**Task1**

**Conceptual Design through FTR**

Aim:

Using basic database design methodology and ER modeler, design Entity Relationship Diagram by satisfying the following sub tasks:

* 1. **a** Identifying the entities.

1. **b** Identifying the attributes.

**1. c** Identification of relationships, cardinality, type of relationship.

**1. d** Reframing the relations with keys and constraint.

1. **e** Using create, develop ER/ER diagram

**1.a Identifying the entities**

1.a.1 CricketBoard

1.a.2 Team

1.a.3 Player

1.a.4 Match

1.a.5 Ground

1.a.6 Umpire

**1.b Identifyingtheattributes**

1.b.1 CricketBoard(BoardID, Name, Address, Contact\_No)

1.b.2 Team(TeamID, Name, Coach, Captain)

1.b.3 Player(PlayerID, FName, LName, Age, DateofBirth, PlayingRole, email, contact\_no)

1.b.4 Match( MatchID, Date, Time, Result)

1.b.5 Ground(GroundID, Name, Location, Capacity)

# 1.b.6 Umpire(UmpireID, FName, LName, Age, DateofBirth, Country, email, contact\_no)

# 1.c Identification of relationships, cardinality, type of relationship.

1.c.1 **Board-Team Relationship:** The Board will have a **one-to-many** relationship with Teams since the board can have multiple teams affiliated with it, but a team can only be associated with one board.

1.c.2 **Team-Player Relationship:** Teams and Players will have a **many-to-many** relationship since a team can have multiple players, and a player can be a part of multiple teams over time.

1.c,3. **Match-Team Relationship:** Matches will have a **many-to-many** relationship with Teams, as a match involves two teams, and a team can participate in multiple matches.

1.c.4. **Match-Ground Relationship:** Matches will have a **one-to-one** relationship with Grounds, as each match takes place in one specific ground.

**1.d Reframing the relations with keys and constraint**

**1.d.1 Create Table CricketBoard:**

**SQL>create table CricketBoard(**BoardID varchar(10) PRIMARY KEY, Name varchar(30), Address varchar(50), Contact\_No number);

Table Created

SQL>DESC CricketBoard

|  |  |  |
| --- | --- | --- |
| Column | NULL | TYPE |
| BoardID | NOT NULL | VARCHAR(10) |
| Name | - | VARCHAR(30) |
| Address | - | VARCHAR(50) |
| Contact\_No | - | NUMBER |

**1.d.2 Create Table Team:**

**SQL> create table Team(TeamID varchar(6) PRIMARY KEY, BoardID varchar(10), Name varchar(30), Coach varchar(30), Captain varchar(30), FOREIGN KEY(BoardID) REFERENCES CricketBoard(BoardID));**

**Table created.**

**SQL> DESC TEAM**

Name Null? Type

----------------------------------------- -------- ----------------------------

TEAMID NOT NULL VARCHAR2(6)

BOARDID NOT NULL VARCHAR2(10)

NAME - VARCHAR2(30)

COACH - VARCHAR2(30)

CAPTAIN - VARCHAR2(30)

**1.d.3 Create Table Player:**

SQL> CREATE table Player(PlayerID varchar(6) PRIMARY KEY, TeamID varchar(6), FName varchar(30), LName varchar(30), Age number(5,2), DateofBirth date, PlayingRole varchar(25), email varchar(40), contact\_no number, FOREIGN KEY(TeamID) REFERENCES Team(TeamID));

Table created.

SQL> DESC PLAYER

Name Null? Type

----------------------------------------- -------- ----------------------------

PLAYERID NOT NULL VARCHAR2(6)

TEAMID NOT NULL VARCHAR2(6)

FNAME VARCHAR2(30)

LNAME VARCHAR2(30)

AGE NUMBER(5,2)

DATEOFBIRTH DATE

PLAYINGROLE VARCHAR2(25)

EMAIL VARCHAR2(40)

CONTACT\_NO NUMBER

**1.d.4 Create Table Match:**

SQL> create table Match( MatchID varchar(10), TeamID1 varchar(6), TeamID2 varchar(6), PlayerID varchar(6), Match\_Date date, Time1 number, Result varchar(20), PRIMARY KEY(MatchID,PlayerID), FOREIGN KEY(TeamID1) REFERENCES team(TeamID), FOREIGN KEY(TeamID2) REFERENCES team(TeamID), FOREIGN KEY(PlayerID) REFERENCES Player(PLayerID));

Table created.

SQL> DESC Match

Name Null? Type

----------------------------------------- -------- ----------------------------

MATCHID NOT NULL VARCHAR2(10)

TEAMID1 NOT NULL VARCHAR2(6)

TEAMID2 NOT NULL VARCHAR2(6)

PLAYERID NOT NULL VARCHAR2(6)

MATCH\_DATE DATE

TIME1 NUMBER

RESULT VARCHAR2(20)

**1.d.5 Create Table Ground:**

SQL> create table Ground(GroundID varchar(10) PRIMARY KEY, MatchID Varchar(10), Name varchar(30), Location varchar(30), Capacity number, FOREIGN KEY(MatchID) REFERENCES Match(MatchID));

Table created.

SQL> DESC Ground

Name Null? Type

----------------------------------------- -------- ----------------------------

GROUNDID NOT NULL VARCHAR2(10)

MATCHID NOT NULL VARCHAR2(10)

NAME VARCHAR2(30)

LOCATION VARCHAR2(30)

CAPACITY NUMBER

**1.d.6 Create Table Umpire:**

**SQL> Create Table Umpire(UmpireID varchar(10) PRIMARY KEY, FName varchar(30), LName varchar(30), Age number(5,2), DateofBirth date, Country varchar(30), email varchar(40), contact\_no number);**

SQL> DESC Umpire

Name Null? Type

----------------------------------------- -------- ----------------------------

UMPIREID NOT NULL VARCHAR2(10)

FNAME VARCHAR2(30)

LNAME VARCHAR2(30)

AGE NUMBER(5,2)

DATEOFBIRTH DATE

COUNTRY VARCHAR2(30)

EMAIL VARCHAR2(40)

CONTACT\_NO NUMBER

**1.d.6 Create Table Umpire\_Umpired:**

SQL> create table Umpire\_Umpired(UmpireID varchar(10), MatchID Varchar(10), GroundID varchar(10), FOREIGN KEY(UmpireID) REFERENCES Umpire(UmpireID), FOREIGN KEY(MatchID) REFERENCES Match(MatchID), FOREIGN KEY(GroundID) REFERENCES Ground(GroundID));

Table created.

SQL> DESC Umpire

Name Null? Type

----------------------------------------- -------- ----------------------------

UMPIREID NOT NULL VARCHAR2(10)

GROUNDID NOT NULL VARCHAR2(10)

MATCHID NOT NULL VARCHAR2(10)

**Result:**

Thus the database design methodology and ER Model design diagram has been completed successfully.

**TASK2**

**Generating Design of other traditional database model**

**Aim:**

Creating Hierarchical/Network model of the database by enhancing the sound abstract data by performing following tasks using forms of inheritance:

2. a Identify the specificity of each relationship, find and form surplus relations.

2. b Check is-a hierarchy/has-a hierarchy and performs generalization and/or specialization relationship.

2. c Find the domain of the attribute and perform check constraint to the applicable.

2. d Rename the relations.

2. e Perform SQL Relations using DDL, DCL commands.

**2. a Identify the specificity of each relationship, find and form surplus relations.**

**Relationship: Cricket Board manages Team (one-to-many)**

Specificity: One Cricket Board manages one or more Teams, but each Team is managed by only one Cricket Board.

Surplus Relation: No surplus relation is needed for this relationship since it is already one-to-many.

**Relationship: Team has Player (many-to-one)**

Specificity: One Team can have many Players, but each Player belongs to only one Team.

Surplus Relation: No surplus relation is needed for this relationship since it is already many-to-one.

**Relationship: Match involves Team (many-to-many)**

Specificity: One Match involves two Teams, and each Team can participate in multiple Matches.

Surplus Relation: No surplus relation is needed for this relationship since it is already many-to-many.

**Relationship: Match has Umpire (many-to-many)**

Specificity: One Match can have multiple Umpires, and each Umpire can officiate multiple Matches.

Surplus Relation: No surplus relation is needed for this relationship since it is already many-to-many.

Based on the specificity analysis, all the relationships in the ER diagram are appropriately represented, and there are no surplus relations required for this particular model. Each relationship reflects the correct cardinality and participation constraints as per the description provided earlier.

**2.b Check is-a hierarchy/has -a hierarchy and performs generalization and/or specialization relationship.**

Generalization

In the ER diagram for the Tamil Nadu Cricket Board (TNCA) described earlier, we can identify potential generalizations based on common attributes or relationships among entities. Here's an example of a possible generalization:

**Entities:**

Player

Umpire

**Attributes:**

The above entities have common attributes like First\_Name, Last\_Name, Date\_of\_Birth, age, Contact\_No, and Email.

**Potential Generalization:**

Create a superclass called "Person" to represent the common attributes shared by Player and Umpire. The "Person" entity would have the following attributes:

Person\_ID (primary key)

First\_Name

Last\_Name

Date\_of\_Birth

Age

Contact\_Number

Email

**Subclasses:**

Player: Inherited attributes from "Person" and add specific attributes like Player\_ID.

Umpire: Inherited attributes from "Person" and add specific attributes like Umpire\_ID.

Umpire

Umpire\_ID, FName, LName, Age, DateofBirth, email, contact\_no

Player

Player\_ID, FName, LName, Age, DateofBirth, email, contact\_no

Person

Person\_ID,

FName, LName, Age, DateofBirth, email, contact\_no,

Role

By using generalization, we can reduce data redundancy, improve data integrity, and simplify the structure of the ER diagram. This approach also allows for easier maintenance and updates, as changes made to the attributes shared by all "Person" entities will be automatically reflected in the subclasses.

**Specialization**

In the context of Entity-Relationship (ER) diagrams, specialization refers to the process of defining subtypes within an entity type. It allows, to represent entities that have specific attributes or relationships distinct from the general attributes or relationships of the parent entity.

In the case of the Tamil Nadu Cricket Board Association, let's consider the specialization of the "Player" entity into two subtypes: "Batsman" and "Bowler." This specialization is based on the specific roles that players can have in cricket.

Here's the modified ER diagram with the specialization:

Player

PlayerID, FName, LName, Age, DateofBirth, email, contact\_no

Bowler

Bowler\_ID

Bowling\_avg

Batsman

Batsman\_ID

Batting\_avg

**2. c Find the domain of the attribute and perform check constraint to the applicable.**

For the purpose of illustration, I'll assume we are considering the "age" attribute of the "Player" entity from the ER diagram of the Tamil Nadu Cricket Association.

Finding the domain of the "age" attribute:

The "age" attribute typically represents the age of a player, and it should be a positive integer or a non-negative integer depending on how you handle the birth dates of players. For the sake of simplicity, let's assume it's a positive integer.

Check constraint to enforce the domain:

To enforce the domain on the "age" attribute and ensure that only valid values are allowed, we can create a check constraint in the database schema. The check constraint will specify the condition that the "age" attribute must satisfy.

Suppose your database schema language is SQL, here's an example of how you can add the check constraint:

**SQL> ALTER TABLE Player ADD CONSTRAINT check\_con CHECK (age>= 20);**

**Table altered.**

**2.d** Rename the relations:

Renaming a table (relation) in SQL can be accomplished using the ALTER TABLE statement with the RENAME TO clause. The specific syntax for renaming tables varies slightly between different database management systems.

Here's the syntax for renaming a column in the Table:

SQL> Alter table Umpire RENAME column contact\_no TO phone\_no;

Table altered.

SQL> DESC Umpire

Name Null? Type

----------------------------------------- -------- ----------------------------

UMPIREID VARCHAR2(10)

FNAME VARCHAR2(30)

LNAME VARCHAR2(30)

AGE NUMBER(5,2)

DATEOFBIRTH DATE

COUNTRY VARCHAR2(30)

EMAIL VARCHAR2(40)

PHONE\_NO NUMBER

**2.e** Perform SQL Relations using DDL, DCL commands.

DCL stands for "Data Control Language," which is a subset of SQL (Structured Query Language) used to control access to data in a database. DCL commands are responsible for managing user permissions, granting privileges, and controlling data security within a database system. There are two primary DCL commands:

1. Grant
2. Revoke

**GRANT:**

The GRANT command is used to provide specific privileges to users or roles, allowing them to perform certain actions on database objects (e.g., tables, views, procedures). Privileges may include SELECT, INSERT, UPDATE, DELETE, EXECUTE, and more.

**SQL> create user Raj identified by kumar;**

User created.

SQL> grant resource to raj;

Grant succeeded.

SQL> grant create session to raj;

Grant succeeded.

SQL> conn

Enter user-name: raj

Enter password:

Connected.

SQL> create table emp(eno number,ename varchar(10));

Table created.

SQL> conn system/manager

Connected.

SQL> grant all on Umpire to Raj;

Grant succeeded.

**Result:**

Thus the Hierarchical model and Network model has been successfully created.

**TASK 3**

**Using Clauses, Operators and Functions in queries**

Aim:

To perform the query processing on databases for different retrieval results of queries using DML, DRL operations using aggregate, date, string, indent functions, set clauses and operators.

* To retrieve the location of a particular match conducted by its MatchID
* To retrieve the Players detail whose name start with ‘A’.
* Add a column Batting and Bowling in Player table.
* To count the number of right-hand batsman in a team.
* To display the CricketBoard details for the BoardIDs 'BID01', 'BID03', and 'BID06'.
* To select the names and IDs of players who are left-hand bowlers.
* To find the UmpireID of umpires who have not umpired any match.

**CricketBoard:**

|  |  |  |  |
| --- | --- | --- | --- |
| **BoardID** | **Name** | **Address** | **Contact\_No** |
| BID01 | Chennai Cricket Board | Chennai | 9988776699 |
| BID02 | Tiruvallur Cricket Board | Chennai | 9977886699 |
| BID03 | Viluppuram Cricket Board | Viluppuram | 9966886699 |
| BID04 | Trichy Cricket Board | Trichy | 9955886699 |
| BID05 | Madurai Cricket Board | Madurai | 9944886699 |
| BID06 | Tuticorin Cricket Board | Tuticorin | 9933886699 |
| BID07 | Selam Cricket Board | Selam | 9922886699 |
| BID08 | Tiruppur Cricket Board | Tiruppur | 9911886699 |

**Team:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TeamID** | **BoardID** | **Name** | **Coach** | **Captain** |
| CCB01 | BID01 | ABS EXPRESS | G.D.RAMESH | SAMPATH KUMAR |
| CCB02 | BID01 | AVG EXPRESS | T.KARTHIK | Y.JOHN |
| TCB01 | BID02 | ANGRY BARD | TOM BABU | CINIL JOHN |
| TCB02 | BID02 | TIGER ROCK | S.KANNAN | BEN GEORGE |
| TRICB01 | BID04 | ROCK | K.PAUL | K.MUTHU |
| VCB01 | BID03 | RAINBOW | S.RAJESHKUMAR | MANIMARAN |
| MCB01 | BID05 | PANTHER | SARAVANAN | R.SUNILKUMAR |
| TUCB01 | BID06 | THUNDER | D ALEX | BARATHI |
| SCB01 | BID07 | EAGLE | SOMU | SRI HARI |
| TICB01 | BID08 | KINGS | D ANAND | MATHAN |

**Player:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PlayerID** | **TeamID** | **FName** | **LName** | **Age** | **DateofBirth** | **PlayingRole** | **email** | **contact\_no** | **Batting** | **Bowling** |
| 1 | CCB01 | Raj | N | 27 | 29-JUN-1996 | Bowler | rajn@gmail.com | 9191910101 | null | left-hand |
| 33 | CCB01 | Balaji | D | 23 | 02-JAN-1999 | Batsman | balajid@gmail.com | 9191910031 | right-hand | *Null* |
| 02 | CCB02 | Krishna | R | 23 | 02-JAN-1999 | Bowler | krishnar@gmail.com | 9191930103 | *null* | right-hand |
| 18 | CCB02 | Kishore | K | 24 | 02-SEP-1998 | ALL ROUNDER | kishorek@gmail.com | 9291930105 | left-hand | left-hand |
| 19 | TCB01 | Karthick | K | 24 | 14-SEP-1998 | Batsman | karthickk@gmail.com | 9292930107 | right-hand | *Null* |
| 62 | TCB01 | Amar | J | 22 | 21-SEP-1998 | Batsman | Amarj@gmail.com | 9292930508 | right-hand | *Null* |
| 102 | TCB02 | Akash | G | 21 | 26-SEP-1999 | Batsman | Amarj@gmail.com | 9292930510 | right-hand | *Null* |
| 12 | TCB02 | Premkumar | S | 21 | 13-OCT-1999 | Bowler | Premkumars@gmail.com | 9592930517 | *null* | right-hand |
| 01 | VCB02 | Prem | V | 23 | 13-APR-1997 | Bowler | Premkumars@gmail.com | 9592950517 | *null* | left-hand |
| 21 | VCB02 | Kali | J | 21 | 11-APR-2002 | Batsman | Kalij@gmail.com | 9592950630 | right-hand | *Null* |
| 61 | VCB01 | Kamalesh | A | 22 | 21-JUN-2001 | Batsman | Kamalesha@gmail.com | 9592958730 | right-hand | *Null* |
| 66 | VCB01 | Ganesh | V | 24 | 21-JUN-1998 | Batsman | Ganeshv@gmail.com | 9592958790 | right-hand | *Null* |
| 303 | TRICB01 | Arun | T | 24 | 21-OCT-1998 | Batsman | Ganeshv@gmail.com | 9592958450 | right-hand | *Null* |
| 313 | TRICB01 | Srinivasan | N | 24 | 21-OCT-1998 | Batsman | srinivasann@gmail.com | 9992958450 | right-hand | *Null* |

**Match:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MatchID** | **TeamID** | **TeamID** | **Match\_Date** | **Time1** | **Result** |
| M01 | CCB01 | TCB01 | 22-JUN-2022 | 1.3 | TCB01 - WIN |
| M02 | CCB02 | TCB02 | 22-JUN-2022 | 8.3 | CCB01 - WIN |
| M03 | TRIBCB02 | TCB01 | 24-JUN-2022 | 8.3 | TCB01 - WIN |
| M04 | TRIBCB01 | TCB02 | 25-JUN-2022 | 8.3 | TRICB01 - WIN |

**Ground:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GroundID** | **MatchID** | **Name** | **Location** | **Capacity** |
| GID01 | M01 | Nehru | Chennai | 10000 |
| GID02 | M02 | GK | Coimbatore | 10000 |
| GID03 | M03 | Sankar | Nellai | 6000 |

**Umpire:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **UmpireID** | **FName** | **LName** | **Age** | **DateofBirth** | **Country** | **email** | **contact\_no** |
| UID01 | Venkatesh | T | 45 | 21-JUN-1978 | INDIA | venkatesh@gmail.com | 9665571435 |
| UID02 | Muthukumar | R | 46 | 01-JUN-1979 | INDIA | mutukumarr@gmail.com | 9665571460 |
| UID03 | Somu | K | 42 | 01-JUN-1983 | INDIA | somuk@gmail.com | 9664471460 |

**Umpire\_Umpired:**

|  |  |  |
| --- | --- | --- |
| **UmpireID** | **MatchID** | **GroundID** |
| UID01 | M01 | GID01 |
| UID02 | M03 | GID02 |

3.1: To retrieve the location of a particular match conducted by its MatchID

SQL> SELECT LOCATION FROM GROUND WHERE MatchID=’M03’;

Resul:

|  |
| --- |
| **Location** |
| Nellai |

3.2: To retrieve the Players detail whose name start with ‘A’.

SQL> Select \* from Player where FName like 'A%';

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PlayerID** | **TeamID** | **FName** | **LName** | **Age** | **DateofBirth** | **PlayingRole** | **Email** | **contact\_no** |
| 62 | TCB01 | Amar | J | 22 | 21-SEP-1998 | Batsman | [Amarj@gmail.com](mailto:Amarj@gmail.com) | 9292930508 |
| 102 | TCB02 | Akash | G | 21 | 26-SEP-1999 | Batsman | [Amarj@gmail.com](mailto:Amarj@gmail.com) | 9292930510 |
| 303 | TRICB01 | Arun | T | 24 | 21-OCT-1998 | Batsman | [Ganeshv@gmail.com](mailto:Ganeshv@gmail.com) | 9592958450 |

3.3: Add a column Batting and Bowling in Player table.

SQL> Alter table player add Batting varchar(10);

Table Altered

SQL> Alter table player add Bowling varchar(10);

Table Altered

3.4: To count the number of right-hand batsman in a team.

SQL> Select count(\*) from Player where Batting='right-hand';

Result:

|  |
| --- |
| **count(\*)** |
| 9 |

3.5: To display the CricketBoard details for the BoardIDs 'BID01', 'BID03', and 'BID06'.

SQL> Select \* from CricketBoard where BoardID in('BID01','BID03','BID06');

|  |  |  |  |
| --- | --- | --- | --- |
| **BoardID** | **Name** | **Address** | **Contact\_No** |
| BID01 | Chennai Cricket Board | Chennai | 9988776699 |
| BID03 | Viluppuram Cricket Board | Viluppuram | 9966886699 |
| BID06 | Tuticorin Cricket Board | Tuticorin | 9933886699 |

3.6: To select the names and IDs of players who are left-hand bowlers.

SQL> Select playerID, FName, LName from player where Bowling='left-hand';

|  |  |  |
| --- | --- | --- |
| **PlayerID** | **FName** | **LName** |
| 1 | Raj | N |
| 18 | Kishore | K |
| 01 | Prem | V |

3.7: To find the UmpireID of umpires who have not umpired any match.

SQL> select a.UmpireID from Umpire a where UmpireID NOT IN(select UmpireID from Umpire\_Umpired);

Result:

|  |
| --- |
| **UmpireID** |
| UID03 |

**Result:**

Thus the query processing on database for different retrieval result of query using Clauses, Operators and Functions in queries has been performed successfully.

**TASK 4**

**Using Functions in Queries and Writing Sub Queries**

Aim:

To perform the advanced query processing and test its heuristics using designing of optimal correlated and nested sub queries such as finding summary statistics.

1. To retrieve all team details, including the count of winning matches for each team
2. To retrieve the total number of 'Tie' matches in a team-wise manner.
3. To retrieve the team details who won the matches.
4. To retrieve players and match details of players who are above 25 years old.
5. To retrieve the details of Team who have not played any matches.
6. To retrieve the teamid, boardid, teamname, and playername for a particular playerid given.
   1. To retrieve all team details, including the count of winning matches for each team.

SQL> SELECT t.TeamID, t.Name AS TeamName, t.Coach, t.Captain, COUNT(m.MatchID) AS WinningMatchCount FROM Team t LEFT JOIN Match m ON t.TeamID = substr(m.result,1,5) GROUP BY t.TeamID, t.Name, t.Coach, t.Captain;

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TeamID** | **TeamName** | **Coach** | **Captain** | **WinningMatchCount** |
| CCB01 | ABS EXPRESS | G.D.RAMESH | SAMPATH KUMAR | 1 |
| CCB02 | AVG EXPRESS | T.KARTHIK | Y.JOHN | 0 |
| MCB01 | PANTHER | SARAVANAN | R.SUNILKUMAR | 0 |
| SCB01 | EAGLE | SOMU | SRI HARI | 0 |
| TCB01 | ANGRY BARD | TOM BABU | CINIL JOHN | 2 |
| TCB02 | TIGER ROCK | S.KANNAN | BEN GEORGE | 0 |
| TICB01 | KINGS | D ANAND | MATHAN | 0 |
| TRICB01 | ROCK | K.PAUL | K.MUTHU | 0 |
| TUCB01 | THUNDER | D ALEX | BARATHI | 0 |
| VCB01 | RAINBOW | S.RAJESHKUMAR | MANIMARAN | 0 |

4.2 To retrieve the total number of 'Tie' matches in a team-wise manner.

SQL> SELECT t.Name AS TeamName, COUNT(\*) AS TotalTieMatches FROM Team t JOIN Match\_result mt ON t.TeamID = mt.TeamID JOIN Match\_result m ON mt.MatchID = m.MatchID WHERE m.Result = 'Tie' GROUP BY t.Name;

|  |  |
| --- | --- |
| **TeamName** | **TotalTieMatches** |
| ROCK | 1 |

* 1. To retrieve the team details who won the matches.

SQL> select \* from team where teamID in (select mr.teamID from match\_result mr left join team t on mr.teamId=t.teamID where mr.result='Win');

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TeamID** | **BoardID** | **Name** | **Coach** | **Captain** |
| CCB01 | BID01 | ABS EXPRESS | G.D.RAMESH | SAMPATH KUMAR |
| TCB01 | BID02 | ANGRY BARD | TOM BABU | CINIL JOHN |
| TRICB01 | BID04 | ROCK | K.PAUL | K.MUTHU |

* 1. To retrieve players and match details of players who are above 25 years old.

SQL> SELECT p.PlayerID, p.FName AS PlayerName, p.Age, m.MatchID, m.match\_Date, m.Time1, m.Result FROM Player p, match m where p.playerID in(select playerID from player where age>25);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PlayerID** | **PlayerName** | **Age** | **MatchID** | **Match\_Date** | **Time1** | **Result** |
| 1 | Raj | 27 | M01 | 22-JUN-2022 | 1.3 | TCB01 - WIN |
| 1 | Raj | 27 | M02 | 22-JUN-2022 | 8.3 | CCB01 - WIN |
| 1 | Raj | 27 | M03 | 24-JUN-2022 | 8.3 | TCB01 - WIN |
| 1 | Raj | 27 | M04 | 25-JUN-2022 | 8.3 | TRICB01 - WIN |
| 1 | Raj | 27 | M05 | 04-APR-2023 | 7.3 | Tie |

* 1. To retrieve the details of Team who have not played any matches.

SQL> select \* from team where teamID not in(select teamid from match Union select playerId from match);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TeamID** | **BoardID** | **Name** | **Coach** | **Captain** |
| VCB01 | BID03 | RAINBOW | S.RAJESHKUMAR | MANIMARAN |
| MCB01 | BID05 | PANTHER | SARAVANAN | R.SUNILKUMAR |
| TUCB01 | BID06 | THUNDER | D ALEX | BARATHI |
| TICB01 | BID08 | KINGS | D ANAND | MATHAN |

* 1. To retrieve the teamid, boardid, teamname, and playername for a particular playerid given.

SQL> SELECT t.teamid, t.boardid, t.name, p.fname FROM team t JOIN player p ON t.teamid = p.teamid WHERE p.playerid = '66';

|  |  |  |  |
| --- | --- | --- | --- |
| **TeamID** | **BoardID** | **Name** | **FName** |
| VCB01 | BID03 | RAINBOW | Ganesh |

Result:

Thus the query using joins and writing subqueries has been done successfully.

**TASK 5**

**Writing Join Queries, equivalent, and/or recursive queries:**

(Tool: SQL/ Oracle, ALM: Flipped Classroom)

**Aim:** To Perform the advanced query processing and test its heuristics using designing of optimal correlated and nested sub queries such as finding summary statistics.

* 1. To retrieve all cricket boards and their teams.
  2. To list all matches along with the teams and their captains.
  3. To count the number of matches played by each team.
  4. To find all the players who are part of the team named " TIGER ROCK ".
  5. To retrieve all team details, including the count of winning matches for each team.
  6. To retrieve the total number of 'Tie' matches in a team-wise manner.
  7. To retrieve the team details who won the matches.
  8. To retrieve players and match details of players who are above 25 years old.
  9. To retrieve the details of Team who have not played any matches.
  10. To retrieve the teamid, boardid, teamname, and playername for a particular playerid given.
  11. To retrieve all cricket boards and their teams.

SQL> SELECT cb.Name AS CricketBoard, t.Name AS Team FROM CricketBoard cb JOIN Team t ON cb.BoardID = t.BoardID;

|  |  |
| --- | --- |
| **CricketBoard** | **Team** |
| Chennai Cricket Board | ABS EXPRESS |
| Chennai Cricket Board | AVG EXPRESS |
| Tiruvallur Cricket Board | ANGRY BARD |
| Tiruvallur Cricket Board | TIGER ROCK |
| Trichy Cricket Board | ROCK |
| Viluppuram Cricket Board | RAINBOW |
| Madurai Cricket Board | PANTHER |
| Tuticorin Cricket Board | THUNDER |
| Selam Cricket Board | EAGLE |
| Tiruppur Cricket Board | KINGS |

* 1. List all matches along with the teams and their captains.

SQL> SELECT m.match\_Date, m.Time1, m.matchID, t1.name AS team1\_name, t1.captain AS team1\_captain, t2.name AS team2\_name, t2.captain AS team2\_captain FROM match m JOIN team t1 ON m.teamID = t1.teamID JOIN team t2 ON m.playerID = t2.teamID;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Match\_Date** | **Time1** | **MatchID** | **team1\_name** | **team1\_captain** | **team2\_name** | **team2\_captain** |
| 22-JUN-2022 | 1.3 | M01 | ABS EXPRESS | SAMPATH KUMAR | ANGRY BARD | CINIL JOHN |
| 22-JUN-2022 | 8.3 | M02 | AVG EXPRESS | Y.JOHN | TIGER ROCK | BEN GEORGE |
| 24-JUN-2022 | 8.3 | M03 | ROCK | K.MUTHU | ANGRY BARD | CINIL JOHN |
| 25-JUN-2022 | 8.3 | M04 | ROCK | K.MUTHU | TIGER ROCK | BEN GEORGE |
| 04-APR-2023 | 7.3 | M05 | EAGLE | SRI HARI | AVG EXPRESS | Y.JOHN |

* 1. Count the number of matches played each team.

SQL> SELECT t.Name AS Team, COUNT(mt.TeamID) AS MatchesPlayed FROM Team t LEFT JOIN Match mt ON t.TeamID = mt.TeamID GROUP BY t.Name;

|  |  |
| --- | --- |
| **Team** | **MatchesPlayed** |
| ABS EXPRESS | 1 |
| ANGRY BARD | 0 |
| AVG EXPRESS | 1 |
| EAGLE | 1 |
| KINGS | 0 |
| PANTHER | 0 |
| RAINBOW | 0 |
| ROCK | 2 |
| THUNDER | 0 |
| TIGER ROCK | 0 |

* 1. To find all the players who are part of the team named "TIGER ROCK".

SQL> SELECT p.playerID, p.fname, p.teamID, t.coach, t.captain FROM player p JOIN team t ON p.teamID = t.teamID WHERE t.name = 'TIGER ROCK';

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PlayerID** | **FName** | **TeamID** | **Coach** | **Captain** |
| 102 | Akash | TCB02 | S.KANNAN | BEN GEORGE |
| 12 | Premkumar | TCB02 | S.KANNAN | BEN GEORGE |

* 1. To retrieve all team details, including the count of winning matches for each team.

SQL> SELECT t.TeamID, t.Name AS TeamName, t.Coach, t.Captain, COUNT(m.MatchID) AS WinningMatchCount FROM Team t LEFT JOIN Match m ON t.TeamID = substr(m.result,1,5) GROUP BY t.TeamID, t.Name, t.Coach, t.Captain;

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TeamID** | **TeamName** | **Coach** | **Captain** | **WinningMatchCount** |
| CCB01 | ABS EXPRESS | G.D.RAMESH | SAMPATH KUMAR | 1 |
| CCB02 | AVG EXPRESS | T.KARTHIK | Y.JOHN | 0 |
| MCB01 | PANTHER | SARAVANAN | R.SUNILKUMAR | 0 |
| SCB01 | EAGLE | SOMU | SRI HARI | 0 |
| TCB01 | ANGRY BARD | TOM BABU | CINIL JOHN | 2 |
| TCB02 | TIGER ROCK | S.KANNAN | BEN GEORGE | 0 |
| TICB01 | KINGS | D ANAND | MATHAN | 0 |
| TRICB01 | ROCK | K.PAUL | K.MUTHU | 0 |
| TUCB01 | THUNDER | D ALEX | BARATHI | 0 |
| VCB01 | RAINBOW | S.RAJESHKUMAR | MANIMARAN | 0 |

5.6 To retrieve the total number of 'Tie' matches in a team-wise manner.

SQL> SELECT t.Name AS TeamName, COUNT(\*) AS TotalTieMatches FROM Team t JOIN Match\_result mt ON t.TeamID = mt.TeamID JOIN Match\_result m ON mt.MatchID = m.MatchID WHERE m.Result = 'Tie' GROUP BY t.Name;

|  |  |
| --- | --- |
| **TeamName** | **TotalTieMatches** |
| ROCK | 1 |

5.7To retrieve the team details who won the matches.

SQL> select \* from team where teamID in (select mr.teamID from match\_result mr left join team t on mr.teamId=t.teamID where mr.result='Win');

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TeamID** | **BoardID** | **Name** | **Coach** | **Captain** |
| CCB01 | BID01 | ABS EXPRESS | G.D.RAMESH | SAMPATH KUMAR |
| TCB01 | BID02 | ANGRY BARD | TOM BABU | CINIL JOHN |
| TRICB01 | BID04 | ROCK | K.PAUL | K.MUTHU |

* 1. To retrieve players and match details of players who are above 25 years old.

SQL> SELECT p.PlayerID, p.FName AS PlayerName, p.Age, m.MatchID, m.match\_Date, m.Time1, m.Result FROM Player p, match m where p.playerID in(select playerID from player where age>25);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PlayerID** | **PlayerName** | **Age** | **MatchID** | **Match\_Date** | **Time1** | **Result** |
| 1 | Raj | 27 | M01 | 22-JUN-2022 | 1.3 | TCB01 - WIN |
| 1 | Raj | 27 | M02 | 22-JUN-2022 | 8.3 | CCB01 - WIN |
| 1 | Raj | 27 | M03 | 24-JUN-2022 | 8.3 | TCB01 - WIN |
| 1 | Raj | 27 | M04 | 25-JUN-2022 | 8.3 | TRICB01 - WIN |
| 1 | Raj | 27 | M05 | 04-APR-2023 | 7.3 | Tie |

* 1. To retrieve the details of Team who have not played any matches.

SQL> select \* from team where teamID not in(select teamid from match Union select playerId from match);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TeamID** | **BoardID** | **Name** | **Coach** | **Captain** |
| VCB01 | BID03 | RAINBOW | S.RAJESHKUMAR | MANIMARAN |
| MCB01 | BID05 | PANTHER | SARAVANAN | R.SUNILKUMAR |
| TUCB01 | BID06 | THUNDER | D ALEX | BARATHI |
| TICB01 | BID08 | KINGS | D ANAND | MATHAN |

* 1. To retrieve the teamid, boardid, teamname, and playername for a particular playerid given.

SQL> SELECT t.teamid, t.boardid, t.name, p.fname FROM team t JOIN player p ON t.teamid = p.teamid WHERE p.playerid = '66';

|  |  |  |  |
| --- | --- | --- | --- |
| **TeamID** | **BoardID** | **Name** | **FName** |
| VCB01 | BID03 | RAINBOW | Ganesh |

**Result:**

Thus the query using Join Queries, equivalent, and/or recursive queries has been done successfully.

**TASK 6: Procedures, Function and Loops**

**Aim:** To write a programming using PL/SQL Procedures, Functions and loops on Number theory and business scenarios like.

1. Write a PL/SQL block that calculates the average age of players and displays the result.
2. Write a PL/SQL block that inserts a new player record into the Player table.
3. To create a function that returns the total number of teams in a particular Cricket Board.
4. To write a non-recursive PL/SQL procedure to retrieve even-numbered PlayerIDs registered for any tournament.

**Write a PL/SQL block that calculates the average age of players and displays the result.**

DECLARE

total\_age NUMBER := 0;

num\_players NUMBER := 0;

avg\_age NUMBER := 0;

BEGIN

-- Using a cursor to loop through all players

FOR player\_rec IN (SELECT Age FROM Player) LOOP

total\_age := total\_age + player\_rec.Age; -- Summing up the ages

num\_players := num\_players + 1; -- Counting the number of players

END LOOP;

-- Calculating the average age

IF num\_players > 0 THEN

avg\_age := total\_age / num\_players;

END IF;

-- Displaying the result

DBMS\_OUTPUT.PUT\_LINE('Total Players: ' || num\_players);

DBMS\_OUTPUT.PUT\_LINE('Total Age: ' || total\_age);

DBMS\_OUTPUT.PUT\_LINE('Average Age: ' || avg\_age);

END;

Output:

Total Players: 14

Total Age: 342

Average Age: 24.42

**Write a PL/SQL block that inserts a new player record into the Player table.**

DECLARE

v\_PlayerID VARCHAR(6) := &PlayerID’; -- You can generate a unique PlayerID as needed

v\_TeamID VARCHAR(6) := '&TEAMID'; -- Replace with the actual TeamID

v\_FName VARCHAR(30) := '&Fname';

v\_LName VARCHAR(30) := '&Lname';

v\_Age NUMBER(5,2) := &age;

v\_DateofBirth DATE := TO\_DATE('&DOB', 'YYYY-MM-DD'); -- Replace with the actual DateofBirth

v\_PlayingRole VARCHAR(25) := '&PlayingRole';

v\_email VARCHAR(40) := '&email';

v\_contact\_no NUMBER := &phone; -- Replace with the actual contact number

BEGIN

INSERT INTO Player (PlayerID, TeamID, FName, LName, Age, DateofBirth, PlayingRole, email, contact\_no)

VALUES (v\_PlayerID, v\_TeamID, v\_FName, v\_LName, v\_Age, v\_DateofBirth, v\_PlayingRole, v\_email, v\_contact\_no);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Player record inserted successfully.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

Enter the PlayerID: 676

Enter the TeamID: CCB01

Enter the FName: Rahul

Enter the LName: Sharma

Enter the Age: 23

Enter the DateofBirth: 17-07-1999

Enter the PlayingRole: AllRounder

Enter the email: rahulsharma@gmail.com

Enter the Contact\_no: 9797181815

Player record inserted successfully.

**To create a function that returns the total number of teams in a particular Cricket Board.**

CREATE OR REPLACE FUNCTION GetTotalTeamsInBoard(BoardID VARCHAR2) RETURN NUMBER IS

v\_TotalTeams NUMBER := 0;

BEGIN

SELECT COUNT(\*) INTO v\_TotalTeams FROM Team WHERE BoardID = BoardID;

RETURN v\_TotalTeams;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Handle the case when the board doesn't exist or has no teams

RETURN 0;

WHEN OTHERS THEN

-- Handle other exceptions as needed

RETURN -1; -- Return a negative value to indicate an error

END GetTotalTeamsInBoard;

/

**Function successfully created.**

**SQL>**

Declare

**number res;**

**Begin  
res:=** GetTotalTeamsInBoard(‘BID01’);

DBMS\_OUTPUT.PUT\_LINE(‘No of teams: ‘||res);

END;

/

No of teams: 2

**To write a non-recursive PL/SQL procedure to retrieve even-numbered PlayerIDs registered for any tournament.**

CREATE OR REPLACE PROCEDURE GetEvenNumberedPlayerIDs IS

BEGIN

FOR player\_rec IN ( SELECT PlayerID FROM Player WHERE TO\_NUMBER(PlayerID) MOD 2 = 0)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Even-Numbered PlayerID: ' || player\_rec.PlayerID);

END LOOP;

END GetEvenNumberedPlayerIDs;

/

**Result:**

Thus the PL/SQL Procedures, Functions and loops on Number theory and business scenarios experiment was successfully completed and results are verified.

**TASK 7**: **Triggers, Views and Exceptions**

**Aim:**

To Conduct events, views, exceptions on CRUD operations for restricting phenomenon.

a) To create a trigger in PL/SQL that automatically inserts a new record in the match\_result table when a new record is inserted into the match table.

b) To create a view that displays the details of players along with their team details.

c) To write a non-recursive PL/SQL procedure to retrieve even-numbered PlayerIDs registered for any tournament.

**To create a trigger in PL/SQL that automatically inserts a new record in the match\_result table when a new record is inserted into the match table.**

CREATE OR REPLACE TRIGGER insert\_match\_result

AFTER INSERT ON match

FOR EACH ROW

BEGIN

INSERT INTO match\_result (MatchID, TeamID, Result)

VALUES (:new.MatchID, :new.TeamID, 'Pending'); -- Assuming a default value of 'Pending' for Result

END;

/

**To create a view that displays the details of players along with their team details.**

SQL> CREATE VIEW PlayerTeamDetails AS SELECT p.playerID, p.fname AS PlayerName, p.teamID, p.coach AS PlayerCoach, p.captain AS PlayerCaptain, (SELECT t.name FROM team t WHERE t.teamID = p.teamID) AS TeamName, (SELECT t.coach FROM team t WHERE t.teamID = p.teamID) AS TeamCoach, (SELECT t.captain FROM team t WHERE t.teamID = p.teamID) AS TeamCaptain FROM player p;

SQL> Select \* from PlayerTeamDetails;

**To write a non-recursive PL/SQL procedure to retrieve even-numbered PlayerIDs registered for any tournament.**

CREATE OR REPLACE PROCEDURE GetEvenPlayerIDsForTournament(in\_tournament\_id NUMBER, out\_even\_player\_ids SYS.ODCINUMBERLIST) AS

BEGIN

out\_even\_player\_ids := SYS.ODCINUMBERLIST(); -- Initialize the collection

-- Populate the collection with even-numbered PlayerIDs for the specified tournament

FOR player\_rec IN (SELECT PlayerID FROM Player WHERE TournamentID = in\_tournament\_id AND MOD(PlayerID, 2) = 0) LOOP

out\_even\_player\_ids.EXTEND;

out\_even\_player\_ids(out\_even\_player\_ids.COUNT) := player\_rec.PlayerID;

END LOOP;

END;

/

DECLARE

tournament\_id NUMBER := 123; -- Replace with the desired tournament ID

even\_player\_ids SYS.ODCINUMBERLIST;

BEGIN

GetEvenPlayerIDsForTournament(tournament\_id, even\_player\_ids);

-- You can now use the even\_player\_ids collection as needed.

-- For example, to print the even PlayerIDs:

FOR i IN 1..even\_player\_ids.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Even PlayerID: ' || even\_player\_ids(i));

END LOOP;

END;

/

**Result:**

Thus the Triggers, Views and Exceptions experiment was successfully completed results are verified.

**TASK 8**

**CRUD operations in Document databases**

**AIM:**

To Perform Mongoose using NPM design on MongoDB designing document database and performing CRUD operations like creating, inserting, querying, finding, removing operations

**STEPS:**

Step 1: Install Mongo db using following link

https://www.mongodb.com/try/download/community

Step 2: install Mongosh using the below link

https://www.mongodb.com/docs/mongodb-shell/#download-and-install-mongosh

Step 3: To add the MongoDB Shell binary's location to your PATH environment variable:

3.1 Open the Control Panel.

3.2 In the System and Security category, click System.

3,3 Click Advanced system settings. The System Properties modal displays.

3.4 Click Environment Variables.

3.5 In the System variables section, select path and click Edit. The Edit environment

variable modal displays.

3.6 Click New and add the filepath to your mongosh binary.

3.7 Click OK to confirm your changes. On each other modal, click OK to confirm

your changes.

Step 4: To confirm that your PATH environment variable is correctly configured to find

mongosh, open a command prompt and enter the mongosh --help command. If your

PATH is configured correctly, a list of valid commands displays.

Step 5: Open mongo shell 4.0 from c:\programfiles\mongoDB\server\bin\mongod.exe

Step 6: Type the CRUD(CREATE READ UPDATE DELETE) COMMANDS GIVEN IN

TEXT FILE.

**CRUD OPERATIONS:**

db.createCollection("CricketBorad")

{ "ok" : 1 }

> db.CricketBoard.insertOne({BoardID: "BID01", Name:"Chennai Cricket Board", Address:"Chennai",Phone:9988776699});

{

"acknowledged" : true,

"insertedId" : ObjectId("651cf1726ebbfe7993adf909")

}

db.mylab.find({BoardID:"BID01"})

{ "\_id" : ObjectId("651cf1726ebbfe7993adf909"), "BoardID" : "BID01", "Name" : "Chennai Cricket Board", "Address" : "Chennai", "Phone" : 9988776699 }

db.CricketBoard.insertMany([{BoardID:"BID02",Name:"Tiruvallur Cricket Board",Address:"Chennai",Phone:9977886699},{BoardID:"BID03",Name:"Viluppuram Cricket Board",Address:"Viluppuram",Phone:9966886699},{BoardID:"BID04",Name:"Trichy Cricket Board",Address:"Trichy",Phone:9955886699},{BoardID:"BID05",Name:"Madurai Cricket Board",Address:"Madurai",Phone:9944886699}]);

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("651ceee36ebbfe7993adf904"),

ObjectId("651ceee36ebbfe7993adf905"),

ObjectId("651ceee36ebbfe7993adf906"),

ObjectId("651ceee36ebbfe7993adf907")

]

}

db.CricketBoard.find()

{ "\_id" : ObjectId("651ceee36ebbfe7993adf904"), "BoardID" : "BID02", "Name" : "Tiruvallur Cricket Board", "Address" : "Chennai", "Phone" : 9977886699 }

{ "\_id" : ObjectId("651ceee36ebbfe7993adf905"), "BoardID" : "BID03", "Name" : "Viluppuram Cricket Board", "Address" : "Viluppuram", "Phone" : 9966886699 }

{ "\_id" : ObjectId("651ceee36ebbfe7993adf906"), "BoardID" : "BID04", "Name" : "Trichy Cricket Board", "Address" : "Trichy", "Phone" : 9955886699 }

{ "\_id" : ObjectId("651ceee36ebbfe7993adf907"), "BoardID" : "BID05", "Name" : "Madurai Cricket Board", "Address" : "Madurai", "Phone" : 9944886699 }

{ "\_id" : ObjectId("651cf1726ebbfe7993adf909"), "BoardID" : "BID01", "Name" : "Chennai Cricket Board", "Address" : "Chennai", "Phone" : 9988776699 }

db.CricketBoard.find().pretty()

{

"\_id" : ObjectId("651ceee36ebbfe7993adf904"),

"BoardID" : "BID02",

"Name" : "Tiruvallur Cricket Board",

"Address" : "Chennai",

"Phone" : 9977886699

}

{

"\_id" : ObjectId("651ceee36ebbfe7993adf905"),

"BoardID" : "BID03",

"Name" : "Viluppuram Cricket Board",

"Address" : "Viluppuram",

"Phone" : 9966886699

}

{

"\_id" : ObjectId("651ceee36ebbfe7993adf906"),

"BoardID" : "BID04",

"Name" : "Trichy Cricket Board",

"Address" : "Trichy",

"Phone" : 9955886699

}

{

"\_id" : ObjectId("651ceee36ebbfe7993adf907"),

"BoardID" : "BID05",

"Name" : "Madurai Cricket Board",

"Address" : "Madurai",

"Phone" : 9944886699

}

{

"\_id" : ObjectId("651cf1726ebbfe7993adf909"),

"BoardID" : "BID01",

"Name" : "Chennai Cricket Board",

"Address" : "Chennai",

"Phone" : 9988776699

}

db.CricketBoard.deleteOne({BoardID:"BID03"})

{ "acknowledged" : true, "deletedCount" : 1 }

db.CricketBoard.find().pretty()

{

"\_id" : ObjectId("651ceee36ebbfe7993adf904"),

"BoardID" : "BID02",

"Name" : "Tiruvallur Cricket Board",

"Address" : "Chennai",

"Phone" : 9977886699

}

{

"\_id" : ObjectId("651ceee36ebbfe7993adf906"),

"BoardID" : "BID04",

"Name" : "Trichy Cricket Board",

"Address" : "Trichy",

"Phone" : 9955886699

}

{

"\_id" : ObjectId("651ceee36ebbfe7993adf907"),

"BoardID" : "BID05",

"Name" : "Madurai Cricket Board",

"Address" : "Madurai",

"Phone" : 9944886699

}

{

"\_id" : ObjectId("651cf1726ebbfe7993adf909"),

"BoardID" : "BID01",

"Name" : "Chennai Cricket Board",

"Address" : "Chennai",

"Phone" : 9988776699

}

**Result:**

Thus CRUD using NPM design on MongoDB designing document database and performing CRUD operations like creating, inserting, querying, finding, removing operations are performed.

**TASK 9**

**CRUD operations in Graph databases**

**AIM:**

To perform CRUD operations like creating, inserting, querying, finding, deleting operations on graph spaces.

**The steps to get started with Neo4j's Aura Graph Database:**

**Step1:** Copy and paste the following link into your web browser: https://neo4j.com/cloud/platform/aura-graph-database/?ref=docs-get-started-dropdown

**Step2:** Click on "Start Free."

**Step3:** Choose the option to "Continue with Google."

**Step4:** Click the "Open" button.

**Step5:** After clicking "Open," a text file will be automatically downloaded. This file contains your user ID and password details.

**Step6:** Copy the password from the downloaded text file and paste it where required.

**Step7:** Close the "Get started with Neo4j with beginner guides" if it's open.

**Sep8:** You're now ready to begin practicing with the Graph Database.

## Create Node with Properties

Properties are the key-value pairs using which a node stores data. Create a node with properties using the CREATE clause and need to specify these properties separated by commas within the flower braces “{ }”.

**Syntax**

CREATE (node:label { key1: value, key2: value, . . . . . . . . . }) return node

To verify the creation of the node, type and execute the following query in the dollar prompt.

**Syntax:**

MATCH (n) RETURN n

**Creating Relationships**

To create a relationship using the CREATE clause and specify relationship within the square braces “[ ]” depending on the direction of the relationship it is placed between hyphen “ - ” and arrow “ → ” as shown in the following syntax.

**Syntax:**

CREATE (node1)-[:RelationshipType]->(node2)

**Syntax:**

MATCH (a:LabeofNode1), (b:LabeofNode2)

WHERE a.name = "nameofnode1" AND b.name = " nameofnode2"

CREATE (a)-[: Relation]->(b) RETURN a,b

**Deleting a Particular Node**

To delete a particular node and need to specify the details of the node in the place of “n” in the above query.

**Syntax:**

MATCH (node:label {properties . . . . . . . . . . }) DELETE node

Create a graph database for student course registration, create student and dept node and insert values of properties.

**Create a CrickerBoard Node:**

create(cb:CricketBoard{BoardID:'BID01',Name:'Chennai Cricket Board', Address:'Chennai', Phone:9988776699}) return cb

**Create Team Nodes:**

create(t1:Team{teamID:'CCB01',BoardID:'BID01',name:'ABS EXPRESS', Coach:'G.D.RAMESH', Captain:'SAMPATH KUMAR'}) return t1

create(t2:Team{teamID:'CCB02',BoardID:'BID01',name:'AVG EXPRESS',Coach: 'T.KARTHIKH', Captain:'Y.JOHN'}) return t2

**Create Player Nodes:**

create(p1:Player{PlayerID:'1',TeamID:'CCB01',Name:'Raj',Age:23,DateofBirth:'29-JUN-1996', PlayingRole:'Bowler',email:'rajn@gmail.com'}) return p1

create(p2:Player{PlayerID:'33',TeamID:'CCB01',Name:'Anand',Age:23,DateofBirth:'02-JAN-1999', PlayingRole:'Batsman',email:'balajid@gmail.comm'}) return p2

create(p3:Player{PlayerID:'65',TeamID:'CCB02',Name:'Suresh',Age:27,DateofBirth:'02-JUN-1996', PlayingRole:'Batsman',email:'sureshd@gmail.comm'}) return p3

create(p4:Player{PlayerID:'75',TeamID:'CCB02',Name:'Rohit',Age:33,DateofBirth:'02-JUN-1991', PlayingRole:'Batsman',email:'srohit@gmail.comm'}) return p4

**Creating Relationship among CricketBoard and Teams:**

match(cb:CricketBoard{BoardID:'BID01'}),(t1:Team{teamID:'CCB01'}) create(cb)-[r:has]->(t1) return cb,r,t1

match(cb:CricketBoard{BoardID:'BID01'}),(t2:Team{teamID:'CCB02'}) create(cb)-[r:has]->(t2) return cb,r,t2

**Creating Relationship among Players and Teams:**

match(p1:Player{PlayerID:'1'}),(t1:Team{teamID:'CCB01'}) create(p1)-[r1:playfor]->(t1) return p1,r1,t1

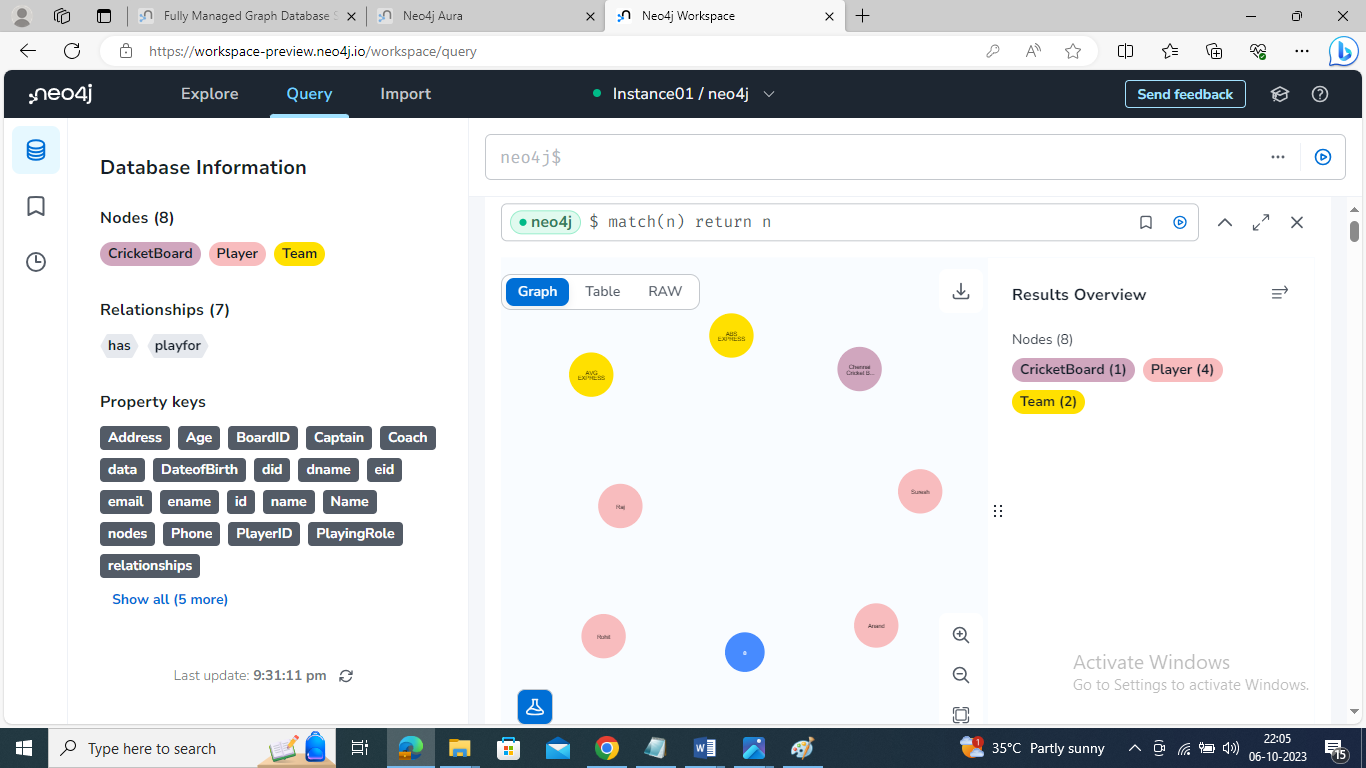
match(p2:Player{PlayerID:'33'}),(t1:Team{teamID:'CCB01'}) create(p2)-[r2:playfor]->(t1) return p2,r2,t1

match(p3:Player{PlayerID:'65'}),(t2:Team{teamID:'CCB02'}) create(p3)-[r3:playfor]->(t2) return p3,r3,t2

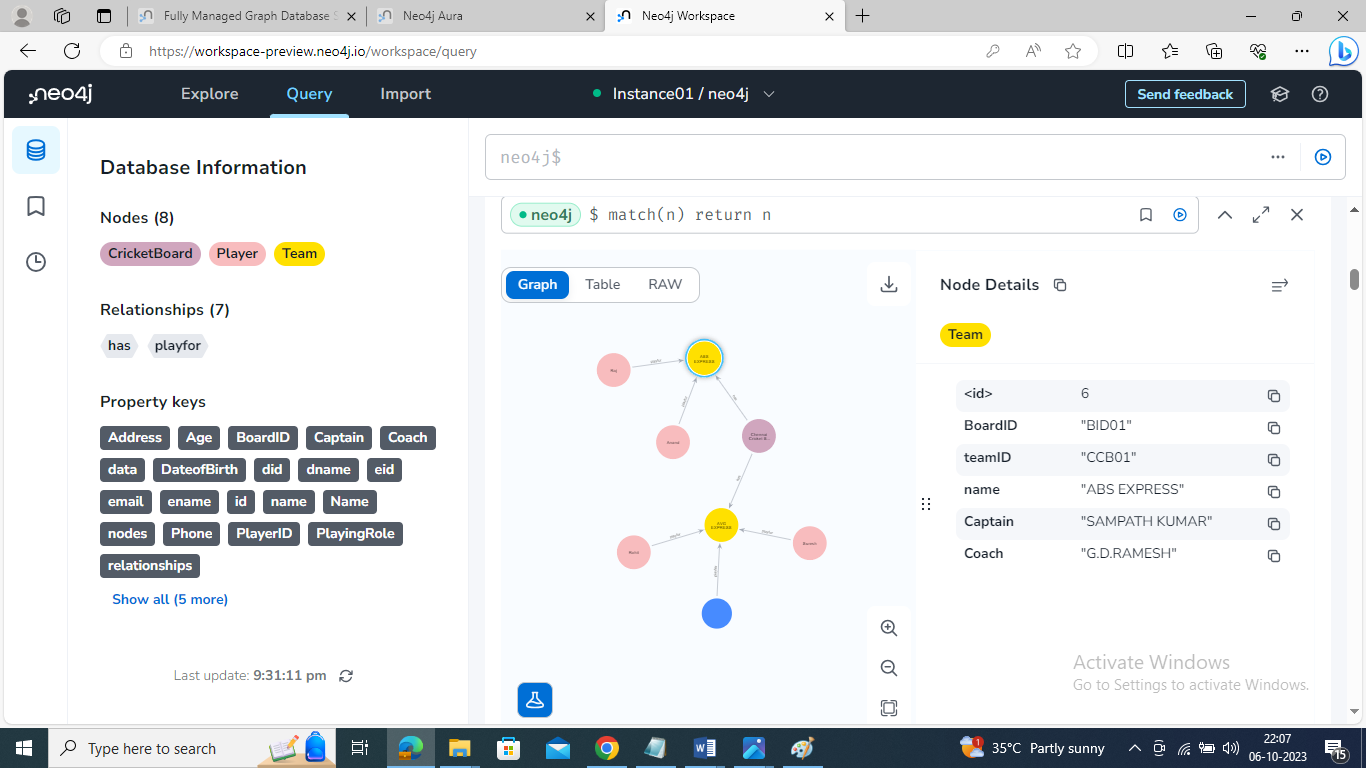
match(p4:Player{PlayerID:'75'}),(t2:Team{teamID:'CCB02'}) create(p3)-[r4:playfor]->(t2) return p4,r4,t2

**Display All nodes:** match(n) return n

**Output:**

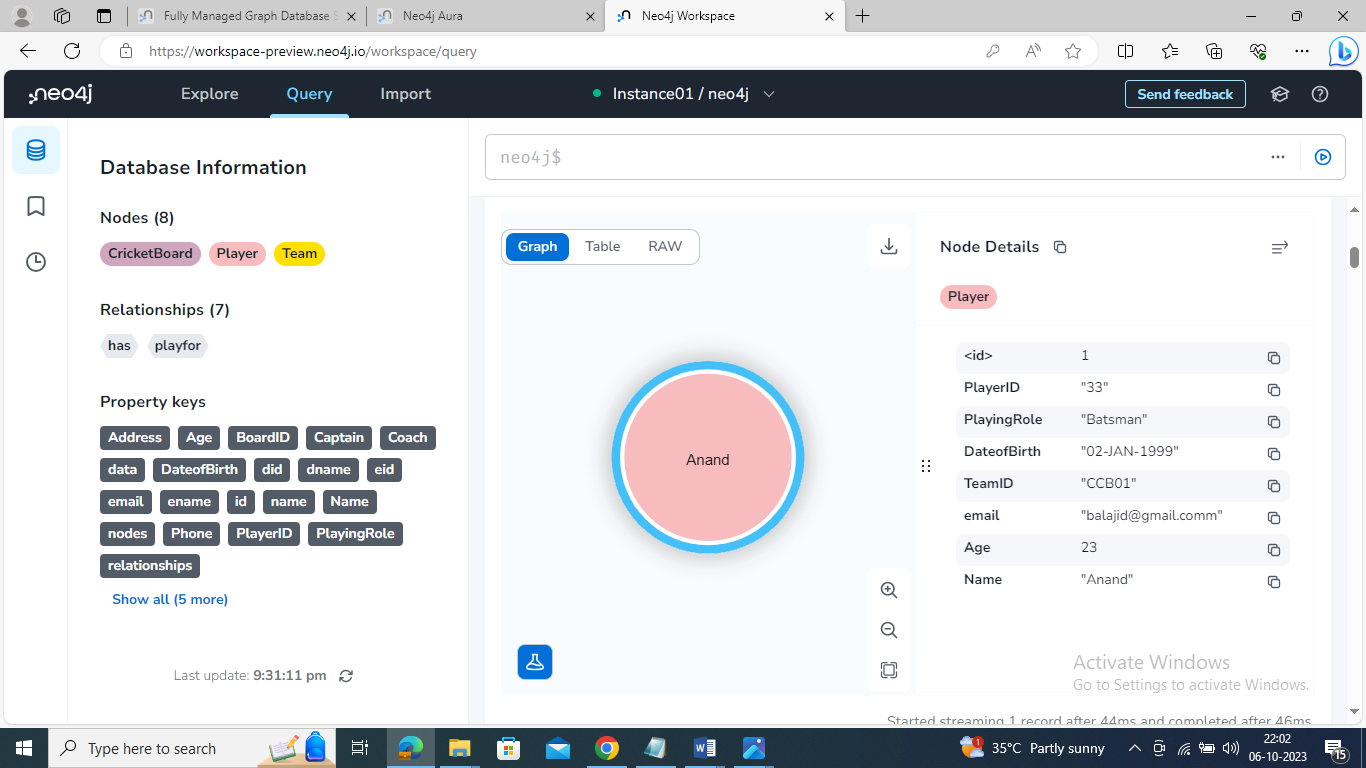
****

**OUTPUT:**



**Retrieve particular player details:**

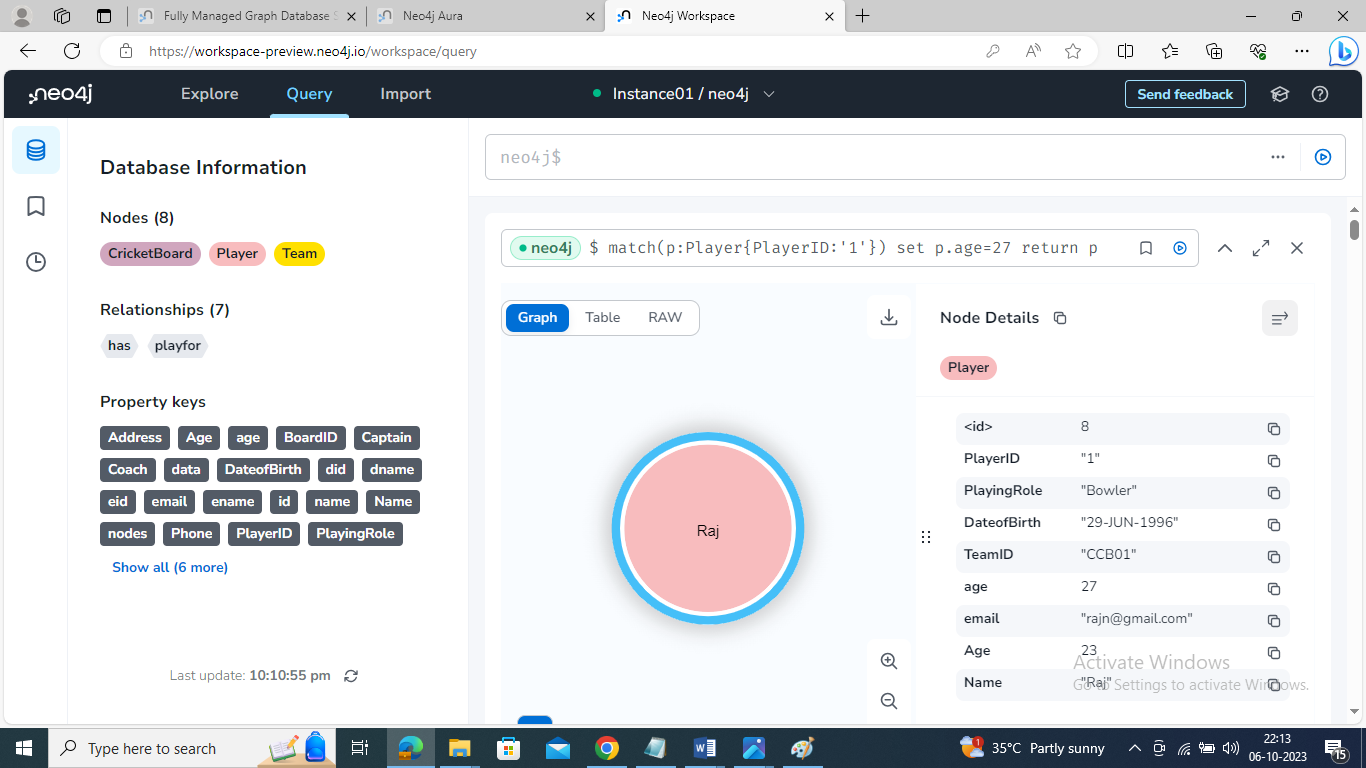
match(p:Player{PlayerID:'33'}) return p



**Update particular player details:**

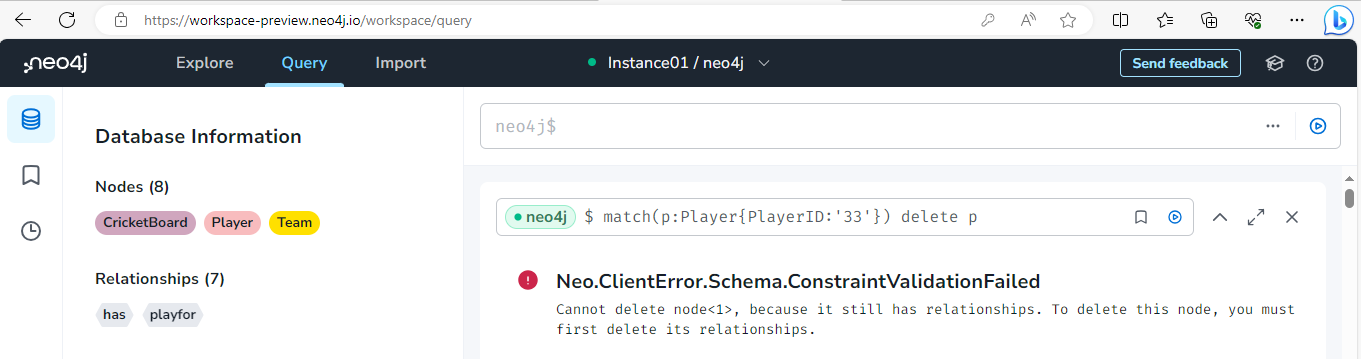
match(p:Player{PlayerID:'1'}) set p.age=27 return p

**Output:**

****

**Delete particular player from the team:**

match(p:Player{PlayerID:'33'}) delete p

****

**Result:**

Thus the CRUD operations like creating, inserting, querying, finding, deleting operations on graph spaces were executed successfully.

**Task 10**

**Normalizing databases using functional dependencies upto Third Normal Form**

**Aim:** To normalize the below relation and create the simplified table with suitable constraint.

CricketBoard(BoardID, Name, Address, Contact\_No, TeamID, TName, Coach, Captain, PlayerID, PFName, PLName, Age, PDateofBirth, PlayingRole, email, contact\_no, Batting, Bowling, MatchID, Match\_Date, Time1, Result, GroundID, GName, Location, Capacity, UmpireID, UFName, ULName, UAge, UDateofBirth, Country, Uemail, Ucontact\_no).

1. Apply the functional dependency, normalize to 1NF
2. Normalize the relations using FD+ and α+.
3. Find the minimal cover, canonical cover.
4. Normalize to 2NF, add/alter constraints if necessary.
5. Normalize to 3NF, add/alter constraints if necessary.

**Procedure:**

Normalize the given relation and create simplified tables with suitable constraints, we need to identify the functional dependencies and separate them into different tables. Normalization involves breaking down the data into smaller, related tables to minimize data redundancy and maintain data integrity. Let's identify the functional dependencies:

**Functional Dependency:**

BoardID → Name, Address, Contact\_No

TeamID → TName, Coach, Captain

PlayerID → PFName, PLName, Age, PDateofBirth, PlayingRole, email, contact\_no, Batting, Bowling

MatchID → Match\_Date, Time1, Result, GroundID

GroundID → GName, Location, Capacity

UmpireID → UFName, ULName, UAge, UDateofBirth, Country, Uemail, Ucontact\_no

Now, we can create simplified tables:

CricketBoard (BoardID [PK], Name, Address, Contact\_No)

CricketTeam (TeamID [PK], TName, Coach, Captain)

CricketPlayer (PlayerID [PK], TeamID [FK], PFName, PLName, Age, PDateofBirth, PlayingRole, email, contact\_no, Batting, Bowling)

CricketMatch (MatchID [PK], TeamID [FK], Match\_Date, Time1, Result, GroundID [FK])

CricketGround (GroundID [PK], GName, Location, Capacity)

CricketUmpire (UmpireID [PK], UFName, ULName, UAge, UDateofBirth, Country, Uemail, Ucontact\_no)

In these tables, [PK] denotes the primary key, [FK] denotes the foreign key, and suitable constraints should be added to maintain data integrity.

**Create tables for all non-prime attributes using α+**

α+ (Alpha Plus) allows to group attributes based on their functional dependencies and candidate keys. And create tables for each set of attributes that functionally depend on a candidate key. The candidate keys in this case are BoardID, TeamID, PlayerID, MatchID, and UmpireID.

Board Table: BoardID (PK), Name, Address, Contact\_No

Team Table: TeamID (PK), TName, Coach, Captain

Player Table: PlayerID (PK), TeamID (FK), PFName, PLName, Age, PDateofBirth, PlayingRole,

Email, contact\_no, Batting, Bowling

Match Table: MatchID (PK), TeamID (FK), Match\_Date, Time1, Result

Ground Table: GroundID (PK), GName, Location, Capacity

Umpire Table: UmpireID (PK), UFName, ULName, UAge, UDateofBirth, Country, Uemail, Ucontact\_no

Create additional tables to represent transitive dependencies.

Already addressed transitive dependencies in previous normalization steps by introducing the MatchVenue table for the transitive dependency between MatchID and GroundID through the Result attribute.

MatchVenue Table: MatchID (PK, FK), GroundID (FK)

**First Normal Form:**

The given relation into the First Normal Form (1NF), to need to ensure that each attribute (column) contains atomic (indivisible) values, and there are no repeating groups or arrays.

Based on the provided relation, it appears that each attribute already contains atomic values, so there are no repeating groups to eliminate.

**Second Normal Form:**

To determine whether the given relation is in the Second Normal Form (2NF), we need to check two conditions:

The relation must already be in 1NF (First Normal Form).

All non-prime attributes (attributes not part of any candidate key) must be fully functionally dependent on the entire primary key.

First, let's identify the potential candidate key(s) from the given relation based on functional dependencies:

It appears that the potential candidate keys could be:

1. BoardID
2. TeamID
3. PlayerID
4. MatchID
5. UmpireID

Next, we need to check if all non-prime attributes are fully functionally dependent on their respective candidate key(s).

**Third Normal Form:**

To determine whether the given relation is in the Third Normal Form (3NF), need to check two conditions:

1. The relation must already be in the Second Normal Form (2NF).
2. There should be no transitive dependencies between non-prime attributes and candidate keys.

The given relation satisfies the conditions of the Second Normal Form (2NF). Now, let's check for transitive dependencies:

Now, let's analyze each functional dependency and check for transitive dependencies:

BoardID → Name, Address, Contact\_No

There are no transitive dependencies in this case, as Name, Address, and Contact\_No are directly dependent on BoardID.

TeamID → TName, Coach, Captain

There are no transitive dependencies here either, as TName, Coach, and Captain are directly dependent on TeamID.

PlayerID → PFName, PLName, Age, PDateofBirth, PlayingRole, email, contact\_no, Batting, Bowling

There are no transitive dependencies for PlayerID, as all the mentioned attributes are directly dependent on PlayerID.

MatchID → Match\_Date, Time1, Result, GroundID

There is a transitive dependency between MatchID and GroundID through the Result attribute. To resolve this, we create a new table called MatchVenue:

MatchVenue (MatchID [PK], GroundID [FK])

GroundID → GName, Location, Capacity

There are no transitive dependencies for GroundID, as GName, Location, and Capacity are directly dependent on GroundID.

UmpireID → UFName, ULName, UAge, UDateofBirth, Country, Uemail, Ucontact\_no

There are no transitive dependencies for UmpireID, as UFName, ULName, UAge, UDateofBirth, Country, Uemail, and Ucontact\_no are directly dependent on UmpireID.

With the introduction of the MatchVenue table to resolve the transitive dependency, the relation now satisfies the conditions of the Third Normal Form (3NF).

**Result:**

Thus the normalization of the given relation is created the simplified tables with suitable constraint successfully.

**TASK 11**

**Menus, Forms and Reports**

**Aim:**

To designing an application with Oracle Forms, Menus and Report Builder involves creating a user interface (UI) using Forms and generating reports using Report Builder.

**Install Oracle Forms and Report Builder:**

Ensure that the Oracle Forms and Report Builder installed on your development machine.

**Design the Data Model:**

In Oracle Forms to define the data model that connects to the database schema. Use the Data Block Wizard to create data blocks that represent the tables or views. Ensure that the set up data blocks for all the data to need to work with in the application.

**Create Menus**

Menus provide the navigation structure for application. To create menus in Oracle Forms:

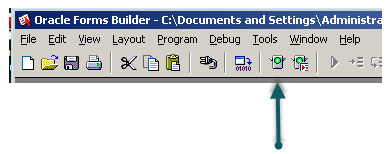
**Step1:** Open Oracle Forms Builder.

**Step2:** Create a new form for menu or use an existing one.

**Step3:** Add menu items for each function or feature of application.

**Step4:** Define the menu hierarchy and assign triggers or procedures to handle menu item actions.

**Step5:** Compile and run the menu form to test the navigation.



**Design Forms**

Forms are used to capture, display, and edit data. To design forms in Oracle Forms:

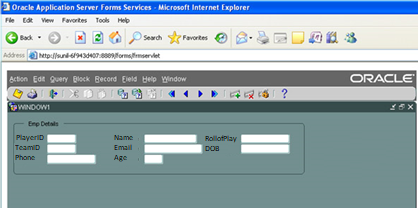
**Step1:** Create a new form for each major component of application.

**Step2:** Add form elements like text fields, buttons, and lists to forms.

**Step3:** Use the Property Palette to configure the properties of form elements and data blocks.

**Step4:** Write PL/SQL code to handle business logic and data validation.

**Step5:** Test the forms within the Forms Builder environment.



**Create Reports**

Reports provide a way to present data from application. To create reports using Oracle Report Builder:

**Step1:** Open Oracle Report Builder.

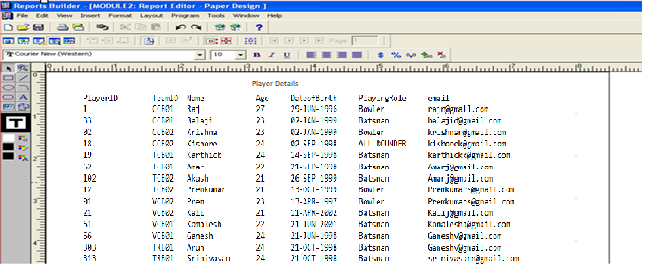
**Step2:** Create a new report or use an existing one.

**Step3:** Define the data source for the report (e.g., a database query or PL/SQL procedure).

**Step4:** Design the report layout, including headers, footers, and data columns.

**Step5:** Add report parameters if needed to allow users to customize the report.

**Step6:** enerate and preview the report to ensure it meets the requirements.



**RESULT**

Thus designing an application with Oracle Forms, Menus and Report Builder involves creating a user interface (UI) using Forms and generating reports using Report Builder has done successfully.