Users guide

Assignment 2

Group 9

Course: DBS311

Fall 2023 Semester

Contents

Object 1: spPlayersInsert

1. Required Input Parameters:

- p_playerId (Type: NUMBER, Meaning: Player ID)
- p_regnumber (Type: VARCHAR2, Meaning: Registration Number)
- p_lastname (Type: VARCHAR2, Meaning: Last Name)
- p_firstname (Type: VARCHAR2, Meaning: First Name)
- p_isactive (Type: NUMBER, Meaning: Active status, 1 for active, 0 for inactive)

2. Expected Outputs:

- out_result (OUT parameter of type NUMBER)
- Returns p_playerId if the insertion is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If p_playerId is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the insert, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Inserts a new player into the 'players' table, provided that the p_playerId is valid and positive.
- Handles exceptions for invalid input or any other unforeseen issues during the insertion.
- Returns the inserted player's ID if the insertion is successful.

5. Example of Non-saved Procedural Code:

```
---------------------------- TEST spTeamInsert -----------------------------
DECLARE
    out_pk NUMBER;

BEGIN
    spTeamInsert(231,'Liverpool',1, 'Red',out_pk);

        IF out_pk = -1 THEN
                DBMS_OUTPUT.PUT_LINE('Insertion failed.');
            ELSE
                DBMS_OUTPUT.PUT_LINE('Insert a new team ID: '||out_pk);
        END IF;
END;
-------------------------------------------------------------------------
```

Object 2: spPlayersUpdate

1. Required Input Parameters:

- p_playerId (Type: NUMBER, Meaning: Player ID)
- p_regnumber (Type: VARCHAR2, Meaning: Registration Number)
- p_lastname (Type: VARCHAR2, Meaning: Last Name)
- p_firstname (Type: VARCHAR2, Meaning: First Name)
- p_isactive (Type: NUMBER, Meaning: Active status, 1 for active, 0 for inactive)

2. Expected Outputs:

- out_result (OUT parameter of type NUMBER)
- Returns p_playerId if the update is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If p_playerId is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the update, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Updates player information in the 'players' table based on the provided input parameters.
- Checks the validity of the primary key (p_playerId) before proceeding with the update.
- Handles exceptions for invalid input or any other unforeseen issues during the update.
- Returns the updated player's ID if the update is successful.

5. Example of Non-saved Procedural Code:

```
-------------------------- TEST spPlayersUpdate --------------------------------
DECLARE
    v_result NUMBER;

BEGIN
    spPlayersUpdate(2323745,'X1110','Parker','Peter',1,v_result);

    -- Check the result
    IF v_result = -1 THEN
        DBMS_OUTPUT.PUT_LINE('Update failed.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Update successful for Player ID: ' || v_result);
    END IF;

END;
--------------------------------------------------------------------------------
```

Object 3 : spPlayersDelete

1. Required Input Parameters:

p_playerId (Type: NUMBER, Meaning: Player ID)

2. Expected Outputs:

- out_result (OUT parameter of type NUMBER)
- Returns p_playerId if the deletion is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If p_playerId is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the delete operation, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Deletes a player from the 'players' table based on the provided player ID (p_playerId).
- Checks the validity of the primary key (p_playerId) before proceeding with the deletion.
- Handles exceptions for invalid input or any other unforeseen issues during the deletion.
- Returns the deleted player's ID if the deletion is successful.

5. Example of Non-saved Procedural Code:

```
-------------------------- TEST spPlayersDelete --------------------------------
DECLARE
    v_result NUMBER;

BEGIN

    spPlayersDelete(2323743,v_result);

    -- Check the result
    IF v_result = -1 THEN
        DBMS_OUTPUT.PUT_LINE('Delete failed.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Delete successful for Player ID: ' || v_result);
    END IF;

END;
--------------------------------------------------------------------------------
```

Object 4 : spPlayersSelect

1. Required Input Parameters:

p_playerId (Type: NUMBER, Meaning: Player ID)

2. Expected Outputs:

- out_regnumber (OUT parameter of type VARCHAR2, Meaning: Registration Number)
- out_lastname (OUT parameter of type VARCHAR2, Meaning: Last Name)
- out_firstname (OUT parameter of type VARCHAR2, Meaning: First Name)
- out_isactive (OUT parameter of type NUMBER, Meaning: Active status, 1 for active, 0 for inactive)
- out_result (OUT parameter of type NUMBER)
- Returns p_playerId if the selection is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If p_playerId is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the selection, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Selects player information from the 'players' table based on the provided player ID (p_playerId).
- Checks the validity of the primary key (p_playerId) before proceeding with the selection.
- Handles exceptions for invalid input or any other unforeseen issues during the selection.
- Returns the selected player's ID if the selection is successful.

5. Example of Non-saved Procedural Code

```
---------------------------- TEST SELECT function------------------------------
DECLARE
    v_regnumber VARCHAR2(50);
    v_lastname VARCHAR2(50);
    v_firstname VARCHAR2(50);
    v_isactive NUMBER;
    v_result NUMBER;

BEGIN
    spPlayersSelect(1302,v_regnumber,v_lastname,v_firstname,v_isactive,v_result);

    -- Check the result
    IF v_result = -1 THEN
        DBMS_OUTPUT.PUT_LINE('Select failed or Player ID not found.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Select successful for Player ID: ' || v_result);
        DBMS_OUTPUT.PUT_LINE('RegNumber: ' || v_regnumber);
        DBMS_OUTPUT.PUT_LINE('LastName: ' || v_lastname);
        DBMS_OUTPUT.PUT_LINE('FirstName: ' || v_firstname);
        DBMS_OUTPUT.PUT_LINE('IsActive: ' || v_isactive);
    END IF;
END;
--------------------------------------------------------------------------------
```

Object 5 : spTeamInsert

1. Required Input Parameters:

- t_teamId (Type: NUMBER, Meaning: Team ID)
- t_teamName (Type: VARCHAR2, Meaning: Team Name)
- t_isActive (Type: NUMBER, Meaning: Active status, 1 for active, 0 for inactive)
- t_jerseyColor (Type: VARCHAR2, Meaning: Jersey Color)

2. Expected Outputs:

- out_result (OUT parameter of type NUMBER)
- Returns t_teamId if the insertion is successful.
- Returns -1 in case of errors (invalid primary key, duplicate rows, or other exceptions).

3. Potential Error Codes:

- If t_teamId is less than or equal to 0, sets out_result to -1.
- If duplicate rows are found during the insert, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Inserts a new team into the 'teams' table, provided that the t_teamId is valid and positive.
- Checks for duplicate rows and raises an exception if found.
- Handles exceptions for invalid input or any other unforeseen issues during the insertion.
- Returns the inserted team's ID if the insertion is successful.

5. **Example of Non-saved Procedural Code:**

```
--------------------------- TEST spTeamInsert -------------------------------
DECLARE
    out_pk NUMBER;

BEGIN
    spTeamInsert(231,'Liverpool',1, 'Red',out_pk);

        IF out_pk = -1 THEN
                DBMS_OUTPUT.PUT_LINE('Insertion failed.');
            ELSE
                DBMS_OUTPUT.PUT_LINE('Insert a new team ID: '||out_pk);
        END IF;
END;
-------------------------------------------------------------------------------
```

Object 6 : spTeamsUpdate

1. Required Input Parameters:

- t_teamId (Type: NUMBER, Meaning: Team ID)
- t_teamName (Type: VARCHAR2, Meaning: Updated Team Name)
- t_isActive (Type: NUMBER, Meaning: Updated Active status, 1 for active, 0 for inactive)
- t_jerseyColour (Type: VARCHAR2, Meaning: Updated Jersey Color)

2. Expected Outputs:

- out_result (OUT parameter of type NUMBER)
- Returns t_teamId if the update is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If t_teamId is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the update, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Updates the information of an existing team in the 'teams' table, provided that the t_teamId is valid and positive.
- Handles exceptions for invalid input or any other unforeseen issues during the update.
- Returns the updated team's ID if the update is successful.

5. Example of Non-saved Procedural Code:

```
------------------------- TEST spPlayersUpdate --------------------------------
DECLARE
    v_result NUMBER;

BEGIN
    spTeamsUpdate(230,'Man City',1,'Blue',v_result);

    -- Check the result
    IF v_result = -1 THEN
        DBMS_OUTPUT.PUT_LINE('Update failed.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Update successful for team ID: ' || v_result);
    END IF;
END;
-------------------------------------------------------------------------------
```

Object 7 : spTeamsDelete

1. Required Input Parameters:

p_teamId (Type: NUMBER, Meaning: Team ID)

2. Expected Outputs:

- out_result (OUT parameter of type NUMBER)
- Returns p_teamId if the deletion is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If p_teamId is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the deletion, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Deletes an existing team from the 'teams' table, provided that the p_teamId is valid and positive.
- Handles exceptions for invalid input or any other unforeseen issues during the deletion.
- Returns the deleted team's ID if the deletion is successful.

5. Example of Non-saved Procedural Code:

```
-------------------------Test spTeamsDelete-------------------------------------
DECLARE
    v_result NUMBER;

BEGIN

    spTeamsDelete(230,v_result);

    -- Check the result
    IF v_result = -1 THEN
        DBMS_OUTPUT.PUT_LINE('Delete failed.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Delete successful for Team ID: ' || v_result);
    END IF;

END;
-------------------------------------------------------------------------------
```

Object 8 : spTeamsSelect

1. Required Input Parameters:

t_teamId (Type: NUMBER, Meaning: Team ID)

2. Expected Outputs:

- out_teamName (OUT parameter of type VARCHAR2, Meaning: Team Name)
- out_isActive (OUT parameter of type NUMBER, Meaning: Active status, 1 for active, 0 for inactive)
- out_jerseyColour (OUT parameter of type VARCHAR2, Meaning: Jersey Colour)
- out_result (OUT parameter of type NUMBER)
- Returns t_teamId if the selection is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If t_teamId is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the selection, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Selects data of an existing team from the 'teams' table based on the provided t_teamId.
- Handles exceptions for invalid input or any other unforeseen issues during the selection.
- Returns the selected team's ID if the selection is successful.

5. Example of Non-saved Procedural Code:

```
---------------------------- TEST spTeamsSelect function-------------------------
DECLARE
    v_teamName VARCHAR2(50);
    v_isActive NUMBER;
    v_jerseyColour VARCHAR2(50);
    v_result  NUMBER;
BEGIN
    spTeamsSelect(210,v_teamName,v_isActive,v_jerseyColour,v_result);

    -- Check the result
    IF v_result = -1 THEN
        DBMS_OUTPUT.PUT_LINE('Select failed or Team ID not found.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Select successful for Team ID: ' || v_result);
        DBMS_OUTPUT.PUT_LINE('Team name: ' || v_teamName);
        DBMS_OUTPUT.PUT_LINE('IsActive: ' || v_isActive);
        DBMS_OUTPUT.PUT_LINE('Jersey colour: ' || v_jerseyColour);
    END IF;
END;

--------------------------------------------------------------------------------
```

Object 9 : spRosterInsert

1. Required Input Parameters:

- r_playerid (Type: NUMBER, Meaning: Player ID)
- r_teamId (Type: NUMBER, Meaning: Team ID)
- r_isactive (Type: NUMBER, Meaning: Active status, 1 for active, 0 for inactive)
- r_jerseyNumber (Type: NUMBER, Meaning: Jersey Number)

2. Expected Outputs:

- out_result (OUT parameter of type NUMBER)
- Returns r_playerid if the insertion is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If r_playerid is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the insert, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Inserts a new roster entry into the 'rosters' table, provided that the r_playerid is valid and positive.
- Handles exceptions for invalid input or any other unforeseen issues during the insertion.
- Returns the inserted player's ID if the insertion is successful.

5. Example of Non-saved Procedural Code:

```
---------------------------- TEST spTeamInsert --------------------------------
DECLARE
    out_pk NUMBER;

BEGIN
    spRosterInsert(2323743,223,1,10,out_pk);

        IF out_pk = -1 THEN
                DBMS_OUTPUT.PUT_LINE('Insertion failed.');
            ELSE
                DBMS_OUTPUT.PUT_LINE('Insert a new roster for player id: '||out_pk);

            END IF;
END;
-------------------------------------------------------------------------------
```

Object 10 : sprRosterUpdate

1. Required Input Parameters:

- r_rosterid (Type: NUMBER, Meaning: Roster ID)
- r_playerid (Type: NUMBER, Meaning: Player ID)
- r_teamId (Type: NUMBER, Meaning: Team ID)
- r_isactive (Type: NUMBER, Meaning: Active status, 1 for active, 0 for inactive)
- r_jerseyNumber (Type: NUMBER, Meaning: Jersey Number)

2. Expected Outputs:

- out_result (OUT parameter of type NUMBER)
- Returns r_rosterid if the update is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If r_rosterid is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the update, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Updates an existing roster entry in the 'rosters' table, provided that the r_rosterid is valid and positive.
- Handles exceptions for invalid input or any other unforeseen issues during the update.
- Returns the updated roster ID if the update is successful.

5. Example of Non-saved Procedural Code:

```
-------------------------- TEST spPlayersUpdate --------------------------------
DECLARE
    v_result NUMBER;

BEGIN
    sprRosterUpdate(230,1150,223,1,27,v_result);

    -- Check the result
    IF v_result = -1 THEN
        DBMS_OUTPUT.PUT_LINE('Update failed.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Update successful for roster ID: ' || v_result);
    END IF;
END;
--------------------------------------------------------------------------------
```

Object 11 : spRostersDelete

1. Required Input Parameters:

r_rosterId (Type: NUMBER, Meaning: Roster ID)

2. Expected Outputs:

- out_result (OUT parameter of type NUMBER)
- Returns r_rosterId if the deletion is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If r_rosterId is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the deletion, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Deletes an existing roster entry from the 'rosters' table, provided that the r_rosterId is valid and positive.
- Handles exceptions for invalid input or any other unforeseen issues during the deletion.
- Returns the deleted roster ID if the deletion is successful.

5. Example of Non-saved Procedural Code:

```
-------------------------Test spTeamsDelete-----------------------------------
DECLARE
    v_result NUMBER;

BEGIN

    spRostersDelete(230,v_result);

    -- Check the result
    IF v_result = -1 THEN
        DBMS_OUTPUT.PUT_LINE('Delete failed.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Delete successful for Team ID: ' || v_result);
    END IF;

END;
-------------------------------------------------------------------------------
```

Object 12 : spRosterSelect

1. Required Input Parameters:

r_rosterId (Type: NUMBER, Meaning: Roster ID)

2. Expected Outputs:

- r_playerId (OUT parameter of type NUMBER)
- r_teamId (OUT parameter of type NUMBER)
- r_isactive (OUT parameter of type NUMBER)
- r_jerseyNumber (OUT parameter of type NUMBER)
- out_result (OUT parameter of type NUMBER)
- Returns r_rosterId if the selection is successful.
- Returns -1 in case of errors (invalid primary key, no rows affected, or other exceptions).

3. Potential Error Codes:

- If r_rosterId is less than or equal to 0, sets out_result to -1.
- If no rows were affected during the selection, sets out_result to -1.
- If any other exception occurs during execution, sets out_result to -1.

4. Purpose:

- Selects data associated with a specific roster entry from the 'rosters' table, provided that the r_rosterId is valid and positive.
- Retrieves player ID, team ID, active status, and jersey number associated with the specified roster entry.
- Handles exceptions for invalid input or any other unforeseen issues during the selection.
- Returns the selected roster ID if the selection is successful.

5. Example of Non-saved Procedural Code:

```
--------------------------- TEST spTeamsSelect function-----------------------
DECLARE
    v_rosterid NUMBER;
    v_playerid NUMBER;
    v_teamid NUMBER;
    v_isactive NUMBER;
    v_jerseyNumber NUMBER;
    v_result  NUMBER;
BEGIN
    spRosterSelect(254,v_playerid,v_teamid,v_isactive,v_jerseyNumber,v_result);

    -- Check the result
    IF v_result = -1 THEN
        DBMS_OUTPUT.PUT_LINE('Select failed or Team ID not found.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Select successful for Roster ID: ' || v_result);
        DBMS_OUTPUT.PUT_LINE('Player id: ' || v_playerid);
        DBMS_OUTPUT.PUT_LINE('Team id: ' || v_teamid);
        DBMS_OUTPUT.PUT_LINE('IsActive: ' || v_isactive);
        DBMS_OUTPUT.PUT_LINE('Jersey Number: ' || v_jerseyNumber);
    END IF;

END;
```

Object 13: spPlayersSelectAll

1. Required Input Parameters: None.

2. Expected Outputs: None.

3. Potential Error Codes:

- No specific error codes are defined for this procedure.
- Handles the case where no data is found in the Players table.

4. Purpose:

- Retrieves and prints all records from the 'players' table.
- Displays Player ID, Registration Number, Last Name, First Name, and Active Status for each player.
- Provides a generic procedure to view all players' information.

5. Example of Non-saved Procedural Code:

```
75  -- Test spPlayersSelectAll
76  BEGIN
77      spPlayersSelectAll;
78  END;
```

Object 14: spTeamsSelectAll

1. Required Input Parameters: None.

2. Expected Outputs: None.

3. Potential Error Codes:

- No specific error codes are defined for this procedure.
- Handles the case where no data is found in the Teams table.

4. Purpose:

- Retrieves and prints all records from the 'teams' table.
- Displays Team ID, Team Name, Active Status, and Jersey Colour for each team.
- Provides a generic procedure to view all teams' information.

5. Example of Non-saved Procedural Code:

```
-- Test spTeamsSelectAll
BEGIN
    spTeamsSelectAll;
END;
```

Object 15: spRostersSelectAll

1. Required Input Parameters: None.

2. Expected Outputs: None.

3. Potential Error Codes:

- No specific error codes are defined for this procedure.
- Handles the case where no data is found in the Teams table.

4. Purpose:

- Retrieves and prints all records from the 'teams' table.
- Displays Team ID, Team Name, Active Status, and Jersey Colour for each team.
- Provides a generic procedure to view all teams' information.

5. Example of Non-saved Procedural Code:

```
-- Test spRostersSelectAll
BEGIN
    spRostersSelectAll;
END;
```

Object 16: spPlayersSelectAll (Optional)

1. Required Input Parameters:

p_result (OUT parameter of type SYS_REFCURSOR): Cursor variable to hold the result set.

2. Expected Outputs: None.

3. Potential Error Codes:

- No specific error codes are defined for this procedure.
- Handles the case where no data is found in the 'players' table.

4. Purpose:

- Retrieves all records from the 'players' table.
- Utilizes a SYS_REFCURSOR to provide a reference to the result set.
- Can be used to fetch and display all player records in a more flexible manner.

5. Example of Non-saved Procedural Code:

```
--------------------------- Test spPlayersSelectAll ----------------------------
DECLARE
    v_player_id NUMBER;
    v_reg_number VARCHAR2(50);
    v_last_name VARCHAR2(50);
    v_first_name VARCHAR2(50);
    v_is_active NUMBER;
    v_result SYS_REFCURSOR;
BEGIN
    -- Call the stored procedure
    spPlayersSelectAll(p_result => v_result);

    -- Fetch the result set
    LOOP
        FETCH v_result
        INTO v_player_id,
             v_reg_number,
             v_last_name,
             v_first_name,
             v_is_active;
        EXIT WHEN v_result%NOTFOUND;

        -- Output the fetched data to the script window
        DBMS_OUTPUT.PUT_LINE('Player ID: ' || v_player_id);
        DBMS_OUTPUT.PUT_LINE('RegNumber: ' || v_reg_number);
        DBMS_OUTPUT.PUT_LINE('LastName: ' || v_last_name);
        DBMS_OUTPUT.PUT_LINE('FirstName: ' || v_first_name);
        DBMS_OUTPUT.PUT_LINE('IsActive: ' || v_is_active);
        DBMS_OUTPUT.PUT_LINE('------------------------');
    END LOOP;
    -- Close the cursor
    CLOSE v_result;
END;
--------------------------------------------------------------------------------
```

Object 17: spTeamsSelectAll (Optional)

1. Required Input Parameters:

t_result (OUT parameter of type SYS_REFCURSOR): Cursor variable to hold the result set.

2. Expected Outputs: None.

3. Potential Error Codes:

- No specific error codes are defined for this procedure.
- Handles the case where no data is found in the 'teams' table.

4. Purpose:

- Retrieves all records from the 'teams' table.
- Utilizes a SYS_REFCURSOR to provide a reference to the result set.
- Can be used to fetch and display all team records in a more flexible manner.

5. Example of Non-saved Procedural Code:

```
--------------------------- Test spTeamsSelectAll ----------------------------
DECLARE
    v_team_id NUMBER;
    v_team_name VARCHAR2(50);
    v_is_active NUMBER;
    v_jersey_colour VARCHAR2(50);
    v_result SYS_REFCURSOR;

BEGIN
    -- Call the stored procedure
    spTeamsSelectAll(t_result => v_result);

    -- Fetch the result set
    LOOP
        FETCH v_result
        INTO v_team_id,
             v_team_name,
             v_is_active,
             v_jersey_colour;
        EXIT WHEN v_result%NOTFOUND;

        -- Output the fetched data to the script window
        DBMS_OUTPUT.PUT_LINE('Team ID: ' || v_team_id);
        DBMS_OUTPUT.PUT_LINE('Team Name: ' || v_team_name);
        DBMS_OUTPUT.PUT_LINE('IsActive: ' || v_is_active);
        DBMS_OUTPUT.PUT_LINE('Jersey Colour: ' || v_jersey_colour);
        DBMS_OUTPUT.PUT_LINE('------------------------');
    END LOOP;
    -- Close the cursor
    CLOSE v_result;
END;
--------------------------------------------------------------------------------
```

Object 18: spRostersSelectAll (Optional)

1. Required Input Parameters:

r_result (OUT parameter of type SYS_REFCURSOR): Cursor variable to hold the result set.

2. Expected Outputs: None.

3. Potential Error Codes:

- No specific error codes are defined for this procedure.
- Handles the case where no data is found in the 'rosters' table.

4. Purpose:

- Retrieves all records from the 'rosters' table.
- Utilizes a SYS_REFCURSOR to provide a reference to the result set.
- Can be used to fetch and display all roster records in a more flexible manner.

5. Example of Non-saved Procedural Code:

```
--------------------------- Test spRostersSelectAll ----------------------------
DECLARE
    v_roster_id NUMBER;
    v_player_id NUMBER;
    v_team_id NUMBER;
    v_is_active NUMBER;
    v_jersey_number NUMBER;
    v_result SYS_REFCURSOR;
BEGIN
    -- Call the stored procedure
    spRostersSelectAll(r_result => v_result);
    -- Fetch the result set
    LOOP
        FETCH v_result
        INTO v_roster_id,
             v_player_id,
             v_team_id,
             v_is_active,
             v_jersey_number;
        EXIT WHEN v_result%NOTFOUND;
        -- Output the fetched data to the script window
        DBMS_OUTPUT.PUT_LINE('Roster ID: ' || v_roster_id);
        DBMS_OUTPUT.PUT_LINE('Player ID: ' || v_player_id);
        DBMS_OUTPUT.PUT_LINE('Team ID: ' || v_team_id);
        DBMS_OUTPUT.PUT_LINE('IsActive: ' || v_is_active);
        DBMS_OUTPUT.PUT_LINE('Jersey Number: ' || v_jersey_number);
        DBMS_OUTPUT.PUT_LINE('-------------------------');
        DBMS_OUTPUT.PUT_LINE(CHR(10));
    END LOOP;
    CLOSE v_result;
END;
--------------------------------------------------------------------------------
```

Object 19: vwPlayerRosters

1. Object Information:

- Object Type: VIEW
- Object Name: vwPlayerRosters

2. Description:

- The vwPlayerRosters view is a virtual table that combines information from the players, rosters, and teams tables.
- It provides a comprehensive overview of player details, roster details, and team details in a single result set.

3. Columns:

- playerid (Type: NUMBER): Player ID.
- regnumber (Type: VARCHAR2): Registration Number of the player.
- lastname (Type: VARCHAR2): Last Name of the player.
- firstname (Type: VARCHAR2): First Name of the player.
- player_isactive (Type: NUMBER): Active status of the player (1 for active, 0 for inactive).
- rosterid (Type: NUMBER): Roster ID.
- teamid (Type: NUMBER): Team ID.
- roster_isactive (Type: NUMBER): Active status of the roster (1 for active, 0 for inactive).
- jerseynumber (Type: NUMBER): Jersey Number assigned to the player in the roster.
- teamname (Type: VARCHAR2): Name of the team.
- team_isactive (Type: NUMBER): Active status of the team (1 for active, 0 for inactive).
- jerseycolour (Type: VARCHAR2): Colour of the team's jersey.

4. Example Usage:

```
--- Test vwPlayerRosters
SELECT * FROM vwPlayerRosters;
```

Object 20: spTeamRosterByID

1. Required Input Parameters:

p_teamId (Type: NUMBER): Team ID for which the roster information is to be retrieved.

2. Expected Outputs:

The procedure doesn't return a value but uses DBMS_OUTPUT.PUT_LINE to display player, roster, and team information.

3. Potential Error Codes:

- If there is an issue with the execution, it may raise a NO_DATA_FOUND exception.
- If no data is found for the specified team, a custom error message is displayed.

4. Purpose:

- Retrieves and displays player, roster, and team information for a specified team from the vwPlayerRosters view.
- Offers a convenient way to view details about players, rosters, and teams associated with a specific team ID.

5. Example of Non-saved Procedural Code:

```
--------------------------- Test spTeamRosterByID -----------------------------

BEGIN
    spTeamRosterByID(210);
    spTeamRosterByID(0);
END;
```

Object 21: spTeamRosterByName

1. Required Input Parameters:

p_teamName (Type: VARCHAR2): Team Name or a part of it to search for.

2. Expected Outputs:

The procedure doesn't return a value but uses DBMS_OUTPUT.PUT_LINE to display player, roster, and team information.

3. Potential Error Codes:

- If there is an issue with the execution, it may raise a NO_DATA_FOUND exception.
- If no data is found for the specified team name, a custom error message is displayed.

4. Purpose:

- Retrieves and displays player, roster, and team information for teams whose names contain the specified string.
- Offers a convenient way to view details about players, rosters, and teams associated with a specific team name.

5. Example of Non-saved Procedural Code:

```
---------------------- TEST spTeamRosterByName---------------------------------
BEGIN
    spTeamRosterByName('Bo');
    spTeamRosterByName('OO');
END;
```

Object 22: vwTeamsNumPlayers

1. Object Information:

Name: vwTeamsNumPlayers

Type: VIEW

2. Description:

This view provides information about the number of players currently registered on each team.

3. Columns:

- teamid (Type: Data Type): The unique identifier for the team.
- teamname (Type: Data Type): The name of the team.
- number_players (Type: Data Type): The count of players currently registered on the team.

4. Example Usage:

```
-------------------------- Test vwTeamsNumPlayers view --------------------------
SELECT* FROM vwTeamsNumPlayers;
```

Object 23: fncNumPlayersByTeamID

1. Required Input Parameters:

p_teamId (Type: NUMBER): The unique identifier for the team.

2. Expected Outputs:

- Return Type: NUMBER
- Description: Returns the number of players currently registered on the specified team.

3. Potential Error Codes:

If no data is found for the provided teamId, the function returns NULL.

4. Purpose:

- This function retrieves the number of players currently registered on a specified team using the vwTeamsNumPlayers view.
- It provides a convenient way to obtain the player count for a given team.

5. Example of Non-saved Procedural Code:

```
---------------------- TEST fncNumPlayersByTeamID-------------------------------
DECLARE
   v_numPlayers NUMBER;
   v_teamId NUMBER;
BEGIN
   v_teamId :=222;
   -- Replace with the actual team ID
   v_numPlayers := fncNumPlayersByTeamID(v_teamId);

   IF v_numPlayers IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('TeamId:'|| v_teamId||' has ' ||
                           v_numPlayers ||' players.');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Team not found.');
   END IF;

END;
--------------------------------------------------------------------------------
```

Object 24: vwSchedule

1. Object Information:

Name: vwSchedule

Type: VIEW

2. Description:

The vwSchedule view provides information about scheduled games, including details such as game ID, date and time, home team, away team, and location.

3. Columns:

- gameid (Type: NUMBER): Unique identifier for the game.
- gamedate (Type: DATE): Date and time of the scheduled game.
- hometeamid (Type: NUMBER): Team ID of the home team.
- hometeam (Type: VARCHAR2): Name of the home team.
- awayteamid (Type: NUMBER): Team ID of the away team.
- awayteam (Type: VARCHAR2): Name of the away team.
- locationid (Type: NUMBER): Unique identifier for the location.
- locationname (Type: VARCHAR2): Name of the game location.

4. Example Usage:

```
-------------------------- Test vwSchedule view --------------------------------
SELECT* FROM vwSchedule;
```

Object 25: spSchedUpcomingGames

1. Required Input Parameters:

n (Type: NUMBER, Meaning: Number of days to look ahead)

Example: If n is set to 7, the procedure will retrieve and display games played in the next 7 days.

2. Expected Outputs:

- The procedure leverages DBMS_OUTPUT.PUT_LINE to neatly display information about upcoming games.
- Each upcoming game is presented in the following format:

Game: [GameID], Date: [FormattedDate]

[GameID]: Unique identifier of the game.

[FormattedDate]: Date and time of the game in the format MM/DD/YYYY HH24:MI:SS.

If no upcoming games are found within the specified timeframe, the procedure outputs:

3. Potential Error Codes:

- TOO_MANY_ROWS: If there are too many rows in the result set.
- NO_DATA_FOUND: If no upcoming games are found in the specified timeframe.

4. Purpose:

The spSchedUpcomingGames procedure retrieves and displays information about upcoming games scheduled within the next n days.

5. Example of Non-saved Procedural Code:

```
--- Test spSchedUpcomingGames
BEGIN
  spSchedUpcomingGames(5); -- Assuming you want to find games in the next 5 days
END;
```

Object 26: spSchedPastGames

1. Required Input Parameters:

N(Type: Number) : Number of days to look back for past games. It must be a positive integer value.

Example: If n is set to 7, the procedure will retrieve and display games played in the last 7 days.

2. Expected Outputs:

- Each past game is presented in the format: Game: [GameID], Date: [FormattedDate].

[GameID]: Unique identifier of the game.

[FormattedDate]: Date and time of the game in the format MM/DD/YYYY HH24:MI:SS.

If no past games are found within the specified timeframe, it outputs: No past games in the last [n] days.

[n]: The number of days specified as the input parameter.

3. Potential Error Codes:

- TOO_MANY_ROWS: If there are too many rows in the result set.
- NO_DATA_FOUND: If no upcoming games are found in the specified timeframe.

4. Purpose:

- Retrieves and displays information about games played within a specified number of past days . Aids users in reviewing recent games for reference or analysis.
- The spSchedPastGames procedure retrieves and displays information about past games that occurred within the last n days.

5. Example of Non-saved Procedural Code:

```
-- Assuming you want to find games played in the last 7 days
BEGIN
  spSchedPastGames(1);
END;
```

Object 27: spRunStandings

1. Required Input Parameters:

None.

2. Expected Outputs:

- The tempstandings table is updated with the calculated standings data. The procedure prints 'Tempstanding table has been updated!' to the DBMS_OUTPUT buffer upon successful execution

3. Potential Error Codes:

TOO_MANY_ROWS: If there are too many rows in the result set.

4. Purpose:

The spRunStandings procedure updates the tempstandings table with the latest standings data based on the game results.

5. Example of Non-saved Procedural Code:

```
--------------------------------Test spRunStandings ----------------------------
BEGIN
    spRunStandings;
END;
--------------------------------------------------------------------------------
```

Object 28: trgUpdateStandings (Trigger)

1. Object Information:

Name: trgUpdateStandings

Type: AFTER INSERT OR UPDATE Trigger

Table Affected: games

2. Description:

- The trgUpdateStandings trigger fires after an INSERT or UPDATE operation on the games table.
- It updates the tempstandings table with the latest standings data based on the game results.

3. Example Usage:

- This trigger is automatically invoked after the insertion or update of scores and the isPlayed status in the games table.
- It ensures that the tempstandings table is updated with the latest standings information.

```
-----------------------------Test the triggger ----------------------------------
-- Update scores
-- Game 1
-- hometeam 218 has pts 14 , awayteam 212 has pts 12
UPDATE games SET homescore = 6, visitscore = 1 WHERE gameid = 1;
```

Object 29: sp_updateScores

1. Required Input Parameters:

- p_game_id (Type: INT, Meaning: Game ID)
- p_home_score (Type: INT, Meaning: New score for the home team)
- p_visit_score (Type: INT, Meaning: New score for the visiting team)

2. Expected Outputs:

- If the update is successful, the procedure outputs 'Scores updated successfully!'
- Users can use this procedure confidently to manage and reflect the latest score changes in the database standings

3. Potential Error Codes:

If there is an error during the update, the procedure outputs an error message using DBMS_OUTPUT.PUT_LINE('Error updating scores: ' || SQLERRM);.

4. Purpose:

- The purpose of this procedure is to update the scores for a specified game in the games table.
- It provides a way to modify game scores based on the provided parameters

5. Example of Non-saved Procedural Code:

```
--- Test sp_updateScores
BEGIN
    sp_updateScores(21, 5, 1); -- Replace with actual values and game ID
END;
```