Ex. No: 06	**PL/SQL – FUNCTIONS AND PROCEDURES**
Date	20-02-2024

Objective

To create procedure, function and cursor and perform various operations.

Description

PL/SQL supports variables, conditions, loops and exceptions, arrays are also supported, though in a somewhat unusual way, involving the use of PL/SQL collections. PL/SQL collections are a slightly advanced topic. Implementations from version 8 of Oracle Database onwards have included features associated with orientation.PL/SQL program units (essentially code containers) can be compiled into the Oracle database. Programmers can thus embed PL/SQL units of functionality into the database directly. They also can write scripts containing PL/SQL program units that can be read into the database using the Oracle SQL *Plus tool.

Once the program units have been stored into the database, they become available for execution at a later time.While programmers can readily embed Data Manipulation Language(DML) statements directly into their PL/SQL code using straight forward SQL statements, Data Definition Language (DDL) requires more complex "Dynamic SQL" statements to be written in the PL/SQL code. However, DML statements underpin the majority of PL/SQL code in typical software applications.

In the case of PL/SQL dynamic SQL, early versions of the Oracle Database required the use of a complicated Oracle DBMS_SQL package library. More recent versions have however introduced a simpler "Native Dynamic SQL", along with an associated EXECUTE IMMEDIATE syntax. Oracle Corporation customarily extends package functionality with each successive release of the Oracle Database.

Detailed Procedure:

Anonymous blocks form the basis of the simplest PL/SQL code, and have the following structure:

```
<<label>>

DECLARE

    TYPE / item / FUNCTION / PROCEDURE declarations
```

```
BEGIN
   Statements
EXCEPTION
         EXCEPTION handlers
END label;
```

The <<label>> and the DECLARE and EXCEPTION sections are optional.

Exceptions, errors which arise during the execution of the code, have one of two types:

1. Predefined exceptions
2. User-defined exceptions.

User-defined exceptions are always raised explicitly by the programmers, using the RAISE or RAISE_APPLICATION_ERROR commands, in any situation where they have determined that it is impossible for normal execution to continue. RAISE command has the syntax:

RAISE <exception name>. Oracle Corporation has pre-defined several exceptions like NO_DATA_FOUND, TOO_MANY_ROWS, *etc.* Each exception has a SQL Error Number and SQL Error Message associated with it. Programmers can access these by using the SQLCODE and SQLERRM functions. The DECLARE section defines and (optionally) initializes variables. If not initialized specifically, they default to NULL.

For example:

```
DECLARE
 number1 NUMBER(2);
 number2 NUMBER(2)    := 17;           -- value default
 text1   VARCHAR2 (12):= 'Hello world';
 text2   DATE         := SYSDATE;      -- current date and time
BEGIN
 SELECT street_number
   INTO number1
   FROM address
  WHERE name = 'INU';
END;
```

The symbol := functions as an assignment operator to store a value in a variable.

The major datatypes in PL/SQL include NUMBER, INTEGER, CHAR, VARCHAR2, DATE, TIMESTAMP, TEXT *etc.*

Functions

Functions in PL/SQL are a collection of SQL and PL/SQL statements that perform a task and should return a value to the calling environment. User defined functions are used to supplement the many hundreds of functions built in by Oracle.

There are two different types of functions in PL/SQL. The traditional function is written in the form:

```
CREATE OR REPLACE FUNCTION <function_name> [(input/output variable declarations)]
RETURN return_type
[AUTHID <CURRENT USER | DEFINER>] <IS|AS>
        [declaration block]
BEGIN
        <PL/SQL block WITH RETURN statement>
    RETURN <return_value>;
[EXCEPTION
        EXCEPTION block]
    RETURN <return_value>;
END;
```

Pipelined Table Functions are written in the form:

```
CREATE OR REPLACE FUNCTION <function_name> [(input/output variable declarations)]
RETURN return_type

[AUTHID <CURRENT USER | DEFINER>] [<AGGREGATE | PIPELINED>] <IS|USING>

        [declaration block]

BEGIN

        <PL/SQL block WITH RETURN statement>

    PIPE ROW <return type>;

    RETURN;

[EXCEPTION

        EXCEPTION block]

    PIPE ROW <return type>;

    RETURN;

END;
```

There are three types of parameters: IN, OUT and IN OUT. An IN parameter is used as input only. An IN parameter is passed by copy and thus cannot be changed by the called program. An OUT parameter is initially NULL. The program assigns the parameter a value and that value is returned to the calling program. An IN OUT parameter may or may not have an initial value. That initial value may or may not be modified by the called program. Any changes made to the parameter are returned to the calling program by default by copying but, with the NOCOPY hint may be passed by reference.

Procedures

Procedures are similar to Functions; in that they can be executed to perform work. The primary difference is that procedures cannot be used in a SQL statement and although they can have multiple out parameters they do not "RETURN" a value.

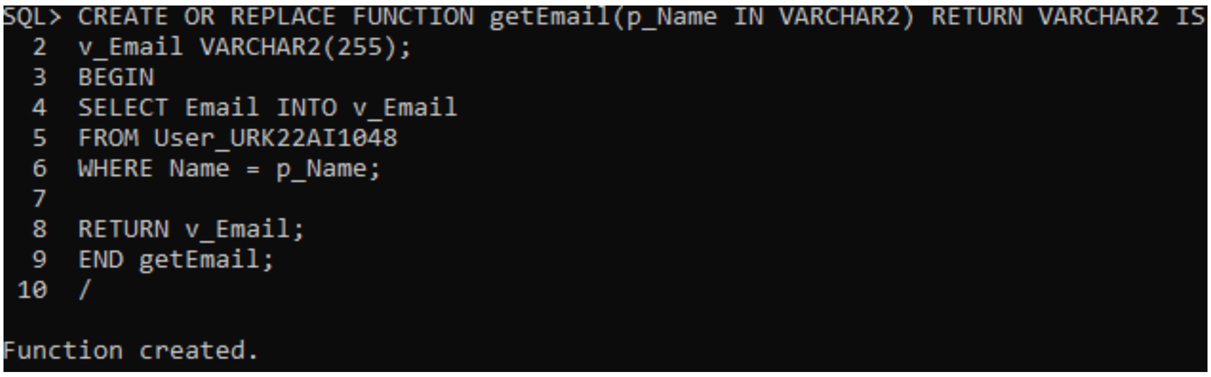
Procedures are traditionally the workhorse of the coding world and functions are traditionally the smaller, more specific pieces of code. PL/SQL maintains many of the distinctions between functions and procedures found in many general-purpose programming languages, but in addition, functions can be called from SQL, while procedures cannot.

These are the real time usages of PL/SQL.

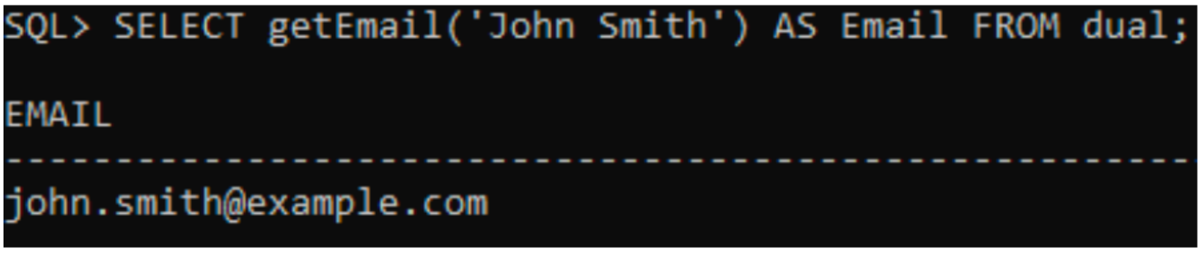
- Block Structures: PL SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.
- Procedural Language Capability**:** PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).
- Better Performance**:** PL SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.
- Error Handling: PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

Questions:

1.Write a PL/SQL function named getUserEmail that takes a user's name as input and returns their email address as output. Assume you have a table named users with the following columns: UserID, Name, Email, Password, and Phone. The function should use the input name to retrieve the corresponding email address from the table and return it.

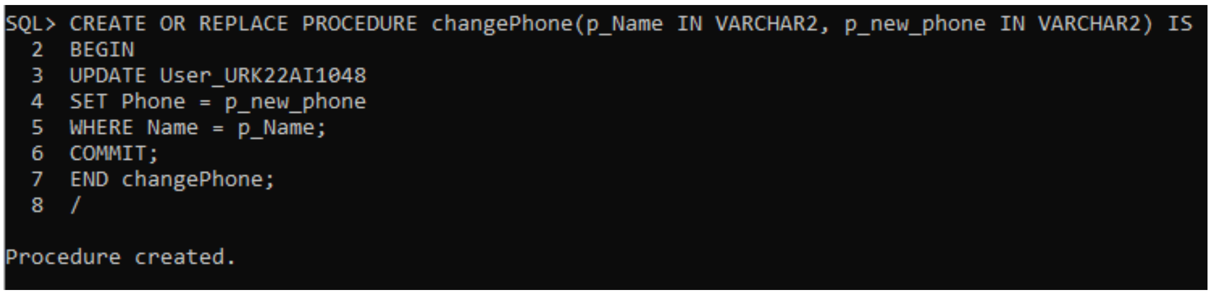


```
SQL> CREATE OR REPLACE FUNCTION getEmail(p_Name IN VARCHAR2) RETURN VARCHAR2 IS
  2  v_Email VARCHAR2(255);
  3  BEGIN
  4  SELECT Email INTO v_Email
  5  FROM User_URK22AI1048
  6  WHERE Name = p_Name;
  7
  8  RETURN v_Email;
  9  END getEmail;
 10  /

Function created.
```

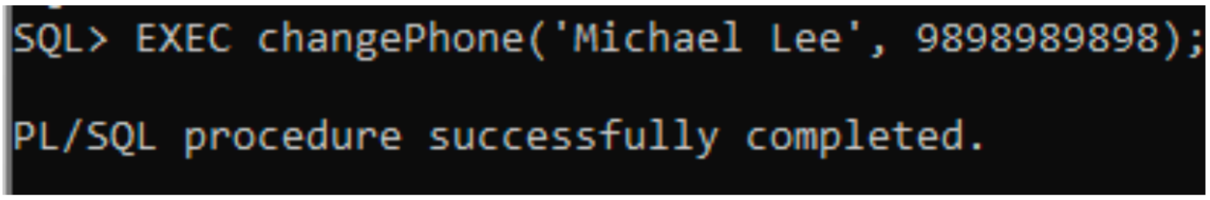


```
SQL> SELECT getEmail('John Smith') AS Email FROM dual;

EMAIL
--------------------------------------------------------
john.smith@example.com
```

2. Write a PL/SQL procedure named changePhoneNumber that takes a user's name and a new phone number as input and updates the phone number in the users table. Assume you have a table named users with the following columns: UserID, Name, Email, Password, and Phone. The procedure should use the provided name to locate the user and update their phone number with the new one.

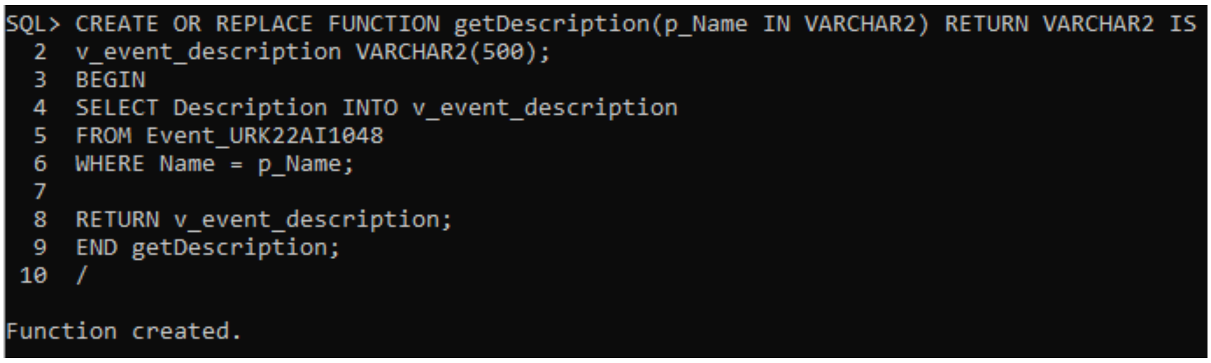


```
SQL> CREATE OR REPLACE PROCEDURE changePhone(p_Name IN VARCHAR2, p_new_phone IN VARCHAR2) IS
BEGIN
UPDATE User_URK22AI1048
SET Phone = p_new_phone
WHERE Name = p_Name;
COMMIT;
END changePhone;
/

Procedure created.
```

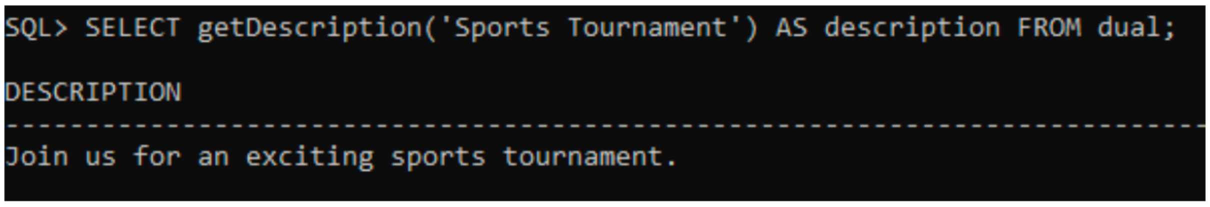


```
SQL> EXEC changePhone('Michael Lee', 9898989898);

PL/SQL procedure successfully completed.
```

3. Write a PL/SQL function named getEventDescription that takes an event name as input and returns its description as output. Assume you have a table named events with the following columns: EventID, Name, Date, Time, VenueID, and Description. The function should use the input event name to retrieve the corresponding event description from the table and return it.



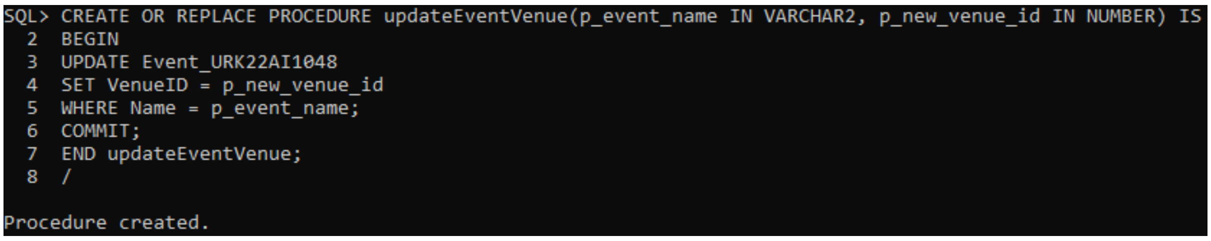
```
SQL> CREATE OR REPLACE FUNCTION getDescription(p_Name IN VARCHAR2) RETURN VARCHAR2 IS
v_event_description VARCHAR2(500);
BEGIN
SELECT Description INTO v_event_description
FROM Event_URK22AI1048
WHERE Name = p_Name;

RETURN v_event_description;
END getDescription;
/

Function created.
```



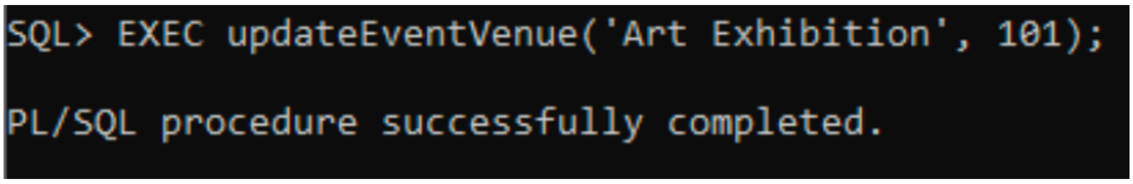
```
SQL> SELECT getDescription('Sports Tournament') AS description FROM dual;

DESCRIPTION
--------------------------------------------------------------------------------
Join us for an exciting sports tournament.
```

4. Write a PL/SQL procedure named updateEventVenue that takes an event name and a new venue ID as input and updates the venue ID in the events table. Assume you have a table named

events with the following columns: EventID, Name, Date, Time, VenueID, and Description. The procedure should use the provided event name to locate the event and update its venue ID with the new one.

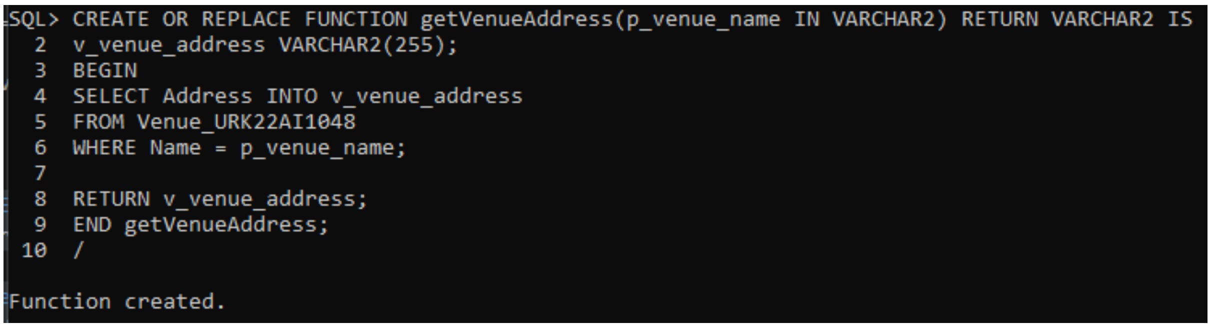


```
SQL> CREATE OR REPLACE PROCEDURE updateEventVenue(p_event_name IN VARCHAR2, p_new_venue_id IN NUMBER) IS
BEGIN
UPDATE Event_URK22AI1048
SET VenueID = p_new_venue_id
WHERE Name = p_event_name;
COMMIT;
END updateEventVenue;
/

Procedure created.
```

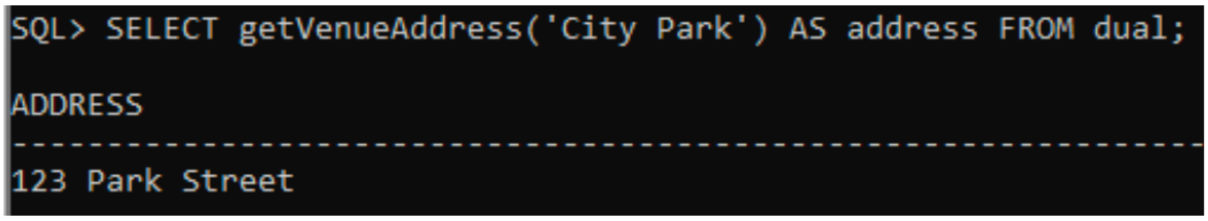


```
SQL> EXEC updateEventVenue('Art Exhibition', 101);

PL/SQL procedure successfully completed.
```

5. Write a PL/SQL function named getVenueAddress that takes a venue name as input and returns its address as output. Assume you have a table named venues with the following columns: VenueID, Name, Address, City, State, and Country. The function should use the input venue name to retrieve the corresponding venue address from the table and return it.

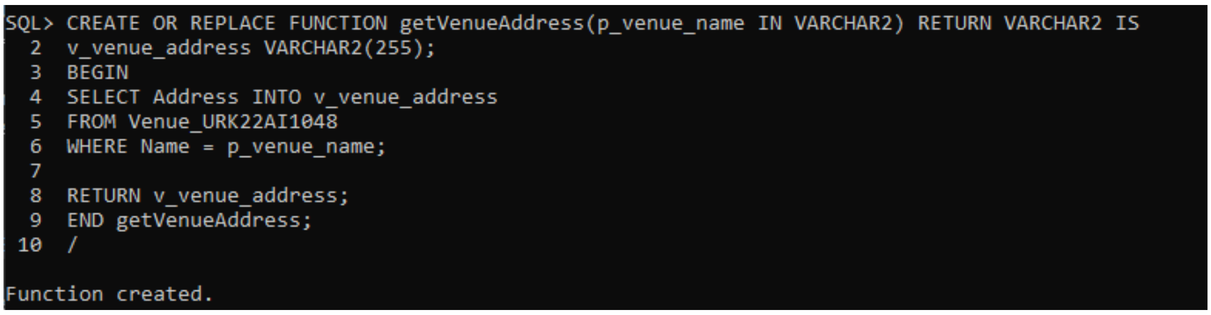


```
SQL> CREATE OR REPLACE FUNCTION getVenueAddress(p_venue_name IN VARCHAR2) RETURN VARCHAR2 IS
v_venue_address VARCHAR2(255);
BEGIN
SELECT Address INTO v_venue_address
FROM Venue_URK22AI1048
WHERE Name = p_venue_name;

RETURN v_venue_address;
END getVenueAddress;
/

Function created.
```

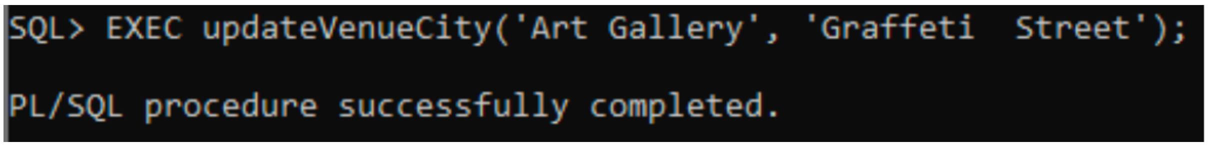


```
SQL> SELECT getVenueAddress('City Park') AS address FROM dual;

ADDRESS
--------------------------------------------------------------
123 Park Street
```

6. Write a PL/SQL procedure named updateVenueCity that takes a venue name and a new city as input and updates the city in the venues table. Assume you have a table named venues with the following columns: VenueID, Name, Address, City, State, and Country. The procedure should use the provided venue name to locate the venue and update its city with the new one.

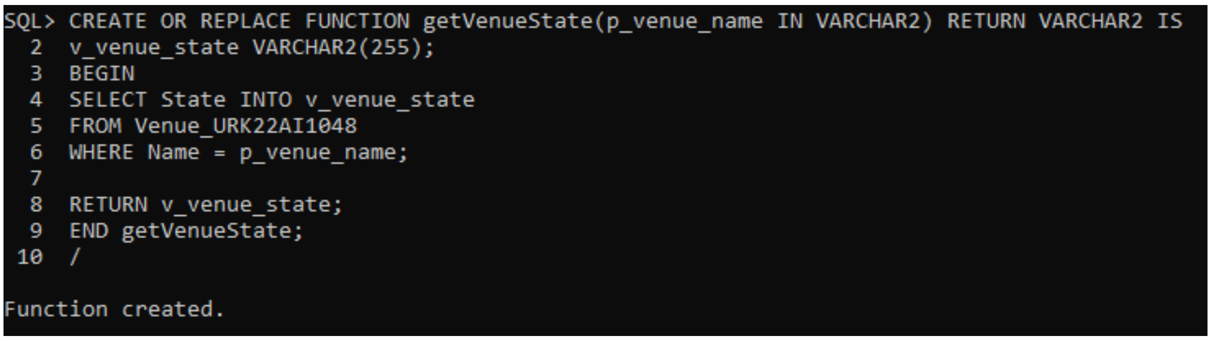


```
SQL> CREATE OR REPLACE FUNCTION getVenueAddress(p_venue_name IN VARCHAR2) RETURN VARCHAR2 IS
v_venue_address VARCHAR2(255);
BEGIN
SELECT Address INTO v_venue_address
FROM Venue_URK22AI1048
WHERE Name = p_venue_name;

RETURN v_venue_address;
END getVenueAddress;
/

Function created.
```

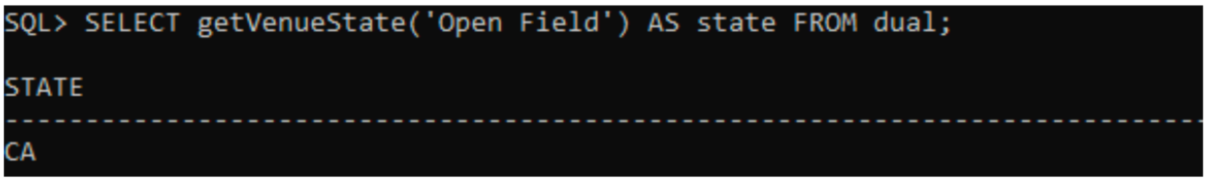


```
SQL> EXEC updateVenueCity('Art Gallery', 'Graffeti  Street');

PL/SQL procedure successfully completed.
```

7. Write a PL/SQL function named getVenueState that takes a venue name as input and returns its state as output. Assume you have a table named venues with the following columns: VenueID, Name, Address, City, State, and Country. The function should use the input venue name to retrieve the corresponding venue state from the table and return it.

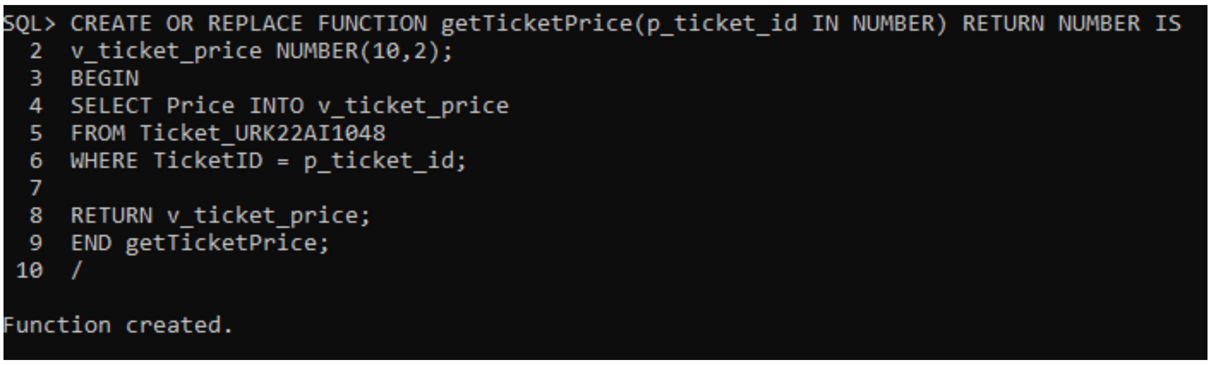


```
SQL> CREATE OR REPLACE FUNCTION getVenueState(p_venue_name IN VARCHAR2) RETURN VARCHAR2 IS
v_venue_state VARCHAR2(255);
BEGIN
SELECT State INTO v_venue_state
FROM Venue_URK22AI1048
WHERE Name = p_venue_name;

RETURN v_venue_state;
END getVenueState;
/

Function created.
```

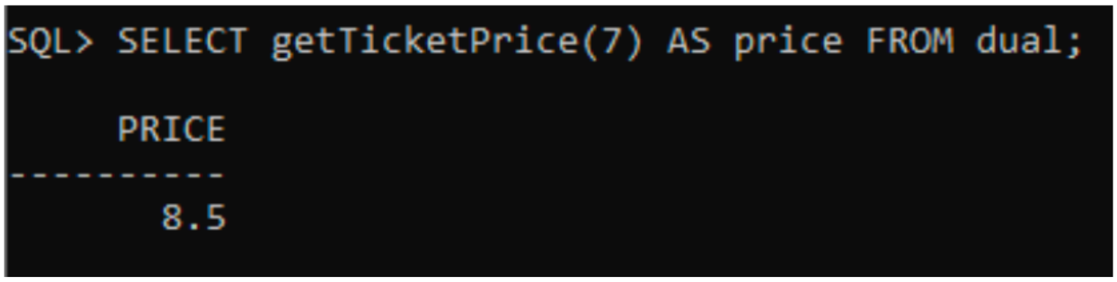


```
SQL> SELECT getVenueState('Open Field') AS state FROM dual;

STATE
--------------------------------------------------------------------
CA
```

8. Write a PL/SQL function named getTicketPrice that takes a ticket ID as input and returns the price of the ticket as output. Assume you have a table named tickets with the following columns: TicketID, EventID, UserID, SeatNumber, Price, and Status. The function should use the input ticket ID to retrieve the corresponding ticket price from the table and return it.

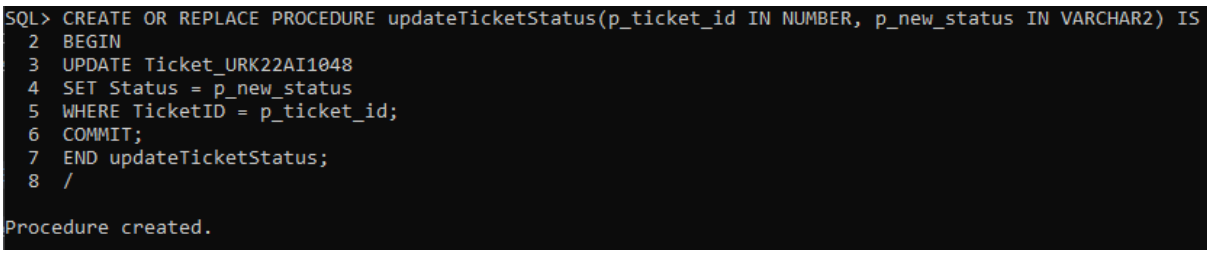


```
SQL> CREATE OR REPLACE FUNCTION getTicketPrice(p_ticket_id IN NUMBER) RETURN NUMBER IS
  v_ticket_price NUMBER(10,2);
  BEGIN
  SELECT Price INTO v_ticket_price
  FROM Ticket_URK22AI1048
  WHERE TicketID = p_ticket_id;

  RETURN v_ticket_price;
  END getTicketPrice;
  /

Function created.
```

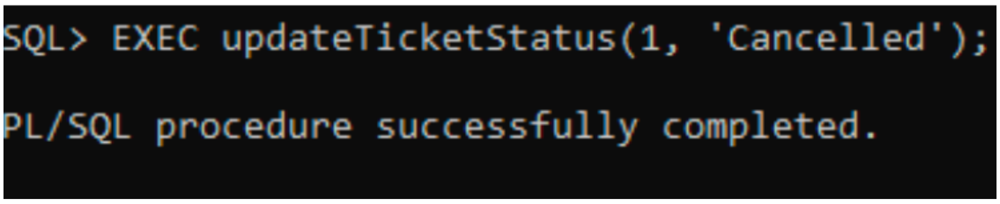


```
SQL> SELECT getTicketPrice(7) AS price FROM dual;

     PRICE
----------
       8.5
```

9. Write a PL/SQL procedure named updateTicketStatus that takes a ticket ID and a new status as input and updates the status in the tickets table. Assume you have a table named tickets with the following columns: TicketID, EventID, UserID, SeatNumber, Price, and Status. The procedure should use the provided ticket ID to locate the ticket and update its status with the new one.

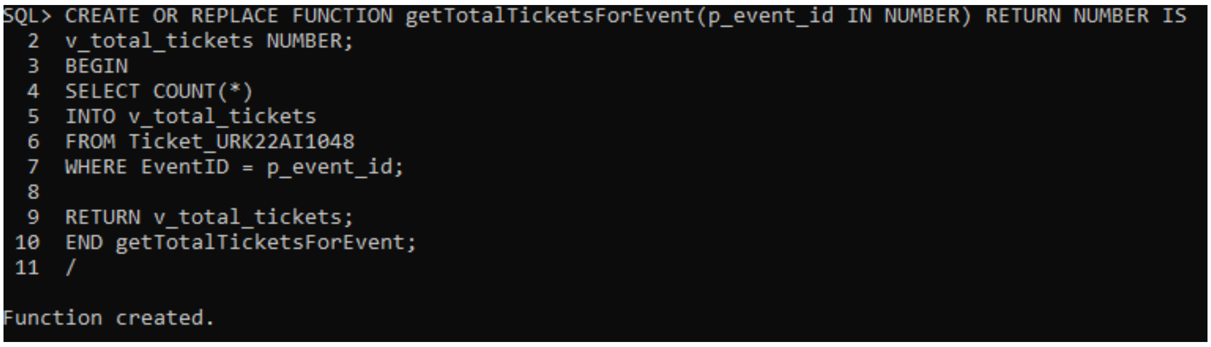


```
SQL> CREATE OR REPLACE PROCEDURE updateTicketStatus(p_ticket_id IN NUMBER, p_new_status IN VARCHAR2) IS
  BEGIN
  UPDATE Ticket_URK22AI1048
  SET Status = p_new_status
  WHERE TicketID = p_ticket_id;
  COMMIT;
  END updateTicketStatus;
  /

Procedure created.
```

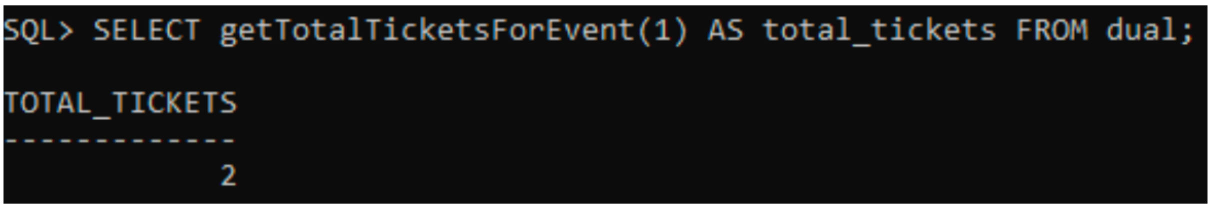


```
SQL> EXEC updateTicketStatus(1, 'Cancelled');

PL/SQL procedure successfully completed.
```

10. Write a PL/SQL function named getTotalTicketsForEvent that takes an event ID as input and returns the total number of tickets booked for that event as output. Assume you have a table named tickets with the following columns: TicketID, EventID, UserID, SeatNumber, Price, and Status. The function should use the input event ID to count the number of tickets with the specified event ID in the table and return that count.



```
SQL> CREATE OR REPLACE FUNCTION getTotalTicketsForEvent(p_event_id IN NUMBER) RETURN NUMBER IS
  v_total_tickets NUMBER;
  BEGIN
  SELECT COUNT(*)
  INTO v_total_tickets
  FROM Ticket_URK22AI1048
  WHERE EventID = p_event_id;

  RETURN v_total_tickets;
  END getTotalTicketsForEvent;
  /

Function created.
```



```
SQL> SELECT getTotalTicketsForEvent(1) AS total_tickets FROM dual;

TOTAL_TICKETS
-------------
            2
```

Result:

The given queries executed by the set operations and joins successfully.