Table of Contents

1 Table of Contents

2 Introduction

2.1 Overview of the Database System

In the dynamic world of data management, proficiency in handling and manipulating data forms the bedrock of effective database systems. Our report delves into an intricately designed database system, built around the core tables: `Players`, `Teams`, and `Rosters`. This system is adept at executing a wide range of operations, from fundamental CRUD (Create, Read, Update, Delete) tasks to intricate data retrieval and scheduling functionalities. The incorporation of advanced stored procedures, user-defined functions, and triggers enhances the system's capability to manage data efficiently and interact dynamically with the database. Each element is meticulously designed to contribute to a unified, efficient, and user-centric experience, emphasizing streamlined data processing and improved accessibility.

2.2 Importance of Error Handling in Stored Procedures

A crucial element of our database system is its sophisticated error handling mechanism. The importance of this feature in maintaining data integrity and operational continuity cannot be overstated. Our system employs a nuanced error handling strategy within its stored procedures, characterized by standardized error codes for accurate and swift issue resolution. These codes include:

- `1` for Success,
- `-1` indicating No Data Found,
- `-2` for scenarios where Many Rows are Returned,
- `-3` when No Row is Inserted/Updated/Deleted,
- `-4` in cases of Value Error,
- `-5` for Others (covering unspecified exceptions),
- `-6` representing an Invalid Cursor.

Such a systematic approach to error handling not only facilitates rapid identification and rectification of issues but also establishes a uniform framework for managing errors. By integrating comprehensive error handling protocols, we aim to preclude potential data inconsistencies, minimize disruptions, and elevate the overall reliability and trustworthiness of the database system.

3 CRUD Operations for Each Table

3.1 Stored Procedures for Players

3.1.1 spPlayersInsert

Purpose: Inserts a new record into the Players table.
Input Parameters:
pID (Type: players.playerID%TYPE): Player ID.
regNum (Type: players.regnumber%TYPE): Registration Number.
lName (Type: players.lastname%TYPE): Last Name.
fName (Type: players.firstname%TYPE): First Name.
isAct (Type: players.isActive%TYPE): Active Status.
codeErr (OUT Parameter, Type: players.playerID%TYPE): Error Code.
Expected Outputs: Confirmation of insertion or error code.
Error Codes:
1 - Success.
-3 - No Row Inserted or Invalid Active Status.
-4 - Value Error.
-5 - Other Errors.
Example Usage:

```
DECLARE
  v_errCode NUMBER;  -- Variable to store the error code
BEGIN
  -- Insert a new player record
  -- If successful, v_errCode will be 1
  -- If there is an error (like invalid active status), an error code will be set
  spPlayersInsert(101, '12345', 'Doe', 'John', 1, v_errCode);
  -- Display the result or error code
  DBMS_OUTPUT.PUT_LINE('Error Code: ' || v_errCode);
END;
```

Expected Output: Displays 'Error Code: 1'.

3.1.2 spPlayersUpdate

- Purpose: Updates an existing record in the `Players` table given the Player ID.
- Input Parameters:
pID (Type: `players.playerID%TYPE`): Player ID.
regnum (Type: `players.regnumber%TYPE`): Registration Number.
lName (Type: `players.lastname%TYPE`): Last Name.
fName (Type: `players.firstname%TYPE`): First Name.
isAct (Type: `players.isActive%TYPE`): Active Status.
codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
-Expected Outputs: Confirmation of update or error code.
-Error Codes
    1 - Success.
    -1 - No Data Found.
    -3 - Invalid `isActive` value.
    -4 - Value Error.
    -5 - Other Errors.
- Example Usage:

```
DECLARE
  v_errCode NUMBER;  -- Variable to store the error code
BEGIN
  -- Update an existing player record
  -- Error code will indicate the status of the operation
  spPlayersUpdate(101, '54321', 'Doe', 'Jane', 0, v_errCode);
  -- Display the result or error code
  DBMS_OUTPUT.PUT_LINE('Error Code: ' || v_errCode);
END;
```

- Expected Output: Displays 'Error Code: 1'

3.1.3 spPlayersDelete

- Purpose: Deletes an existing record from the `Players` table given the Player ID.
- Input Parameters:
  pID (Type: `players.playerID%TYPE`): Player ID.
  codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
- Expected Outputs: Confirmation of deletion or error code.
- Error Codes:
  1 - Success.
  -1 - No Data Found.
  -4 - Value Error.
  -5 - Other Errors.
- Example Usage:

```
DECLARE
  v_errCode NUMBER;  -- Variable to store the error code
BEGIN
  -- Attempt to delete a player with the specified ID
  spPlayersDelete(101, v_errCode);
  -- Output the result or error code
  DBMS_OUTPUT.PUT_LINE('Error Code: ' || v_errCode);
END;
```

- Expected Output: Displays 'Error Code: 1'.

3.1.4 spPlayersSelect

- Purpose: Retrieves all fields in a single row from the `Players` table given a Player ID.
- Input Parameters:
  pID (Type: `players.playerID%TYPE`): Player ID.
regNum (OUT Parameter, Type: `players.regnumber%TYPE`): Registration Number.
lName (OUT Parameter, Type: `players.lastname%TYPE`): Last Name.
fName (OUT Parameter, Type: `players.firstname%TYPE`): First Name.
isAct (OUT Parameter, Type: `players.isActive%TYPE`): Active Status.
codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
- Expected Outputs: Player data or error code.
- Error Codes:
  1 - Success.
  -1 - No Data Found.
  -2 - Many Rows Returned.
  -4 - Value Error.
  -5 - Other Errors.

- Example Usage:

```
DECLARE
  pID NUMBER;             -- Variable to store the player ID
  regNum VARCHAR2(100);   -- Variable to store the registration number
  lName VARCHAR2(100);    -- Variable to store the last name
  fName VARCHAR2(100);    -- Variable to store the first name
  isAct NUMBER;           -- Variable to store the active status
  v_errCode NUMBER;       -- Variable to store the error code
BEGIN
  pID := 374022;          -- Set the player ID
  -- Retrieve player information for the given ID
  spPlayersSelect(pID, regNum, lName, fName, isAct, v_errCode);
  -- Check if the operation was successful and output the data
  IF v_errCode = 1 THEN
    DBMS_OUTPUT.PUT_LINE('Existing Record: ' || regNum || ', ' || lName || ', ' || fName ||
', ' || isAct);
  ELSE
    DBMS_OUTPUT.PUT_LINE('Existing Record Error Code: ' || v_errCode);-- Output error code
if unsuccessful
  END IF;
END;
```

- Output: Existing Record: 38879, Clamp, Daniel, 1.

3.2 Stored Procedures for Teams

3.2.1 spTeamsInsert

- Purpose: Inserts a new record into the Teams table.
- Input Parameters:
  tID (Type: `teams.teamID%TYPE`): Team ID.
  tName (Type: `teams.teamname%TYPE`): Team Name.
  isAct (Type: `teams.isActive%TYPE`): Active Status.
  jColor (Type: `teams.jerseycolour%TYPE`): Jersey Color.
  codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
- Expected Outputs: Confirmation of insertion or error code.
- Error Codes:
  1 - Success.
  -3 - No Row Inserted or Invalid Active Status.
  -4 - Value Error.
  -5 - Other Errors.
- Example Usage:

```
DECLARE
  v_errCode NUMBER;
BEGIN
  -- Attempt to insert a new team
  spTeamsInsert(1001, 'Team A', 1, 'Blue', v_errCode);
  DBMS_OUTPUT.PUT_LINE('Insert Error Code: ' || v_errCode);
  END;
```

- Expected Output: Displays 'Insert Error Code: 1'.

3.2.2 spTeamsUpdate

- Purpose: Updates an existing record in the Teams table given the Team ID.
- Input Parameters:
  tID (Type: `teams.teamID%TYPE`): Team ID.
  tName (Type: `teams.teamname%TYPE`): Team Name.
  isAct (Type: `teams.isActive%TYPE`): Active Status.
  jColor (Type: `teams.jerseycolour%TYPE`): Jersey Color.
  codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
- Expected Outputs: Confirmation of update or error code.
- Error Codes:
  1 - Success.
  -1 - No Data Found.
  -3 - Invalid `isActive` value.
  -4 - Value Error.
  -5 - Other Errors.
- Example Usage:

```
DECLARE
  v_errCode NUMBER;
BEGIN
  -- Update an existing team
  spTeamsUpdate(1001, 'Team ANew', 0, 'Green', v_errCode);
  DBMS_OUTPUT.PUT_LINE('Update Error Code: ' || v_errCode);
    END;
```

- Expected Output: Displays 'Update Error Code: 1'.

3.2.3 spTeamsDelete

- Purpose: Deletes an existing record from the Teams table given the Team ID.
- Input Parameters:
  tID (Type: `teams.teamID%TYPE`): Team ID.
  codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
- Expected Outputs: Confirmation of deletion or error code.
- Error Codes:
  1 - Success.
  -3 - No Row Deleted.
  -4 - Value Error.
  -5 - Other Errors.
- Example Usage:

```
DECLARE
  v_errCode NUMBER;
BEGIN
  -- Delete an existing team
  spTeamsDelete(1001, v_errCode);
  DBMS_OUTPUT.PUT_LINE('Delete Error Code: ' || v_errCode);
    END;
```

- Expected Output: Displays 'Delete Error Code: 1'.

3.2.4 spTeamsSelect

- Purpose: Retrieves all fields in a single row from the Teams table given a Team ID.
- Input Parameters:
  tID (Type: `teams.teamID%TYPE`): Team ID.
  tName (OUT Parameter, Type: `teams.teamname%TYPE`): Team Name.
  isAct (OUT Parameter, Type: `teams.isActive%TYPE`): Active Status.
  jColor (OUT Parameter, Type: `teams.jerseycolour%TYPE`): Jersey Color.
  codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
- Expected Outputs: Team data or error code.
- Error Codes:
  1 - Success.
  -3 - No Data Found.
  -4 - Value Error.
  -5 - Other Errors.
- Example Usage:

```
BEGIN
  -- Select an existing team
  spTeamsSelect(210, tName, isAct, jColor, v_errCode);
  IF v_errCode = 1 THEN
    DBMS_OUTPUT.PUT_LINE('Team Name: ' || tName || ', IsActive: ' || isAct || ', Jersey Color:
' || jColor);
  ELSE
    DBMS_OUTPUT.PUT_LINE('Select Error Code: ' || v_errCode);
  END IF;
  END;
```

- Expected Output: Team Name: Rangers, IsActive: 1, Jersey Color: Green

3.3 Stored Procedures for Rosters

3.3.1 spRostersInsert

- Purpose: Inserts a new record into the Rosters table with checks for existing player and team.
- Input Parameters:
  rID (Type: `rosters.rosterID%TYPE`, IN OUT): Roster ID.
  pID (Type: `rosters.playerID%TYPE`): Player ID.
  tID (Type: `rosters.teamID%TYPE`): Team ID.
  isAct (Type: `rosters.isActive%TYPE`): Active Status.
  jNum (Type: `rosters.jerseynumber%TYPE`): Jersey Number.
  codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
- Expected Outputs: Confirmation of insertion or error code.
- Error Codes:
  1 - Success.
  -1 - No Data Found (Player or Team doesn't exist).
  -3 - No Row Inserted or Duplicate value.
  -4 - Value Error.
  -5 - Other Errors.
  -6 - Invalid `isActive` value.
- Example Usage:

```
DECLARE
  rID rosters.rosterID%TYPE := 1100;
  v_errCode rosters.rosterID%TYPE;
BEGIN
  -- Insert a new roster record (assuming 1100 doesn't exist)
  spRostersInsert(rID, 963874, 221, 1, 99, v_errCode);
  DBMS_OUTPUT.PUT_LINE('Insert Error Code (New Record): ' || v_errCode);
END;
```

- Expected Output: Displays 'Insert Error Code: 1'.

3.3.2 spRostersUpdate

- Purpose: Updates an existing record in the Rosters table given the Roster ID.
- Input Parameters:
  rID (Type: `rosters.rosterID%TYPE`): Roster ID.
  pID (Type: `rosters.playerID%TYPE`): Player ID.
  tID (Type: `rosters.teamID%TYPE`): Team ID.
  isAct (Type: `rosters.isActive%TYPE`): Active Status.
  jNum (Type: `rosters.jerseynumber%TYPE`): Jersey Number.
  codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
- Expected Outputs: Confirmation of update or error code.
- Error Codes:
  1 - Success.
  -3 - No rows updated or Duplicate value.
  -4 - Value Error.
  -5 - Other Errors.
- Example Usage:

```
DECLARE
  v_errCode NUMBER;
BEGIN
  spRostersUpdate(1100, 11, 21, 0, 98, v_errCode);
  DBMS_OUTPUT.PUT_LINE('Update Error Code (Existing Record): ' || v_errCode);
END;
```

- Expected Output: Displays 'Update Error Code: 1'.

3.3.3 spRostersDelete

- Purpose: Deletes an existing record from the Rosters table given the Roster ID.
- Input Parameters:
  rID (Type: `rosters.rosterID%TYPE`): Roster ID.
  codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
- Expected Outputs: Confirmation of deletion or error code.
- Error Codes:
  1 - Success.
  -1 - No rows deleted.
  -4 - Value Error.
  -5 - Other Errors.
- Example Usage:

```
DECLARE
  v_errCode NUMBER;
BEGIN
  spRostersDelete(1100, v_errCode);
  DBMS_OUTPUT.PUT_LINE('Delete Error Code (Existing Record): ' || v_errCode);
END;
```

- Expected Output: Displays 'Delete Error Code: 1'.

3.3.4 spRostersSelect

- Purpose: Retrieves all fields in a single row from the Rosters table given a Roster ID.
- Input Parameters:
  rID (Type: `rosters.rosterID%TYPE`): Roster ID.
  pID (OUT Parameter, Type: `rosters.playerID%TYPE`): Player ID.
  tID (OUT Parameter, Type: `rosters.teamID%TYPE`): Team ID.
  isAct (OUT Parameter, Type: `rosters.isActive%TYPE`): Active Status.
  jNum (OUT Parameter, Type: `rosters.jerseynumber%TYPE`): Jersey Number.
  codeErr (OUT Parameter, Type: `NUMBER`): Error Code.
- Expected Outputs: Roster data or error code.
- Error Codes:
  1 - Success.
  -1 - No Data Found.
  -2 - Many Rows Returned.
  -4 - Value Error.
  -5 - Other Errors.
- Example Usage:

```
DECLARE
  pID NUMBER;
  tID NUMBER;
  isAct NUMBER;
  jNum NUMBER;
  v_errCode NUMBER;
BEGIN
  -- Select an existing roster (assuming roster ID 100 exists)
  spRostersSelect(100, pID, tID, isAct, jNum, v_errCode);
  IF v_errCode = 1 THEN
    DBMS_OUTPUT.PUT_LINE('Existing Roster: Player ID: ' || pID || ', Team ID: ' || tID || ',
IsActive: ' || isAct || ', Jersey Number: ' || jNum);
  ELSE
    DBMS_OUTPUT.PUT_LINE('Select Error Code (Existing Record): ' || v_errCode);
  END IF;
END;
```

- Output: Existing Roster: Player ID: 2131238, Team ID: 215, IsActive: 1, Jersey Number: 29.

4 Data Retrieval and Display

4.1 spTableNameSelectAll Procedures

4.1.1 spPlayersSelectAll

- Purpose: Outputs the contents of the entire Players table to the script window.
- Required input parameters:
  - codeErr (OUT Parameter, Type: `players.playerID%TYPE`): Error Code.
- Expected outputs: The entire Players table content displayed via `DBMS_OUTPUT`.
- Potential error codes:
1 - Success.
-5 - Other Errors.
- Example of non-saved procedural code to execute the object:

```
DECLARE
    codeErr players.playerID%TYPE;
BEGIN
    spPlayersSelectAll(codeErr);
END;
```

4.1.2 spTeamsSelectAll

- Purpose: Outputs the contents of the entire Teams table to the script window.
- Required input parameters:
  - codeErr (OUT Parameter, Type: `teams.teamID%TYPE`): Error Code.
- Expected outputs: The entire Teams table content displayed via `DBMS_OUTPUT`.
- Potential error codes:
1 - Success.
-5 - Other Errors.
- Example of non-saved procedural code to execute the object:

```
DECLARE
    codeErr teams.teamID%TYPE;
BEGIN
    spTeamsSelectAll(codeErr);
END;
```

4.1.3 spRostersSelectAll

- Purpose: Outputs the contents of the entire Rosters table to the script window.
- Required input parameters:
  - codeErr (OUT Parameter, Type: `rosters.rosterID%TYPE`): Error Code.
- Expected outputs: The entire Rosters table content displayed via `DBMS_OUTPUT`.
- Potential error codes:
1 - Success.
-5 - Other Errors.
- Example of non-saved procedural code to execute the object:

```
DECLARE
    codeErr rosters.rosterID%TYPE;
BEGIN
    spRostersSelectAll(codeErr);
END;
```

4.2 Outputting Table Contents to Script Window

4.2.1 Output of spPlayersSelectAll

374022, 38879, Clamp, Daniel, 1
375256, 82775, Reid, Sean, 1
376490, 24007, Dan , Weber, 1
.....
370320, 97528, Czajko, Mike, 1
371554, 99218, Maidment, Joel, 1
372788, 45077, Clamp, Mike, 1

4.2.2 Output of spTeamsSelectAll

210, Rangers, 1, Green
211, Bosna, 1, Pink
212, Aurora , 1, Dark Blue
214, Kickers, 1, Yellow
215, Strikers, 1, Red
216, Eagles, 1, Light Blue
218, Staffords, 1, Dark Green
220, Bankers, 1, Green
221, United, 1, Purple
222, Sonics, 1, White
223, Sonnys, 1, Gold
225, Lions, 1, BR Stripes
400, Noobs, 1, Orange

4.2.3 Output of spRostersSelectAll

1, 963874, 212, 1, 66
2, 1000894, 221, 1, 58
3, 1003362, 221, 1, 23
.....
468, 2321274, 212, 1, 9
469, 2322508, 212, 1, 27
470, 2323742, 223, 1, 52

5 Advanced Stored Procedure Operations

5.1 Retrieving Table Contents as SP Output

5.1.1 spPlayersSelectTable

- Purpose: Retrieves and outputs the entire contents of the Players table.
- Required Input Parameters:
  `c` (OUT Parameter, Type: SYS_REFCURSOR): A cursor to store and return the query results.
  `codeErr` (OUT Parameter, Type: `players.playerID%TYPE`): Error code indicating the status of the procedure execution.
- Expected Outputs: A cursor containing all records from the Players table.
- Potential Error Codes:
  1: Success.
  -4: Value Error.
  -5: Other Errors.
  -6: Invalid Cursor.
- Example of Non-saved Procedural Code:

```
DECLARE
  c SYS_REFCURSOR;
  player players%ROWTYPE;
  codeErr players.playerID%TYPE;
BEGIN
  spPlayersSelectTable(c, codeErr);

  DBMS_OUTPUT.PUT_LINE('Player ID | Registration Number | Last Name | First Name | Is Active');
  DBMS_OUTPUT.PUT_LINE('----------+--------------------+-----------+------------+----------
');

  LOOP
      FETCH c INTO player;
      EXIT WHEN c%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(
          LPAD(player.playerID, 9) || ' | ' ||
          LPAD(player.regNumber, 18) || ' | ' ||
          LPAD(player.lastName, 9) || ' | ' ||
          LPAD(player.firstName, 10) || ' | ' ||
          LPAD(player.isActive, 9)
      );
  END LOOP;
  CLOSE c;

  IF codeErr != 1 THEN
    DBMS_OUTPUT.PUT_LINE('Error occurred with code: ' || codeErr);
  END IF;
END;
```

5.1.2 spTeamsSelectTable

-Purpose Retrieves and outputs the entire contents of the Teams table.
- Required Input Parameters:
  `c` (OUT Parameter, Type: SYS_REFCURSOR).
  `codeErr` (OUT Parameter, Type: `teams.teamID%TYPE`).

- Expected Outputs: A cursor with all records from the Teams table.
- Potential Error Codes:
  1: Success.
  -4: Value Error.
  -5: Other Errors.
  -6: Invalid Cursor.
- Example of Non-saved Procedural Code:

```
DECLARE
  c SYS_REFCURSOR;
  team teams%ROWTYPE;
  codeErr teams.teamID%TYPE;
BEGIN
  spTeamsSelectTable(c, codeErr);

  DBMS_OUTPUT.PUT_LINE('Team ID | Team Name        | Is Active | Jersey Colour');
  DBMS_OUTPUT.PUT_LINE('--------+------------------+-----------+--------------');

  LOOP
      FETCH c INTO team;
      EXIT WHEN c%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(
          LPAD(team.teamID, 7) || ' | ' ||
          RPAD(team.teamName, 16) || ' | ' ||
          LPAD(team.isActive, 9) || ' | ' ||
          RPAD(team.jerseycolour, 13)
      );
  END LOOP;
  CLOSE c;

  IF codeErr != 1 THEN
    DBMS_OUTPUT.PUT_LINE('Error occurred with code: ' || codeErr);
  END IF;
END;
```

5.1.3 spRostersSelectTable

-Purpose Retrieves and outputs the entire contents of the Rosters table.
-Required Input Parameters:
  `c` (OUT Parameter, Type: SYS_REFCURSOR).
  `codeErr` (OUT Parameter, Type: `rosters.rosterID%TYPE`).
- Expected Outputs: A cursor with all records from the Rosters table.
- Potential Error Codes:
  1: Success.
  -4: Value Error.
  -5: Other Errors.
  -6: Invalid Cursor.
- Example of Non-saved Procedural Code:

```
  DECLARE
  c SYS_REFCURSOR;
  roster rosters%ROWTYPE;
  codeErr rosters.rosterID%TYPE;
BEGIN
  spRostersSelectTable(c, codeErr);

  DBMS_OUTPUT.PUT_LINE('Roster ID | Player ID | Team ID | Is Active | Jersey Number');
  DBMS_OUTPUT.PUT_LINE('----------+-----------+---------+-----------+---------------');

  LOOP
      FETCH c INTO roster;
      EXIT WHEN c%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(
          LPAD(roster.rosterID, 9) || ' | ' ||
          LPAD(roster.playerID, 9) || ' | ' ||
          LPAD(roster.teamID, 7) || ' | ' ||
          LPAD(roster.isActive, 9) || ' | ' ||
          RPAD(roster.jerseynumber, 15)
      );
  END LOOP;
  CLOSE c;

  IF codeErr != 1 THEN
    DBMS_OUTPUT.PUT_LINE('Error occurred with code: ' || codeErr);
  END IF;
END;
```

5.2 Output of Procedures

5.2.1 Output of spPlayersSelectTable

Player ID	Registration Number	Last Name	First Name	Is Active
374022	38879	Clamp	Daniel	1
375256	82775	Reid	Sean	1
376490	24007	Dan	Weber	1
377724	37235	Celli	Lou	1
378958	26890	Campbell	Jason	1
381426	59361	Callaghan	Johnston	1
382660	53643	BOURGOIN	Rino	1
383894	23219	DICKERT	Jeffery	1
385128	47054	Hughes	Dave	1
386362	78929	SCHALM	Andrew	1
387596	34329	Alves	Anthony	1
388830	90006	Balbach	Frank	1
390064	25793	Caceros	Byron	1
391298	41477	Martinez	Andy	1
392532	15044	McDermott	Ian	1
393766	50641	McDonald	John	1
395000	10713	Caal	Juan	1
396234	69622	Rakanovic	Michael	1

Figure 1 One part of spPlayersSelectTable's output

5.2.2 Output of spTeamsSelectTable

Team ID	Team Name	Is Active	Jersey Colour
210	Rangers	1	Green
211	Bosna	1	Pink
212	Aurora	1	Dark Blue
214	Kickers	1	Yellow
215	Strikers	1	Red
216	Eagles	1	Light Blue
218	Staffords	1	Dark Green
220	Bankers	1	Green
221	United	1	Purple
222	Sonics	1	White
223	Sonnys	1	Gold
225	Lions	1	BR Stripes
400	Noobs	1	Orange

Figure 2 Output of spTeamsSelectTable

5.2.3 Output of spRostersSelectTable

Roster ID	Player ID	Team ID	Is Active	Jersey Number
241	963874	212	1	66
242	1000894	221	1	58
243	1003362	221	1	23
244	1019404	214	1	9
245	1032978	214	1	48
246	1341478	221	1	56
247	1401944	214	1	2
248	1585810	218	1	67
249	1589512	218	1	10
250	1599384	218	1	69
251	1600618	218	1	55
252	1726486	218	1	79
253	1743762	214	1	13
254	1746230	221	1	31
255	1761038	216	1	30
256	1809164	215	1	21

Figure 3 One part of spRosterSelectTable's output

6 Creation and Utilization of Views

6.1 vwPlayerRosters View

vwPlayerRosters View

- Purpose: This view consolidates player and team information, showing players on teams, including all relevant details from the players, rosters, and teams tables.
- View Structure:
  - Fields: `playerID`, `firstname`, `lastname`, `regnumber`, `playerActive`, `rosterID`, `jerseynumber`, `rosterActive`, `teamID`, `teamname`, `jerseycolour`, `teamActive`.
  - Joins: The view joins the `players`, `rosters`, and `teams` tables.
  - Criteria: Includes records with exact matches across these tables.
- Example Query:

```
SELECT
    vw.playerID,
    vw.firstname,
    vw.lastname,
    vw.regnumber,
    vw.teamname
FROM
    vwPlayerRosters vw
WHERE
    vw.teamname LIKE '%Aurora%';
```

- Output:

	PLAYERID	FIRSTNAME	LASTNAME	REGNUMBER	TEAMNAME
1	963874	Marc	Hodges	37888	Aurora
2	963874	Marc	Hodges	37888	Aurora
3	2017710	Dave	Portela	77319	Aurora
4	2017710	Dave	Portela	77319	Aurora
5	2020178	Kevin	Andrade	54901	Aurora
6	2020178	Kevin	Andrade	54901	Aurora
7	2021412	Ryan	Pereira	74866	Aurora
8	2021412	Ryan	Pereira	74866	Aurora
9	2025114	Luca	Desantis	36960	Aurora
10	2025114	Luca	Desantis	36960	Aurora

Figure 4 One part of vwPlayerRosters's output

6.2 vwTeamsNumPlayers View

vwTeamsNumPlayers View

- Purpose: This view presents the number of players currently registered in each team.
- View Structure:
  - Fields: `teamID`, `numOfPlayers` (the count of players in each team).
  - Grouping: Data is grouped by `teamID`.
- Example Query:

```
SELECT *
FROM vwTeamsNumPlayers;
```

- Output:

	TEAMID	NUMOFPLAYERS
1	210	36
2	211	34
3	212	30
4	214	32
5	215	32
6	216	50
7	218	28
8	220	40
9	221	40
10	222	42
11	223	50
12	225	46

Figure 5 TeamNumPlayers's output

6.3 vwSchedule View

-Purpose: Displays game schedules, including team names and location names, alongside other game details.

- View Structure:
  - Fields: `gameid`, `gamenum`, `gamedatetime`, `homeTeamID`, `homeTeamName`, `homescore`, `visitTeamID`, `visitTeamName`, `visitscore`, `locationid`, `locationname`, `isplayed`, `notes`.
  - Joins: Combines data from the `games`, `teams`, and `sllocations` tables.
- Example Query:

```
SELECT gameid,
       gamenum,
       gamedatetime,
       hometeamid,
       locationname
FROM   vwschedule;
```

- Output:

	GAMEID	GAMENUM	GAMEDATETIME	HOMETEAMID	LOCATIONNAME
1	151	53	23-11-23	220	Bechtel 1
2	51	53	23-10-30	220	Bechtel 1
3	113	13	23-09-02	220	Bechtel 2
4	123	25	23-09-13	220	Bechtel 2
5	126	28	23-09-20	220	Bechtel 2
6	136	38	23-10-04	220	Bechtel 2
7	148	50	23-10-18	220	Bechtel 2
8	161	63	23-11-07	220	Bechtel 2
9	177	79	23-12-05	220	Bechtel 2
10	187	89	23-12-19	220	Bechtel 2

Figure 6 One part of vwschedule's output

7 Team-Specific Stored Procedures

7.1 spTeamRosterByID

Version 1: Using SYS_REFCURSOR
- Flexibility: Better for scenarios where data needs to be processed outside the procedure.
- Usage: Ideal for applications that can handle cursors.
- Complexity: Slightly more complex but offers more versatility.

Version 2: Using Explicit Cursor
- Simplicity: Easier to implement and understand, especially for direct SQL users.

- Self-contained: All processing and outputting are done within the procedure.
- Flexibility: Less flexible in terms of external data manipulation.

Expected Outputs: A cursor containing the roster details of the specified team.

- Potential Error Codes:
    1: Success.
    -1: No data found.
    -2: More than one row found.
    -4: Value Error.
    -5: Other Errors.
    -6: Invalid Cursor.
- Example of Non-saved Procedural Code:

Version 1

```
DECLARE
    myCursor SYS_REFCURSOR;
    rosterRecord vwPlayerRosters%ROWTYPE;
    v_errCode NUMBER;
BEGIN
    spTeamRosterByID(215, myCursor, v_errCode);

    IF v_errCode = 1 THEN
        LOOP
            FETCH myCursor INTO rosterRecord;
            EXIT WHEN myCursor%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE('Player  Name:  '  ||  rosterRecord.firstname  ||  '  '  ||
rosterRecord.lastname ||
                                 ', Team Name: ' || rosterRecord.teamname || ', Jersey Colour:
' || rosterRecord.jerseycolour ||
                                 ', Jersey Number: ' || rosterRecord.jerseynumber);
        END LOOP;
        CLOSE myCursor;
    ELSE
        DBMS_OUTPUT.PUT_LINE('Error Code: ' || v_errCode);
    END IF;
END;
```

Version 2

```
DECLARE
    v_errCode NUMBER;
BEGIN
    spTeamRosterByID(215, v_errCode);  -- Replace 215 to ...
    IF v_errCode != 1 THEN
        DBMS_OUTPUT.PUT_LINE('Error Code: ' || v_errCode);
    END IF;
END;
```

- Output:

```
Player Name: Wesley Thomson, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 21
Player Name: Wesley Thomson, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 21
Player Name: Hugo Filipe, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 90
Player Name: Hugo Filipe, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 90
Player Name: Cory Moore, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 57
Player Name: Cory Moore, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 57
Player Name: Nick Pecarski, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 87
Player Name: Nick Pecarski, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 87
Player Name: Steven Schaefer, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 24
Player Name: Steven Schaefer, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 24
Player Name: Brandon Kohler, Team Name: Strikers, Jersey Colour: Red, Jersey Number: 45
```

Figure 7 One part of spTeamRosterByID's output

7.2 spTeamRosterByName

- Purpose: To retrieve and display the roster details for teams based on a search string that partially or fully matches the team name.
- Input Parameters:
  - `searchStr` (Type: VARCHAR2): A string parameter to search within the team names.
  - `mycursor` (OUT, Type: SYS_REFCURSOR): A cursor to store and return the query results.
  - `codeErr` (OUT, Type: NUMBER): Error code for the procedure execution status.
- Functionality:
  - The procedure searches the `vwPlayerRosters` view for teams whose names match the `searchStr`.
  - The search is case-insensitive and can match any part of the team name.
  - It returns the results through the `mycursor` for further processing or display.
- Error Handling:
  - Manages invalid cursors, value errors, and other unspecified exceptions.
- Example Usage:

```
DECLARE
    mycursor SYS_REFCURSOR;
    rosterRow vwPlayerRosters%ROWTYPE;
    searchStr VARCHAR2(100);
    codeErr NUMBER;
BEGIN
    searchStr := '&searchStr';  -- Prompt for the search string input
    spTeamRosterByName(searchStr, mycursor, codeErr);

    -- Print header
    DBMS_OUTPUT.PUT_LINE('-----------------------------------------');
    DBMS_OUTPUT.PUT_LINE('Player Name           Team Name        Jersey');
    DBMS_OUTPUT.PUT_LINE('-----------------------------------------');

    LOOP
        FETCH mycursor INTO rosterRow;
        EXIT WHEN mycursor%NOTFOUND;
        -- Format the output
        DBMS_OUTPUT.PUT_LINE(
            RPAD(rosterRow.firstname || ' ' || rosterRow.lastname, 23) ||
            RPAD(rosterRow.teamname, 15) ||
            rosterRow.jerseycolour || ' ' || rosterRow.jerseynumber
        );
    END LOOP;

    -- Print footer
```

```
    DBMS_OUTPUT.PUT_LINE('-----------------------------------------------');

    CLOSE mycursor;

    EXCEPTION
        WHEN OTHERS
        THEN codeErr := -5;
END;
```

Output: when input “Bosna”

Player Name	Team Name	Jersey
Cory Aarons	Bosna	Pink 94
Cory Aarons	Bosna	Pink 94
Emre Arif	Bosna	Pink 52
Emre Arif	Bosna	Pink 52
Mirnes Bilalic	Bosna	Pink 88
Mirnes Bilalic	Bosna	Pink 88
Brandon DeSerpa	Bosna	Pink 99
Brandon DeSerpa	Bosna	Pink 99
Reuf Fazlic	Bosna	Pink 34
Reuf Fazlic	Bosna	Pink 34
Damir Hadziavdic	Bosna	Pink 59
Damir Hadziavdic	Bosna	Pink 59
Admir Hodzic	Bosna	Pink 19

Figure 8 One part of spTeamRosterByName's output with "Bosna"

8 User-Defined Function

8.1 fncNumPlayersByTeamID Function

- Purpose: To return the count of players currently registered in a team, identified by the team's primary key (team ID).
- Function Signature:
  - Input Parameter: `tID` (Type: `teams.teamID%TYPE`): The ID of the team.
  - Return Type: `NUMBER`: The number of players in the specified team.
- Functionality:
  - Queries the `vwTeamsNumPlayers` view to find the number of players associated with the given team ID.
  - Returns the number of players.
- Error Handling:
  - Returns specific error codes for different scenarios:
    - -1: No data found.
    - -2: More than one row found (unlikely in this context, but included for completeness).
    - -4: Value error.
    - -5: Other unspecified errors.
- Example Usage:

```
DECLARE
    teamID teams.teamID%TYPE := 215; -- Replace with the desired team ID
    numPlayers NUMBER;
    codeErr NUMBER;
BEGIN
    numPlayers := fncNumPlayersByTeamID(teamID);

    -- Check for specific error conditions
    IF numPlayers < 0 THEN
        DBMS_OUTPUT.PUT_LINE('Error code: ' || numPlayers);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Number of players for Team ' || teamID || ': ' || numPlayers);
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred with code: ' || codeErr);
END;
```

- Output: Number of players for Team 215: 32

9 Scheduling Stored Procedures

9.1 spSchedUpcomingGames

- Purpose: Displays the schedule of games to be played in the next 'n' days.
- Input Parameters:
  - `n` (Type: NUMBER): Number of days into the future to display games.
  - `codeErr` (OUT, Type: NUMBER): Error code.
- Functionality:
  - Uses the `vwSchedule` view to fetch game details scheduled between the current date (`SYSDATE`) and 'n' days into the future.

- Outputs game details such as game ID, teams, scores, and location.
- Error Handling:
  - Manages value errors and other exceptions.
- Example Usage:

```
  DECLARE
    codeErr NUMBER;
BEGIN
    spSchedUpcomingGames(20, codeErr);
    IF codeErr = 1 THEN
        DBMS_OUTPUT.PUT_LINE('Procedure executed successfully.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Error occurred with code: ' || codeErr);
    END IF;
END;
```

- Output:

```
Game ID: 177
Game Number: 79
Game Date and Time: 12/05/2023 15:27:00
Home Team: Bankers
Home Team Score: 0
Visiting Team: Staffords
Visiting Team Score: 0
Location: Bechtel 2
Is Played: 0
Notes:
------------------
Game ID: 177
Game Number: 79
Game Date and Time: 12/05/2023 15:27:00
Home Team: Bankers
Home Team Score: 0
Visiting Team: Staffords
Visiting Team Score: 0
Location: Bechtel 2
Is Played: 0
Notes:
------------------
```

Figure 9 One part of spSchedUpcomingGames's with next 20 day

9.2 spSchedPastGames

- Purpose: Displays the schedule of games that were played in the past 'n' days.
- Input Parameters:
  - `n` (Type: NUMBER): Number of past days to display games.
  - `codeErr` (OUT, Type: NUMBER).
- Functionality:
  - Fetches game details from the `vwSchedule` view for games played between 'n' days ago and the current date.
  - Outputs similar game details as `spSchedUpcomingGames`.
- Error Handling:
  - Handles value errors and other exceptions.
- Example Usage:

```
  DECLARE
    codeErr NUMBER;
BEGIN
    spSchedPastGames(7, codeErr); -- Display games played in the past 7 days
    IF codeErr = 1 THEN
        DBMS_OUTPUT.PUT_LINE('Procedure executed successfully.');
    ELSE
```

```
        DBMS_OUTPUT.PUT_LINE('Error occurred with code: ' || codeErr);
    END IF;
END;
```

- Output:

```
Game ID: 151
Game Number: 53
Game Date and Time: 11/23/2023 15:26:59
Home Team: Bankers
Home Team Score: 0
Visiting Team: Sonics
Visiting Team Score: 0
Location: Bechtel 1
Is Played: 0
Notes:
--------------------
Game ID: 151
Game Number: 53
Game Date and Time: 11/23/2023 15:26:59
Home Team: Bankers
Home Team Score: 0
Visiting Team: Sonics
Visiting Team Score: 0
Location: Bechtel 1
Is Played: 0
Notes:
--------------------
```

Figure 10 One part of spSchedPastGames's output with 7 days

10 Standings Management

10.1 spRunStandings Stored Procedure

- Create the `tempStandings` Table: Before we proceed with the `spRunStandings` stored procedure, the first essential step is to create the `tempStandings` table. This table is crucial for holding the temporary standings data that our stored procedure will populate and update.

```
CREATE TABLE tempStandings (
    TheTeamID NUMBER,
    teamname VARCHAR2(255),
    GP NUMBER,
    W NUMBER,
    L NUMBER,
    T NUMBER,
    Pts NUMBER,
    GF NUMBER,
    GA NUMBER,
    GD NUMBER
);
```

- How It Works:

1. Truncating Existing Data: Initially, we truncate the `tempStandings` table to remove any old data. This ensures the table only contains the most current standings.
2. Calculating Standings: Our procedure then populates the `tempStandings` table with new data. It calculates the standings by aggregating data from the `games` table.
   The calculations include the total number of games played (GP), wins (W), losses (L), ties (T), points (Pts), goals for (GF), goals against (GA), and goal difference (GD) for each team.
   These calculations are based on the outcomes of games that have been played (`isPlayed = 1`).

3. Handling Multiple Perspectives: We include calculations from both the perspective of the home team and the visiting team to ensure comprehensive standings.
4. Error Handling: Our procedure is equipped with an exception block to handle various exceptions, including value errors and other unexpected issues.

10.2 Automated Trigger for Standings Update

- Purpose: To ensure that our `spRunStandings` stored procedure is automatically executed whenever there's an update in the `games` table. This automation keeps the standings up-to-date.
-Trigger Syntax:

```
CREATE OR REPLACE TRIGGER trgUpdateStandings
AFTER INSERT OR UPDATE OF homescore, isPlayed, visitScore ON games
FOR EACH ROW
BEGIN
    spRunStandings;
END;
```

- Steps for Implementation:
1. Create the `tempStandings` Table: We start by executing the CREATE TABLE statement for `tempStandings`.
2. Create the `spRunStandings` Stored Procedure: With the table in place, we then create the `spRunStandings` procedure to update the standings.
3. Create the Trigger `trgUpdateStandings`: Finally, we establish the trigger `trgUpdateStandings` to ensure automatic updates to the standings.

11 Creative Implementation

11.1 Design and Development of a Unique Database Object

12 Conclusion

12.1 Summary of Key Achievements

12.2 Future Prospects and Improvements

13 References