**--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**--Name: Jacky Chun Kit Siu, Song Joo**

**--ID: 134663186, 171443211**

**--Purpose: Assignment 2**

**--Date:23-11-2022**

**--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**spGamesInsert( game\_id games.gameid%TYPE, div\_id games.divid%TYPE, game\_num games.gamenum%TYPE, gameDandT games.gamedatetime%TYPE, homeT games.hometeam%TYPE, homeS games.homescore%TYPE, visitT games.visitteam%TYPE, visitS games.visitscore%TYPE, location\_id games.locationid%TYPE, isPlayed2 games.isplayed%TYPE, notes2 games.notes%TYPE, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | game\_id  div\_id  game\_num  gameDandT  homeT  homeS  visitT  visitS  location\_id  IsPlayed2  notes2  errorCode | NUMBER  NUMBER  NUMBER  DATE  NUMBER  NUMBER  NUMBER  NUMBER  NUMBER  NUMBER  VARCHAR2  OUT NUMBER | Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Size of Date  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  50 bytes  Precision: 38 Scale:0 | Id of a game  division  game Number  date played or to be played  homeTeam ID  points scored by home team  visit team ID  points scored by visit team  ID of the location of the game  Whether the game has been played or not  Notes about the game  Code of error | |
| **expected outputs** |
| error code 10 or 1  Add a record to the table games |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Insert a record into the table games. The Row including fields such as gameid, division, game number, game date and time, home Team ID, home Score, visit Team ID, visit Score, location ID, isplayed, notes will be inserted according to parameters entered into the procedure. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spGamesInsert(122,28,91,TO\_DATE('23-01-29','yy-mm-dd'),216,0,210,0,80,0,NULL,newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spGamesUpdate( game\_id NUMBER, div\_id games.divid%TYPE, game\_num games.gamenum%TYPE, gameDandT games.gamedatetime%TYPE, homeT games.hometeam%TYPE, homeS games.homescore%TYPE, visitT games.visitteam%TYPE, visitS games.visitscore%TYPE, location\_id games.locationid%TYPE, isPlayed2 games.isplayed%TYPE, notes2 games.notes%TYPE, errorCode OUT NUMBER, safety NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | game\_id  div\_id  game\_num  gameDandT  homeT  homeS  visitT  visitS  location\_id  IsPlayed2  notes2  errorCode  safety | NUMBER  NUMBER  NUMBER  DATE  NUMBER  NUMBER  NUMBER  NUMBER  NUMBER  NUMBER  VARCHAR2  OUT NUMBER  NUMBER | Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Size of Date  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  50 bytes  Precision: 38 Scale:0  Precision: 38 Scale:0 | Id of a game  division  game Number  date played or to be played  homeTeam ID  points scored by home team  visit team ID  points scored by visit team  ID of the location of the game  Whether the game has been played or not  Notes about the game  Code of error  Whether to proceed with Update when error code = 3  1= not to proceed  0= to proceed | |
| **expected outputs** |
| error code 10, 1, 2, 3  Update a record In the table games |
| **potential error codes** |
| 10 = success, 2=no row found so no record updated, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Update a record in the table games according to the game\_id entered. The Row including fields such as division, game number, game date and time, home Team ID, home Score, visit Team ID, visit Score, location ID, isplayed and notes will be Updated according to the parameters entered into the procedure. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spGamesUpdate(122,28,91,TO\_DATE('23-02-25', 'yy-mm-dd'), 216, 0, 210, 0, 80, 0, NULL,  newErrorCode, 1);  DBMS\_ OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spGamesDelete( gID games.gameid%TYPE, errorCode OUT NUMBER, safety NUMBER);**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | gId  errorCode  safety | NUMBER  OUT NUMBER  NUMBER | Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0 | Id of a game  division  Code of error  Whether to proceed with delete when error code = 3  1= not to proceed  0= to proceed. | |
| **expected outputs** |
| error code 10, 1, 2, 3  Delete a record In the table games base on game id. |
| **potential error codes** |
| 10 = success, 2=no row found so no record to delete, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| delete a record in the table games according to the game\_id entered. The Row including fields such as division, game number, game date and time, home Team ID, home Score, visit Team ID, visit Score, location ID, isplayed and notes will be deleted. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spGamesUpdate(122,28,91,TO\_DATE('23-02-25', 'yy-mm-dd'), 216, 0, 210, 0, 80, 0, NULL,  newErrorCode, 1);  DBMS\_ OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spGamesSelect(gameID NUMBER,refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | gameID  refcursor  errorCode | NUMBER  refcur\_games.refcur\_g  OUT NUMBER | Precision: 38 Scale:0  Size of the Pointer to the element of the collection  Precision: 38 Scale:0 | Id of a game  division  reference to a cursor that returns a games%ROWTYPE  Code of error | |
| **expected outputs** |
| error code 10, 1  Select a row from the games table base on the game id entered and output it to software. |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE(**gamesoutput(**CURSOR**(**SELECT\*FROM** games ))) **WHERE** game\_id = gameID ; The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output a games record base on game id. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER:=0;  gamesselect\_refcursor refcur\_games.refcur\_g;  game\_id NUMBER(38,0);  div\_id NUMBER(38,0);  game\_num NUMBER(38,0);  gameDandT DATE;  homeT NUMBER(38,0);  homeS NUMBER(38,0);  visitT NUMBER(38,0);  visitS NUMBER(38,0);  location\_id NUMBER(38,0);  isPlayed2 NUMBER(38,0);  notes2 VARCHAR2(50);  BEGIN  spGamesSelect(10,gamesselect\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('game ID',13,' ')||RPAD('divID',13,' ')||  RPAD('game num',13,' ')||RPAD('gameDateTime',13,' ')||  RPAD('homeTeam',13,' ')||RPAD('homeScore',13,' ')||RPAD('visitTeam',13,' ')||  RPAD('visitScore',13,' ')|| RPAD('location\_id',13,' ')||RPAD('isPlayed',13,' ')||  RPAD('notes',13,' '));  LOOP  FETCH gamesselect\_refcursor  INTO game\_id, div\_id, game\_num, gameDandT, homeT, homeS, visitT, visitS, location\_id,  isPlayed2, notes2;  EXIT WHEN gamesselect\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(game\_id, 13, ' ')||RPAD(div\_id,13,' ')||  RPAD(game\_num,13,' ')|| RPAD(gameDandT,13,' ')|| RPAD(homeT,13,' ')||  RPAD(homeS,13,' ')||RPAD(visitT,13,' ')||RPAD(visitS,13,' ')|| RPAD(location\_id,13,' ')||  RPAD(isPlayed2,13,' ')||RPAD(NVL(notes2,'null'),13,' '));  END LOOP;  CLOSE gamesselect\_refcursor;  END; |

**spGoalscorersInsert( goalId goalscorers.goalid%TYPE, gameId goalscorers.gameid%TYPE, playerId goalscorers.playerid%TYPE, teamId goalscorers.teamid%TYPE, numGoals goalscorers.numgoals%TYPE, numAssists goalscorers.numassists%TYPE, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | goalId  gameId  playerId  teamId  numgoals  numassists  errorCode | NUMBER  NUMBER  NUMBER  NUMBER  NUMBER  NUMBER  OUT NUMBER | Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0 | ID of the goal  ID of the game  ID of the player  ID of the team  Number of goals  Number of assists  Code for errors | |
| **expected outputs** |
| error code 10, 1  Insert a record Into the table goalscorers |
| **potential error codes** |
| 10 = success,  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Insert a record in the table goalscorers according to the goal\_id entered. Row includes columns such as goalid, gameid, playerid, teamid, numgoals, numassists will be inserted according to the parameters entered into the procedure. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spGoalscorersInsert(292,99,2024144,211,9,0,newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spgoalscorersUpdate( goal\_Id NUMBER, game\_Id goalscorers.gameid%TYPE, player\_Id goalscorers.playerid%TYPE, team\_Id goalscorers.teamid%TYPE, num\_Goals goalscorers.numgoals%TYPE, num\_Assists goalscorers.numassists%TYPE, errorCode OUT NUMBER, safety NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | goal\_Id  game\_Id  player\_Id  team\_Id  num\_Goals  num\_Assists  errorCode  safety | NUMBER  NUMBER  NUMBER  NUMBER  NUMBER  NUMBER  OUT NUMBER  NUMBER | Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0 | ID of the goal  ID of the game  ID of the player  ID of the team  Number of goals  Number of assists  Code for errors  Whether to proceed with Update when error code = 3  1= not to proceed  0= to proceed | |
| **expected outputs** |
| error code 10, 1, 2, 3  Update a record Into the table goalscorers base on goal id entered. |
| **potential error codes** |
| 10 = success, 2=no row found so no record to delete, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Update a record in the table goalscorers according to the goal\_id entered. Row including columns such as gameid, playerid, teamid, numgoals, numassists will be Updated according to the parameters enter into the procedure. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spgoalscorersUpdate(292,99,2024144,211,100,0, newErrorCode, 1);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spGoalscorersDelete(goal\_ID NUMBER, errorCode OUT NUMBER, safety Number)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | goal\_ID  errorCode  safety | NUMBER  OUT NUMBER  NUMBER | Precision: 38 Scale:0  Precision: 38 Scale:0  Precision: 38 Scale:0 | Id for the row that contain Goal information  Code of error  Whether to proceed with delete when error code = 3  1= not to proceed  0= to proceed. | |
| **expected outputs** |
| error code 10, 1, 2, 3  Delete a record In the table goalscorers base on goal id. |
| **potential error codes** |
| 10 = success, 2=no row found so no record to delete, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| delete a record in the table goalscorers according to the goal\_ID entered. Row includes column such as goalid, gameid, playerid, teamid, numgoals, numassists will be deleted. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spGoalscorersDelete(292,newErrorCode,1);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spGoalscorersSelect(goal\_id2 NUMBER, refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | goal\_id2  refcursor  errorCode | NUMBER  refcur\_goalscorers.refcur\_gs  OUT NUMBER | Precision: 38 Scale:0  Size of the Pointer to the element of the collection  Precision: 38 Scale:0 | Id of a row on the goal table  reference to a cursor that returns a goalscorers%ROWTYPE  Code of error | |
| **expected outputs** |
| error code 10, 1  Select a row from the goalscorers table base on the goal id entered and output it to software. |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(goalscorersoutput(**CURSOR**(**SELECT\*FROM** goalscorers ))) **WHERE** goal\_id = goal\_id2 ; The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output a row from goalscorers base on the goal id. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  goalscorersSelect\_refcursor refcur\_goalscorers.refcur\_gs;  goalscorersRow goalscorers%ROWTYPE;    BEGIN  spGoalscorersSelect(261,goalscorersSelect\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('goal ID',13,' ')||RPAD('game ID',13,' ')||  RPAD('player ID',13,' ')||RPAD('team ID',13,' ')|| RPAD('num Goals',13,' ')||  RPAD('num Assists',13,' '));  LOOP  FETCH goalscorersSelect\_refcursor  INTO goalscorersRow;  EXIT WHEN goalscorersSelect\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(goalscorersRow.goalid, 13, ' ')||  RPAD(goalscorersRow.gameid,13,' ')|| RPAD(goalscorersRow.playerid,13,' ')||  RPAD(goalscorersRow.teamid,13,' ')||RPAD(goalscorersRow.numgoals,13,' ')||  RPAD(goalscorersRow.numassists,13,' '));  END LOOP;  CLOSE goalscorersSelect\_refcursor;  END; |

**spPlayersInsert( player\_ID players.playerid%TYPE, reg\_Num players.regnumber%TYPE, last\_name players.lastname%TYPE, first\_name players.firstname%TYPE, is\_active players.isactive%TYPE, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | player\_ID  reg\_Num  last\_name  first\_name  is\_active  errorCode | NUMBER  VARCHAR2  VARCHAR2  VARCHAR2  NUMBER  OUT NUMBER | Precision: 38 Scale: 0  15 Byte  25 Byte  25 Byte  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of player  Registration Number  Last Name  First Name  Whether a player is active or not  Code for error | |
| **expected outputs** |
| error code 10, 1  Insert a record Into the table players |
| **potential error codes** |
| 10 = success,  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| To insert a record into the table players according to the parameter entered into the procedure. A row including field such as playerid, regnumber, lastname, firstname, isactive will be inserted. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spPlayersInsert(2024144, 982068, 'Siu', 'Jacky', 1, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spPlayersUpdate( player\_ID NUMBER, reg\_Num players.regnumber%TYPE, last\_name players.lastname%TYPE, first\_name players.firstname%TYPE, is\_active players.isactive%TYPE, errorCode OUT NUMBER, safety NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | player\_ID  reg\_Num  last\_name  first\_name  is\_active  errorCode  safety | NUMBER  VARCHAR2  VARCHAR2  VARCHAR2  NUMBER  OUT NUMBER  NUMBER | Precision: 38 Scale: 0  15 Byte  25 Byte  25 Byte  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of player  Registration Number  Last Name  First Name  Whether a player is active or not  Code for error  Whether to proceed with Update when error code = 3  1= not to proceed  0= to proceed. | |
| **expected outputs** |
| error code 10, 1, 2, 3  An Updated row or updated rows |
| **potential error codes** |
| 10 = success, 2=no row found so no record to update, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| To Update a row in the Table player base on the player\_ID. Or it could be multiple rows that is Updated if the errorCode is 3. Fields such as regnumber, lastname, firstname, isactive of a row will be updated if it normal, or rows will be updated if the safety is off and the errorCode is 3. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spPlayersUpdate(2024144,98268,'Wong','Jason',1, newErrorCode, 1);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spPlayersDelete(player\_ID players.playerid%TYPE, errorCode OUT NUMBER, safety NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | player\_ID  errorCode  safety | NUMBER  OUT NUMBER  NUMBER | Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of a player  Error Code  Whether to proceed with delete when error code = 3  1= not to proceed  0= to proceed. | |
| **expected outputs** |
| error code 10, 1,2,3  Delete a record or multiple record from the table players |
| **potential error codes** |
| 10 = success, 2=no row found so no record to delete, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Delete a row according to the player\_ID entered if everything is normal or delete multiple row according to the player\_ID entered if the safety is off and the error code is 3. The rows or rows deleted will be base on the player id entered in the parameter in the procedure. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spPlayersDelete(2024144, newErrorCode, 1);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  END; |

**PROCEDURE spPlayersSelect( player\_ID2 NUMBER, refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)AS**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | player\_ID2  refcursor  errorCode | NUMBER  refcur\_players.refcur\_p  NUMBER | Precision: 38 Scale: 0  Size of pointer to the element of the collection  Precision: 38 Scale: 0 | Id of the player  Reference to a cursor that returns a players%rowtype  Error Code | |
| **expected outputs** |
| error code 10, 1  Select a row from the players table base on the player id entered and output it to software. |
| **potential error codes** |
| 10 = success,  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(playersoutput(**CURSOR**(**SELECT\*FROM** players ))) **WHERE** player\_ID = player\_ID2 ; The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output a row from the table players base on the player id. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  playersselect\_refcursor refcur\_players.refcur\_p;  playersRow players%ROWTYPE;  BEGIN  spPlayersSelect(1302,playersselect\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('playerID',13,' ')||RPAD('regNumber',13,' ')||  RPAD('lastName',13,' ')||RPAD('firstName',13,' ')|| RPAD('isActive',13,' '));  LOOP  FETCH playersselect\_refcursor  INTO playersRow.playerid, playersRow.regnumber, playersRow.lastname, playersRow.firstname, playersRow.isactive;  EXIT WHEN playersselect\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(playersRow.playerid,13,' ')||RPAD(playersRow.regnumber,13,' ')||RPAD(playersRow.lastname,13,' ')||  RPAD(playersRow.firstname,13,' ')||RPAD(playersRow.isactive,13,' '));  END LOOP;  CLOSE playersselect\_refcursor;  END; |

**spTeamsInsert( team\_ID teams.teamid%TYPE, team\_name teams.teamname%TYPE, is\_active teams.isactive%TYPE, jersey\_colour teams.jerseycolour%TYPE, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **team\_ID**  **team\_name**  **is\_active**  **jersey\_colour**  **errorCode** | NUMBER  VARCHAR2  NUMBER  VARCHAR2  NUMBER | Precision: 38 Scale: 0  10 Byte  Precision: 38 Scale: 0  10 Byte  Precision: 38 Scale: 0 | ID of a team  Name of a team  Whether a team is active or not  Jersey Colour  Error Code | |
| **expected outputs** |
| error code 10, 1,  Insert a record Into the table teams |
| **potential error codes** |
| 10 = success,  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| To insert a row into the table teams base on the parameters entered. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spTeamsInsert(540,'MeatWad',1,'Brown', newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spTeamUpdate( team\_ID teams.teamid%TYPE, team\_name teams.teamname%TYPE, is\_active teams.isactive%TYPE, jersey\_colour teams.jerseycolour%TYPE, errorCode OUT NUMBER, safety NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **team\_ID**  **team\_name**  **is\_active**  **jersey\_colour**  **errorCode**  **safety** | NUMBER  VARCHAR2  NUMBER  VARCHAR2  NUMBER  NUMBER | Precision: 38 Scale: 0  10 Byte  Precision: 38 Scale: 0  10 Byte  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of a team  Name of a team  Whether a team is active or not  Jersey Colour  Error Code  Whether to proceed with Update when error code = 3  1= not to proceed  0= to proceed. | |
| **expected outputs** |
| error code 10, 1,2,3  Update a record In the table teams |
| **potential error codes** |
| 10 = success, 2=no row found so no record to update, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Update the fields teamname, isactive and jerseycolour base on the team\_ID and other information entered in the parameter of the procedure. IF safety is off and the errorCode is 3 multiple rows would be updated. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spTeamsUpdate(540,'FryLock',1,'Red', newErrorCode, 1);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spTeamsDelete( team\_ID teams.teamid%type, errorCode OUT NUMBER, safety NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **team\_ID**  **errorCode**  **safety** | NUMBER  NUMBER  NUMBER | Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of a team  Error Code  Whether to proceed with Delete when error code = 3  1= not to proceed  0= to proceed. | |
| **expected outputs** |
| error code 10, 1,2,3  Delete a rows or multiple rows from the table teams |
| **potential error codes** |
| 10 = success, , 2=no row found so no record to delete, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Delete a row from the table teams according to the team\_ID entered if everything is normal or delete multiple row from the table according to the team\_ID entered if the safety is off and the error code is 3. The rows or rows deleted will be base on the player id entered in the parameter in the procedure. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spTeamsDelete(540, newErrorCode, 1);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  END; |

**spTeamsSelect(team\_ID2 teams.teamid%TYPE,refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **team\_ID2**  **refcursor**  **errorCode** | NUMBER  refcur\_teams.refcur\_t  NUMBER | Precision: 38 Scale: 0  Size of pointer to the element of the collection  Precision: 38 Scale: 0 | Id of the team  Reference to a cursor that returns a teams%rowtype  Error Code | |
| **expected outputs** |
| error code 10, 1  Select a row from the teams table base on the team id entered and output it to software. |
| **potential error codes** |
| 10 = success,  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(teamssoutput(**CURSOR**(**SELECT\*FROM** teams ))) **WHERE** team\_ID = team\_ID2 ; The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output a row from the table teams base on the team id. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 10;  teamsSelect\_refcursor refcur\_teams.refcur\_t;  teamRow teams%ROWTYPE;  BEGIN  spTeamsSelect(225, teamsSelect\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('team ID',13,' ')||RPAD('teamName',13,' ')||  RPAD('isActive',13,' ')||RPAD('jerseyColour',13,' '));  LOOP  FETCH teamsSelect\_refcursor  INTO teamRow.teamid, teamRow.teamname, teamRow.isactive, teamRow.jerseycolour;  EXIT WHEN teamsSelect\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(teamRow.teamid, 13, ' ')||  RPAD(teamRow.teamname,13,' ')||RPAD(teamRow.isactive,13,' ')||  RPAD(teamRow.jerseycolour,13,' '));  END LOOP;  CLOSE teamsSelect\_refcursor;  END; |

**spRostersInsert( roster\_ID rosters.rosterid%TYPE, player\_ID rosters.playerid%TYPE, team\_ID rosters.teamid%TYPE, is\_active rosters.isactive%TYPE, jersey\_number rosters.jerseynumber%TYPE, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **roster\_ID**  **player\_ID**  **team\_ID**  **is\_active**  **jersey\_number**  **errorCode** | NUMBER  NUMBER  NUMBER  NUMBER  NUMBER  OUT NUMBER | Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of roster  ID of player  ID of team  Whether the Player is active or not  Jersey Number  Error Code | |
| **expected outputs** |
| error code 10, 1,2,3  Insert a record Into the table rosters |
| **potential error codes** |
| 10 = success,  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| To insert a record into the table rosters according to the parameter entered into the procedure. A row including field such as rosterid, playerid, teamid, isactive, jerseynumber will be inserted. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER:= 10;  BEGIN  spRostersInsert(231,2024144,212,1,88, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '|| newErrorCode);  END; |

**spRostersUpdate( roster\_ID rosters.rosterid%TYPE, player\_ID rosters.playerid%TYPE, team\_ID rosters.teamid%TYPE, is\_Active rosters.isactive%TYPE, jersey\_Number rosters.jerseynumber%TYPE, errorCode OUT NUMBER, safety NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **roster\_ID**  **player\_ID**  **team\_ID**  **is\_active**  **jersey\_number**  **errorCode**  **safety** | NUMBER  NUMBER  NUMBER  NUMBER  NUMBER  OUT NUMBER  NUMBER | Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of roster  ID of player  ID of team  Whether the Player is active or not  Jersey Number  Error Code  Whether to proceed with Update when error code = 3  1= not to proceed  0= to proceed. | |  |  |  |  | |
| **expected outputs** |
| error code 10, 1,2,3  Update a row or multiple rows in the table rosters |
| **potential error codes** |
| 10 = success, ,2=no row found so no record to update, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Update the fields playerid, teamid, isactive, jerseynumber base on the roster\_ID entered and other information entered into the parameters. IF safety is off and the errorCode is 3 multiple rows would be updated. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spRostersUpdate(231,2024144,212,1,95, newErrorCode, 1);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '|| newErrorCode);  END; |

**spRostersDelete( roster\_ID rosters.rosterid%TYPE, errorCode OUT NUMBER, safety NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **roster\_ID**  **errorCode**  **safety** | NUMBER  NUMBER  NUMBER | Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of roster  Error Code  Whether to proceed with deleted when error code = 3  1= not to proceed  0= to proceed | |
| **expected outputs** |
| error code 10, 1,2,3  Delete a row or multiple row from the table rosters |
| **potential error codes** |
| 10 = success, 2=no row found so no record to delete, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Delete a row from the table rosters according to the roster\_ID entered if everything is normal or delete multiple row from the table according to the roster\_ID entered if the safety is off and the error code is 3. The rows or rows deleted will be base on the roster id entered in the parameter in the procedure. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spRostersDelete(231,newErrorCode, 1);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  END; |

**spRostersSelect( roster\_ID2 NUMBER, refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **roster\_ID2**  **refcursor**  **errorCode** | NUMBER  refcur\_rosters.refcur\_r  NUMBER | Precision: 38 Scale: 0  Size of pointer to the element of the collection  Precision: 38 Scale: 0 | Id of the roster  Reference to a cursor that returns a rosters%rowtype  Error Code | |
| **expected outputs** |
| error code 10, 1  Select a row or multiple rows from the table rosters base on roster id and output it to software |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(rostersoutput(**CURSOR**(**SELECT\*FROM** rosters ))) **WHERE** roster\_ID = roster\_ID2 ; The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output a row from the table rosters base on the roster id. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  rostersSelect\_refcursor refcur\_rosters.refcur\_r;  rostersRow rosters%ROWTYPE;  BEGIN  spRostersSelect(230, rostersSelect\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('roster ID',13,' ')||RPAD('player ID',13,' ')||  RPAD('team ID',13,' ')||RPAD('is Active',13,' ')|| RPAD('jersey Number',13,' '));  LOOP  FETCH rostersSelect\_refcursor  INTO rostersRow.rosterid, rostersRow.playerid, rostersRow.teamid, rostersRow.isactive,  rostersRow.jerseynumber;  EXIT WHEN rostersSelect\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(rostersRow.rosterid, 13, ' ')||  RPAD(rostersRow.playerid,13,' ')||RPAD(rostersRow.teamid,13,' ')||  RPAD(rostersRow.isactive,13,' ')|| RPAD(rostersRow.jerseynumber,13,' '));  END LOOP;  CLOSE rostersSelect\_refcursor;  END; |

**spSllocationsInsert( location\_ID sllocations.locationid%TYPE, location\_Name, sllocations.locationname%TYPE, field\_Length sllocations.fieldlength%TYPE, is\_Active sllocations.isactive%TYPE, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **location\_ID**  **location\_Name**  **field\_Length**  **is\_Active**  **errorCode** | NUMBER  VARCHAR2  NUMBER  NUMBER  NUMBER | Precision: 38 Scale: 0  50 Byte  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of location  Location name  Length of the field at that location.  Whether that location is active or not  Error code | |
| **expected outputs** |
| error code 10, 1,2,3  Insert a row into the table sllocations |
| **potential error codes** |
| 10 = success,  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| To insert a record into the table sllocations according to the parameters entered into the procedure. A row including field such as locationid, locationname, fieldlength, isactive will be inserted. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spSllocationsInsert(92,'2537 Victoria Park Ave',110,1,newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  END; |

**spSllocationsUpdate( location\_ID sllocations.locationid%TYPE, location\_Name sllocations.locationname%TYPE, field\_Length sllocations.fieldlength%TYPE, is\_Active sllocations.isactive%TYPE, errorCode OUT NUMBER, safety NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **location\_ID**  **location\_Name**  **field\_Length**  **is\_Active**  **errorCode**  **safety** | NUMBER  VARCHAR2  NUMBER  NUMBER  NUMBER  NUMBER | Precision: 38 Scale: 0  50 Byte  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of location  Location name  Length of the field at that location.  Whether that location is active or not  Error code  Whether to proceed with Update when error code = 3  1= not to proceed  0= to proceed | |
| **expected outputs** |
| error code 10, 1,2,3  Update a row or multiple rows in the table sllocations |
| **potential error codes** |
| 10 = success, 2=no row found so no record to Update, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Update the fields locationname, fieldlength, isactive base on the location\_ID entered and other information entered into the parameters. IF safety is off and the errorCode is 3 multiple rows would be updated. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spSllocationsUpdate(92, '35 Springfield Blvd', 110,1,newErrorCode,1);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '|| newErrorCode);  END; |

**spSllocationsDelete( location\_ID sllocations.locationid%TYPE, errorCode OUT NUMBER, safety NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **location\_ID**  **errorCode**  **safety** | NUMBER  NUMBER  NUMBER | Precision: 38 Scale: 0  Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of location  errorCode  Whether to proceed with delete when error code = 3  1= not to proceed  0= to proceed | |
| **expected outputs** |
| error code 10, 1,2,3  delete a row or multiple rows from the table sllocation |
| **potential error codes** |
| 10 = success, 2=no row found so no record to delete, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Delete a row from the table sllocations according to the location\_ID entered if everything is normal or delete multiple rows from the table according to the location\_ID entered if the safety is off and the error code is 3. The rows or rows deleted will be base on the roster id entered in the parameter in the procedure. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  BEGIN  spSllocationsDelete(92, newErrorCode, 1);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  END; |

**spSllocationsSelect(location\_ID2 NUMBER, refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **location\_ID2**  **refcursor**  **errorCode** | NUMBER  refcur\_sllocations.refcur\_sl  NUMBER | Precision: 38 Scale: 0  Size of pointer to the element of the collection  Precision: 38 Scale: 0 | Id of the roster  Reference to a cursor that returns a sllocations%rowtype  Error Code | |
| **expected outputs** |
| error code 10, 1  Select a row or multiple rows from the table sllocations |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(sllocationsoutput(**CURSOR**(**SELECT\*FROM** sllocations ))) **WHERE** location\_ID = location\_ID2 ; The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output a row from the table sllocations base on the location id. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  sllocationsSelect\_refcursor refcur\_sllocations.refcur\_sl;  sllocationsRow sllocations%ROWTYPE;  BEGIN  spSllocationsSelect(85, sllocationsSelect\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('location ID',13,' ')||RPAD('location Name',50,' ')||  RPAD('field Length',13,' ')||RPAD('is Active',13,' '));  LOOP  FETCH sllocationsSelect\_refcursor  INTO sllocationsRow.locationid, sllocationsRow.locationname, sllocationsRow.fieldlength,  sllocationsRow.isactive;  EXIT WHEN sllocationsSelect\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(sllocationsRow.locationid, 13, ' ')||  RPAD(sllocationsRow.locationname,50,' ')||RPAD(sllocationsRow.fieldlength,13,' ')||  RPAD(sllocationsRow.isactive,13,' '));  END LOOP;  CLOSE sllocationsSelect\_refcursor;  END; |

**Question 2**

**spGamesSelectAll(refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | refcursor  errorCode | refcur\_games.refcur\_g  OUT NUMBER | Size of the Pointer to the element of the collection  Precision: 38 Scale:0 | reference to a cursor that returns a games%ROWTYPE  Code of error | |
| **expected outputs** |
| error code 10, 1  Select all from the games table and output it to software. |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE(**gamesoutput(**CURSOR**(**SELECT\*FROM** games ))); The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output all rows from the table games. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER:=0;  gamesselectAll\_refcursor refcur\_games.refcur\_g;  game\_id NUMBER(38,0);  div\_id NUMBER(38,0);  game\_num NUMBER(38,0);  gameDandT DATE;  homeT NUMBER(38,0);  homeS NUMBER(38,0);  visitT NUMBER(38,0);  visitS NUMBER(38,0);  location\_id NUMBER(38,0);  isPlayed2 NUMBER(38,0);  notes2 VARCHAR2(50);  BEGIN  spGamesSelectAll(gamesselectAll\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('game ID',13,' ')||RPAD('divID',13,' ')||  RPAD('game num',13,' ')||RPAD('gameDateTime',13,' ')||  RPAD('homeTeam',13,' ')||RPAD('homeScore',13,' ')||RPAD('visitTeam',13,' ')||  RPAD('visitScore',13,' ')|| RPAD('location\_id',13,' ')||RPAD('isPlayed',13,' ')||  RPAD('notes',13,' '));  LOOP  FETCH gamesselectAll\_refcursor  INTO game\_id, div\_id, game\_num, gameDandT, homeT, homeS, visitT, visitS, location\_id,  isPlayed2, notes2;  EXIT WHEN gamesselectAll\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(game\_id, 13, ' ')||RPAD(div\_id,13,' ')||  RPAD(game\_num,13,' ')|| RPAD(gameDandT,13,' ')|| RPAD(homeT,13,' ')||  RPAD(homeS,13,' ')||RPAD(visitT,13,' ')||RPAD(visitS,13,' ')|| RPAD(location\_id,13,' ')||  RPAD(isPlayed2,13,' ')||RPAD(NVL(notes2,'null'),13,' '));  END LOOP;  CLOSE gamesselectAll\_refcursor;  END; |

**spGoalscorersSelectAll( refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | refcursor  errorCode | refcur\_goalscorers.refcur\_gs  OUT NUMBER | Size of the Pointer to the element of the collection  Precision: 38 Scale:0 | reference to a cursor that returns a goalscorers%ROWTYPE  Code of error | |
| **expected outputs** |
| error code 10, 1  Select all from the goalscorers table and output it to software. |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(goalscorersoutput(**CURSOR**(**SELECT\*FROM** goalscorers ))); The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output all rows from goalscorers. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  goalscorersSelectAll\_refcursor refcur\_goalscorers.refcur\_gs;  goalscorersRow goalscorers%ROWTYPE;    BEGIN  spGoalscorersSelectAll(goalscorersSelectAll\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('goal ID',13,' ')||RPAD('game ID',13,' ')||  RPAD('player ID',13,' ')||RPAD('team ID',13,' ')|| RPAD('num Goals',13,' ')||  RPAD('num Assists',13,' '));  LOOP  FETCH goalscorersSelectAll\_refcursor  INTO goalscorersRow;  EXIT WHEN goalscorersSelectAll\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(goalscorersRow.goalid, 13, ' ')||  RPAD(goalscorersRow.gameid,13,' ')|| RPAD(goalscorersRow.playerid,13,' ')||  RPAD(goalscorersRow.teamid,13,' ')||RPAD(goalscorersRow.numgoals,13,' ')||  RPAD(goalscorersRow.numassists,13,' '));  END LOOP;  CLOSE goalscorersSelectAll\_refcursor;  END; |

**PROCEDURE spPlayersSelectAll(refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)AS**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | refcursor  errorCode | refcur\_players.refcur\_p  NUMBER | Size of pointer to the element of the collection  Precision: 38 Scale: 0 | Reference to a cursor that returns a players%rowtype  Error Code | |
| **expected outputs** |
| error code 10, 1  Select all rows from the players table and output it to software. |
| **potential error codes** |
| 10 = success,  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(playersoutput(**CURSOR**(**SELECT\*FROM** players ))); The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output all rows from the table players. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  playersselectAll\_refcursor refcur\_players.refcur\_p;  playersRow players%ROWTYPE;  BEGIN  spPlayersSelectAll(playersselectAll\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('playerID',13,' ')||RPAD('regNumber',13,' ')||  RPAD('lastName',13,' ')||RPAD('firstName',13,' ')|| RPAD('isActive',13,' '));  LOOP  FETCH playersselectAll\_refcursor  INTO playersRow.playerid, playersRow.regnumber, playersRow.lastname, playersRow.firstname,  playersRow.isactive;  EXIT WHEN playersselectAll\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(playersRow.playerid,13,' ')||  RPAD(playersRow.regnumber,13,' ')||RPAD(playersRow.lastname,13,' ')||  RPAD(playersRow.firstname,13,' ')||RPAD(playersRow.isactive,13,' '));  END LOOP;  CLOSE playersselectAll\_refcursor;  END; |

**spTeamsSelectAll(refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **refcursor**  **errorCode** | refcur\_teams.refcur\_t  NUMBER | Size of pointer to the element of the collection  Precision: 38 Scale: 0 | Reference to a cursor that returns a teams%rowtype  Error Code | |
| **expected outputs** |
| error code 10, 1  Select all rows from the teams table and output it to software. |
| **potential error codes** |
| 10 = success,  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(teamssoutput(**CURSOR**(**SELECT\*FROM** teams ))); The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output all rows from the table teams. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 10;  teamsSelectAll\_refcursor refcur\_teams.refcur\_t;  teamRow teams%ROWTYPE;  BEGIN  spTeamsSelectAll(teamsSelectAll\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('team ID',13,' ')||RPAD('teamName',13,' ')||  RPAD('isActive',13,' ')||RPAD('jerseyColour',13,' '));  LOOP  FETCH teamsSelectAll\_refcursor  INTO teamRow.teamid, teamRow.teamname, teamRow.isactive, teamRow.jerseycolour;  EXIT WHEN teamsSelectAll\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(teamRow.teamid, 13, ' ')||  RPAD(teamRow.teamname,13,' ')||RPAD(teamRow.isactive,13,' ')||  RPAD(teamRow.jerseycolour,13,' '));  END LOOP;  CLOSE teamsSelectAll\_refcursor;  END; |

**spRostersSelectAll(refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **refcursor**  **errorCode** | refcur\_rosters.refcur\_r  NUMBER | Size of pointer to the element of the collection  Precision: 38 Scale: 0 | Reference to a cursor that returns a rosters%rowtype  Error Code | |
| **expected outputs** |
| error code 10, 1  Select all rows from the table rosters and output them to software |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(rostersoutput(**CURSOR**(**SELECT\*FROM** rosters ))); The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output all rows from the table rosters. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  rostersSelectAll\_refcursor refcur\_rosters.refcur\_r;  rostersRow rosters%ROWTYPE;  BEGIN  spRostersSelectAll( rostersSelectAll\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('roster ID',13,' ')||RPAD('player ID',13,' ')||  RPAD('team ID',13,' ')||RPAD('is Active',13,' ')|| RPAD('jersey Number',13,' '));  LOOP  FETCH rostersSelectAll\_refcursor  INTO rostersRow.rosterid, rostersRow.playerid, rostersRow.teamid, rostersRow.isactive,  rostersRow.jerseynumber;  EXIT WHEN rostersSelectAll\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(rostersRow.rosterid, 13, ' ')||  RPAD(rostersRow.playerid,13,' ')||RPAD(rostersRow.teamid,13,' ')||  RPAD(rostersRow.isactive,13,' ')|| RPAD(rostersRow.jerseynumber,13,' '));  END LOOP;  CLOSE rostersSelectAll\_refcursor;  END; |

**spSllocationsSelectAll(refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **refcursor**  **errorCode** | refcur\_sllocations.refcur\_sl  NUMBER | Size of pointer to the element of the collection  Precision: 38 Scale: 0 | Reference to a cursor that returns a sllocations%rowtype  Error Code | |
| **expected outputs** |
| error code 10, 1  Select all rows from the table sllocations and output them to software |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(sllocationsoutput(**CURSOR**(**SELECT\*FROM** sllocations ))); The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output all rows from the table sllocations. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  sllocationsSelectAll\_refcursor refcur\_sllocations.refcur\_sl;  sllocationsRow sllocations%ROWTYPE;  BEGIN  spSllocationsAllSelect(85, sllocationsSelectAll\_refcursor, newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('location ID',13,' ')||RPAD('location Name',50,' ')||  RPAD('field Length',13,' ')||RPAD('is Active',13,' '));  LOOP  FETCH sllocationsSelectAll\_refcursor  INTO sllocationsRow.locationid, sllocationsRow.locationname, sllocationsRow.fieldlength,  sllocationsRow.isactive;  EXIT WHEN sllocationsSelectAll\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(sllocationsRow.locationid, 13, ' ')||  RPAD(sllocationsRow.locationname,50,' ')||RPAD(sllocationsRow.fieldlength,13,' ')||  RPAD(sllocationsRow.isactive,13,' '));  END LOOP;  CLOSE sllocationsSelectAll\_refcursor;  END; |

**Question 4**

**spTeamRosterByID(team\_ID2 NUMBER, refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **team\_ID2**  **refcursor**  **errorCode** | NUMBER  refcur\_teamroster.refcur\_tr  NUMBER | Precision: 38 Scale: 0  Size of pointer to the element of the collection  Precision: 38 Scale: 0 | ID of a team  Reference to a cursor that returns a vwPlayerRosters%rowtype  Error Code | |
| **expected outputs** |
| error code 10, 1 |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(teamrosteroutput(**CURSOR**(**SELECT\*FROM** vwPlayerRosters ))) **WHERE** team\_ID = team\_ID2; The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output a row or multiple rows from the view vwPlayerRosters base on team ID. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  teamRosterByID\_refcursor refcur\_teamroster.refcur\_tr;  teamRosterRow vwPlayerRosters%ROWTYPE;  BEGIN  spTeamRosterByID(212,teamRosterByID\_refcursor,newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('player ID',13,' ')||RPAD('regNumber',13,' ')||  RPAD('last Name',13,' ')|| RPAD('first name',13,' ')||RPAD('player isActive',16,' ')||  RPAD('roster ID',13,' ')|| RPAD('team ID',13,' ')||RPAD('roster isActive',16,' ')||  RPAD('jersey Number',15,' ')|| RPAD('team Name',13,' ')||  RPAD('team isActive',14,' ')||RPAD('jerseyColour',13,' '));  LOOP  FETCH teamRosterByID\_refcursor  INTO teamRosterRow.playerid,teamRosterRow.regnumber, teamRosterRow.lastname,  teamRosterRow.firstname, teamRosterRow.player\_isactive, teamRosterRow.rosterid,  teamRosterRow.teamid, teamRosterRow.roster\_isactive,  teamRosterRow.jerseynumber,teamRosterRow.teamname, teamRosterRow.team\_isactive,  teamRosterRow.jerseycolour;  EXIT WHEN teamRosterByID\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(teamRosterRow.playerid,13,' ')||RPAD(teamRosterRow.regnumber,13,' ')||  RPAD(teamRosterRow.lastname,13,' ')||  RPAD(teamRosterRow.firstname,13,' ')||RPAD(teamRosterRow.player\_isactive,16,' ')||  RPAD(teamRosterRow.rosterid,13,' ')||RPAD(teamRosterRow.teamid,13,' ')||  RPAD( teamRosterRow.roster\_isactive,16,' ')||RPAD(teamRosterRow.jerseynumber,15,' ')||  RPAD(teamRosterRow.teamname,13,' ')||RPAD(teamRosterRow.team\_isactive,14,' ')||  RPAD(teamRosterRow.jerseycolour,13,' '));  END LOOP;  CLOSE teamRosterByID\_refcursor;  END; |

**Question 5**

**spTeamRosterByID(team\_Name2 varchar2, refcursor OUT SYS\_REFCURSOR, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **team\_Name2**  **refcursor**  **errorCode** | VARCHAR2  refcur\_teamroster.refcur\_tr  NUMBER | 10 Byte  Size of pointer to the element of the collection  Precision: 38 Scale: 0 | Name of a team  Reference to a cursor that returns a vwPlayerRosters%rowtype  Error Code | |
| **expected outputs** |
| error code 10, 1 |
| **potential error codes** |
| 10 = success  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| This Select procedure contains a query that uses a pipelined table function. It also uses an OUT SYS\_REFCURSOR to fetch the result from that query and outputs it to software. A *pipelined* table function returns a collection in an iterative, pipelined way.  A table function that accepts a REF CURSOR as an argument can use it to fetch the input rows, maybe perform some transformation on them, and then pipeline the results out. A *pipelined* table function includes a special instruction to pipeline results (single elements of the collection) out of the function instead of returning the whole collection as a single value. When used in a query a table function itself specifies the partitioning approach that is appropriate for it. In this example: **SELECT** \* **FROM TABLE**(teamrosteroutput(**CURSOR**(**SELECT\*FROM** vwPlayerRosters ))) **WHERE UPPER**(team\_Name) **LIKE** (‘%’||**UPPER**(team\_Name2)||’%’); The function is then executed in a two-stage operation. First, one set of slave processes partitions the data as directed in the function's declaration; then a second set of slave scans executes the table function in parallel on the partitioned data. The two slave process operate concurrently with each other to redistribute the rows and produces the result you see. This is how the desire output is pipelined out using a query and then outputted from the procedure by a REF CURSOR that fetches it. In this case the query would output a row or multiple rows from the view vwPlayerRosters base on team name. |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  teamRosterByName\_refcursor refcur\_teamroster.refcur\_tr;  teamRosterRow vwPlayerRosters%ROWTYPE;  BEGIN  spTeamRosterByName(‘kicker’,teamRosterByID\_refcursor,newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '||newErrorCode);  DBMS\_OUTPUT.PUT\_LINE(RPAD('player ID',13,' ')||RPAD('regNumber',13,' ')||  RPAD('last Name',13,' ')|| RPAD('first name',13,' ')||RPAD('player isActive',16,' ')||  RPAD('roster ID',13,' ')|| RPAD('team ID',13,' ')||RPAD('roster isActive',16,' ')||  RPAD('jersey Number',15,' ')|| RPAD('team Name',13,' ')||  RPAD('team isActive',14,' ')||RPAD('jerseyColour',13,' '));  LOOP  FETCH teamRosterByName\_refcursor  INTO teamRosterRow.playerid,teamRosterRow.regnumber, teamRosterRow.lastname,  teamRosterRow.firstname, teamRosterRow.player\_isactive, teamRosterRow.rosterid,  teamRosterRow.teamid, teamRosterRow.roster\_isactive,  teamRosterRow.jerseynumber,teamRosterRow.teamname, teamRosterRow.team\_isactive,  teamRosterRow.jerseycolour;  EXIT WHEN teamRosterByName\_refcursor%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(RPAD(teamRosterRow.playerid,13,' ')||  RPAD(teamRosterRow.regnumber,13,' ')||  RPAD(teamRosterRow.lastname,13,' ')||  RPAD(teamRosterRow.firstname,13,' ')||RPAD(teamRosterRow.player\_isactive,16,' ')||  RPAD(teamRosterRow.rosterid,13,' ')||RPAD(teamRosterRow.teamid,13,' ')||  RPAD( teamRosterRow.roster\_isactive,16,' ')||RPAD(teamRosterRow.jerseynumber,15,' ')||  RPAD(teamRosterRow.teamname,13,' ')||RPAD(teamRosterRow.team\_isactive,14,' ')||  RPAD(teamRosterRow.jerseycolour,13,' '));  END LOOP;  CLOSE teamRosterByName\_refcursor;  END; |

**Question 7**

**fncNumPlayersByTeamID(team\_ID2 NUMBER, errorCode OUT NUMBER)**

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | | **team\_ID2**  **errorCode** | NUMBER  NUMBER | Precision: 38 Scale: 0  Precision: 38 Scale: 0 | ID of a team  Error Code | |
| **expected outputs** |
| error code 10, 1,2,3  A number representing the number of players in a team. |
| **potential error codes** |
| 10 = success, 2=no row found so no record to output, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
| Return the number of players in a team from the vwTeamsNumPlayers |
| **an example of non-saved procedural code to execute the given object** |
| DECLARE  newErrorCode NUMBER := 0;  numPlayers NUMBER;  BEGIN  numPlayers := fncNumPlayersByTeamID(315,newErrorCode);  DBMS\_OUTPUT.PUT\_LINE('Error Code: '|| newErrorCode||' Number of Players: '||numPlayers);  END; |

|  |
| --- |
| **Required input parameters (type, size, and meaning)** |
| |  |  |  |  | | --- | --- | --- | --- | | Input parameter | Type | Size | Meaning | |  |  |  |  | |
| **expected outputs** |
| error code 10, 1,2,3 |
| **potential error codes** |
| 10 = success, 2=no row found so no record to delete, 3= more than 1 row found  1 = OTHERs which traps all remaining error and out put 1 as a result |
| **a description of its’ purpose** |
|  |
| **an example of non-saved procedural code to execute the given object** |
|  |