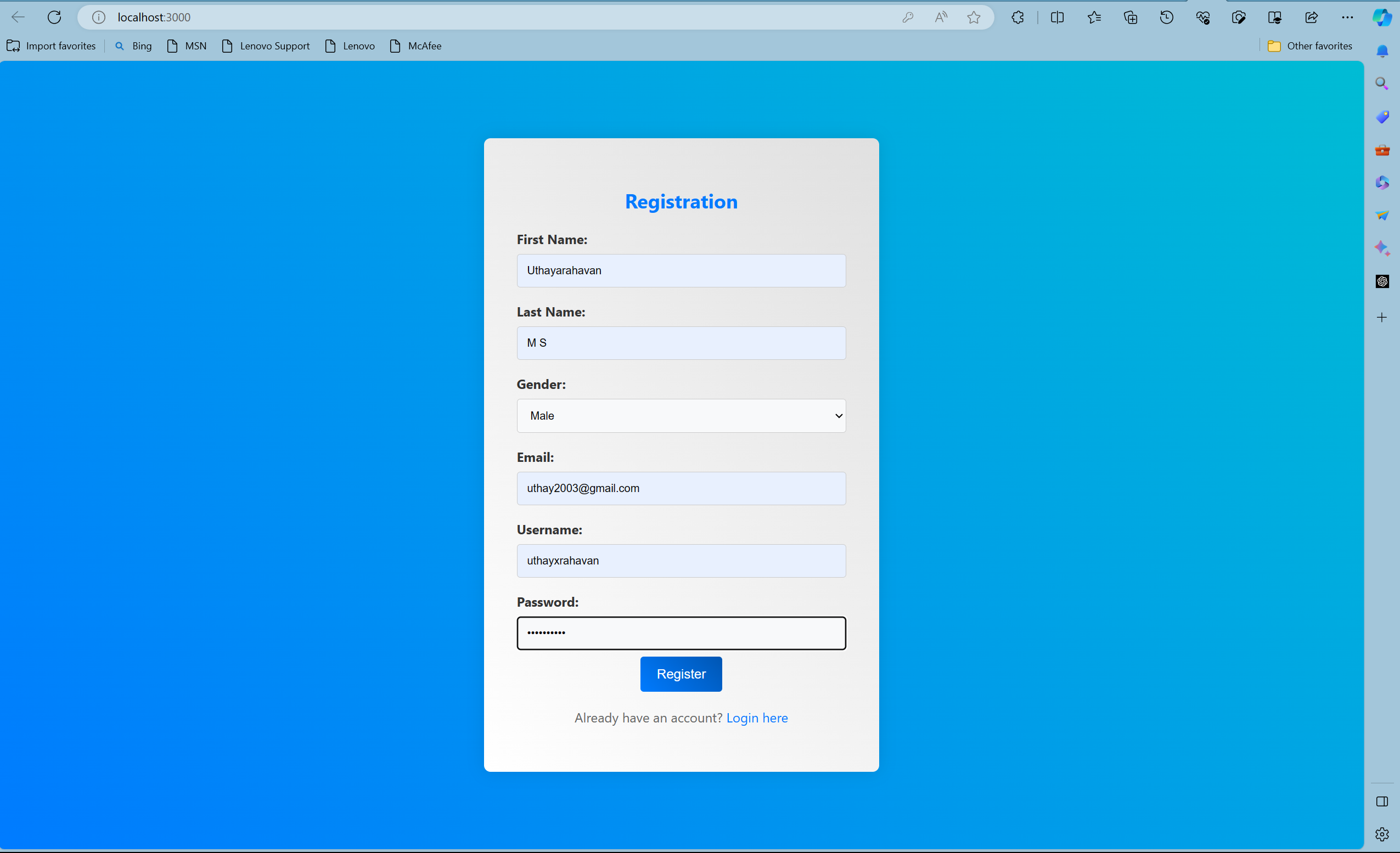
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Form Name** | **Category(Master**  **/Transaction)** | **Table Name associated with the form** | **Form Name /css**  **The Css is embedded in the html file.** | **File Name for reference in the shared drive** | **Type of operations**  **(Insert/Update/Delete**  **/Search/Display)** |
| **Login and registration.** | **Master** | **users** | **Login.html , registration.html** |  | **insert** |
| **Ma\_movies** | **Master** | **Master\_movies** | **Ma\_movies.html** |  | **Insert** |
| **Ma\_books** | **Master** | **Master\_books** | **Ma\_books.html** |  | **Insert** |
| **Ma\_products** | **Master** | **Master\_products** | **Ma\_products.html** |  | **Insert** |
| **Ma\_restaurants** | **Master** | **Master\_restaurants** | **Ma\_restaurants.html** |  | **Insert** |
| **Ma\_courses** | **Master** | **Master\_courses** | **Ma\_courses.html** |  | **Insert** |
| **tr\_movies** | **transaction** | **transactions\_movies** | **tr\_movies.html** |  | **Display,update** |
| **tr\_books** | **transaction** | **transactions\_books** | **tr\_books.html** |  | **Display,update** |
| **tr\_products** | **transaction** | **Transaction s\_products** | **tr\_products.html** |  | **Display,update** |
| **tr\_restaurants** | **transaction** | **transactions\_restaurants** | **tr\_restaurants.html** |  | **Display,update** |
| **tr\_courses** | **transaction** | **transactions\_courses** | **tr\_courses.html** |  | **Display,update** |
| **tr\_miscellaneous** | **transaction** | **Transactions\_miscellaneous** | **tr\_miscellaneous** |  | **Display,update** |

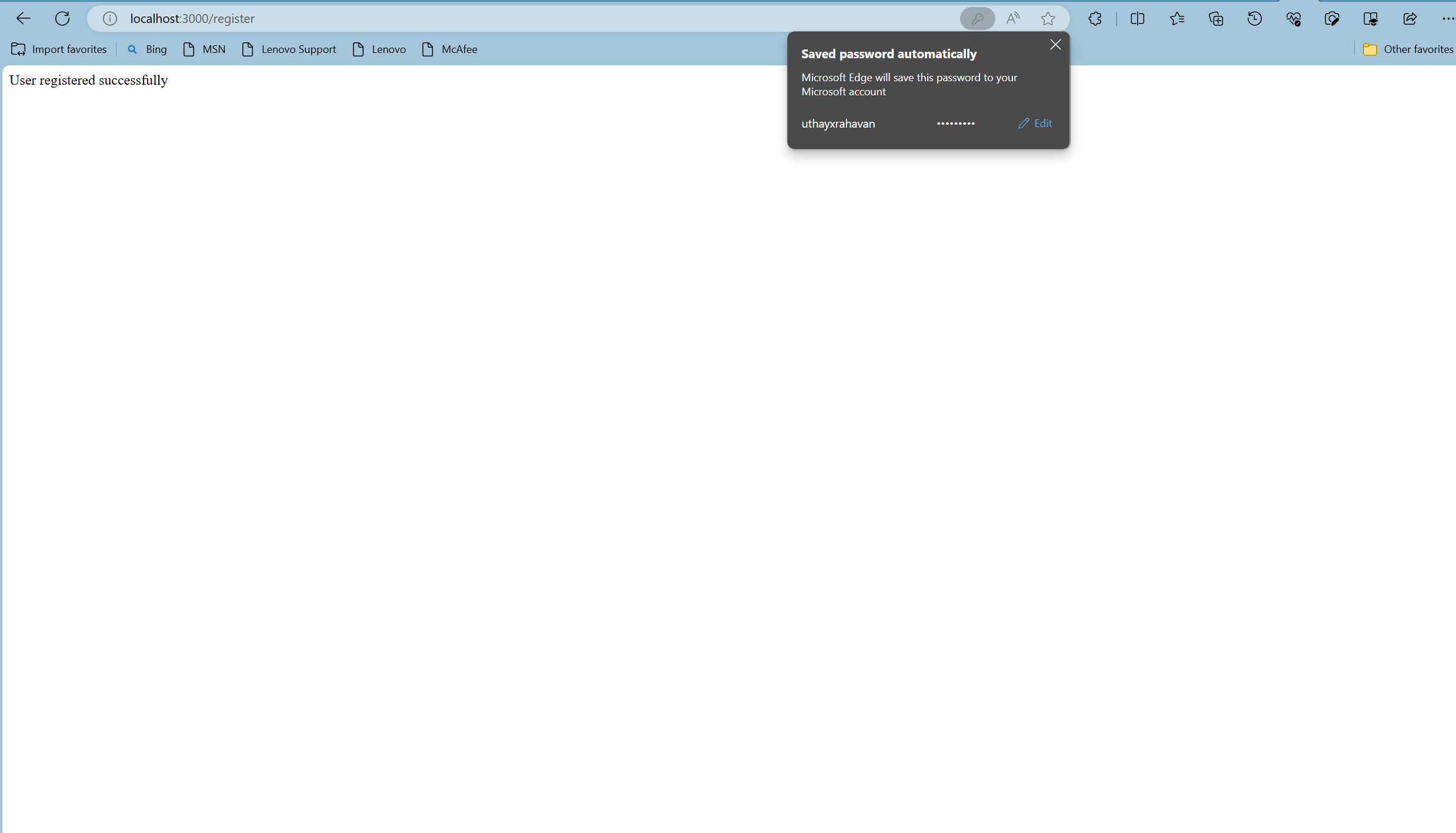
**Operations :**

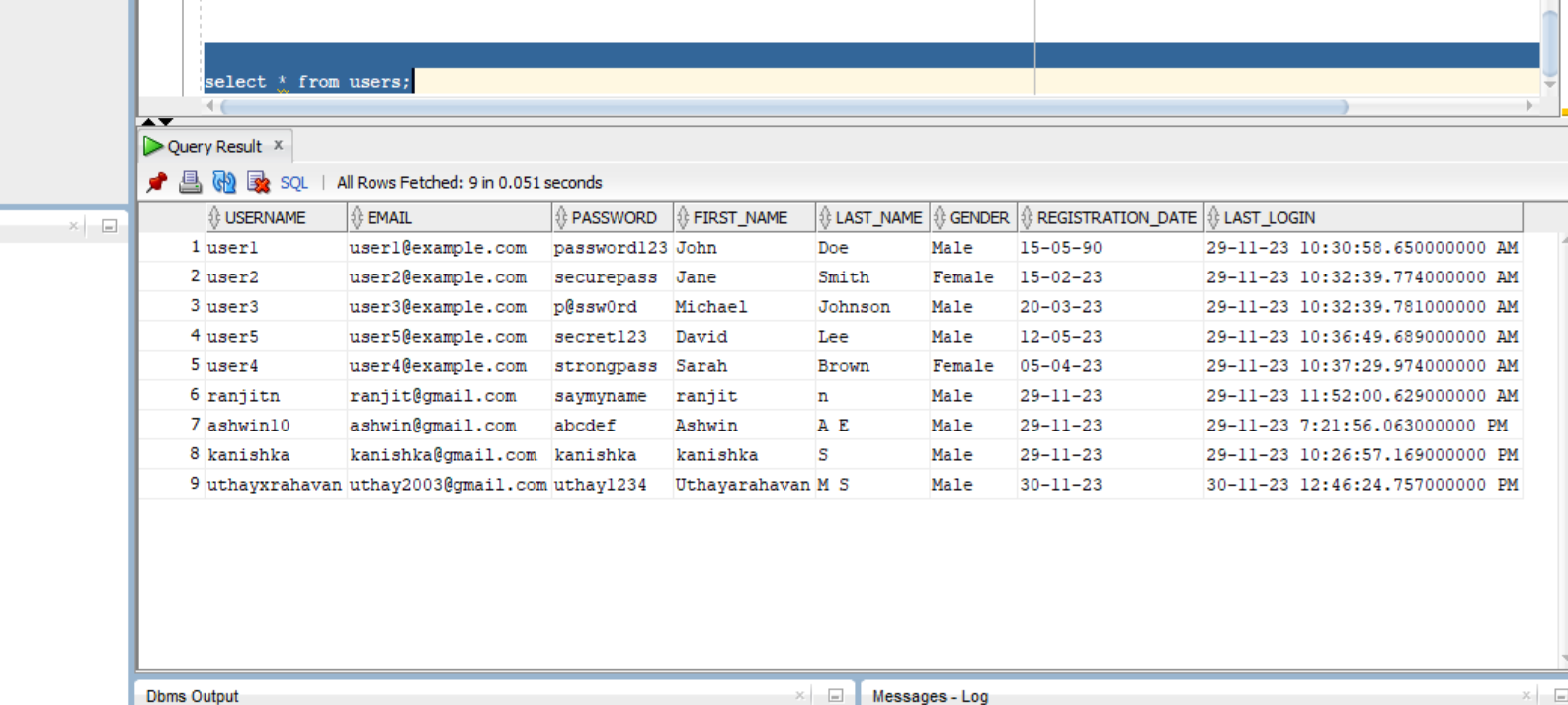
**<<Repeat the 1.insert for all the operations delete,update, search,display>>**

**1. Insert**

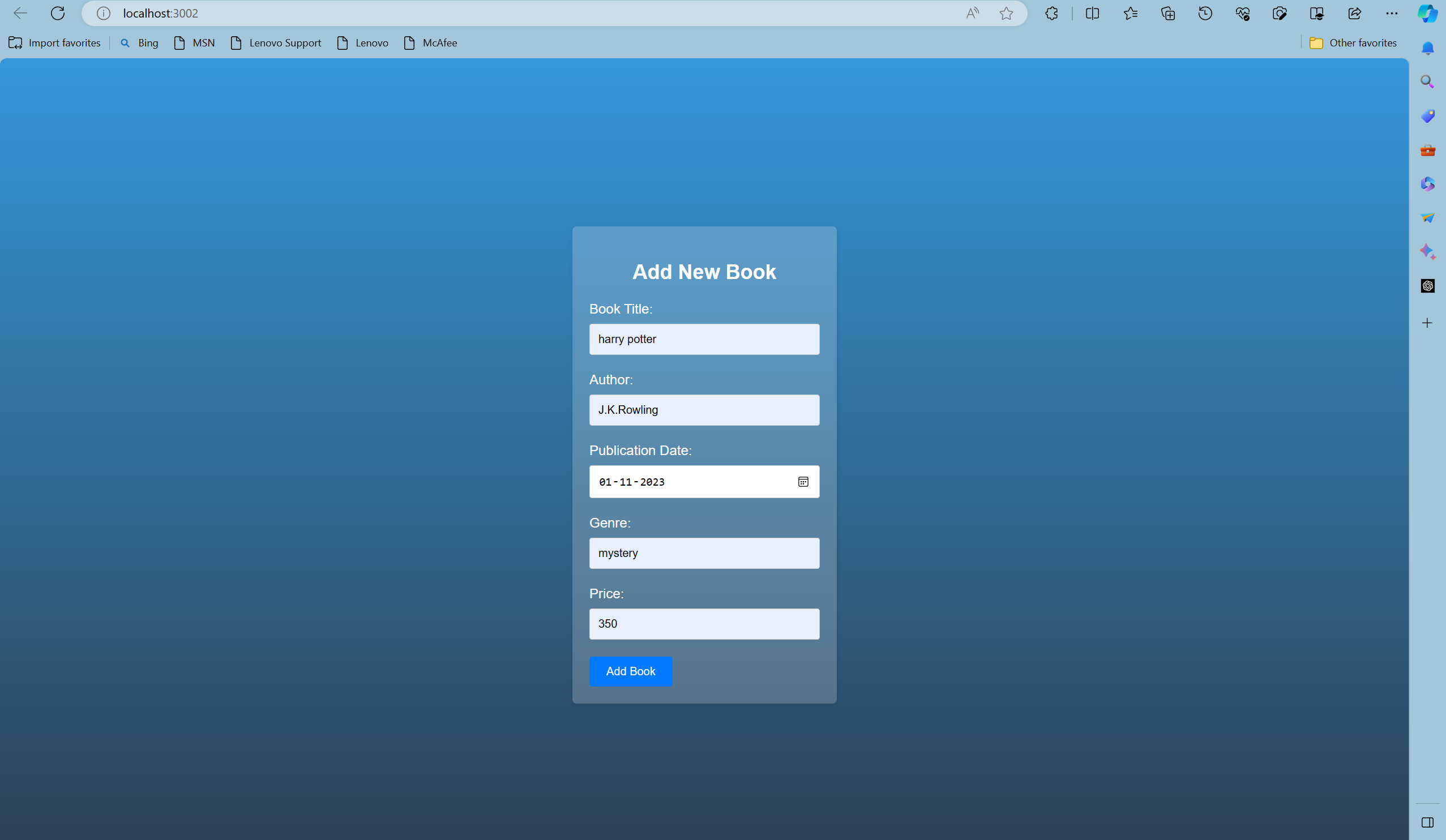
**<<keep the screenshot for the form operations for all the forms>>**

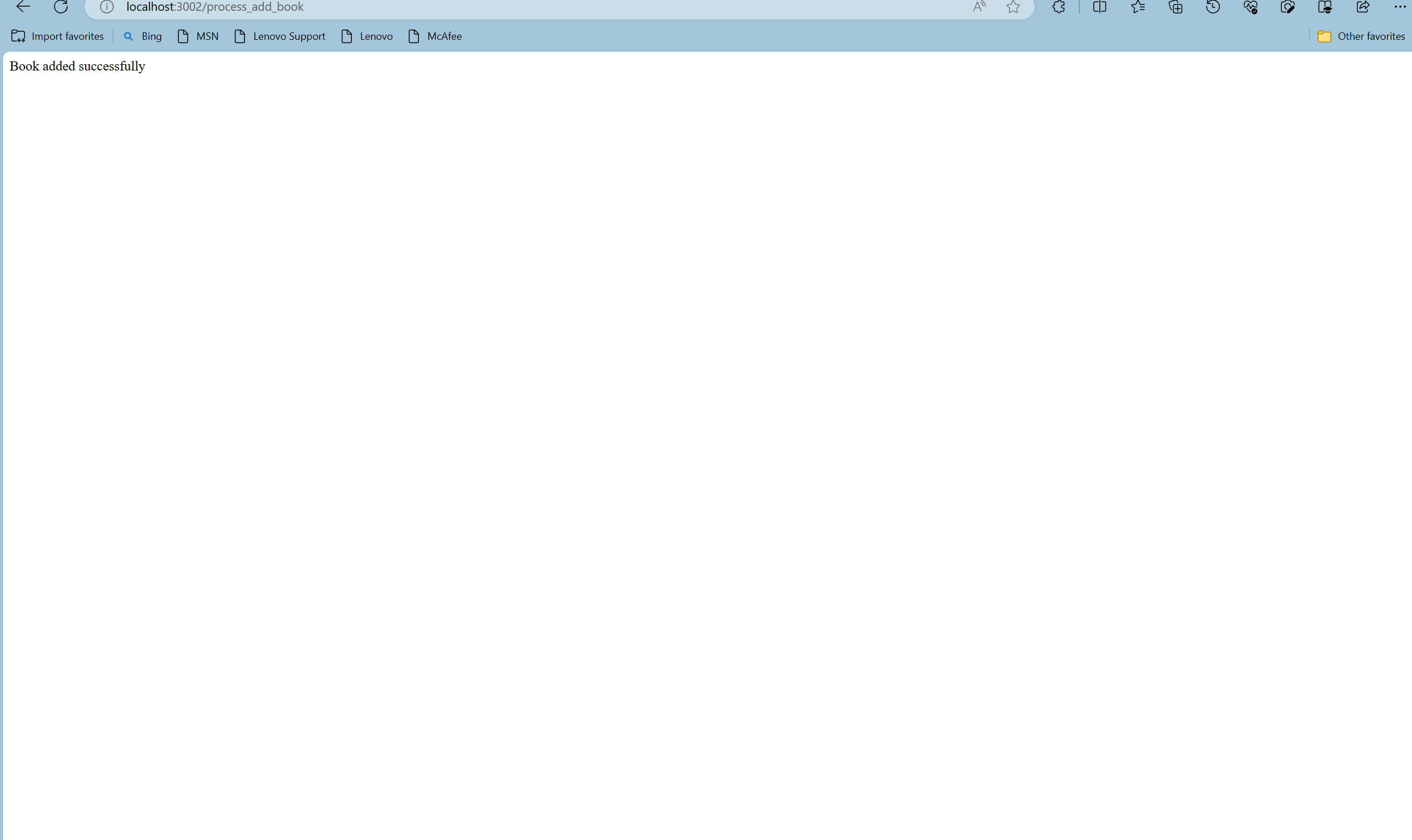
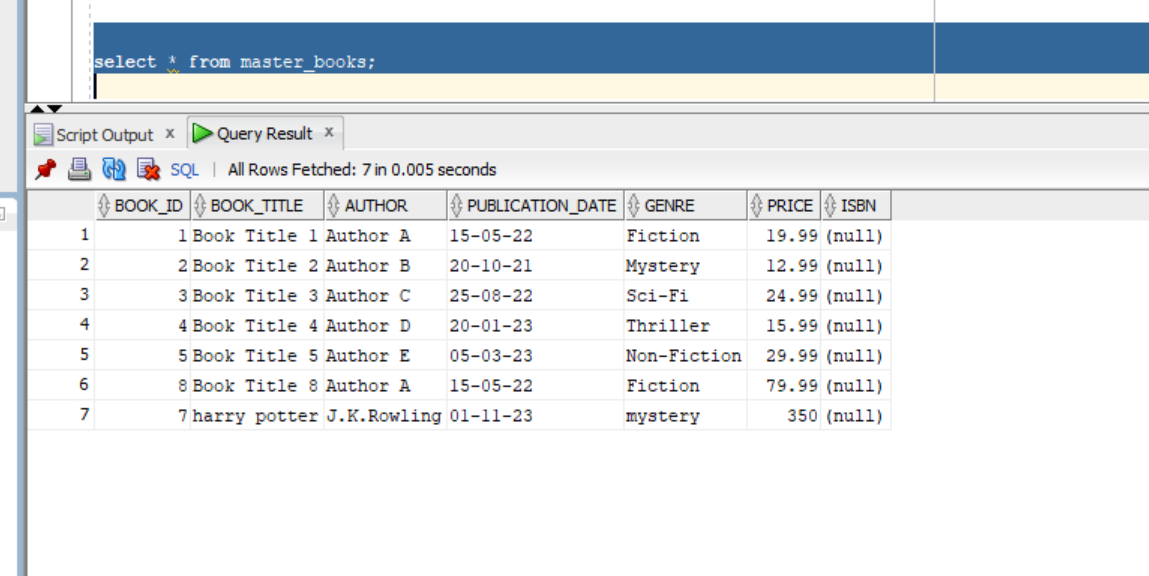
****

****

****

**Adding books:**

****

**** ****

**Inference**

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

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

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

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

**Technologies:**

|  |  |
| --- | --- |
| **Front End** | **HTML** |
| **Back End** | **MongoDB** |
| **Editor** | **VS Code** |
| **Language** | **JavaScript** |
| **Framework** | **Node.js** |
|  |  |
|  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Form Name** | **Category(Master**  **/Transaction)** | **Table Name associated with the form** | **Form Name** | **File Name for reference in the shared drive** | **Type of operations**  **(Insert/Update/Delete**  **/Search/Display)** |
| **Ma\_movies** | **Master** | **Master\_movies** | **Ma\_movies.html**  **Ma\_movies.js** | **Project files – NoSQL, Videos** | **Insert** |

**-- 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** |  |