Library Management System - Part – 5

(Functions, Procedures, Package, and Triggers)

Updations made for Part – 5:

A few new assumptions that we had to include in the table structures are highlighted here below in the DDL queries -

- We are including the On-time factor, lost_books, and fine-amount in the user's table to keep track of these respective things based on the user.
- Also, in the return table added new columns on Time, and Late for easier access to data that helps analyze other necessary details.

```
create table Users(
  User_ID NUMBER(8) NOT NULL,
  User_name VARCHAR2(25),
  User_contact NUMBER(20) NOT NULL UNIQUE,
  User_street VARCHAR2(40) NOT NULL,
  User_city VARCHAR2(15) NOT NULL,
  User_State VARCHAR2(15) NOT NULL,
  User_country VARCHAR2(40),
  User_zip_code VARCHAR2(9),
  Late_Factor NUMBER(2),
  On_time_factor NUMBER(2),
  Lost_books NUMBER(2),
  Fine_Amount NUMBER(5),
  PRIMARY KEY (User_ID)
  );

insert into Users values (40011, 'K. Christina', 3426757979, 'W. Oak', 'San Antonio', 'Texas', 'US',
'78015',2,2,14,0);
insert into Users values (40012, 'Kirk', 7896757979, 'A. Oak', 'San Francisco', 'California', 'US',
'76207',3,4,2,0);
insert into Users values (40013, 'Mary', 0136757979, 'A. Oak', 'San Francisco', 'California', 'US',
'76207',5,3,9,0);
insert into Users values (40014, 'Rosy', 0146757979, 'A. Oak', 'San Francisco', 'California', 'US',
'76207',1,4,5,0);
```

```
insert into Users values (40015, 'Chris', 0156757979, 'A. Oak', 'San Francisco', 'California', 'US',
'76207',4,2,15,0);
insert into Users values (40016, 'Tracy', 0166757979, 'Fry St', 'Galvaston', 'Texas', 'US',
'76207',8,5,11,0);
insert into Users values (40017, 'Bruce', 0176757979, 'Bleeker St', 'Gotham City', 'Michigan',
'US', '76207',2,2,5,0);
insert into Users values (40018, 'Garg', 0186757979, '10 Downing St', 'Boston', 'Massachusetts',
'US', '76207',4,6,16,0);
insert into Users values (40019, 'Henry', 0196757979, 'A. Oak', 'Houston', 'Texas', 'US',
'76207',8,1,12,0);
insert into Users values (40020, 'Feng', 0206757979, 'JFK st', 'Queens', 'New York', 'US',
'76207',6,2,8,0);

create table Return(
  Book_ID NUMBER(8),
  User_ID NUMBER(8),
  Expected_Return_date DATE,
  Actual_Return_date DATE,
  OnTime NUMBER(1),
  Late NUMBER(1),
  PRIMARY KEY (Book_ID, User_ID)
  );
```

Functions:

The purpose of functions is to help calculate/analyze and aggregate data while maintaining security and avoiding redundancy in the system. Below are a few functions for the library management system with descriptions for each of them.

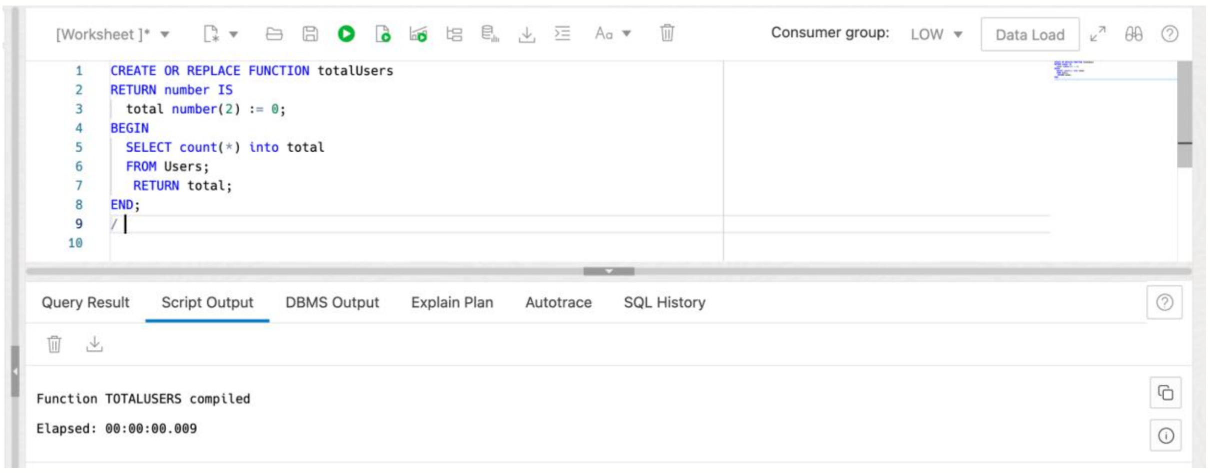
Function 1: Total count of the Users.

Description: Display the count of the total users.

Implementation:

```
CREATE OR REPLACE FUNCTION totalUsers
```

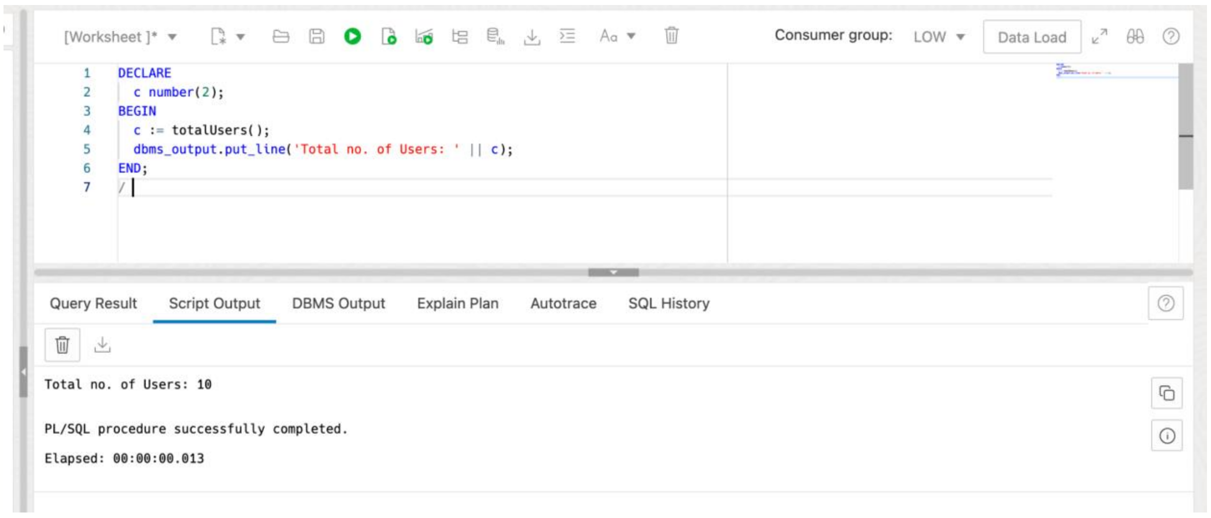
```
RETURN number IS
 total number(2) := 0;
BEGIN
 SELECT count(*) into total
 FROM Users;
  RETURN total;
END;
/
```



```
CREATE OR REPLACE FUNCTION totalUsers
RETURN number IS
  total number(2) := 0;
BEGIN
  SELECT count(*) into total
  FROM Users;
   RETURN total;
END;
/
```

```
Function TOTALUSERS compiled
Elapsed: 00:00:00.009
```

```
DECLARE
 c number(2);
BEGIN
 c := totalUsers();
 dbms_output.put_line('Total no. of Users: ' || c);
END;
/
```



```
DECLARE
  c number(2);
BEGIN
  c := totalUsers();
  dbms_output.put_line('Total no. of Users: ' || c);
END;
/
```

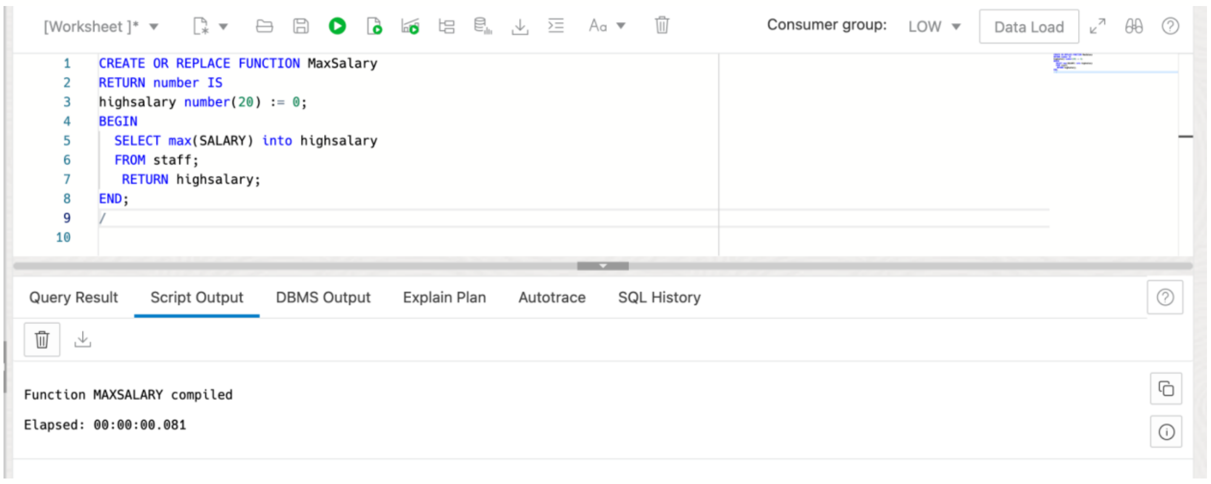
```
Total no. of Users: 10

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.013
```

Function 2: Highest Salary of the Staff.

Description: To calculate the highest salary of staff.

Implementation:

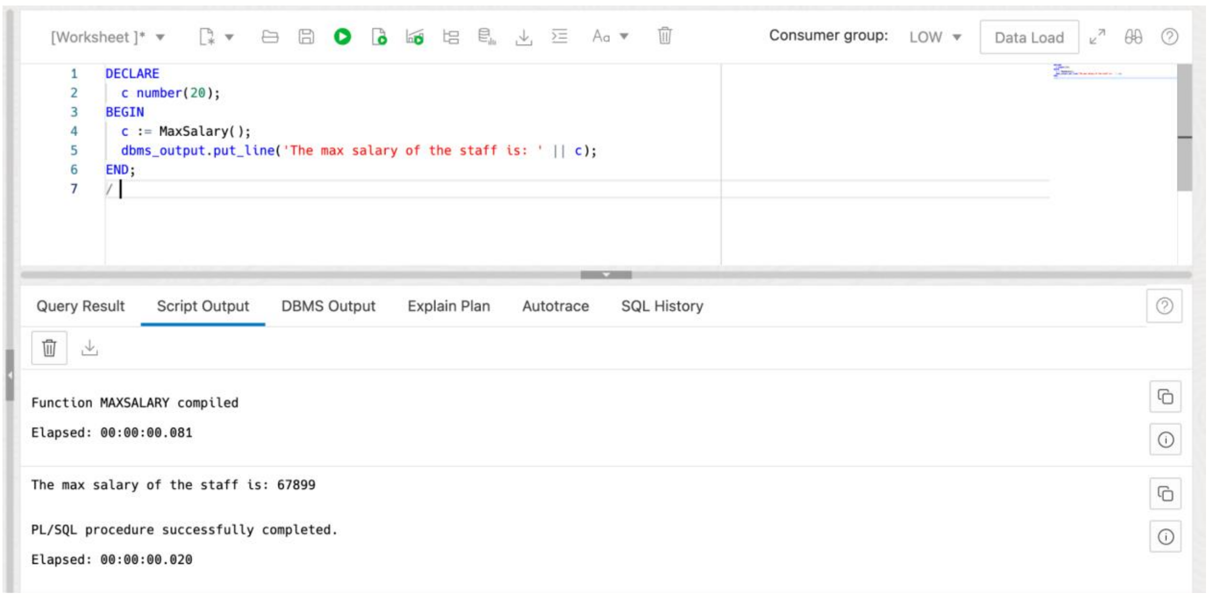
```
CREATE OR REPLACE FUNCTION MaxSalary
RETURN number IS
highsalary number(20) := 0;
BEGIN
 SELECT max(SALARY) into highsalary
 FROM staff;
  RETURN highsalary;
END;
/
```



```
CREATE OR REPLACE FUNCTION MaxSalary
RETURN number IS
highsalary number(20) := 0;
BEGIN
  SELECT max(SALARY) into highsalary
  FROM staff;
   RETURN highsalary;
END;
/
```

```
Function MAXSALARY compiled

Elapsed: 00:00:00.081
```

```
DECLARE
 c number(20);
BEGIN
 c := MaxSalary();
 dbms_output.put_line('The max salary of the staff is: ' || c);
END;
/
```



```
DECLARE
  c number(20);
BEGIN
  c := MaxSalary();
  dbms_output.put_line('The max salary of the staff is: ' || c);
END;
/
```

```
Function MAXSALARY compiled

Elapsed: 00:00:00.081

The max salary of the staff is: 67899

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.020
```

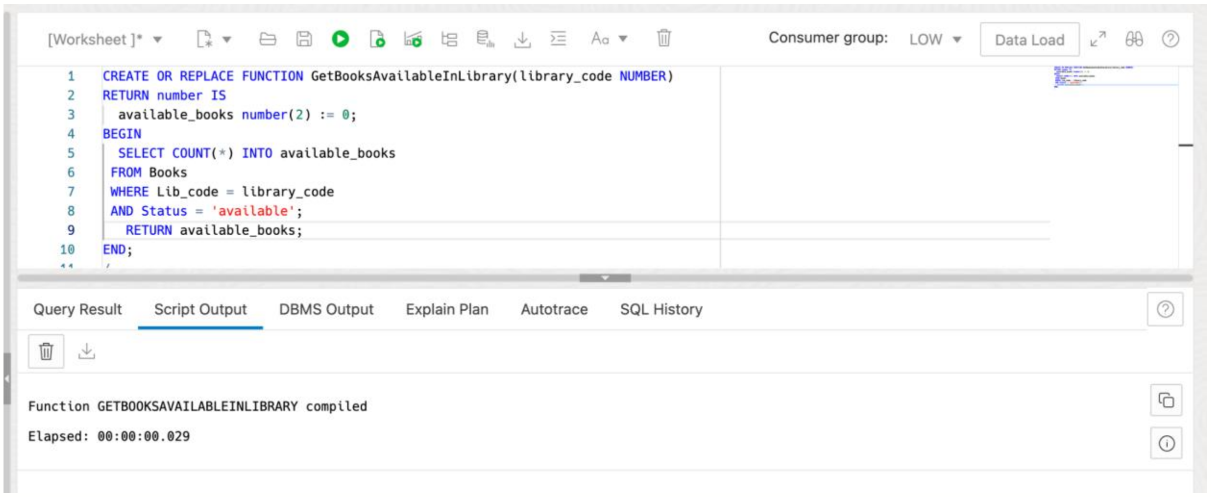
Function 3: Get available books in the Library.

Description: This function returns the available books in the library.

Implementation:

```
CREATE OR REPLACE FUNCTION GetBooksAvailableInLibrary(library_code NUMBER)
RETURN number IS
```

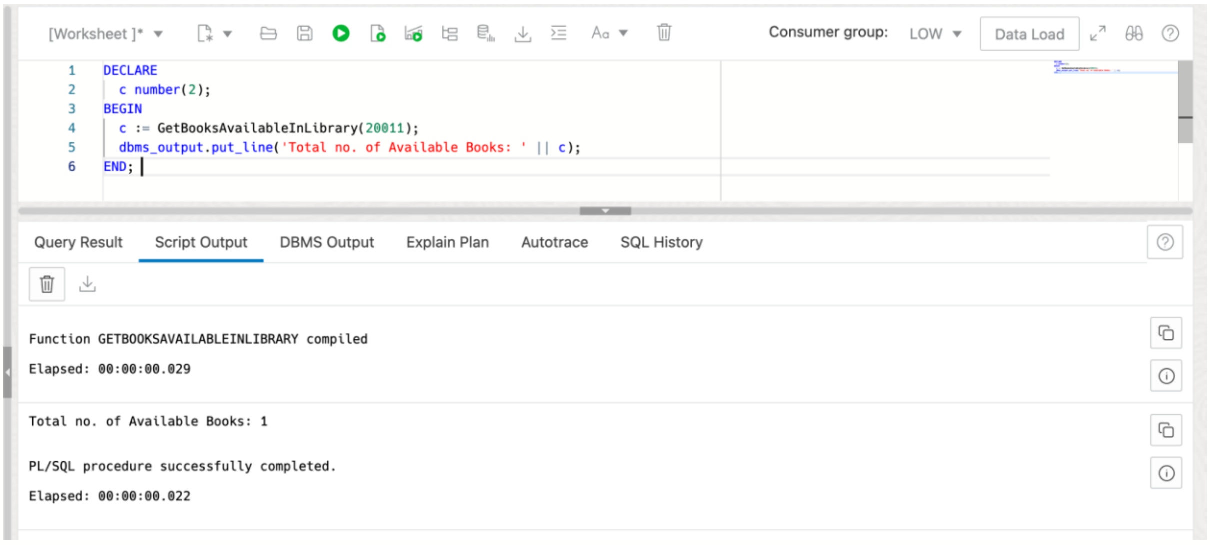
```
 available_books number(2) := 0;
BEGIN
 SELECT COUNT(*) INTO available_books
 FROM Books
 WHERE Lib_code = library_code
 AND Status = 'available';
  RETURN available_books;
END;
/
```



```
CREATE OR REPLACE FUNCTION GetBooksAvailableInLibrary(library_code NUMBER)
RETURN number IS
  available_books number(2) := 0;
BEGIN
  SELECT COUNT(*) INTO available_books
 FROM Books
 WHERE Lib_code = library_code
 AND Status = 'available';
   RETURN available_books;
END;
```

```
Function GETBOOKSAVAILABLEINLIBRARY compiled

Elapsed: 00:00:00.029
```

```
DECLARE
 c number(2);
BEGIN
 c := GetBooksAvailableInLibrary(20011);
 dbms_output.put_line('Total no. of Available Books: ' || c);
END;
```



```
DECLARE
  c number(2);
BEGIN
  c := GetBooksAvailableInLibrary(20011);
  dbms_output.put_line('Total no. of Available Books: ' || c);
END;
```

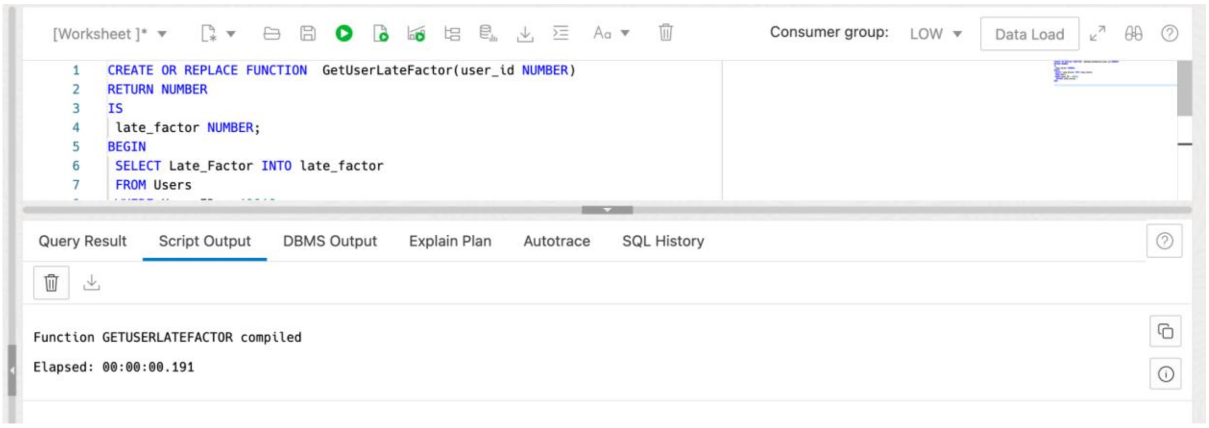
```
Function GETBOOKSAVAILABLEINLIBRARY compiled

Elapsed: 00:00:00.029

Total no. of Available Books: 1

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.022
```

Function 4: Get the late factor of a particular user.

Description: This function takes a user ID as input and returns the late factor for that user, which is calculated based on the number of late returns they have had in the past.

Implementation:

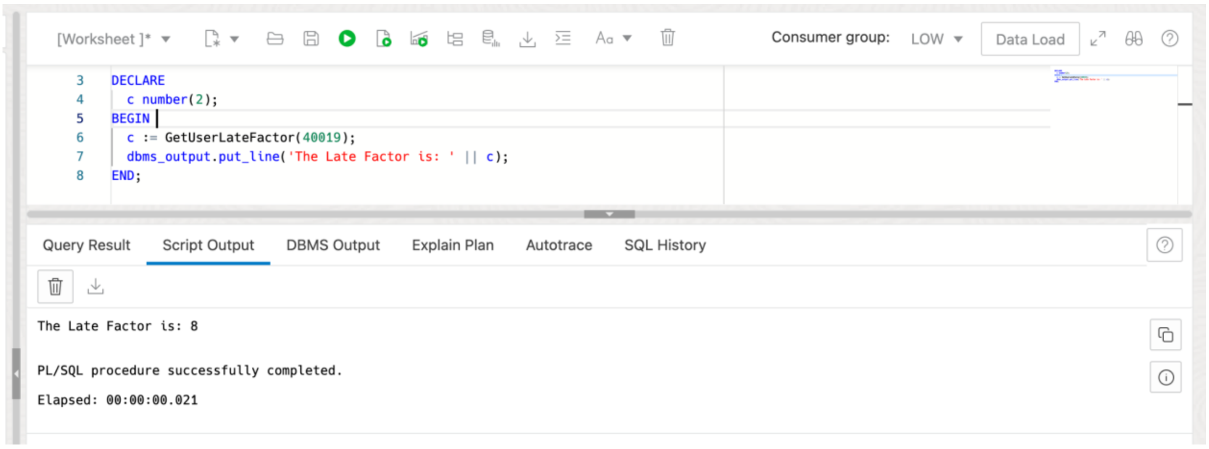
```
CREATE OR REPLACE FUNCTION  GetUserLateFactor(user_id NUMBER)
RETURN NUMBER
IS
 late_factor NUMBER;
BEGIN
 SELECT Late_Factor INTO late_factor
 FROM Users
 WHERE User_ID = 40019;
  RETURN late_factor;
END;
/
```



```
CREATE OR REPLACE FUNCTION  GetUserLateFactor(user_id NUMBER)
RETURN NUMBER
IS
 late_factor NUMBER;
BEGIN
 SELECT Late_Factor INTO late_factor
 FROM Users
```

```
Function GETUSERLATEFACTOR compiled

Elapsed: 00:00:00.191
```

DECLARE
 c number(2);
BEGIN
 c := GetUserLateFactor(40019);
 dbms_output.put_line('The Late Factor is: ' || c);
END;



```
DECLARE
  c number(2);
BEGIN
  c := GetUserLateFactor(40019);
  dbms_output.put_line('The Late Factor is: ' || c);
END;
```

```
The Late Factor is: 8

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.021
```

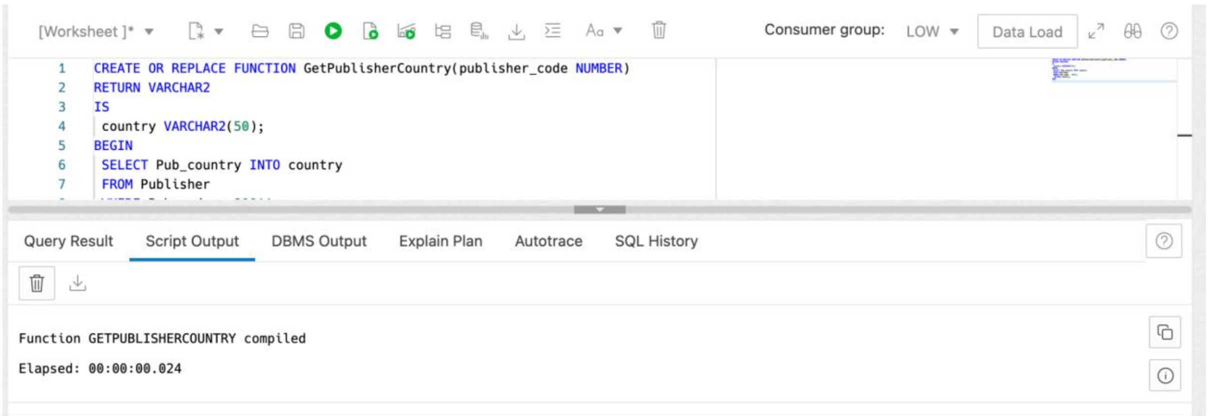
Function 5: Get the country of the publisher.

Description: This function takes a publisher code as input and returns the country where the publisher is located.

Implementation:

CREATE OR REPLACE FUNCTION GetPublisherCountry(publisher_code NUMBER)
RETURN VARCHAR2
IS

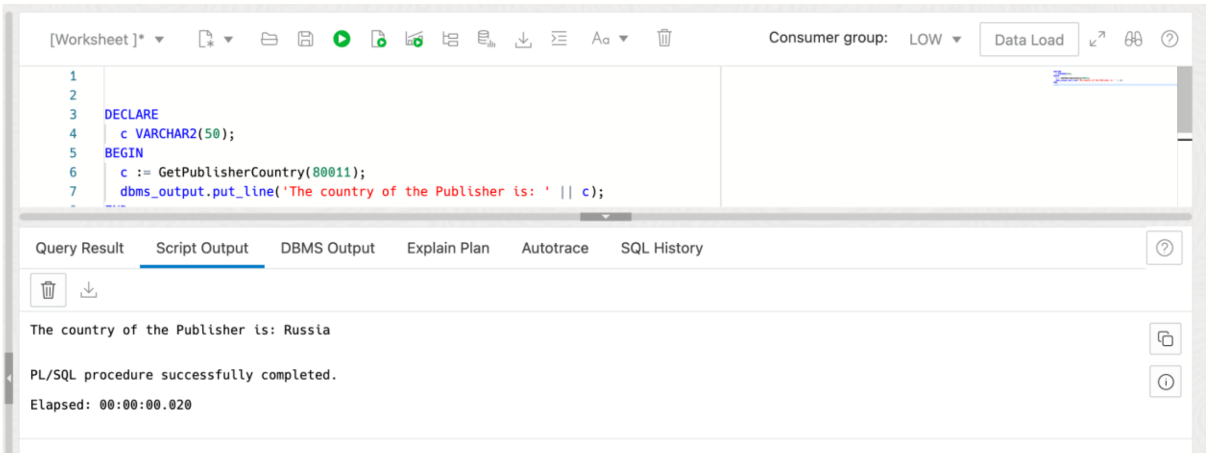
```
 country VARCHAR2(50);
BEGIN
 SELECT Pub_country INTO country
 FROM Publisher
 WHERE Pub_code = 80011;
  RETURN country;
END;
/
```



```
CREATE OR REPLACE FUNCTION GetPublisherCountry(publisher_code NUMBER)
RETURN VARCHAR2
IS
 country VARCHAR2(50);
BEGIN
 SELECT Pub_country INTO country
 FROM Publisher
```

```
Function GETPUBLISHERCOUNTRY compiled
Elapsed: 00:00:00.024
```

```
DECLARE
  c VARCHAR2(50);
BEGIN
  c := GetPublisherCountry(80011);
  dbms_output.put_line('The country of the Publisher is: ' || c);
END;
/
```



```


DECLARE
  c VARCHAR2(50);
BEGIN
  c := GetPublisherCountry(80011);
  dbms_output.put_line('The country of the Publisher is: ' || c);
```

```
The country of the Publisher is: Russia

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.020
```

Procedures:

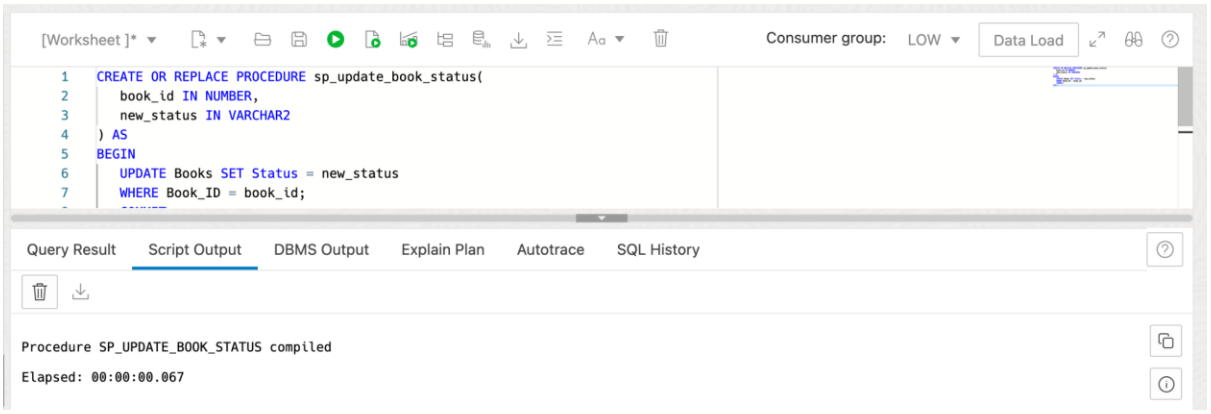
These can be used to execute the SQL statements dynamically using immediately execute statements that are just compiled once and stored in an executable form. Following are a few procedures that help us achieve this-

Procedure 1: Update the book status.

Description: This procedure will update the status of the book.

Implementation:

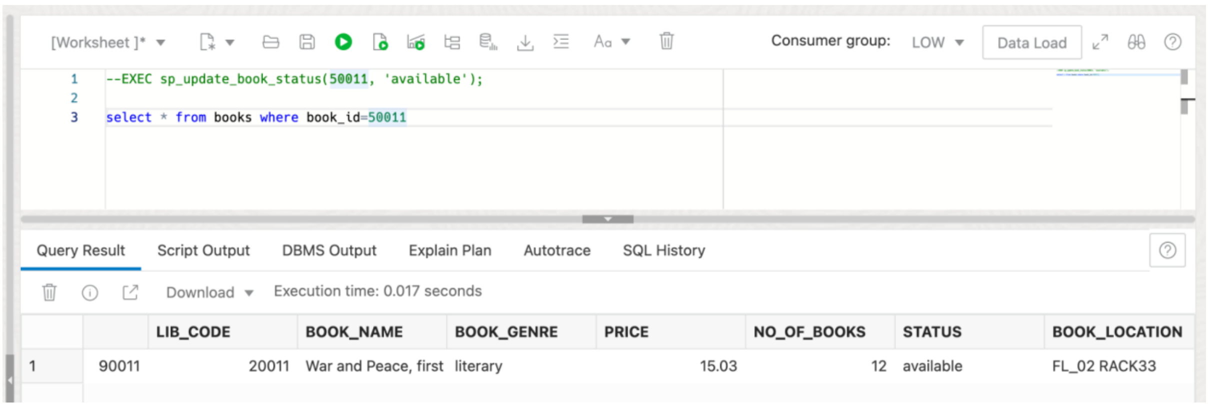
```
CREATE OR REPLACE PROCEDURE sp_update_book_status(
  book_id IN NUMBER,
  new_status IN VARCHAR2
) AS
BEGIN
  UPDATE Books SET Status = new_status
  WHERE Book_ID = book_id;
  COMMIT;
END;
/
```



```
CREATE OR REPLACE PROCEDURE sp_update_book_status(
   book_id IN NUMBER,
   new_status IN VARCHAR2
) AS
BEGIN
   UPDATE Books SET Status = new_status
   WHERE Book_ID = book_id;
```

```
Procedure SP_UPDATE_BOOK_STATUS compiled
Elapsed: 00:00:00.067
```

EXEC sp_update_book_status(50011, 'available');



```
--EXEC sp_update_book_status(50011, 'available');

select * from books where book_id=50011
```

Execution time: 0.017 seconds

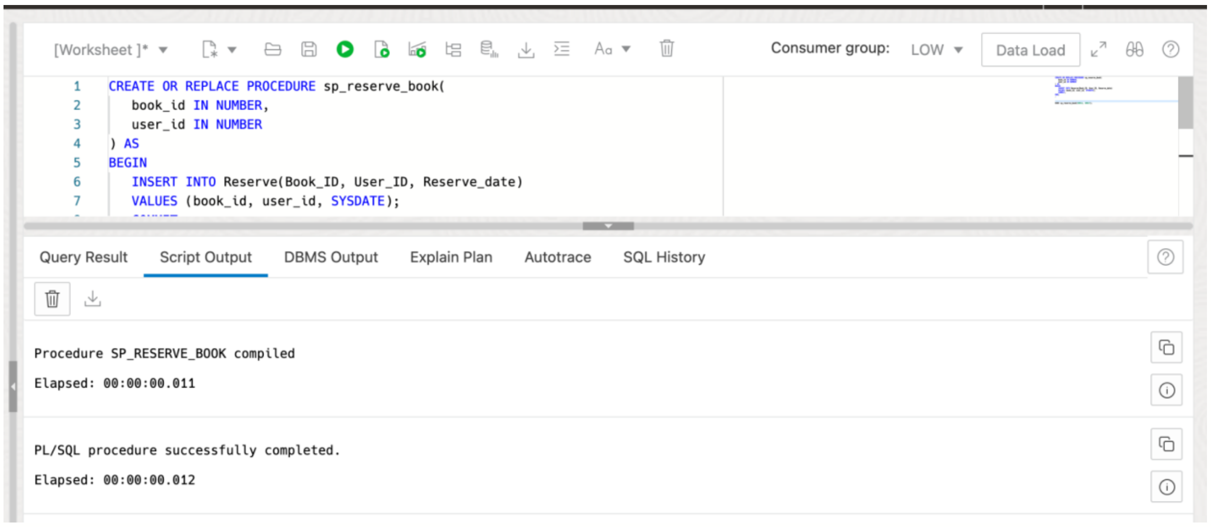
	LIB_CODE	BOOK_NAME	BOOK_GENRE	PRICE	NO_OF_BOOKS	STATUS	BOOK_LOCATION
1	90011	20011	War and Peace, first	literary	15.03	12 available	FL_02 RACK33

Procedure 2: Reserve the book.

Description: This procedure will reserve the book.

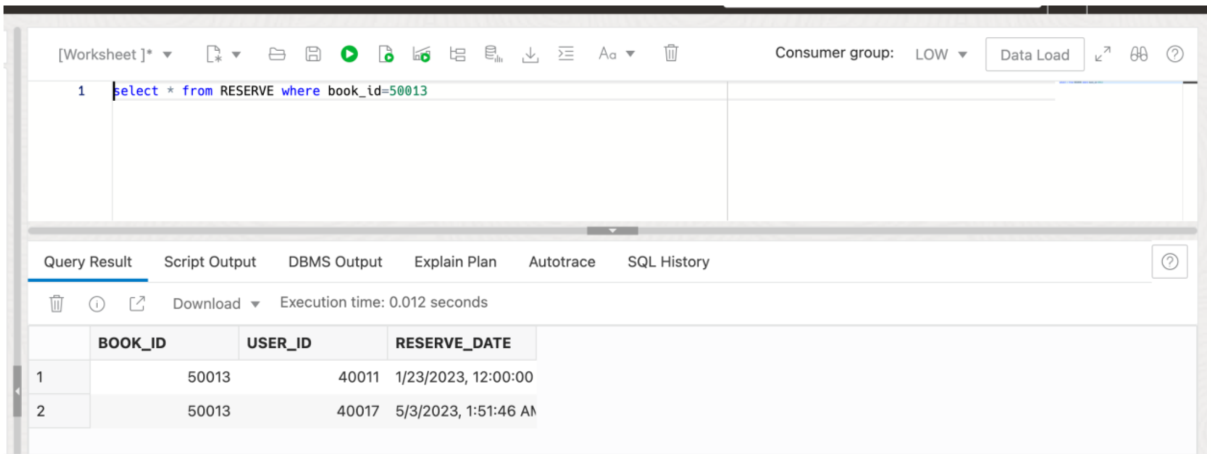
Implementation:

```
CREATE OR REPLACE PROCEDURE sp_reserve_book(
  book_id IN NUMBER,
  user_id IN NUMBER
) AS
BEGIN
  INSERT INTO Reserve(Book_ID, User_ID, Reserve_date)
  VALUES (book_id, user_id, SYSDATE);
  COMMIT;
END;
/

EXEC sp_reserve_book(50011, 40017);
```



```
CREATE OR REPLACE PROCEDURE sp_reserve_book(
   book_id IN NUMBER,
   user_id IN NUMBER
) AS
BEGIN
   INSERT INTO Reserve(Book_ID, User_ID, Reserve_date)
   VALUES (book_id, user_id, SYSDATE);
```

```
Procedure SP_RESERVE_BOOK compiled

Elapsed: 00:00:00.011

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.012
```



```
select * from RESERVE where book_id=50013
```

	BOOK_ID	USER_ID	RESERVE_DATE
1	50013	40011	1/23/2023, 12:00:00
2	50013	40017	5/3/2023, 1:51:46 AM

Procedure 3: Update the book status.

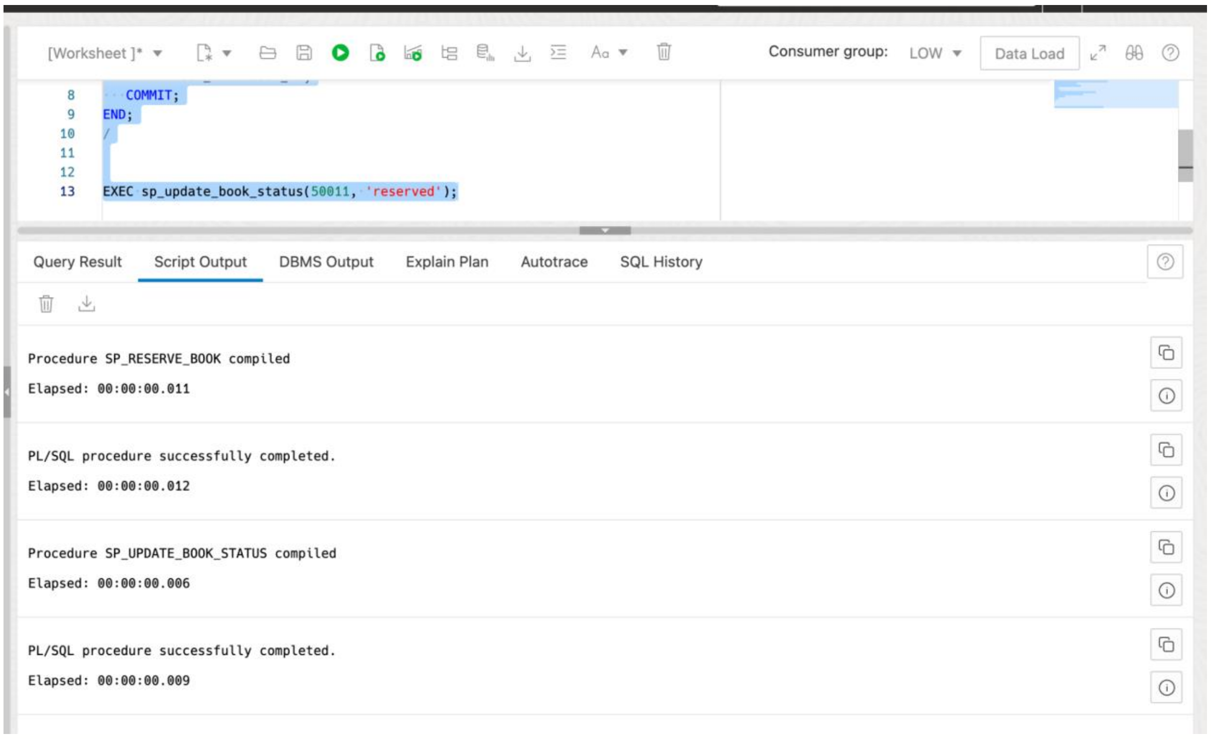
Description: This procedure will update the status of the book.

Implementation:

```
CREATE OR REPLACE PROCEDURE sp_update_book_status(
  book_id IN NUMBER,
  new_status IN VARCHAR2
) AS
BEGIN
  UPDATE Books SET Status = new_status
  WHERE Book_ID = book_id;
  COMMIT;
END;
```

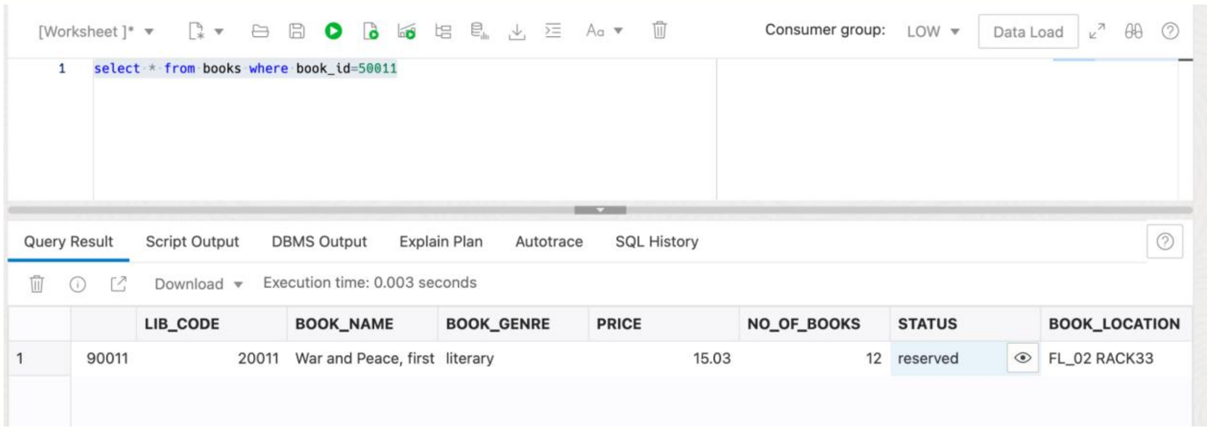
/

EXEC sp_update_book_status(50011, 'reserved');



```
    COMMIT;
END;
/


EXEC sp_update_book_status(50011, 'reserved');
```

```
Procedure SP_RESERVE_BOOK compiled
Elapsed: 00:00:00.011

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.012

Procedure SP_UPDATE_BOOK_STATUS compiled
Elapsed: 00:00:00.006

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.009
```



```
select * from books where book_id=50011
```

Execution time: 0.003 seconds

	LIB_CODE	BOOK_NAME	BOOK_GENRE	PRICE	NO_OF_BOOKS	STATUS	BOOK_LOCATION
1	90011 20011	War and Peace, first	literary	15.03	12	reserved	FL_02 RACK33

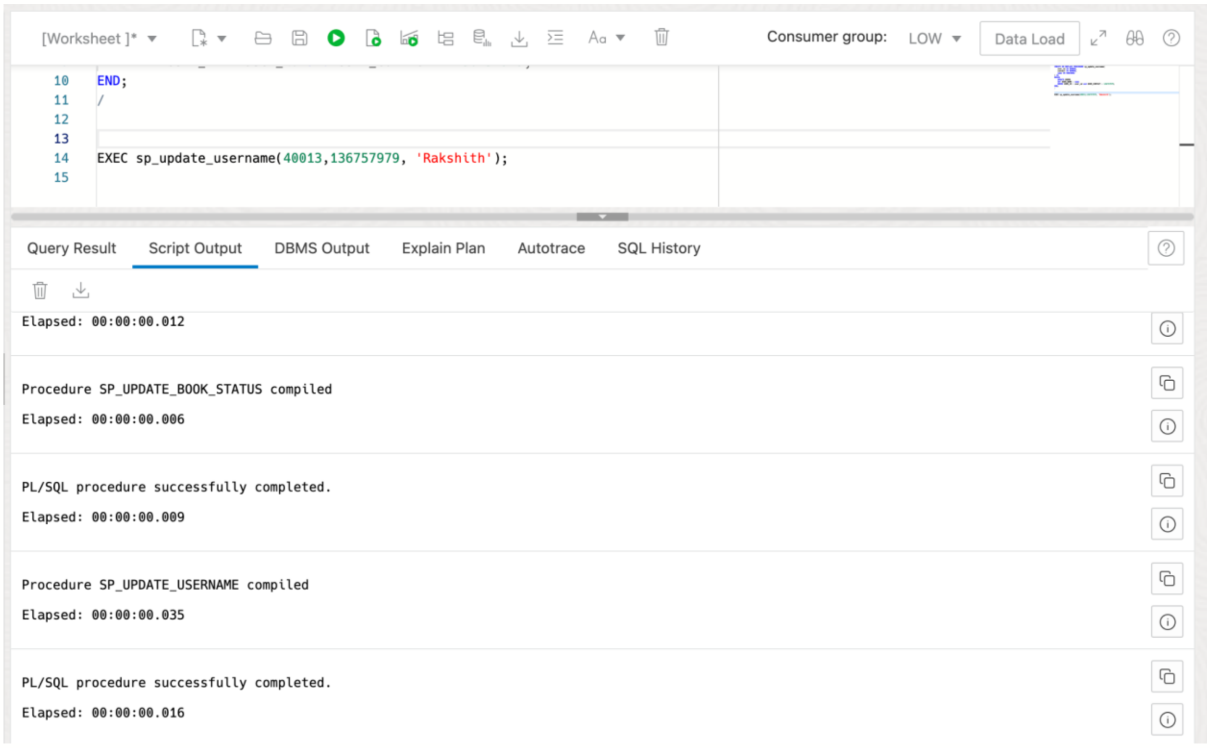
Procedure 4: Update the Name of the user.

Description: This procedure will update the Name of the user.

Implementation:

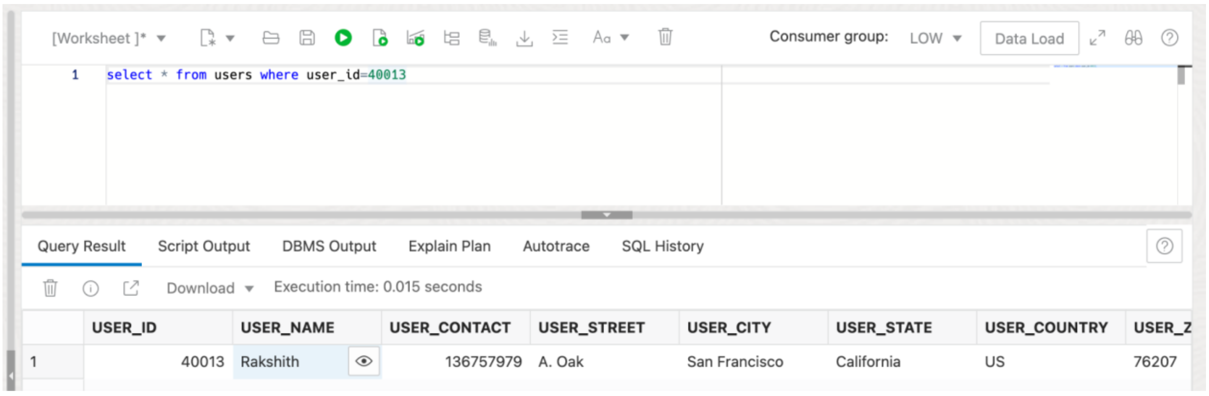
CREATE OR REPLACE PROCEDURE sp_update_username(

```
  user_id IN NUMBER,
  contact IN NUMBER,
  name IN VARCHAR2
) AS
BEGIN
  UPDATE USERS
  SET USER_NAME = name
  WHERE USER_ID = user_id and USER_CONTACT = 136757979;
END;
/

EXEC sp_update_username(40013,136757979, 'Rakshith');
```



```
END;
/

EXEC sp_update_username(40013,136757979, 'Rakshith');
```

```
Elapsed: 00:00:00.012

Procedure SP_UPDATE_BOOK_STATUS compiled

Elapsed: 00:00:00.006

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.009

Procedure SP_UPDATE_USERNAME compiled

Elapsed: 00:00:00.035

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.016
```



```
select * from users where user_id=40013
```

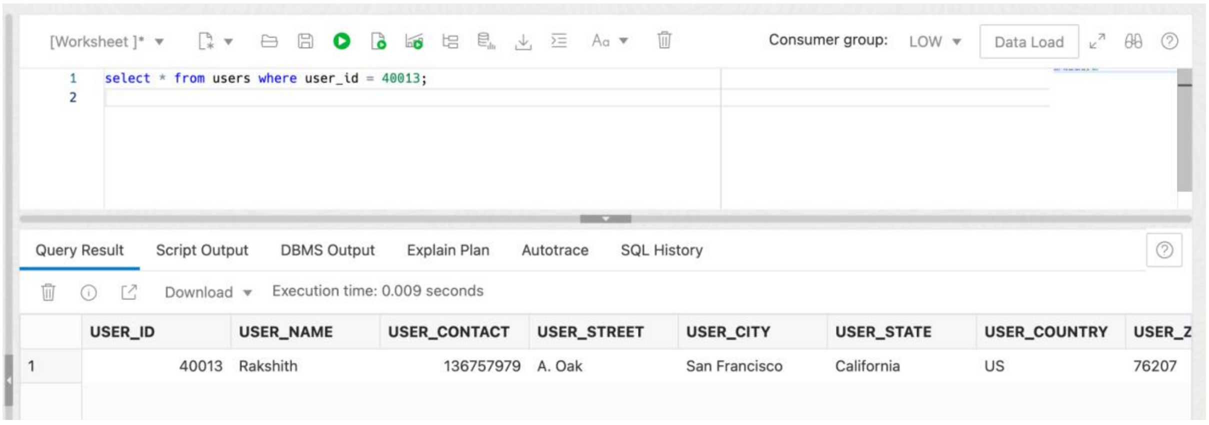
	USER_ID	USER_NAME	USER_CONTACT	USER_STREET	USER_CITY	USER_STATE	USER_COUNTRY	USER_Z
1	40013	Rakshith	136757979	A. Oak	San Francisco	California	US	76207

Procedure 5: Delete the user.

Description: This procedure will Delete the user.

Implementation:

```
select * from users where user_id = 40013;
```

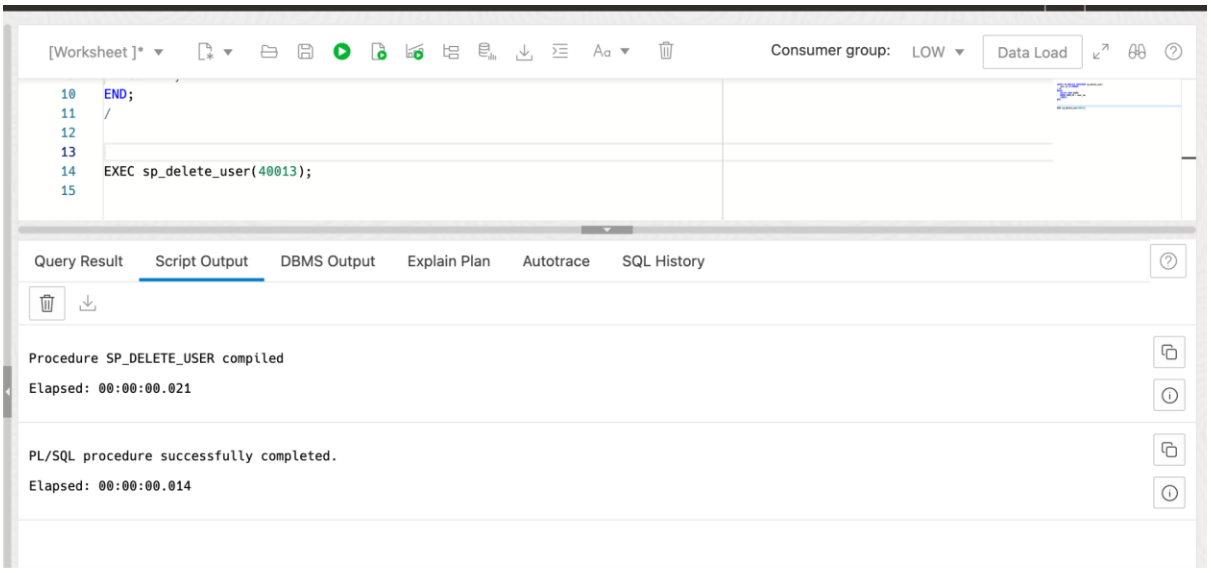


```
select * from users where user_id = 40013;
```

	USER_ID	USER_NAME	USER_CONTACT	USER_STREET	USER_CITY	USER_STATE	USER_COUNTRY	USER_Z
1	40013	Rakshith	136757979	A. Oak	San Francisco	California	US	76207

```
CREATE OR REPLACE PROCEDURE sp_delete_user(
  user_id IN NUMBER
) AS
BEGIN
  DELETE FROM USERS
  WHERE USER_ID = user_id;
  COMMIT;
END;
/
```

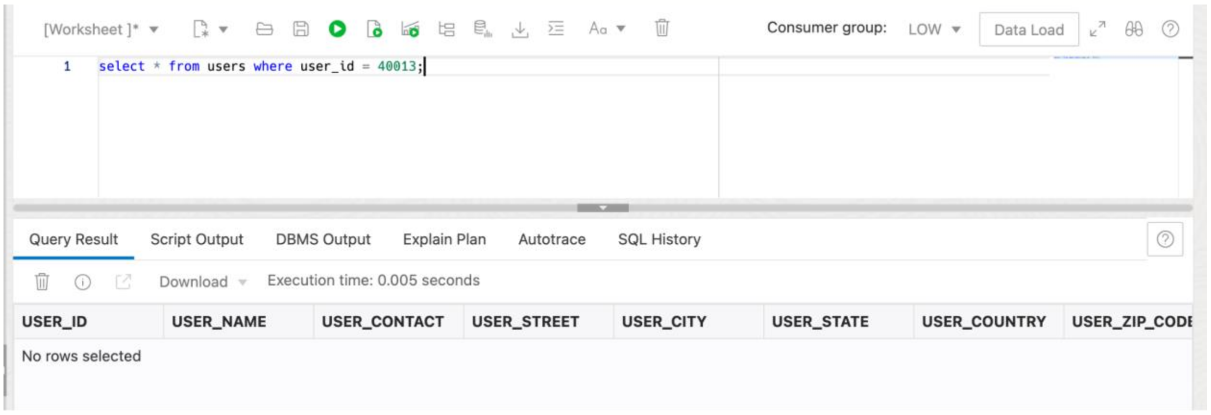
EXEC sp_delete_user(40013);



```
END;
/


EXEC sp_delete_user(40013);
```

```
Procedure SP_DELETE_USER compiled
Elapsed: 00:00:00.021

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.014
```

select * from users where user_id = 40013;



```
select * from users where user_id = 40013;
```

Execution time: 0.005 seconds

USER_ID	USER_NAME	USER_CONTACT	USER_STREET	USER_CITY	USER_STATE	USER_COUNTRY	USER_ZIP_CODE
No rows selected							

Triggers:

In general, Triggers are procedures that are stored in the database and implicitly run, or fired, when something happens. Here few triggers that help our Library management system.

Trigger 1: Late factor trigger.

CREATE OR REPLACE TRIGGER TRG_INCREMENT_LATE_FACTOR

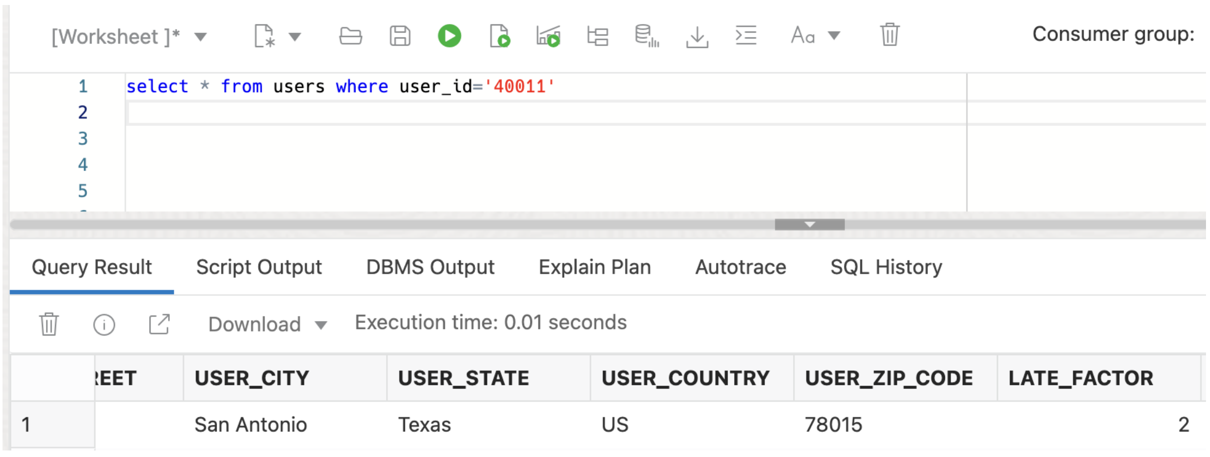
```
AFTER INSERT ON RETURN
FOR EACH ROW
WHEN (NEW.Late = 1)
BEGIN
  UPDATE Users
  SET Late_Factor = Late_Factor + 1
  WHERE User_ID = :NEW.User_ID;
END;
/
```

Description:

Late factor: which is an integer type with a range of 2 digits (max value 99).
When a record is inserted into the Return table, like a user is returning a book on a particular date which is later than the expected date of return this is called late submission,
We have a column late factor in the user's table and if a user is submitting any book late, we will make an entry in the return table, which will trigger an add +1 to the late factor for that user and update the same in the user's table.

Implementation:

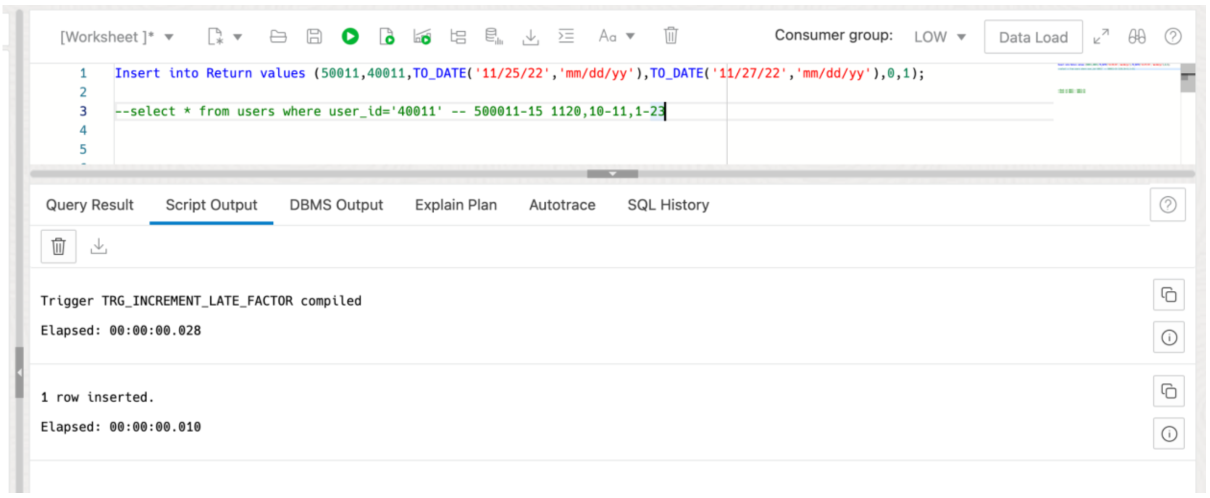
We can see the late factor for the below user is 2.



```
select * from users where user_id='40011'
```

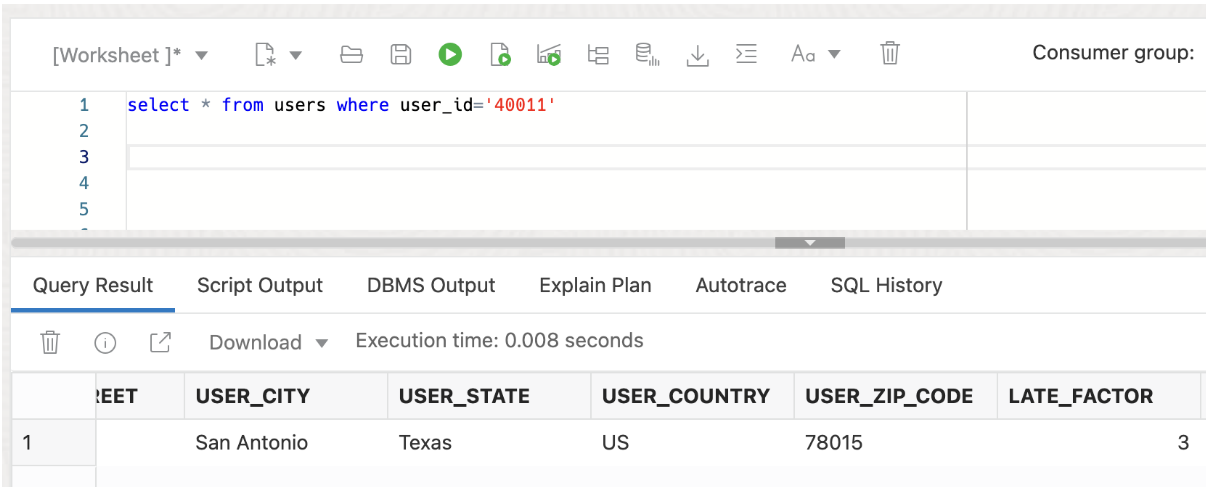
	:EET	USER_CITY	USER_STATE	USER_COUNTRY	USER_ZIP_CODE	LATE_FACTOR
1		San Antonio	Texas	US	78015	2

Then we will, update the records accordingly in reserves and return table then try to initiate this trigger. Which will in-turn increase this value 2 to 3.


```
Insert into Return values (50011,40011,TO_DATE('11/25/22','mm/dd/yy'),TO_DATE('11/27/22','mm/dd/yy'),0,1);

--select * from users where user_id='40011' -- 500011-15 1120,10-11,1-23
```

```
Trigger TRG_INCREMENT_LATE_FACTOR compiled

Elapsed: 00:00:00.028


1 row inserted.

Elapsed: 00:00:00.010
```

The late factor is now updated as 3.


```
select * from users where user_id='40011'
```

Execution time: 0.008 seconds

	EET	USER_CITY	USER_STATE	USER_COUNTRY	USER_ZIP_CODE	LATE_FACTOR
1		San Antonio	Texas	US	78015	3

Trigger 2: On-time trigger.

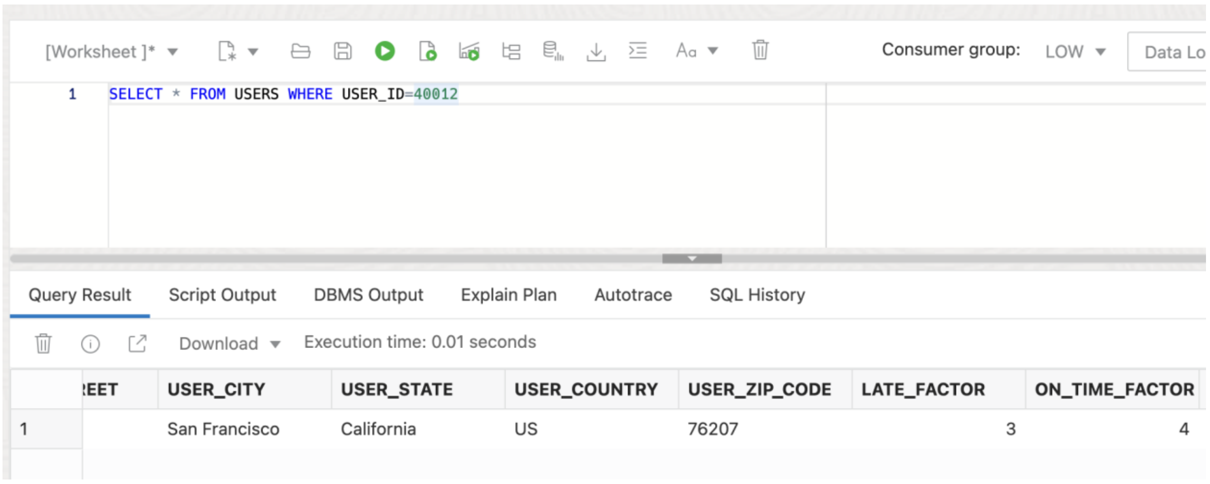
```
CREATE OR REPLACE TRIGGER TRG_INCREMENT_ONTIME_FACTOR
AFTER INSERT ON RETURN
FOR EACH ROW
WHEN (NEW.OnTime = 1)
BEGIN
 UPDATE Users
 SET On_time_factor = On_time_factor + 1
 WHERE User_ID = :NEW.User_ID;
END;
/
```

Description:

Added on_time_factor: which is an integer type with a range of 2 digits (max value 99).
When a record is inserted into the Return table, like a user is returning a book on a particular date which is way on or before the expected date of return this is called on-time submission,
We have a column on time factor in the user's table and if a user is submitting any book on time, we will make an entry in the return table, which will initiate the trigger and add +1 to the on-time factor for that user and update the same in the user's table.

Implementation:

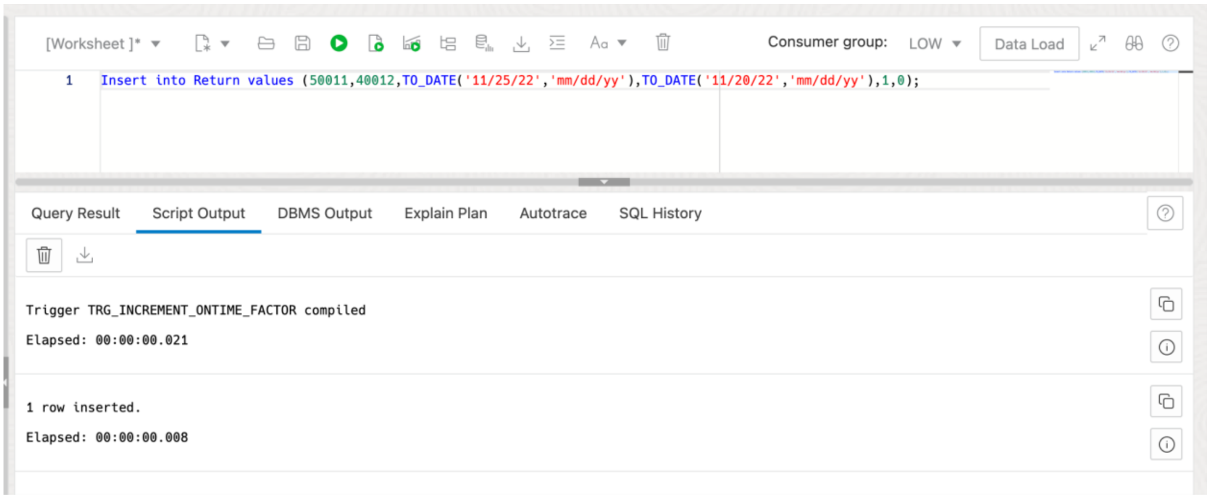
Updating on time factor for user 40012 with the on-time trigger.



```sql
SELECT * FROM USERS WHERE USER_ID=40012
```

	:EET	USER_CITY	USER_STATE	USER_COUNTRY	USER_ZIP_CODE	LATE_FACTOR	ON_TIME_FACTOR
1		San Francisco	California	US	76207	3	4

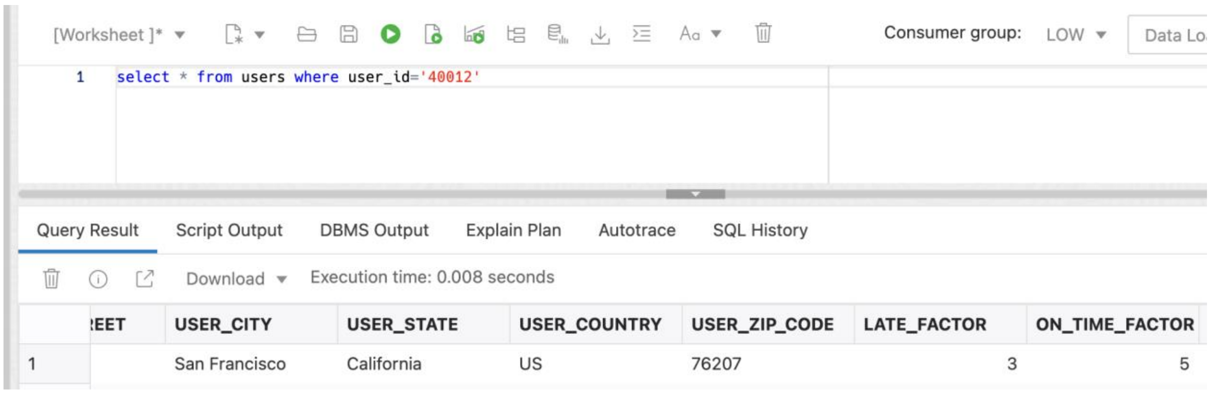
Making an entry where the user returned the book on time.



```sql
Insert into Return values (50011,40012,TO_DATE('11/25/22','mm/dd/yy'),TO_DATE('11/20/22','mm/dd/yy'),1,0);
```

```
Trigger TRG_INCREMENT_ONTIME_FACTOR compiled

Elapsed: 00:00:00.021

1 row inserted.

Elapsed: 00:00:00.008
```

We can see the on-time factor is now updated to 5.

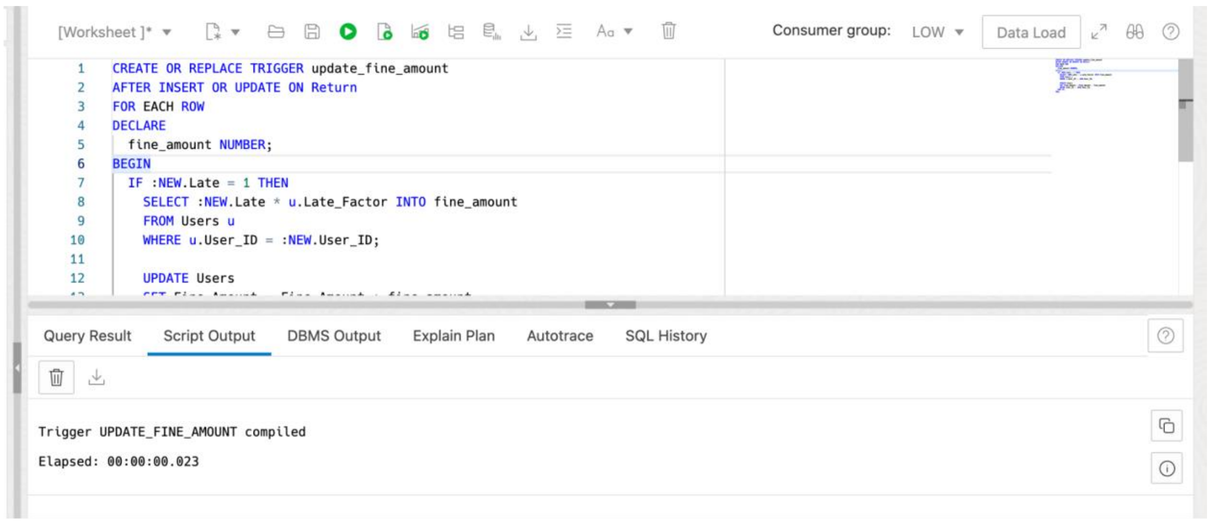


```
select * from users where user_id='40012'
```

	EET	USER_CITY	USER_STATE	USER_COUNTRY	USER_ZIP_CODE	LATE_FACTOR	ON_TIME_FACTOR
1		San Francisco	California	US	76207	3	5

Trigger 3: Creating a Trigger for updating the fine amount.

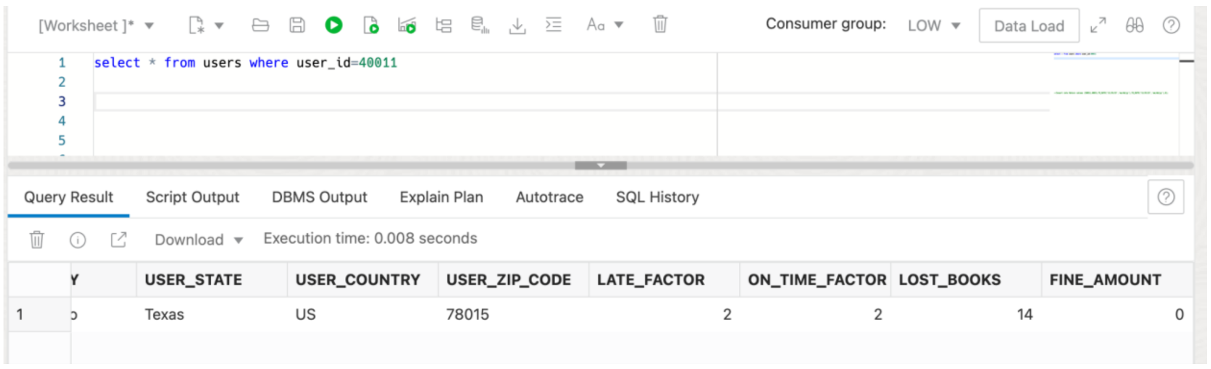
Implementation:



```
CREATE OR REPLACE TRIGGER update_fine_amount
AFTER INSERT OR UPDATE ON Return
FOR EACH ROW
DECLARE
  fine_amount NUMBER;
BEGIN
  IF :NEW.Late = 1 THEN
    SELECT :NEW.Late * u.Late_Factor INTO fine_amount
    FROM Users u
    WHERE u.User_ID = :NEW.User_ID;

    UPDATE Users
```

```
Trigger UPDATE_FINE_AMOUNT compiled

Elapsed: 00:00:00.023
```

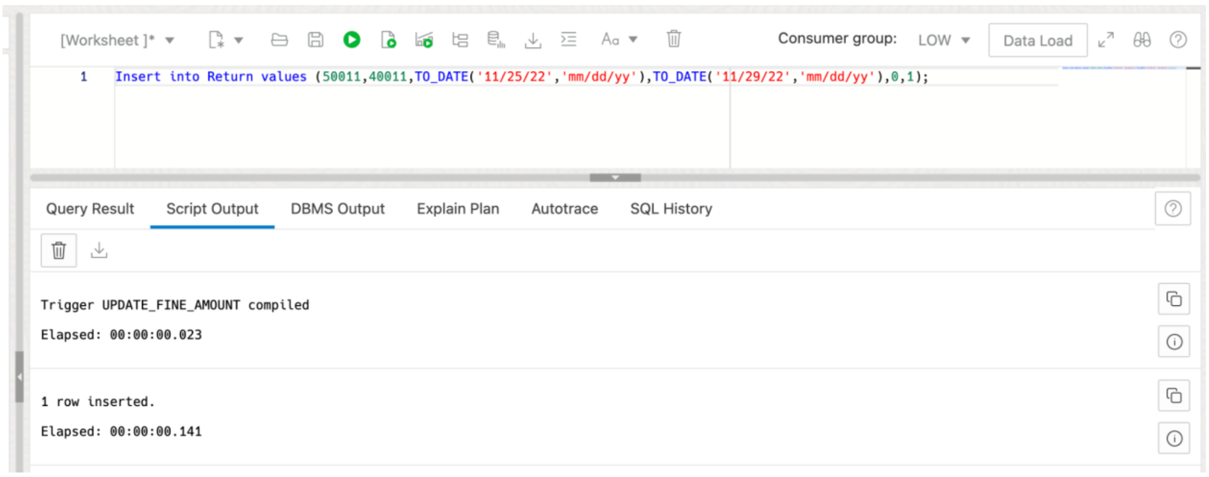
User 40011 has fine amount 0.



```
select * from users where user_id=40011
```

	Y	USER_STATE	USER_COUNTRY	USER_ZIP_CODE	LATE_FACTOR	ON_TIME_FACTOR	LOST_BOOKS	FINE_AMOUNT
1	ɔ	Texas	US	78015	2	2	14	0

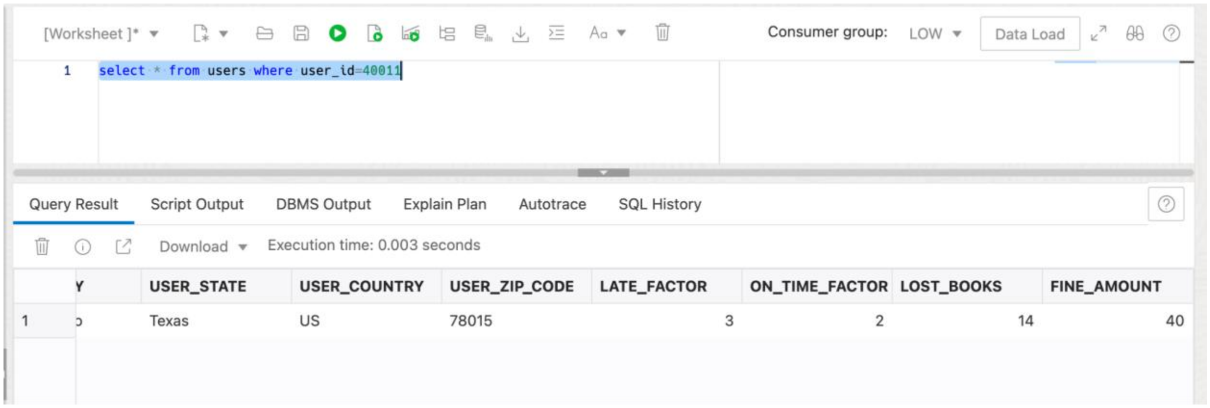
Making user 40011 return a book as 4 days late.



```
Insert into Return values (50011,40011,TO_DATE('11/25/22','mm/dd/yy'),TO_DATE('11/29/22','mm/dd/yy'),0,1);
```

```
Trigger UPDATE_FINE_AMOUNT compiled
Elapsed: 00:00:00.023

1 row inserted.
Elapsed: 00:00:00.141
```

Penalty of 40 units is applied for 4 days and updated for user 40011 in the users table.



```
select * from users where user_id=40011
```

Execution time: 0.003 seconds

	Y	USER_STATE	USER_COUNTRY	USER_ZIP_CODE	LATE_FACTOR	ON_TIME_FACTOR	LOST_BOOKS	FINE_AMOUNT
1	ɔ	Texas	US	78015	3	2	14	40

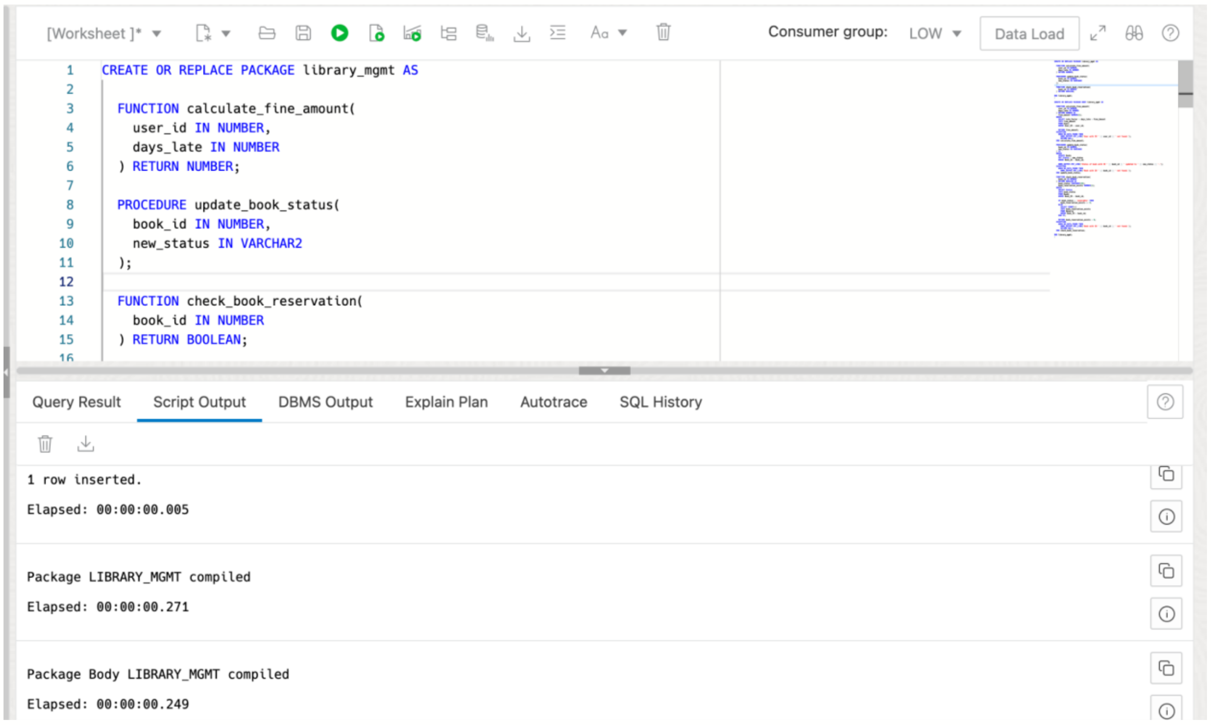
Package:

Packages are helpful as we get to make use of these with already-defined modules which improves simplicity in large applications. Here is a package that includes two functions and a procedure that can be executed depending on the need by passing the required input params.

Description:

```
CREATE OR REPLACE PACKAGE library_mgmt AS


 FUNCTION calculate_fine_amount(
   user_id IN NUMBER,
   days_late IN NUMBER
 ) RETURN NUMBER;
  PROCEDURE update_book_status(
```

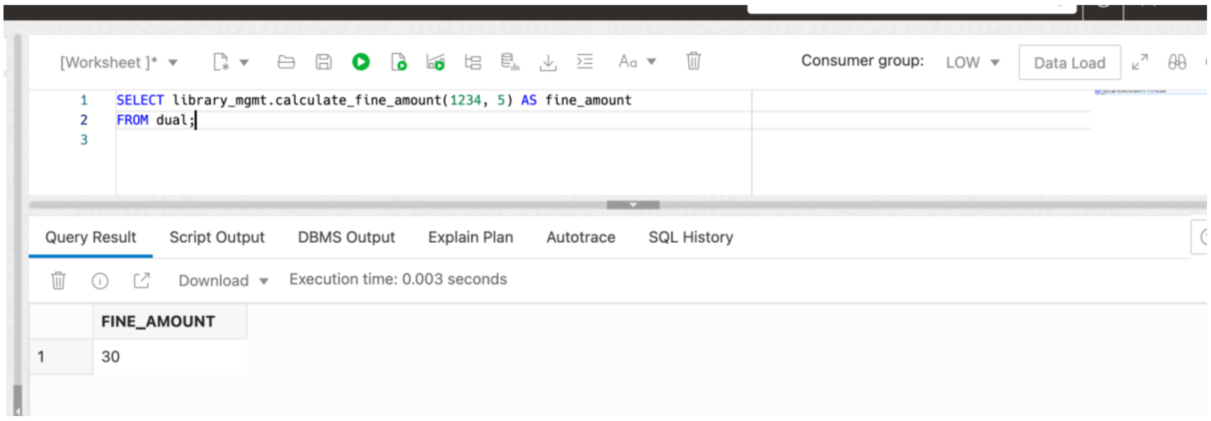
```sql
  book_id IN NUMBER,
  new_status IN VARCHAR2
 );
 FUNCTION check_book_reservation(
  book_id IN NUMBER
 ) RETURN BOOLEAN;
END library_mgmt;
/

CREATE OR REPLACE PACKAGE BODY library_mgmt AS

 FUNCTION calculate_fine_amount(
  user_id IN NUMBER,
  days_late IN NUMBER
 ) RETURN NUMBER AS
  fine_amount NUMBER(5);
 BEGIN
  SELECT Late_Factor * days_late * Fine_Amount
  INTO fine_amount
  FROM Users
  WHERE User_ID = user_id;

  RETURN fine_amount;
 EXCEPTION
  WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('User with ID ' || user_id || ' not found.');
   RETURN NULL;
 END calculate_fine_amount;
 PROCEDURE update_book_status(
  book_id IN NUMBER,
  new_status IN VARCHAR2
 ) AS
 BEGIN
  UPDATE Books
  SET Status = new_status
  WHERE Book_ID = book_id;
  DBMS_OUTPUT.PUT_LINE('Status of book with ID ' || book_id || ' updated to ' || new_status ||
'.');
 EXCEPTION
  WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('Book with ID ' || book_id || ' not found.');
 END update_book_status;
  FUNCTION check_book_reservation(
  book_id IN NUMBER
 ) RETURN BOOLEAN AS
  book_status VARCHAR2(20);
  book_reservation_exists NUMBER(1);
 BEGIN
  SELECT Status
  INTO book_status
  FROM Books
  WHERE Book_ID = book_id;

  IF book_status = 'available' THEN
   book_reservation_exists := 0;
  ELSE
   SELECT COUNT(*)
   INTO book_reservation_exists
   FROM Reserve
   WHERE Book_ID = book_id;
  END IF;

  RETURN book_reservation_exists = 0;
 EXCEPTION
  WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('Book with ID ' || book_id || ' not found.');
   RETURN NULL;
 END check_book_reservation;

END library_mgmt;
/
```



```
CREATE OR REPLACE PACKAGE library_mgmt AS

  FUNCTION calculate_fine_amount(
    user_id IN NUMBER,
    days_late IN NUMBER
  ) RETURN NUMBER;

  PROCEDURE update_book_status(
    book_id IN NUMBER,
    new_status IN VARCHAR2
  );

  FUNCTION check_book_reservation(
    book_id IN NUMBER
  ) RETURN BOOLEAN;
```

```
1 row inserted.

Elapsed: 00:00:00.005

Package LIBRARY_MGMT compiled

Elapsed: 00:00:00.271

Package Body LIBRARY_MGMT compiled

Elapsed: 00:00:00.249
```

Accessing and activating the procedures from packages:

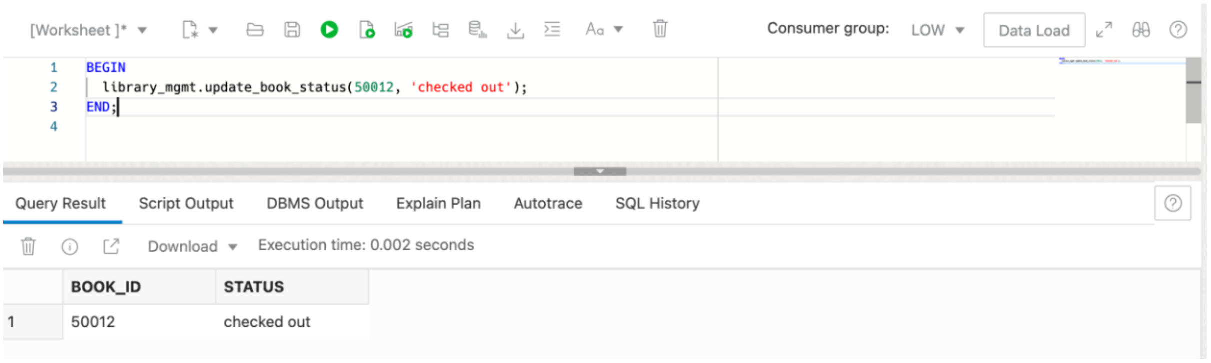


```
SELECT library_mgmt.calculate_fine_amount(1234, 5) AS fine_amount
FROM dual;
```

Execution time: 0.003 seconds

	FINE_AMOUNT
1	30

Changing the status of the book to check out:

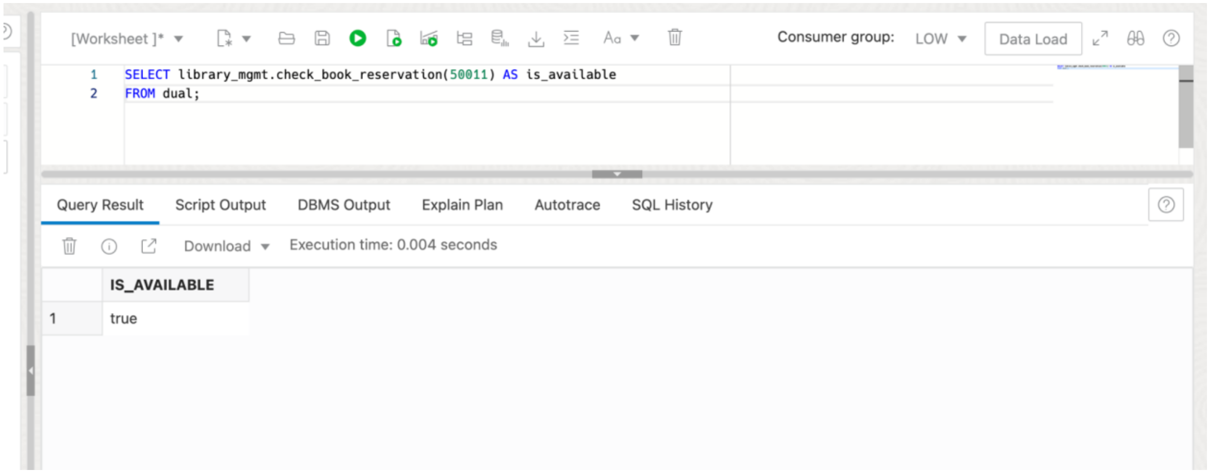


```
BEGIN
  library_mgmt.update_book_status(50012, 'checked out');
END;
```

Execution time: 0.002 seconds

```
   BOOK_ID   STATUS
1  50012     checked out
```

To check if a book is available or not:



```
SELECT library_mgmt.check_book_reservation(50011) AS is_available
FROM dual;
```

Execution time: 0.004 seconds

```
   IS_AVAILABLE
1  true
```

Individual Contribution:

As part of Project Part 5, I have been involved in creating Functions, procedures, and triggers.

I have created Function 3 for getting available books, where I created a function that returns the total number of available books in a library, given a library code as input. And Procedure 3 for Updating the booking status, I developed a procedure that updates a book's status in the Books table, taking a book ID and a new status as inputs, and Trigger 3 for updating the fine amount. I have designed a trigger that automatically updates the fine amount when necessary, such as when a book's status changes from 'checked out' to 'returned'. I have also been involved in executing them and presenting the screenshots in the report as part of the documentation.

In summary, my contributions to part 5 of the project involve improving book availability management, streamlining book status updates, and automating fine calculations using triggers.