

# **CRICKET TOURNAMENT DATABASE**

**A PROJECT REPORT**

*Submitted by*

**YAFFIN S [RA2211032010053]**

**SAKINA RIZVI [RA2211032010073]**

**MITUN M [RA2211032010090]**

*Under the Guidance of*

**Dr. THANGA REVATHI S**

Assistant Professor, Department of Networking and Communications

*In partial fulfilment of the requirements for the degree of*

**BACHELOR OF TECHNOLOGY in**

**COMPUTER SCIENCE AND ENGINEERING**

**with a specialization in Internet of Things**



**DEPARTMENT OF NETWORKING AND COMMUNICATIONS**

**COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR – 603 203**

**MAY 2024**



**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR – 603 203**

**BONAFIDE CERTIFICATE**

Certified that this B.Tech project report titled “**Cricket Tournament Database**” is the bonafide work of “**YAFFIN S[RA2211032010053], SAKINA RIZVI[RA2211032010073] and MITUN M[RA2211032010090]**” who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

**Dr. THANGA REVATHI S**  
**SUPERVISOR**  
Assistant Professor  
Department of Networking and  
Communications.

**Dr. ANNAPURANI K**  
**HEAD OF THE DEPARTMENT**  
Professor  
Department of Networking and  
Communications.



**Department of Networking and Communications  
SRM Institute of Science and Technology**

**Own Work Declaration Form**

**Degree/ Course : B.Tech in Computer Science and Engineering with a  
specialization in Internet of Things.**

**Student Names : YAFFIN S, SAKINA RIZVI, MITUN M**

**Registration Number: RA2211032010053, RA2211032010073, RA2211032010090**

**Title of Work : CRICKET TOURNAMENT DATABASE**

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

We understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

**DECLARATION:**

We are aware of and understand the University's policy on Academic misconduct and plagiarism and we certify that this assessment is our own work, except where indicated by referring, and that we have followed the good academic practices noted above.

Student 1 Yaffin S (RA2211032010053)

Student 2 Sakina Rizvi (RA2211032010073)

Student 3 Mitun M(RA2211032010090)

## ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. Annapurani K**, Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We register our immeasurable thanks to our Faculty Advisor, **Dr. Praveena Akki**, Assistant Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. Thanga Revathi**, Associate Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Networking and Communications Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

**YAFFIN S[RA2211032010053]**

**SAKINA RIZVI[RA2211032010073]**

**MITUN M[RA2211032010090]**

## **ABSTRACT**

Cricket is the most popular sport in South Asian countries and the second most popular sport globally. Businesses have grown enormously based on cricketing sports events from the last decade. Also, coaches, sports analysts, and technicians get game facts and ideas about other teams, which help them make decisions and change plans accordingly. The Cricket Tournament Database System is a comprehensive and efficient solution designed to manage and streamline the operations of cricket tournaments. This system leverages modern database technologies to provide a centralized platform for storing, retrieving, and managing various aspects related to cricket tournaments. The primary objective is to enhance the overall efficiency, transparency, and organization of cricket tournaments, catering to the needs of tournament organizers, teams, players, and spectators.

## **PROBLEM STATEMENT**

The existing problem in managing cricket tournaments lies in the manual and disjointed processes of handling team information, player data, match details, and statistical analysis. Currently, there is a lack of centralized and automated systems to manage these aspects efficiently, leading to errors, data inconsistencies, and inefficiencies. The need for a Cricket Tournament Management System database arises from these challenges, aiming to streamline and automate the management of teams, players, matches, and statistics. By implementing a database system using SQL technologies like MySQL or PostgreSQL, the objective is to eliminate manual data handling, ensure data accuracy and integrity, and provide functionalities for easy retrieval, analysis, and reporting of tournament-related information.

## **TABLE OF CONTENTS**

<b>Ch. NO</b>	<b>TITLE</b>	<b>PAGE NO.</b>
1.	Introduction	<b>1</b>
2.	Objectives	<b>2</b>
3.	Requirement Analysis	<b>3</b>
4.	E-R Diagram	<b>4</b>
5.	Relational Database Schema	<b>8</b>
6.	Tables	<b>9</b>
7.	Working with Complex SQL queries	<b>23</b>
8.	Creating Views	<b>26</b>
9.	Working with PLSQL	<b>28</b>
10.	Working with Triggers	<b>33</b>
11.	Finding Functional Dependencies	<b>36</b>
12.	Normalization	<b>38</b>
13.	Working with the Transactions	<b>44</b>
14.	Conclusion	<b>47</b>
	References	<b>48</b>
	Annexure-1	<b>49</b>
	Annexure-2	<b>50</b>
	Annexure-3	<b>59</b>

# **CHAPTER 1**

## **INTRODUCTION**

This DBMS project is based on Cricket Tournament management. It provides various information about the various teams participating in the Tournament, in which various teams participate. It also provides us with information about the various players participating in the tournament. The database contains details of players, coaches and umpires among others. All the useful information about the entire World Cup can be found here.

Managing the intricacies of cricket tournaments, ranging from scheduling matches to tracking player statistics, has become a complex task. To address the challenges associated with organizing and overseeing cricket tournaments, the Cricket Tournament Database System serves as a pivotal solution.

Managing cricket tournament data poses several challenges, including fragmentation across various sources, inconsistent data quality, limited accessibility, manual data entry processes, lack of standardization, limited analytical capabilities, and scalability concerns. Addressing these challenges requires the development of a centralized database with comprehensive coverage, high data quality, accessible for all stakeholders, automated management processes, standardized formats, advanced analytics, and scalable infrastructure. This database would enhance the cricketing experience, drive innovation, and support growth within the sport.

The system caters to the diverse needs of tournament organizers, teams, players, and enthusiasts by providing a centralized and efficient database infrastructure.



## CHAPTER 2

### OBJECTIVES

❖ **Efficient Tournament Management:**

Allow organizers to define tournament formats, rules, and regulations, tailoring the system to meet the specific requirements of each event.

❖ **Comprehensive Information Management:**

Maintain a centralized database containing detailed information about participating teams and players, including historical statistics, performance metrics, and player profiles. Store and manage comprehensive data about matches, including schedules, venues, umpires, and match results.

❖ **Data Analysis and Reporting:**

Provide tools for in-depth analysis of tournament data, enabling organizers, teams, and analysts to derive valuable insights. Generate customizable reports on various aspects such as player performances, team standings, and overall tournament statistics.

❖ **Data Analysis and Reporting:**

Maintain an extensive historical archive of past tournaments, matches, and player performances, facilitating the tracking of cricketing milestones and trends over time. Allow users to retrieve and analyze historical data for research, comparison, and benchmarking purposes.

## **CHAPTER 3**

### **REQUIREMENT ANALYSIS**

#### **NEED FOR CRICKET TOURNAMENT DB:**

The need for a Cricket Database System aims to address the lack of an efficient and centralized means to manage and organize the vast and intricate data associated with the sport of cricket. Without such a system, there are challenges in maintaining comprehensive records of players, teams, matches, and related statistics. The absence of a structured platform makes it difficult for stakeholders, including players, coaches, analysts, fans, and administrators, to access and analyze historical and real-time cricket data easily. Manual record-keeping may lead to inaccuracies, and the absence of a standardized system can impede data-driven insights and decision-making processes.

#### **SOLUTION:**

The solution to the challenge of inefficient cricket data management lies in the development and implementation of a robust Cricket Database System. This system serves as a centralized hub, meticulously organizing comprehensive information about players, teams, matches, and statistics. By creating an intuitive and user-friendly interface, accessible to players, coaches, analysts, and administrators, the system ensures ease of use and widespread adoption. Detailed player profiles, team management modules, and match scheduling functionalities contribute to a comprehensive and well-structured database.

## CHAPTER 4

### ENTITY RELATIONSHIP DIAGRAM

#### 4.1 Entities

- 1) **Team** is an entity type which has many attributes like Team Name which uses the data type varchar. Every team has been given a Team ID which is the primary key which is of data type varchar. Team Ranking, Number of Batsmen and Number of Bowlers are of the data type number. There is another attribute - Wicketkeeper which is of multivalued type and accepts varchar data type. Primary key cannot have null value.
- 2) **Players** is an entity type which has an attribute – Player Name which is of the data type varchar. It has a primary key, Player ID, which cannot have null value. It has a foreign key, Team ID which is the primary key of the entity, Team. There is a complex attribute, Number of matches played, which comprises of Number of Test Matches, Number of T20 Matches, Number of World Cup Matches and Number of ODIs
- 3) **Batsman** is an entity type which has the attributes – Number of sixes hit, Number of Fours hit, the batting average, and the total runs scored. All of these attributes are of the data type number.
- 4) **Bowler** is an entity type which has the attribute – type of batsman with varchar data type. It also includes number of wickets and economy which are of the data type number.
- 5) **Umpire** is an entity type which has the attributes name and country of origin of data type varchar. The primary key of this is Umpire Id which is of varchar data type. It also has an attribute Number of matches of data type number.

- 6) **Coach** is an entity type with a foreign key, Team ID, which is a primary key of entity type, Team. It has a primary key, Coach ID, of data type varchar. It also has another attribute of data type varchar, Name.
- 7) **Captain** is an entity type with a primary key, Captain ID of data type varchar. It has two foreign keys, i) Player id from table Players and ii) Team ID from table Team. Number of years of captaincy and Number of wins are also attributes of this table of data type number.
- 8) **Matches** is an entity type with a primary key, match ID, of varchar data type. It has attributes like Team1 Name, Team2 Name, Stadium, Winner Team and Loser Team of data type varchar. Match date is an attribute which uses the datatype date. Match time is an attribute which is of the data type time.

## 4.2 RELATIONSHIPS

### 1) Cricket player plays in team (N-1)

A cricket player can play in only one team but a team can have many players in it but a team must have players in it. So, the relationship becomes (N-1).

### 2) Coach manages team(1-N)

Coach can manage a single team, but each team can have many coaches (like batting coach, fielding coach, bowling coach). But it is compulsory for a team to have a coach. So, the relationship is 1-N

### 3) Team plays match(M-N)

Team can play many matches and a match can be played by two teams. So, the relationship is M-N.

### 4) Matches are umpired by Umpire(M-N)

An umpire can umpire in many matches and a match can have two umpires. So, the relationship is M-N

### 5) Team headed by a Captain (1-1)

A team has 1 captain and a captain is from single team only. So the relationship is 1-1.

The below figure 4.1 depicts the Entity Relationship Diagram for the Cricket Tournament database.

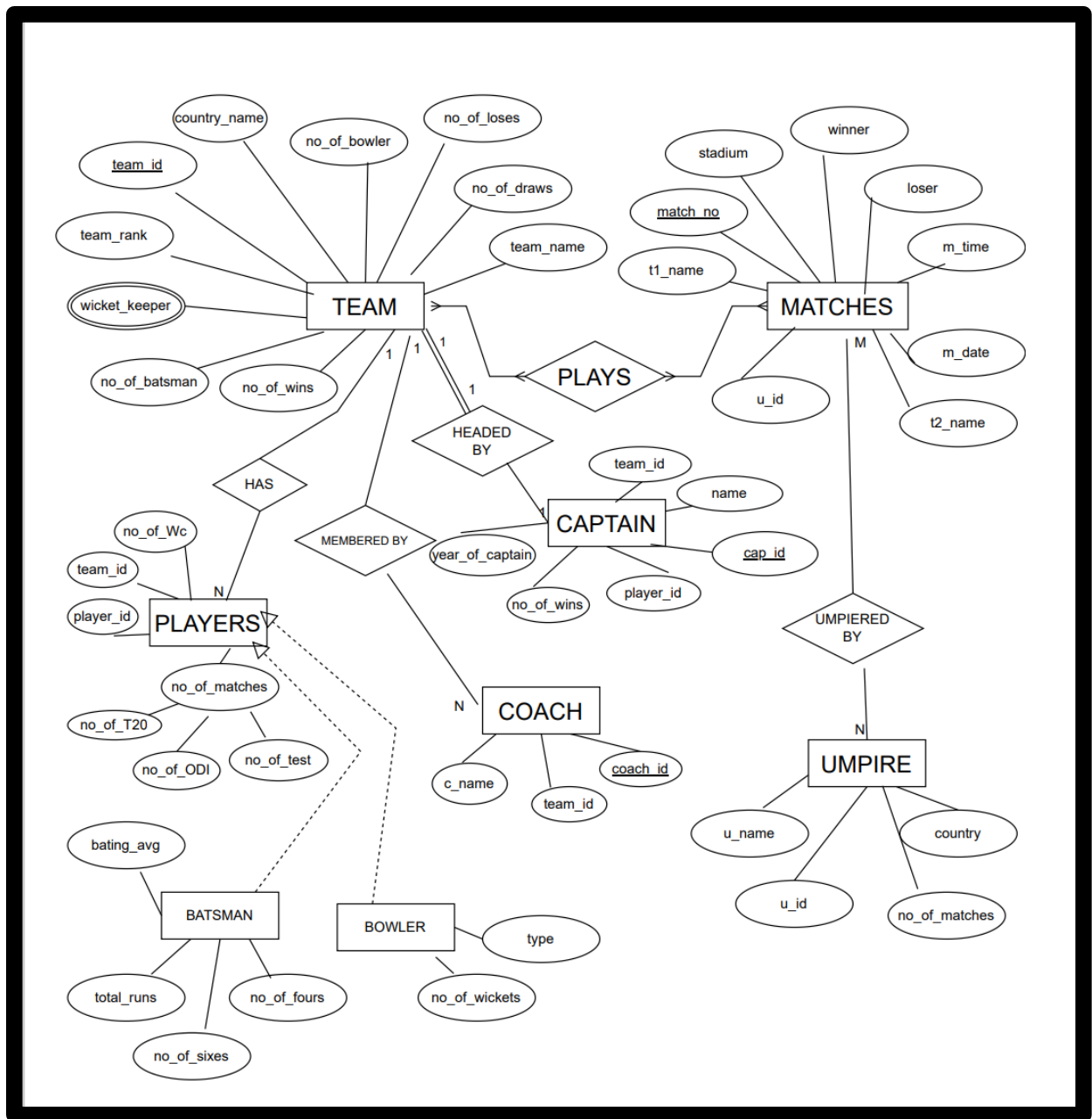


Figure 4.1

## CHAPTER 5

### RELATIONAL DATABASE SCHEMA

The below figure 5.1 shows the Relational Database Schema for Cricket Tournament Database

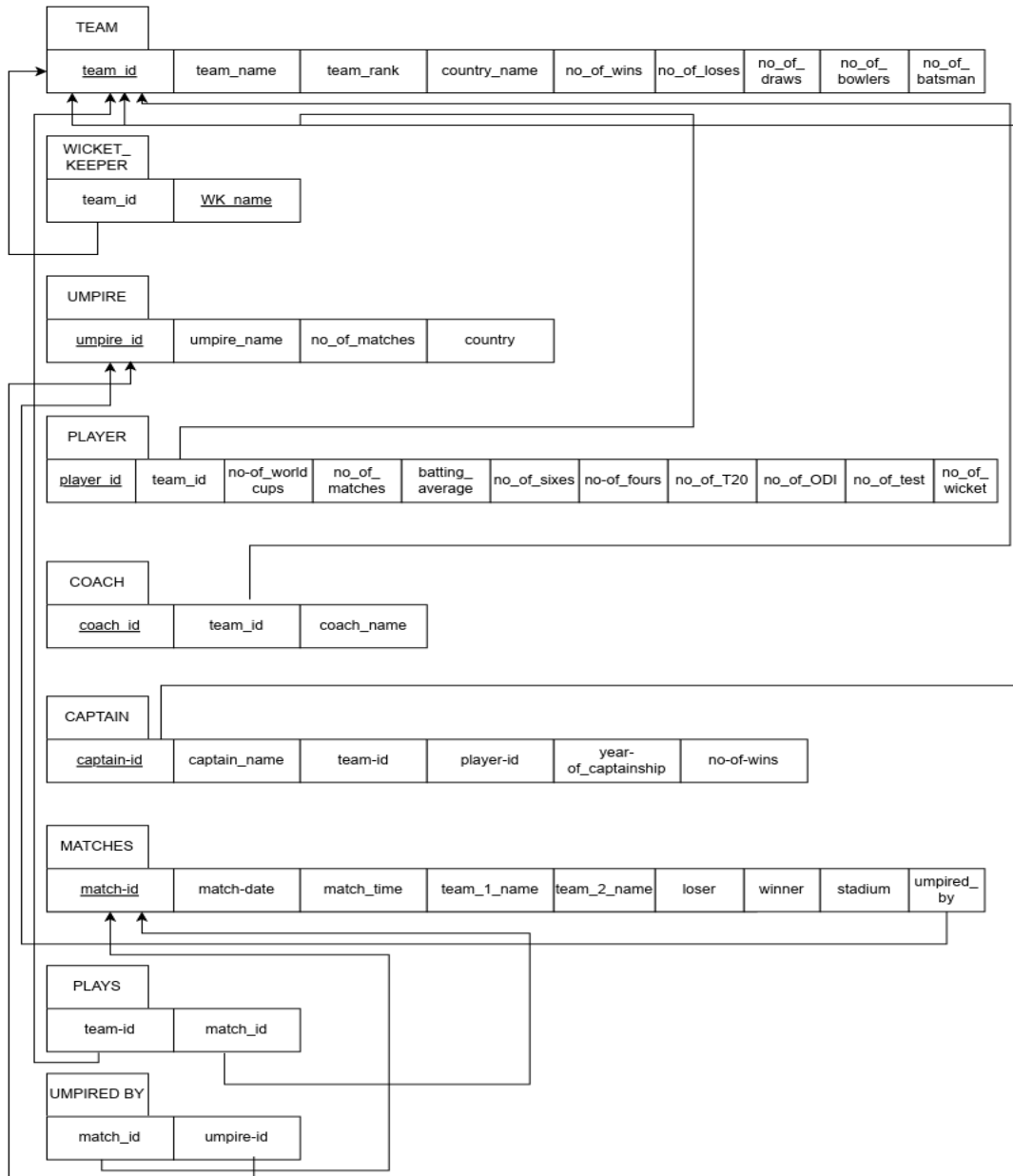


Figure 5.1

## CHAPTER 6

### TABLES

- Team
- Wicket\_Keeper
- Umpire
- Player
- Coach
- Captain
- Matches
- Plays
- Umpired\_By

#### 6.1. TABLE CREATION

##### 1. TABLE TEAM

The table TEAM has columns team\_id, team\_rank, team\_name, country\_name, no\_of\_wins, no\_of\_loses, no\_of\_draws, no\_of\_bowlers, no\_of\_batsman. as shown in fig 6.1.1

```
SQL> create table TEAM(  
 2  team_id varchar(30) primary key,  
 3  team_rank number(3),  
 4  teaam_name varchar(20) not null,  
 5  country_name varchar(20),  
 6  no_of_wins number(3),  
 7  no_of_loses number(3),  
 8  no_of_draws number(3),  
 9  no_of_bowlers number(2),  
10  no_of_batsmans number(2)  
11 );
```

Table created.

Figure 6.1.1



Describing the schema of the table TEAM as shown in fig 6.1.2

```
SQL> desc TEAM;
Name                                Null?    Type
-----
TEAM_ID                            NOT NULL VARCHAR2(30)
TEAM_RANK                           NUMBER(3)
TEAAM_NAME                          NOT NULL VARCHAR2(20)
COUNTRY_NAME                        VARCHAR2(20)
NO_OF_WINS                           NUMBER(3)
NO_OF_LOSES                          NUMBER(3)
NO_OF_DRAWS                          NUMBER(3)
NO_OF_BOWLERS                         NUMBER(2)
NO_OF_BATSMANS                       NUMBER(2)
```

Figure 6.1.2

## 2.TABLE WICKET\_KEEPER

The table WICKET\_KEEPER has the columns team\_id,wk\_name. as shown in fig 6.2.1

```
SQL> create table WICKET_KEEPER(
  2  team_id references TEAM,
  3  wk_name varchar(30)
  4  );

Table created.
```

Figure 6.2.1

Describing the schema of the table WICKET\_KEEPER as shown in fig 6.2.2

```
SQL> desc WICKET_KEEPER;
Name                                Null?    Type
-----
TEAM_ID                            VARCHAR2(30)
WK_NAME                            VARCHAR2(30)
```

Figure 6.2.2

### 3.TABLE UMPIRE

The table UMPIRE has the columns umpire\_id, umpire\_name, no\_of\_matches as shown in fig 6.3.1

```
SQL> create table UMPIRE(  
2  umpire_id varchar(30) primary key,  
3  umpire_name varchar(30),  
4  no_of_matches number(4),  
5  country varchar(20)  
6  );
```

Table created.

*Figure 6.3.1*

Describing the structure of the table UMPIRE as shown in fig 6.3.2

```
SQL> desc UMPIRE;  
Name                               Null?   Type  
-----  
UMPIRE_ID                          NOT NULL VARCHAR2(30)  
UMPIRE_NAME                        VARCHAR2(30)  
NO_OF_MATCHES                      NUMBER(4)  
COUNTRY                           VARCHAR2(20)
```

*Figure 6.3.2*

#### 4.TABLE PLAYER

The table player has the attributes player\_id, team\_id, no\_of\_worldcups, no\_of\_matches, batting\_average, no\_of\_sixes, no\_of\_fours, no\_of\_totalruns, no\_of\_t20, no\_of\_odi, no\_of\_test, no\_of\_wickets, type\_of\_bowler as shown in fig 6.4.1

```
SQL> create table PLAYER(  
  2  player_id varchar(30) primary key,  
  3  team_id references TEAM,  
  4  no_of_worldcups number(2),  
  5  no_of_matches number(3),  
  6  batting_average number(3),  
  7  no_of_sixes number(3),  
  8  no_of_fours number(3),  
  9  no_of_totalruns number(4),  
10  no_of_t20 number(3),  
11  no_of_odi number(3),  
12  no_of_test number(3),  
13  no_of_wickets number(2),  
14  type_of_bowler varchar(30),  
15  economy number(3)  
16 );
```

Table created.

Figure 6.4.1

Describing the table PLAYER as shown in the fig 6.4.2

```
SQL> desc PLAYER;  
Name                               Null?   Type  
-----  
PLAYER_ID                          NOT NULL VARCHAR2(30)  
TEAM_ID                            VARCHAR2(30)  
NO_OF_WORLDDCUPS                    NUMBER(2)  
NO_OF_MATCHES                       NUMBER(3)  
BATTING_AVERAGE                    NUMBER(3)  
NO_OF_SIXES                         NUMBER(3)  
NO_OF_FOURS                         NUMBER(3)  
NO_OF_TOTALRUNS                     NUMBER(4)  
NO_OF_T20                           NUMBER(3)  
NO_OF_ODI                           NUMBER(3)  
NO_OF_TEST                          NUMBER(3)  
NO_OF_WICKETS                       NUMBER(2)  
TYPE_OF_BOWLER                      VARCHAR2(30)  
ECONOMY                             NUMBER(3)
```

Figure 6.4.2

## 5.TABLE COACH

The table COACH has the columns coach\_id, team\_id, coach\_name. as shown in fig 6.5.1

```
SQL> create table COACH(  
2   coach_id varchar(30) primary key,  
3   team_id references TEAM,  
4   coach_name varchar(30)  
5   );
```

Table created.

Figure 6.5.1

Describing the table COACH as shown in fig 6.5.2

```
SQL> desc COACH;  
Name                               Null?      Type  
-----  
COACH_ID                           NOT NULL   VARCHAR2(30)  
TEAM_ID                                       VARCHAR2(30)  
COACH_NAME                                    VARCHAR2(30)
```

Figure 6.5.2

## 6.TABLE CAPTAIN

The table CAPTAIN has the attributes captain-id, captain\_name, team-id, player\_id, year\_of\_captaincy, no\_of\_wins. as shown in fig 6.6.1

```
SQL> create table CAPTAIN(  
2   captain_id varchar(30) primary key,  
3   captain_name varchar(30),  
4   team_id references Team,  
5   player_id varchar(30),  
6   year_of_captaincy number(2),  
7   no_of_wins number(4)  
8   );
```

Table created.

Figure 6.6.1

Describing the table CAPTAIN as shown in fig 6.6.2

```
SQL> desc CAPTAIN;
```

Name	Null?	Type
CAPTAIN_ID	NOT NULL	VARCHAR2(30)
CAPTAIN_NAME		VARCHAR2(30)
TEAM_ID		VARCHAR2(30)
PLAYER_ID		VARCHAR2(30)
YEAR_OF_CAPTAINCY		NUMBER(2)
NO_OF_WINS		NUMBER(4)

Figure 6.6.2

## 7.TABLE UMPIRE

The table MATCHES has the attributes match\_id, match\_date, match\_time, team\_1\_name, Team\_2\_name, loser, winner, stadium, umpire\_id. as shown in fig 6.7.1

```
SQL> create table MATCHES(  
2  match_id varchar(20) primary key,  
3  match_date date,  
4  match_time timestamp(0),  
5  team_1_name varchar(30),  
6  team_2_name varchar(30),  
7  loser varchar(30),  
8  winner varchar(30),  
9  stadium varchar(30),  
10 umpire_id references umpire  
11 );  
  
Table created.
```

Figure 6.7.1

Describing the table MATCHES as shown in fig 6.7.2

```
SQL> desc MATCHES;
```

Name	Null?	Type
MATCH_ID	NOT NULL	VARCHAR2(20)
MATCH_DATE		DATE
MATCH_TIME		TIMESTAMP(0)
TEAM_1_NAME		VARCHAR2(30)
TEAM_2_NAME		VARCHAR2(30)
LOSER		VARCHAR2(30)
WINNER		VARCHAR2(30)
STADIUM		VARCHAR2(30)
UMPIRE_ID		VARCHAR2(30)

Figure 6.7.2

## 8.TABLE PLAYS

The Table PLAYS has the attributes team\_id, match-id as shown in fig 6.8.1

```
SQL> create table PLAYS(  
  2  team_id references TEAM,  
  3  match_id references MATCHES  
  4  );  
  
Table created.
```

Figure 6.8.1

Describing the table PLAYS. as shown in fig 6.8.2

```
SQL> desc PLAYS;  
Name                               Null?   Type  
-----  
TEAM_ID                            VARCHAR2(30)  
MATCH_ID                           VARCHAR2(20)
```

Figure 6.8.2

## 9.TABLE UMPIRED\_BY

The table UMPIRED\_BY has the attributes match\_id, umpire\_id. as shown in fig 6.9.1

```
SQL> create table UMPIRED_BY(  
  2  match_id references MATCHES,  
  3  umpire_id references UMPIRE  
  4  );  
  
Table created.
```

Figure 6.9.1

Describing the table UMPIRED\_BY as shown in fig 6.9.2

```
SQL> desc UMPIRED_BY;  
Name                               Null?   Type  
-----  
MATCH_ID                           VARCHAR2(20)  
UMPIRE_ID                           VARCHAR2(30)
```

Figure 6.9.2

## 6.2 TABLE RECORDS/VALUES

### 1. TABLE TEAM

40 Teams are entered with distinct team\_id along with their team rank, country name, no of wins, no of loses, no of draws, no of bowlers and no of batsman. as shown in fig 6.2.1

```
SQL> select * from TEAM;
```

TEAM_ID	TEAM_RANK	TEAM_NAME	COUNTRY_NAME	NO_OF_WINS	NO_OF_LOSES	NO_OF_DRAWS	NO_OF_BOWLERS	NO_OF_BATSMANS
MI01	1	Mumbai Indians	India	120	90	6	10	12
CSK02	2	Chennai Super Kings	India	110	95	5	11	11
RCB03	3	Royal Challengers Bangalore	India	100	100	4	9	13
KKR04	4	Kolkata Knight Riders	India	95	105	5	10	12
DC05	5	Delhi Capitals	India	90	110	5	9	13
KXIP06	6	Kings XI Punjab	India	85	115	4	10	12
RR07	7	Rajasthan Royals	India	80	120	6	9	13
SRH08	8	Sunrisers Hyderabad	India	85	115	4	10	12
PWI09	9	Pune Warriors	India	70	130	6	8	14
GL10	10	Gujarat Lions	India	75	125	5	9	13
RPS11	11	Rising Pune Supergiants	India	80	120	4	8	14
DC12	12	Deccan Chargers	India	65	135	5	7	15
CSK13	13	Superkings	India	105	95	6	11	11
KKR14	14	Knight Riders	India	100	100	5	10	12
KXIP15	15	Kings	India	95	105	4	9	13
RCB16	16	Royal Challengers	India	90	110	5	9	13
MI17	17	Indians	India	85	115	6	9	13
SRH18	18	Sunrisers	India	90	110	4	10	12
RR19	19	Royals	India	80	120	5	9	13
DC20	20	Capitals	India	85	115	4	10	12
CSK21	21	Super Kings	India	95	105	6	10	12
KKR22	22	Knight Riders 2	India	105	95	5	11	11
RCB23	23	Royal	India	110	90	4	10	12
MI24	24	Indians 2	India	120	80	6	11	11
KXIP25	25	Kings 2	India	115	85	5	11	11
SRH26	26	Sunrisers 2	India	105	95	4	11	11
RR27	27	Royals 2	India	95	105	6	11	11
RCB28	28	Royal 2	India	100	100	5	11	11
MI29	29	Indians 3	India	110	90	4	11	11
KKR30	30	Knight Riders 3	India	120	80	6	11	11
DC31	31	Capitals 2	India	115	85	5	11	11
CSK32	32	Super Kings 2	India	125	75	4	11	11
SRH33	33	Sunrisers 3	India	130	70	6	11	11
RR34	34	Royals 3	India	135	65	5	11	11
RCB35	35	Royal 3	India	140	60	4	11	11
MI36	36	Indians 4	India	145	55	6	11	11
KKR37	37	Knight Riders 4	India	150	50	5	11	11
KXIP38	38	Kings 3	India	155	45	4	11	11
SRH39	39	Sunrisers 4	India	160	40	6	11	11
RR40	40	Royals 4	India	165	35	5	11	11

40 rows selected.

Figure 6.2.1

## 2. TABLE WICKET\_KEEPER

40 Wicket keeper names are added to the table wicket keeper along with their team id. as shown in fig 6.2.2

TEAM_ID	WK_NAME
MI01	John Smith
CSK02	David Johnson
RCB03	Michael Brown
KKR04	Christopher Lee
DC05	Rohit Sharma
KXIP06	Joshua Martin
RR07	Daniel Wilson
SRH08	Andrew Anderson
PWI09	Robert Thompson
GL10	Anthony Harris
RPS11	Thomas White
DC12	Ryan Martinez
CSK13	William Clark
KKR14	Joseph Rodriguez
MI17	Virat Kohli
CSK21	David Hall
RCB16	Brian Young
KKR22	Paul Walker
DC20	James Allen
KXIP15	Steven King
RR19	Jason Green
SRH18	Scott Evans
PWI09	Brandon Martinez
GL10	Eric Johnson
RPS11	Justin Harris
DC05	Christopher Moore
CSK32	Jonathan Thompson
KKR14	Kevin Carter
MI29	Ricky Ponting
CSK13	Samuel Wilson
RCB03	Nathan Lopez
KKR04	Alexander Rodriguez
DC12	Nicholas Garcia
CSK02	Zachary Martinez
KKR14	Benjamin Wilson
MI01	Steve Smith
SRH08	Adam Johnson
RR07	Edward Brown
RCB16	Gregory Martinez
MI29	Shane Watson

Figure 6.2.2

## 3. TABLE UMPIRE

40 records of distinct umpire\_id along with umpire name, no of matches they have umpired to the table umpire as shown in fig 6.2.3

SQL> select * from UMPIRE;			
UMPIRE_ID	UMPIRE_NAME	NO_OF_MATCHES	COUNTRY
UMP65102	Marais Erasmus	82	India
UMP21903	Richard Illingworth	29	India
UMP12704	Paul Reiffel	72	India
UMP93005	Bruce Oxenford	65	India
UMP37486	Nigel Llong	52	India
UMP82907	Joel Wilson	32	India
UMP56208	Rod Tucker	66	India
UMP43109	Chris Gaffaney	47	India
UMP62310	Kumar Dharmasena	103	India
UMP74111	Aleem Dar	207	India
UMP52412	Anil Chaudhary	19	India
UMP85213	Ian Gould	140	India
UMP85214	Tony Hill	96	India
UMP29015	Billy Bowden	200	India
UMP68016	Sundaram Ravi	115	India
UMP57017	Chettithody Shamshuddin	68	India
UMP45018	Michael Gough	30	India
UMP21019	Ruchira Palliyaguruge	13	India
UMP78920	Nitin Menon	23	India
UMP54021	Ajit Agarkar	26	India
UMP90122	Simon Taufel	74	India
UMP32723	Srinivas Venkataraghavan	73	India
UMP20824	Venkat Sundaram	45	India
UMP63425	S. Venkataraghavan	89	India
UMP81526	A. V. Jayaprakash	51	India
UMP92027	Rudi Koertzen	108	India
UMP10528	Billy Bowden	137	India
UMP28629	Steve Bucknor	119	India
UMP29130	Simon Taufel	143	India
UMP84231	Aleem Dar	201	India
UMP44032	Rod Tucker	96	India
UMP52633	Kumar Dharmasena	113	India
UMP75934	Richard Kettleborough	121	India
UMP26735	Nigel Llong	37	India
UMP14936	Chris Gaffaney	69	India
UMP22737	Marais Erasmus	94	India
UMP61038	Paul Reiffel	86	India
UMP76539	Bruce Oxenford	78	India
UMP42640	Joel Wilson	57	India

Figure 6.2.3



#### 4. TABLE PLAYER

Since we have 40 Teams in our cricket database and each team has 22 players so totally  $40 \times 22$  thus 880 player records are added to the table player. As shown in fig 6.2.4

```
SQL> select PLAYER_ID,TEAM_ID,NO_OF_MATCHES,BATTING_AVERAGE,TYPE_OF_BOWLER from PLAYER;
```

PLAYER_ID	TEAM_ID	NO_OF_MATCHES	BATTING_AVERAGE	TYPE_OF_BOWLER
PLR00355	KXIP25	2	44	slow
PLR00356	KXIP25	11	95	legspin
PLR00357	KXIP25	14	37	medium-slow
PLR00358	KXIP25	19	78	slow
PLR00359	KXIP25	12	47	slow
PLR00360	KXIP25	9	60	legspin
PLR00361	KXIP25	15	62	legspin
PLR00362	KXIP25	15	18	slow
PLR00363	KXIP25	3	31	slow
PLR00364	KXIP25	8	15	slow
PLR00365	KXIP25	5	11	slow
PLR00366	KXIP25	7	23	legspin
PLR00367	KXIP25	8	99	slow
PLR00368	KXIP25	8	96	medium-slow
PLR00369	KXIP25	7	69	medium-slow
PLR00370	KXIP25	2	31	legspin
PLR00371	KXIP25	16	68	fast
PLR00372	KXIP25	12	60	slow
PLR00373	KXIP25	13	49	medium
PLR00374	KXIP25	19	96	slow
PLR00375	KXIP38	7	22	slow
PLR00376	KXIP38	5	67	slow
PLR00377	KXIP38	6	41	medium-slow
PLR00378	KXIP38	17	86	medium
PLR00379	KXIP38	12	44	medium
PLR00380	KXIP38	10	71	slow
PLR00381	KXIP38	14	98	slow
PLR00382	KXIP38	7	32	fast
PLR00383	KXIP38	2	99	legspin
PLR00384	KXIP38	14	95	fast
PLR00385	KXIP38	4	52	legspin
PLR00386	KXIP38	10	91	fast
PLR00387	KXIP38	10	86	medium-slow
PLR00388	KXIP38	7	61	medium-slow
PLR00389	KXIP38	6	85	medium-slow
PLR00390	KXIP38	3	15	medium
PLR00391	KXIP38	6	21	fast
PLR00392	KXIP38	13	98	legspin
PLR00393	KXIP38	16	92	medium
PLR00394	KXIP38	9	97	legspin
PLR00395	KXIP38	14	99	medium

Figure 6.2.4

PLR00878	SRH39	2	13	85	33
legspin		4			
PLR00879	SRH39	1	5	93	39
medium-slow		4			
PLR00880	SRH39	1	15	24	33
medium	19				

880 rows selected.

## 5. TABLE COACH

In the table coach 40 records of coaches are added with distinct coach id as shown in fig 6.2.5

```
SQL> select * from COACH;
```

COACH_ID	TEAM_ID	COACH_NAME
CH871	MI01	John Wright
CH932	CSK02	Stephen Fleming
CH576	RCB03	Simon Katich
CH315	KKR04	Brendon McCullum
CH609	DC05	Ricky Ponting
CH834	KXIP06	Anil Kumble
CH249	RR07	Kumar Sangakkara
CH503	SRH08	Tom Moody
CH127	PWI09	Allan Donald
CH669	GL10	Brad Hodge
CH742	RPS11	Stephen Fleming
CH951	DC12	Tom Moody
CH224	CSK13	Stephen Fleming
CH398	KKR14	Brendon McCullum
CH817	KXIP15	Anil Kumble
CH573	RCB16	Simon Katich
CH640	MI17	Mahela Jayawardene
CH122	SRH18	Trevor Bayliss
CH495	RR19	Andrew McDonald
CH308	DC20	Ricky Ponting
CH719	CSK21	Stephen Fleming
CH853	KKR22	Brendon McCullum
CH632	RCB23	Simon Katich
CH491	MI24	Mahela Jayawardene
CH934	KXIP25	Anil Kumble
CH227	SRH26	Trevor Bayliss
CH768	RR27	Andrew McDonald
CH529	RCB28	Simon Katich
CH406	MI29	Mahela Jayawardene
CH635	KKR30	Brendon McCullum
CH792	DC31	Ricky Ponting
CH899	CSK32	Stephen Fleming
CH263	SRH33	Trevor Bayliss
CH514	RR34	Andrew McDonald
CH372	RCB35	Simon Katich
CH920	MI36	Mahela Jayawardene
CH725	KKR37	Brendon McCullum
CH656	KXIP38	Anil Kumble
CH547	SRH39	Trevor Bayliss
CH874	RR40	Andrew McDonald

40 rows selected.

Figure 6.2.5

## 6. TABLE CAPTAIN

40 entries with distinct captain id are added along with team id which is used to map with team table and player id which is used to map with player table. As shown in fig 6.2.6

```
SQL> select * from CAPTAIN;
```

CAPTAIN_ID	CAPTAIN_NAME	TEAM_ID	PLAYER_ID	YEAR_OF_CAPTAINCY	NO_OF_WINS
CAP11333	Virat Kohli	RCB03	PLR33889	6	65
CAP21499	Rohit Sharma	MI01	PLR11223	8	85
CAP30287	Kane Williamson	SRH08	PLR87654	7	75
CAP14892	Shakib Al Hasan	KKR04	PLR66543	5	60
CAP36924	Quinton de Kock	MI17	PLR99321	4	45
CAP41567	David Warner	SRH18	PLR22145	6	70
CAP50432	KL Rahul	KXIP06	PLR12345	5	55
CAP25149	Shreyas Iyer	DC12	PLR87643	3	35
CAP60421	Steve Smith	RR19	PLR98123	7	80
CAP35687	Dinesh Karthik	KKR22	PLR65432	4	40
CAP42365	Ajinkya Rahane	RR27	PLR22334	6	65
CAP55432	Eoin Morgan	KKR30	PLR55667	5	55
CAP28765	Aaron Finch	RCB16	PLR44556	3	30
CAP65142	Faf du Plessis	CSK13	PLR99876	4	45
CAP73215	Jason Holder	SRH26	PLR66578	6	70
CAP87752	Ravichandran Ashwin	DC31	PLR55443	5	60
CAP93312	Moeen Ali	CSK32	PLR45678	4	40
CAP14762	Kieron Pollard	MI36	PLR66555	7	80
CAP26743	Chris Gayle	KXIP38	PLR88777	6	65
CAP36823	AB de Villiers	RCB35	PLR11222	5	55
CAP49923	Bhuvneshwar Kumar	SRH39	PLR33221	4	45
CAP50111	Hardik Pandya	MI24	PLR66778	6	70
CAP79234	Kagiso Rabada	DC05	PLR44789	5	60
CAP65438	Rishabh Pant	DC20	PLR66576	4	40
CAP89651	Jasprit Bumrah	MI29	PLR99887	7	80
CAP71257	Ravindra Jadeja	CSK21	PLR33244	6	65
CAP84762	Sunil Narine	KKR14	PLR88766	5	55
CAP63274	Andre Russell	KKR37	PLR99888	4	45
CAP92381	Imran Tahir	CSK02	PLR66577	6	70
CAP44583	Rahul Chahar	MI17	PLR55466	5	60
CAP85732	Yuzvendra Chahal	RCB23	PLR66789	4	40
CAP67458	Trent Boult	MI01	PLR22111	7	80
CAP23987	Pat Cummins	KKR04	PLR33255	6	65
CAP78546	Jofra Archer	RR07	PLR66579	5	55
CAP92875	Mohammed Shami	KXIP15	PLR22122	4	45
CAP55723	Rashid Khan	SRH08	PLR77665	6	70
CAP44891	Chris Morris	RR34	PLR99889	5	60
CAP63654	David Miller	RR19	PLR66776	4	40
CAP81345	Shikhar Dhawan	DC12	PLR44554	7	80

Figure 6.2.6

## 7. TABLE MATCHES

Here match id is added with unique entries and match date entries are added with date datatype using TO\_DATE('12-03-2023', 'DD-MM-YYYY') and match time using TO\_TIMESTAMP('15:30', 'HH24:MI'). as shown in fig 6.2.7

```
SQL> select MATCH_ID,MATCH_DATE,TEAM_1_NAME,TEAM_2_NAME,WINNER,STADIUM,UMPIRE_ID from MATCHES;
```

MATCH_ID	MATCH_DATE	TEAM_1_NAME	TEAM_2_NAME	WINNER	STADIUM	UMPIRE_ID
MAT101	12-MAR-23	Mumbai Indians	Royal Challengers Bangalore	Mumbai Indians	Feroz Shah Kotla	UMP65102
MAT201	15-MAR-23	Chennai Super Kings	Kings XI Punjab	Kings XI Punjab	Eden Gardens	UMP21903
MAT301	21-MAR-23	Royal Challengers Bangalore	Kolkata Knight Riders	Royal Challengers Bangalore	M.A. Chidambaram	UMP12704
MAT401	23-MAR-23	Kolkata Knight Riders	Sunrisers Hyderabad	Kolkata Knight Riders	Sardar Patel	UMP93005
MAT501	26-MAR-23	Rajasthan Royals	Mumbai Indians	Mumbai Indians	Wankhede	UMP37406
MAT601	29-MAR-23	Mumbai Indians	Sunrisers Hyderabad	Mumbai Indians	M.Chinnaswamy Stadium	UMP82907
MAT701	02-APR-23	Royal Challengers Bangalore	Pune Warriors	Royal Challengers Bangalore	Punjab Cricket Association IS Bindra Stadium	UMP56208
MAT801	05-APR-23	Chennai Super Kings	Delhi Capitals	Delhi Capitals	Rajiv Gandhi International Cricket Stadium	UMP43109
MAT901	08-APR-23	Kolkata Knight Riders	Rajasthan Royals	Kolkata Knight Riders	Holkar Cricket Stadium	UMP62310
MAT1001	11-APR-23	Royal Challengers Bangalore	Kings XI Punjab	Royal Challengers Bangalore	Saurashtra Cricket Association Stadium	UMP74111
MAT1101	14-APR-23	Sunrisers Hyderabad	Chennai Super Kings	Sunrisers Hyderabad	M. A. Chidambaram Stadium	UMP52412
MAT1201	17-APR-23	Mumbai Indians	Kolkata Knight Riders	Mumbai Indians	Wankhede Stadium	UMP85213
MAT1301	20-APR-23	Royal Challengers Bangalore	Delhi Capitals	Royal Challengers Bangalore	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium	UMP55214
MAT1401	23-APR-23	Rajasthan Royals	Royal Challengers Bangalore	Rajasthan Royals	Holkar Cricket Stadium	UMP29015
MAT1501	26-APR-23	Kings XI Punjab	Mumbai Indians	Kings XI Punjab	Saurashtra Cricket Association Stadium	UMP68016
MAT1601	29-APR-23	Kolkata Knight Riders	Sunrisers Hyderabad	Kolkata Knight Riders	M. Chinnaswamy Stadium	UMP57017
MAT1701	02-MAY-23	Royal Challengers Bangalore	Kolkata Knight Riders	Royal Challengers Bangalore	Punjab Cricket Association IS Bindra Stadium	UMP45018
MAT1801	05-MAY-23	Kolkata Knight Riders	Sunrisers Hyderabad	Kolkata Knight Riders	Rajiv Gandhi International Cricket Stadium	UMP21019
MAT1901	08-MAY-23	Delhi Capitals	Rajasthan Royals	Delhi Capitals	Holkar Cricket Stadium	UMP78920
MAT2001	11-MAY-23	Mumbai Indians	Royal Challengers Bangalore	Mumbai Indians	Sardar Patel	UMP54021
MAT2101	14-MAY-23	Royal Challengers Bangalore	Delhi Capitals	Royal Challengers Bangalore	M. A. Chidambaram Stadium	UMP90122
MAT2201	17-MAY-23	Sunrisers Hyderabad	Mumbai Indians	Sunrisers Hyderabad	Wankhede Stadium	UMP32723
MAT2301	20-MAY-23	Rajasthan Royals	Kolkata Knight Riders	Rajasthan Royals	Eden Gardens	UMP20824
MAT2401	23-MAY-23	Kings XI Punjab	Sunrisers Hyderabad	Kings XI Punjab	Punjab Cricket Association IS Bindra Stadium	UMP63425
MAT2501	26-MAY-23	Chennai Super Kings	Delhi Capitals	Chennai Super Kings	Rajiv Gandhi International Cricket Stadium	UMP81526
MAT2601	29-MAY-23	Mumbai Indians	Royal Challengers Bangalore	Mumbai Indians	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium	UMP92027
MAT2701	01-JUN-23	Kolkata Knight Riders	Sunrisers Hyderabad	Kolkata Knight Riders	M. Chinnaswamy Stadium	UMP10528
MAT2801	04-JUN-23	Delhi Capitals	Rajasthan Royals	Delhi Capitals	Punjab Cricket Association IS Bindra Stadium	UMP28629
MAT2901	07-JUN-23	Royal Challengers Bangalore	Kings XI Punjab	Royal Challengers Bangalore	Rajiv Gandhi International Cricket Stadium	UMP29130
MAT3001	10-JUN-23	Kolkata Knight Riders	Sunrisers Hyderabad	Kolkata Knight Riders	M. Chinnaswamy Stadium	UMP84231
MAT3101	13-JUN-23	Mumbai Indians	Delhi Capitals	Mumbai Indians	Wankhede Stadium	UMP44032
MAT3201	16-JUN-23	Sunrisers Hyderabad	Chennai Super Kings	Sunrisers Hyderabad	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium	UMP52633
MAT3301	19-JUN-23	Rajasthan Royals	Royal Challengers Bangalore	Rajasthan Royals	Sardar Patel	UMP75934
MAT3401	22-JUN-23	Delhi Capitals	Mumbai Indians	Delhi Capitals	Wankhede Stadium	UMP36735
MAT3501	25-JUN-23	Kolkata Knight Riders	Sunrisers Hyderabad	Kolkata Knight Riders	Eden Gardens	UMP14936
MAT3601	28-JUL-23	Royal Challengers Bangalore	Mumbai Indians	Royal Challengers Bangalore	M. Chinnaswamy Stadium	UMP22737
MAT3701	01-JUL-23	Kolkata Knight Riders	Delhi Capitals	Kolkata Knight Riders	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium	UMP65038
MAT3801	04-JUL-23	Mumbai Indians	Royal Challengers Bangalore	Mumbai Indians	Rajiv Gandhi International Cricket Stadium	UMP76539
MAT3901	07-JUL-23	Sunrisers Hyderabad	Kolkata Knight Riders	Sunrisers Hyderabad	M. A. Chidambaram Stadium	UMP42640

Figure 6.2.7

## 8. TABLE PLAYS

Each team plays more than one match so the team id cannot be unique and for an match two team are needed here match id also cannot be unique as shown in fig 6.2.8

```
SQL> select * from PLAYS;
```

TEAM_ID	MATCH_ID
MI01	MAT101
MI01	MAT501
CSK02	MAT201
CSK02	MAT401
CSK02	MAT501
CSK02	MAT701
RCB03	MAT301
RCB03	MAT601
KKR04	MAT101
KKR04	MAT401
KKR04	MAT501
KKR04	MAT701
DC05	MAT201
DC05	MAT501
DC05	MAT701
KXIP06	MAT301
KXIP06	MAT601
RR07	MAT101
RR07	MAT501
RR07	MAT701
SRH08	MAT201
SRH08	MAT401
SRH08	MAT701
PWI09	MAT301
PWI09	MAT501
PWI09	MAT601
PWI09	MAT701
GL10	MAT101
GL10	MAT501
GL10	MAT601
RPS11	MAT201
RPS11	MAT401
RPS11	MAT501
RPS11	MAT701
DC12	MAT301
DC12	MAT501
DC12	MAT701
CSK13	MAT101
CSK13	MAT401
CSK13	MAT501

Figure 6.2.8

## 9. TABLE UMPIRED\_BY

Each match is umpired by an umpire so the 40 entries of match id and umpire id are add to the table umpire as shown in fig 6.2.9

```
SQL> select * from UMPIRED_BY;
```

MATCH_ID	UMPIRE_ID
MAT101	UMP65102
MAT201	UMP21903
MAT301	UMP12704
MAT401	UMP93005
MAT501	UMP37406
MAT601	UMP82907
MAT701	UMP56208
MAT801	UMP43109
MAT901	UMP62310
MAT1001	UMP74111
MAT1201	UMP85213
MAT1301	UMP55214
MAT1401	UMP29015
MAT1501	UMP68016
MAT1601	UMP57017
MAT1701	UMP45018
MAT1801	UMP21019
MAT1901	UMP78920
MAT2001	UMP54021
MAT2101	UMP90122
MAT2201	UMP32723
MAT2301	UMP20824
MAT2401	UMP63425
MAT2501	UMP81526
MAT2601	UMP92027
MAT2701	UMP10528
MAT2801	UMP28629
MAT2901	UMP29130
MAT3001	UMP84231
MAT3101	UMP44032
MAT3201	UMP52633
MAT3301	UMP75934
MAT3401	UMP36735
MAT3501	UMP14936
MAT3601	UMP22737
MAT3701	UMP65038
MAT3801	UMP76539
MAT3901	UMP42640

Figure 6.2.9

## CHAPTER 7

### SQL QUERIES USING UPDATE/JOIN/NESTING/SET OPERATIONS

#### 1. USING UPDATE COMMAND

Give an SQL query to add an column TOTAL\_NO\_OF\_MATCHES to the table team and update the rows using total matches= number of WINS+ number of LOSES + number of DRAWS. As shown in fig 7.1

```
SQL> alter table team add total_matches number(5);  
Table altered.  
  
SQL> update team set total_matches=no_of_draws + no_of_wins + no_of_loses;  
40 rows updated.
```

*Figure 7.1*

#### 2. USING EMBEDDED/NESTING SELECT

Write an SQL query to display the UMPIRE NAMES who has umpired the matches in the month of MARCH ?

QUERY: select UMPIRE\_NAME from UMPIRE where UMPIRE\_ID in (select UMPIRE\_ID from MATCHES where MATCH\_DATE like '%MAR%');

Subquery Explanation:

- a. SELECT UMPIRE\_ID FROM MATCHES WHERE MATCH\_DATE LIKE '%MAR%': This subquery selects the UMPIRE\_ID values from the MATCHES table where the MATCH\_DATE column contains the substring 'MAR', indicating matches that occurred in March.

Main Query:

- b. SELECT UMPIRE\_NAME FROM UMPIRE WHERE UMPIRE\_ID IN (...): This main query selects the UMPIRE\_NAME from the UMPIRE table.
- c. The IN clause is used to check if the UMPIRE\_ID from the UMPIRE table exists in the result set obtained from the subquery.

Result:

- d. The query returns the UMPIRE\_NAMES corresponding to the UMPIRE\_IDs retrieved from the subquery.
- e. This provides a list of umpire names who officiated matches that occurred in March. As shown in fig 7.2

```
SQL> select UMPIRE_NAME from UMPIRE where UMPIRE_ID in (select UMPIRE_ID from MATCHES where MATCH_DATE like '%MAR%');

UMPIRE_NAME
-----
Marais Erasmus
Richard Illingworth
Paul Reiffel
Bruce Oxenford
Nigel Llong
Joel Wilson

6 rows selected.
```

*Figure 7.2*

### 3. SELECT USING INNER JOIN

Write an SQL query to display name of coach who has coached a player with total\_runs greater than 500;

QUERY: select distinct coach\_name from coach inner join player on coach.team\_id=player.team\_id where(player.no\_of\_totalruns>500);

- SELECT DISTINCT coach\_name: This part of the main query specifies that you want to select unique coach names (coach\_name).
- FROM coach: Indicates that you are selecting data from the coach table.
- INNER JOIN player ON coach.team\_id = player.team\_id: This is the join condition that connects the coach table with the player table based on the team\_id column. It ensures that only rows with matching team\_id values from both tables are included in the result set.
- WHERE player.no\_of\_totalruns > 500: This filter condition is applied to the joined data. It specifies that you only want rows where the no\_of\_totalruns column in the player table is greater than 500.

Result Explanation:

The query will return a list of distinct coach names (coach\_name) who are associated with teams where at least one player has scored more than 500 total runs. As shown in fig 7.3

```
SQL> select distinct coach_name from coach inner join player on coach.team_id=player.team_id where(player.no_of_totalruns>500);

COACH_NAME
-----
Andrew McDonald
John Wright
Anil Kumble
Brad Hodge
Mahela Jayawardene
Trevor Bayliss
Allan Donald
Ricky Ponting
Kumar Sangakkara
Simon Katich
Brendon McCullum
Stephen Fleming
Tom Moody
13 rows selected
```

Figure 7.3

#### 4. SELECT USING JOINING THE COMMON ATTRIBUTE

Write an SQL query to Display name of wicket keeper who is also the captain of his team

QUERY:           select           wk\_name           from           wicket\_keeper,captain           where  
wicket\_keeper.wk\_name=captain.captain\_name;

- SELECT wk\_name: Specifies that we want to retrieve the wk\_name column from the result set.
- FROM wicket\_keeper, captain: Specifies the tables from which we are retrieving data, using a comma to indicate a Cartesian product (cross join) between the two tables. This means every row from wicket\_keeper is paired with every row from captain.
- WHERE wicket\_keeper.wk\_name = captain.captain\_name: Adds a condition to the cross join. It filters the rows where the wk\_name in wicket\_keeper is equal to captain\_name in captain, effectively joining the two tables based on this condition.

Result:

- The result of this query will be the list of wk\_name values where a wicket keeper is also a team captain, based on the matching names between wicket\_keeper and captain as shown in fig 7.4

```
SQL> select wk_name from wicket_keeper,captain where wicket_keeper.wk_name=captain.captain_name;

WK_NAME
-----
Rohit Sharma
Ajinkya Rahane
Virat Kohli
Chris Gayle
Steve Smith
```

Figure 7.4



## CHAPTER 8

### Creating Views

#### What are Views?

views are virtual tables that represent the result of a stored query. They are not stored as a part of the database schema but are dynamically generated when they are queried. Views can be used to simplify complex queries, provide a layer of security by restricting access to specific columns or rows of a table, and present a customized perspective of the data for different users or applications. They help in separating the logical and physical layers of the database, enhancing data abstraction and organization.

#### View 1

We give an PRESS report about the top ten team of the series we create an view as shown in fig 8.1

```
SQL> create view TOP_10_TEAM as select teaam_name from team where team_rank<=10;
View created.

SQL> SELECT * FROM TOP_10_TEAM;
TEAAM_NAME
-----
Mumbai Indians
Chennai Super Kings
Royal Challengers Bangalore
Kolkata Knight Riders
Delhi Capitals
Kings XI Punjab
Rajasthan Royals
Sunrisers Hyderabad
Pune Warriors
Gujarat Lions

10 rows selected.
```

Figure 8.1

## View 2

Winners for the month of JUNE as shown in fig 8.2

```
SQL> CREATE VIEW WINNERS_OF_JUNE AS SELECT WINNER FROM MATCHES WHERE MATCH_DATE LIKE '%JUN%';
View created.

SQL> SELECT * FROM WINNERS_OF_JUNE;

WINNER
-----
Kolkata Knight Riders
Delhi Capitals
Royal Challengers Bangalore
Kolkata Knight Riders
Mumbai Indians
Sunrisers Hyderabad
Rajasthan Royals
Delhi Capitals
Kolkata Knight Riders
Royal Challengers Bangalore

10 rows selected.
```

*Figure 8.2*

## CHAPTER 9

### WORKING WITH PL/SQL

#### What is PL/SQL?

PL/SQL (Procedural Language/Structured Query Language) is Oracle's extension to SQL that allows developers to write procedural code within the database, enabling tasks such as control flow, loops, exception handling, and the creation of stored procedures and functions. It provides tight integration with SQL for efficient data manipulation and is commonly used for developing database-centric applications on Oracle platforms.

#### 1. PL/SQL Code to add records to the table player.

Declare Variables:

- `i INT := 0`: Initialize a counter for player IDs.
- `team_id VARCHAR2(30);`: Variable to hold each team ID fetched from the team table.
- `team_cursor SYS_REFCURSOR;`: Declare a cursor variable to fetch team IDs from the team table.

Open Cursor:

- `OPEN team_cursor FOR SELECT TEAM_ID FROM team;`: Open a cursor to fetch all `TEAM_ID` values from the team table.

Loop through Teams:

- `LOOP`: Start an infinite loop to iterate through team IDs.
- `FETCH team_cursor INTO team_id;`: Fetch the next `TEAM_ID` from the cursor into the `team_id` variable.
- `EXIT WHEN team_cursor%NOTFOUND;`: Exit the loop when there are no more rows to fetch from the cursor.

Generate Player Data:

- `Nested Loop (FOR j IN 1..22 LOOP)`: This loop generates 22 random player entries for each team.
- Inside the loop, `i` is incremented to generate unique player IDs using `PLR` followed by a padded number.
- Random values are generated for various player attributes such as player type, age, runs, wickets, etc., using `DBMS_RANDOM.VALUE` function.

Insert Data into `PLAYER` Table:

- `INSERT INTO PLAYER ...`: Inserts the randomly generated player data into the `PLAYER` table for each team.

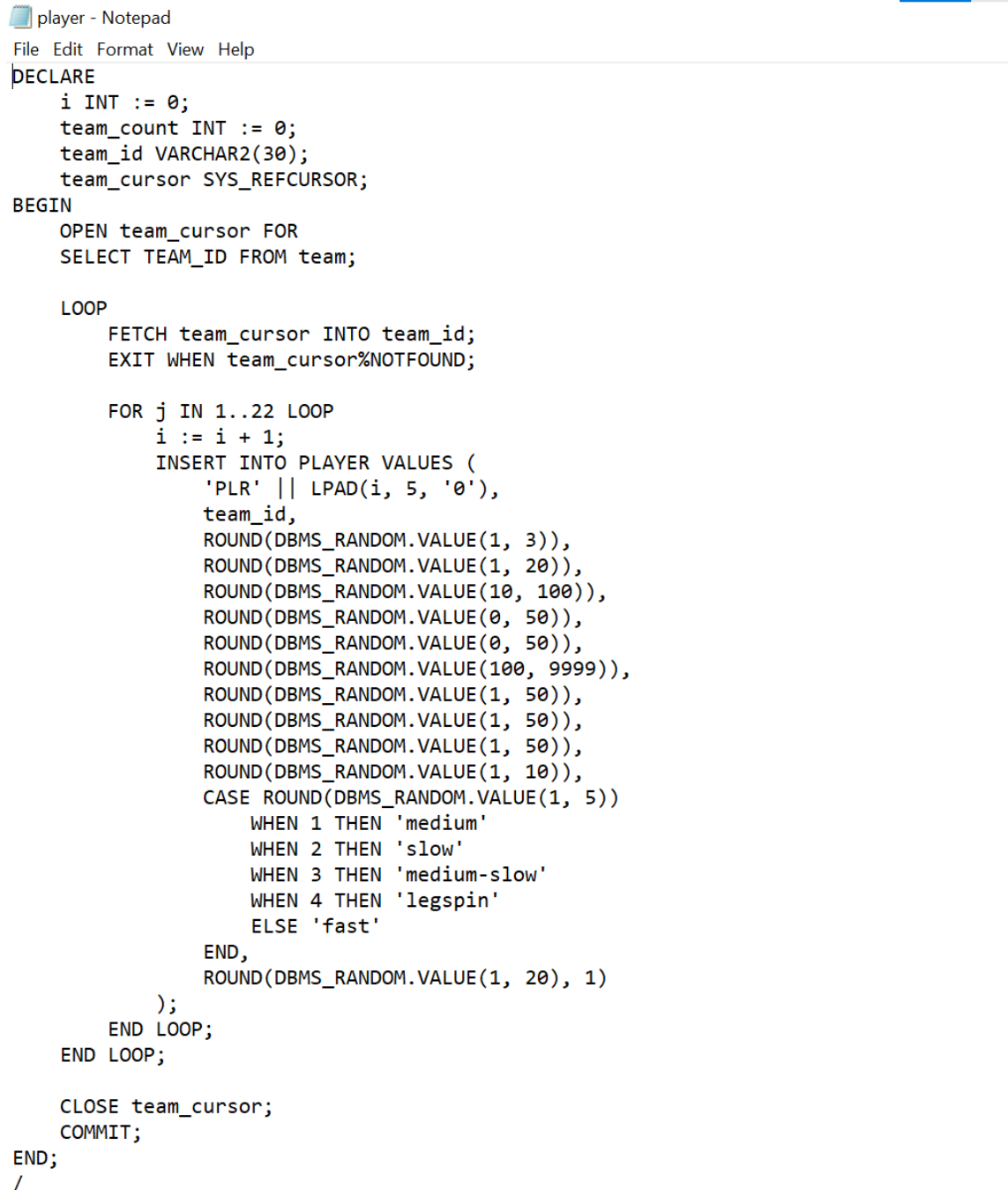
End of Loops:

- The outer loop (`END LOOP;`) continues until all teams are processed.
- The inner loop generates players for each team.

Close Cursor and Commit:

- CLOSE team\_cursor;: Close the cursor after processing all teams.
- COMMIT;: Commit the changes to make them permanent in the database.

The below figure 9.1 depicts the implementation of the above cursor



```
player - Notepad
File Edit Format View Help
DECLARE
    i INT := 0;
    team_count INT := 0;
    team_id VARCHAR2(30);
    team_cursor SYS_REFCURSOR;
BEGIN
    OPEN team_cursor FOR
    SELECT TEAM_ID FROM team;

    LOOP
        FETCH team_cursor INTO team_id;
        EXIT WHEN team_cursor%NOTFOUND;

        FOR j IN 1..22 LOOP
            i := i + 1;
            INSERT INTO PLAYER VALUES (
                'PLR' || LPAD(i, 5, '0'),
                team_id,
                ROUND(DBMS_RANDOM.VALUE(1, 3)),
                ROUND(DBMS_RANDOM.VALUE(1, 20)),
                ROUND(DBMS_RANDOM.VALUE(10, 100)),
                ROUND(DBMS_RANDOM.VALUE(0, 50)),
                ROUND(DBMS_RANDOM.VALUE(0, 50)),
                ROUND(DBMS_RANDOM.VALUE(100, 9999)),
                ROUND(DBMS_RANDOM.VALUE(1, 50)),
                ROUND(DBMS_RANDOM.VALUE(1, 50)),
                ROUND(DBMS_RANDOM.VALUE(1, 50)),
                ROUND(DBMS_RANDOM.VALUE(1, 10)),
                CASE ROUND(DBMS_RANDOM.VALUE(1, 5))
                    WHEN 1 THEN 'medium'
                    WHEN 2 THEN 'slow'
                    WHEN 3 THEN 'medium-slow'
                    WHEN 4 THEN 'legspin'
                    ELSE 'fast'
                END,
                ROUND(DBMS_RANDOM.VALUE(1, 20), 1)
            );
        END LOOP;
    END LOOP;

    CLOSE team_cursor;
    COMMIT;
END;
/
```

Figure 9.

## 2. PL/SQL Code to retrieve details of the matches which are held at Sardar patel stadium.

In this PL/SQL block:

- We declare a variable v\_stadium\_name to hold the stadium name we want to search for.
- The FOR loop fetches all records from the MATCHES table where the stadium name matches v\_stadium\_name.
- Inside the loop, we use DBMS\_OUTPUT.PUT\_LINE to display details of each match such as Match ID, Match Date, Match Time, Team names, Winner, Loser, Stadium, and Umpire ID.
- The loop iterates through each match record that matches the stadium name criteria. The below figure 9.2 depicts the above cursor

```
SQL> @matches.sql
Match ID: MAT401
Match Date: 23-MAR-2023
Match Time: 15:30:00
Team 1: Kolkata Knight Riders
Team 2: Sunrisers Hyderabad
Winner: Kolkata Knight Riders
Loser: Sunrisers Hyderabad
Stadium: Sardar Patel
Umpire ID: UMP93005
-----
Match ID: MAT2001
Match Date: 11-MAY-2023
Match Time: 10:00:00
Team 1: Mumbai Indians
Team 2: Royal Challengers Bangalore
Winner: Mumbai Indians
Loser: Royal Challengers Bangalore
Stadium: Sardar Patel
Umpire ID: UMP54021
-----
Match ID: MAT3301
Match Date: 19-JUN-2023
Match Time: 15:30:00
Team 1: Rajasthan Royals
Team 2: Royal Challengers Bangalore
Winner: Rajasthan Royals
Loser: Royal Challengers Bangalore
Stadium: Sardar Patel
Umpire ID: UMP75934
-----
PL/SQL procedure successfully completed.
```

```

matches - Notepad
File Edit Format View Help
DECLARE
  v_stadium_name VARCHAR2(60) := 'Sardar Patel';
BEGIN
  FOR match_rec IN (
    SELECT *
    FROM MATCHES
    WHERE STADIUM = v_stadium_name
  )
  LOOP
    DBMS_OUTPUT.PUT_LINE('Match ID: ' || match_rec.MATCH_ID);
    DBMS_OUTPUT.PUT_LINE('Match Date: ' || TO_CHAR(match_rec.MATCH_DATE, 'DD-MON-YYYY'));
    DBMS_OUTPUT.PUT_LINE('Match Time: ' || TO_CHAR(match_rec.MATCH_TIME, 'HH24:MI:SS'));
    DBMS_OUTPUT.PUT_LINE('Team 1: ' || match_rec.TEAM_1_NAME);
    DBMS_OUTPUT.PUT_LINE('Team 2: ' || match_rec.TEAM_2_NAME);
    DBMS_OUTPUT.PUT_LINE('Winner: ' || match_rec.WINNER);
    DBMS_OUTPUT.PUT_LINE('Loser: ' || match_rec.LOSER);
    DBMS_OUTPUT.PUT_LINE('Stadium: ' || match_rec.STADIUM);
    DBMS_OUTPUT.PUT_LINE('Umpire ID: ' || match_rec.UMPIRE_ID);
    DBMS_OUTPUT.PUT_LINE('-----');
  END LOOP;
END;
/

```

Figure 9.2

### 3. PL/SQL Code to update the captain id of the captain table.

This PL/SQL block will update the captain ID from "CAP63654" to "CAP99999" in the "CAPTAIN" table. Make sure to execute this script in your SQL environment, and it will display "Captain ID updated successfully." if the update is successful. If there's an error during the update process, it will display the error message. The COMMIT statement is used to commit the changes to the database, and the ROLLBACK statement is there to handle exceptions by rolling back the changes if an error occurs. The below figure 9.3 demonstrates the above

```

SQL> @ captain.sql
Captain ID updated successfully.

PL/SQL procedure successfully completed.

SQL> select * from captain where captain_id='CAP99999';

```

CAPTAIN_ID	CAPTAIN_NAME	TEAM_ID	PLAYER_ID	YEAR_OF_CAPTAINCY	NO_OF_WINS
CAP99999	David Miller	RR19	PLR66776	4	40



captain - Notepad

File Edit Format View Help

```
DECLARE
    v_old_captain_id VARCHAR2(10) := 'CAP63654';
    v_new_captain_id VARCHAR2(10) := 'CAP99999';
BEGIN
    UPDATE CAPTAIN
    SET CAPTAIN_ID = v_new_captain_id
    WHERE CAPTAIN_ID = v_old_captain_id;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Captain ID updated successfully.');
```

EXCEPTION

```
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
        ROLLBACK;
END;
/
```

Figure 9.3

## CHAPTER 10

### Working With Triggers

#### What are Triggers?

Triggers in database management systems are special stored procedures that automatically execute in response to specific events, such as INSERT, UPDATE, DELETE, or user logins. They enforce data integrity, implement business rules, and automate tasks, with row-level triggers acting on each affected row and statement-level triggers operating on the overall event, providing essential automation and logic enforcement in database operations.

#### Trigger 1

When a team is eliminated, do the necessary process and update the elimination table.

- This trigger is designed to capture information about deleted rows from the team table and store it in the elimination table.
- When a row is deleted from team, the trigger fires (AFTER DELETE) and inserts a corresponding row into the elimination table using the old values (:OLD) of the deleted row's columns.
- The trigger's logic assumes that elimination table has columns corresponding to team\_id, country\_name, team\_rank, team\_name, and no\_of\_loses, and it uses the old values (:OLD) of these columns from the deleted row in team to populate the elimination table.

The below figure 10.1.1 and 10.1.2 shows the demonstration of the above trigger.

```
SQL> insert into team values('test12345',55,'test_team','india',1,1,1,22,22,3);
1 row created.

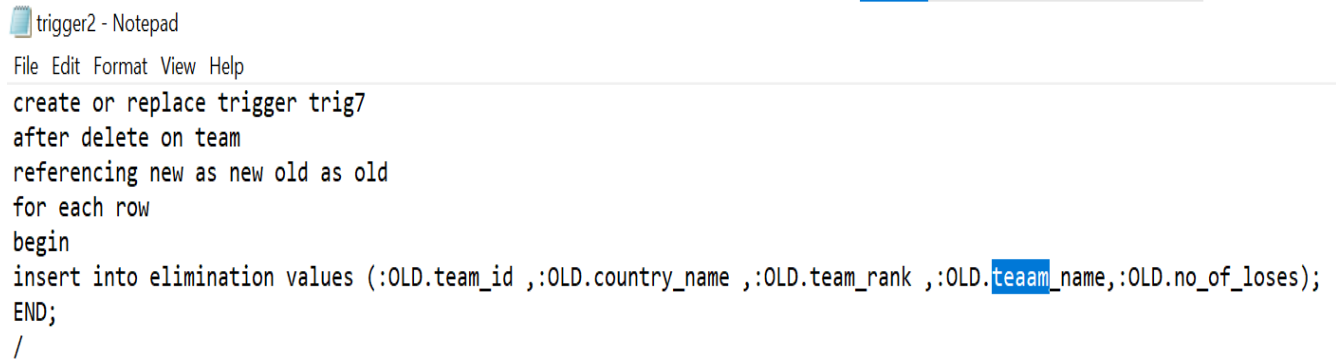
SQL> DELETE FROM team WHERE team_id = 'test12345';
1 row deleted.

SQL> select * from elimination;
```

TEAMID	CNTRY_NAME	RANKK	TEAMNAME	NOLOSES
test123	INDIA	41	testkk	1
test12345	india	55	test_team	1

Figure 10.1.1





```

trigger2 - Notepad
File Edit Format View Help
create or replace trigger trig7
after delete on team
referencing new as new old as old
for each row
begin
insert into elimination values (:OLD.team_id ,:OLD.country_name ,:OLD.team_rank ,:OLD:teaam_name,:OLD.no_of_loses);
END;
/

```

*Figure 10.1.2*

## Trigger 2

Due to some malpractices a team was banned for 2 years . After 2 years when it came back the board of cricket council order to change the team\_ID because of some reasons

Explanation:

- **CREATE OR REPLACE TRIGGER:** This statement creates or replaces a trigger named REFERENCE1.
- **AFTER UPDATE ON team:** The trigger is fired after an UPDATE operation is performed on the team table.
- **FOR EACH ROW:** Indicates that the trigger fires once for each row affected by the UPDATE operation.
- **BEGIN ... END:** Encloses the trigger's executable statements.
- **UPDATE ... SET ... WHERE ...:** These are the SQL statements inside the trigger's body.
  - Each UPDATE statement updates the team\_id in a related table (player, coach, captain, plays, wicket\_keeper) based on the old team\_id (:old.team\_id) and the new team\_id (:new.team\_id) from the team table.
  - For example, when a team\_id is updated in the team table, this trigger ensures that all corresponding records in related tables are also updated to reflect the new team\_id.

This trigger is useful for maintaining data integrity and consistency across related tables when there are changes to the team\_id in the team table. It helps to synchronize the team\_id values across different tables that are linked by this foreign key relationship

```
SQL> update team set team_id='SRH39' where team_id='SRH556';
1 row updated.
SQL> SELECT * FROM PLAYER WHERE TEAM_ID LIKE '%SRH39%';
```

PLAYER_ID	TYPE_OF_BOWLER	TEAM_ID	ECONOMY	NO.
PLR00859	legspin	SRH39	11	
PLR00860	legspin	SRH39	14	
PLR00861	legspin	SRH39	5	
PLR00862	slow	SRH39	13	
PLR00863	medium	SRH39	19	
PLR00864	medium	SRH39	8	
PLR00865	fast	SRH39	2	
PLR00866	slow	SRH39	4	
PLR00867	legspin	SRH39	10	
PLR00868	medium-slow	SRH39	14	
PLR00869	fast	SRH39	9	
PLR00870	medium-slow	SRH39	15	
PLR00871	medium-slow	SRH39	14	
PLR00872	legspin	SRH39	9	
PLR00873	slow	SRH39	10	
PLR00874	medium	SRH39	4	
PLR00875	legspin	SRH39	16	
PLR00876	legspin	SRH39	16	
PLR00877	legspin	SRH39	2	

Figure 10.2.1

```
trigger3 - Notepad
File Edit Format View Help
Create or replace trigger reference1
after update on team
for each row
begin
update player set team_id=:new.team_id where
team_id=:old.team_id;
update coach set team_id=:new.team_id
where team_id=:old.team_id;
update captain set team_id=:new.team_id where
team_id=:old.team_id;
update plays set team_id=:new.team_id where
team_id=:old.team_id;
update wicket_keeper set team_id=:new.team_id where
team_id=:old.team_id;
end;
/ |
```

Figure 10.2.2

## CHAPTER 11

### What are Functional Dependencies?

Functional dependencies are a fundamental concept in database management systems (DBMS) that describe the relationships between attributes (columns) in a relation (table). A functional dependency exists when the value of one attribute uniquely determines the value of another attribute(s) in the same table. In other words, if knowing the value of attribute A allows you to determine the value of attribute B, then A functionally determines B, represented as  $A \rightarrow B$ .

### FINDING FUNCTIONAL DEPENDENCIES

#### 1. Table TEAM

A.  $TEAM\_ID \rightarrow \{TEAM\_RANK, TEAM\_NAME, COUNTRY\_NAME, NO\_OF\_WINS, NO\_OF\_LOSES, NO\_OF\_DRAWS, NO\_OF\_BOWLERS, NO\_OF\_BATSMEN, TOTAL\_MATCHES\}$

#### 2. Table WICKET\_KEEPER

A.  $TEAM\_ID \rightarrow WK\_NAME$   
B.  $WK\_NAME \rightarrow TEAM\_ID$

#### 3. Table UMPIRE

A.  $UMPIRE\_ID \rightarrow \{UMPIRE\_NAME, NO\_OF\_MATCHES, COUNTRY\}$   
B.  $UMPIRE\_NAME \rightarrow \{UMPIRE\_ID, NO\_OF\_MATCHES, COUNTRY\}$   
C.  $UMPIRE\_ID, UMPIRE\_NAME \rightarrow \{NO\_OF\_MATCHES, COUNTRY\}$

#### 4. Table PLAYER

A.  $PLAYER\_ID \rightarrow \{TEAM\_ID, NO\_OF\_WORLD\_CUPS, NO\_OF\_MATCHES, BATTING\_AVERAGE, NO\_OF\_SIXES, NO\_OF\_FOURS, NO\_OF\_TOTAL\_RUNS, NO\_OF\_T20, NO\_OF\_ODI, NO\_OF\_TEST, NO\_OF\_WICKETS, TYPE\_OF\_BOWLER, ECONOMY\}$   
B.  $TEAM\_ID \rightarrow \{PLAYER\_ID\}$

## 5. Table COACH

- A. COACH\_ID -> {TEAM\_ID, COACH\_NAME}
- B. TEAM\_ID -> {COACH\_ID, COACH\_NAME}

## 6. Table CAPTAIN

- A. CAPTAIN\_ID -> {CAPTAIN\_NAME, TEAM\_ID, PLAYER\_ID, YEAR\_OF\_CAPTAINCY, NO\_OF\_WINS}
- B. TEAM\_ID -> {CAPTAIN\_ID, CAPTAIN\_NAME, PLAYER\_ID, YEAR\_OF\_CAPTAINCY, NO\_OF\_WINS}
- C. PLAYER\_ID -> {CAPTAIN\_ID, CAPTAIN\_NAME, TEAM\_ID, YEAR\_OF\_CAPTAINCY, NO\_OF\_WINS}

## 7. Table MATCHES

- A. MATCH\_ID -> {MATCH\_DATE, MATCH\_TIME, TEAM\_1\_NAME, TEAM\_2\_NAME, LOSER, WINNER, STADIUM, UMPIRE\_ID}
- B. TEAM\_1\_NAME, TEAM\_2\_NAME -> {MATCH\_ID, MATCH\_DATE, MATCH\_TIME, LOSER, WINNER, STADIUM, UMPIRE\_ID}
- C. UMPIRE\_ID -> {MATCH\_ID, MATCH\_DATE, MATCH\_TIME, TEAM\_1\_NAME, TEAM\_2\_NAME, LOSER, WINNER, STADIUM}

## CHAPTER 12

### NORMALIZATION

#### 1. Table MATCHES

The initial schema for the table is given by the following figure 12.1 shown below

```
SQL> desc matches;
```

Name	Null?	Type
MATCH_ID	NOT NULL	VARCHAR2(20)
MATCH_DATE		DATE
MATCH_TIME		TIMESTAMP(0)
TEAM_1_NAME		VARCHAR2(30)
TEAM_2_NAME		VARCHAR2(30)
LOSER		VARCHAR2(30)
WINNER		VARCHAR2(30)
STADIUM		VARCHAR2(60)
UMPIRE_ID		VARCHAR2(30)

Figure 12.1

The table is already in **1NF** since all the instances of the attributes are **atomic and no multi values** exist.

Identifying the Functional Dependencies

- A. MATCH\_ID -> MATCH\_DATE, MATCH\_TIME, TEAM\_1\_NAME, TEAM\_2\_NAME, LOSER, WINNER, STADIUM, UMPIRE\_ID (**CLOSURE**)
- B. MATCH\_DATE, MATCH\_TIME -> MATCH\_ID (**PARTIAL DEPENDENCY**)

Since PARTIAL DEPENDENCY exist we are undergoing 2NF and splitting the relation as follows

## First Table: Matches2NF

```
SQL> CREATE TABLE Matches2NF (  
 2     MATCH_ID VARCHAR2(20) PRIMARY KEY,  
 3     MATCH_DATE DATE,  
 4     MATCH_TIME TIMESTAMP(0),  
 5     STADIUM VARCHAR2(60),  
 6     UMPIRE_ID VARCHAR2(30)  
 7 );  
Table created.
```

```
SQL> desc matches2nf;  
Name                               Null?   Type  
-----  
MATCH_ID                           NOT NULL VARCHAR2(20)  
MATCH_DATE                         DATE  
MATCH_TIME                         TIMESTAMP(0)  
STADIUM                            VARCHAR2(60)  
UMPIRE_ID                          VARCHAR2(30)
```

## Second Table: Match Results

```
SQL> CREATE TABLE MatchResults (  
 2     MATCH_ID VARCHAR2(20) PRIMARY KEY,  
 3     TEAM_1_NAME VARCHAR2(30),  
 4     TEAM_2_NAME VARCHAR2(30),  
 5     LOSER VARCHAR2(30),  
 6     WINNER VARCHAR2(30),  
 7     FOREIGN KEY (MATCH_ID) REFERENCES Matches2NF(MATCH_ID)  
 8 );  
Table created.
```

```
SQL> desc matchresults;  
Name                               Null?   Type  
-----  
MATCH_ID                           NOT NULL VARCHAR2(20)  
TEAM_1_NAME                        VARCHAR2(30)  
TEAM_2_NAME                        VARCHAR2(30)  
LOSER                             VARCHAR2(30)  
WINNER                             VARCHAR2(30)
```

In the above decomposition:

- The "Matches2NF" table contains information directly related to each match, with MATCH\_ID as the primary key.
- The "Match Results" table contains information related to the teams, winner, and loser, with MATCH\_ID as a foreign key referencing the "Matches" table.

This decomposition helps in reducing data redundancy and ensures that each table represents a distinct entity without partial dependencies.

Now Analysing the Functional Dependencies for the table Matches2NF

- a. MATCH\_DATE -> MATCH\_TIME

b. UMPIRE\_ID -> MATCH\_ID

The combination of these dependencies create a **transitive dependency**

MATCH\_DATE -> MATCH\_ID <- UMPIRE\_ID. This means that UMPIRE\_ID is **transitively** dependent on MATCH\_DATE through MATCH\_ID.

Since TRANSITIVE DEPENDENCY exist we are undergoing 3NF and splitting the relation as follows

#### First Table: Matches3NF

```
SQL> CREATE TABLE Matches3NF (  
  2     MATCH_ID VARCHAR2(20) PRIMARY KEY,  
  3     MATCH_DATE DATE,  
  4     MATCH_TIME TIMESTAMP(0)  
  5 );  
  
Table created.
```

```
SQL> DESC MATCHES3NF;  


| Name       | Null?    | Type         |
|------------|----------|--------------|
| MATCH_ID   | NOT NULL | VARCHAR2(20) |
| MATCH_DATE |          | DATE         |
| MATCH_TIME |          | TIMESTAMP(0) |


```

#### Second Table: Match Results 3NF

```
SQL> CREATE TABLE MatchDetails3NF (  
  2     MATCH_ID VARCHAR2(20),  
  3     STADIUM VARCHAR2(60),  
  4     UMPIRE_ID VARCHAR2(30),  
  5     PRIMARY KEY (MATCH_ID),  
  6     FOREIGN KEY (MATCH_ID) REFERENCES Matches2NF(MATCH_ID)  
  7 );  
  
Table created.
```

There are no transitive dependencies in these tables, and each table represents a distinct entity with no non-key attributes functionally depending on other non-key attributes within the same table. This structure is in Third Normal Form (3NF).

Hence the Table MATCHES is finally **NORMALIZED**

## 2. Table CAPTAIN

The initial schema for the table is given by the following figure 12.2 shown below

```
SQL> desc captain;
```

Name	Null?	Type
CAPTAIN_ID	NOT NULL	VARCHAR2(30)
CAPTAIN_NAME		VARCHAR2(30)
TEAM_ID		VARCHAR2(30)
PLAYER_ID		VARCHAR2(30)
YEAR_OF_CAPTAINCY		NUMBER(2)
NO_OF_WINS		NUMBER(4)

Figure 12.2

The table is already in **1NF** since all the instances of the attributes are **atomic and no multi values** exist.

Identifying the Functional Dependencies

- A. CAPTAIN\_ID -> {CAPTAIN\_NAME, TEAM\_ID, PLAYER\_ID, YEAR\_OF\_CAPTAINCY, NO\_OF\_WINS} (**CLOSURE**)
- B. CAPTAIN\_ID -> TEAM\_ID (**PARTIAL DEPENDENCY**)
- C. CAPTAIN\_ID -> PLAYER\_ID (**PARTIAL DEPENDENCY**)

Since PARTIAL DEPENDENCY exist we are undergoing 2NF and splitting the relation as follows

### First Table: Captain2NF

SQL> CREATE TABLE Captain2NF ( 2     CAPTAIN_ID VARCHAR2(30) PRIMARY KEY, 3     CAPTAIN_NAME VARCHAR2(30), 4     YEAR_OF_CAPTAINCY NUMBER(2), 5     NO_OF_WINS NUMBER(4) 6 );	SQL> desc Captain2NF;																		
Table created.	<table><tr><th>Name</th><th>Null?</th><th>Type</th></tr><tr><td colspan="3">-----</td></tr><tr><td>CAPTAIN_ID</td><td>NOT NULL</td><td>VARCHAR2(30)</td></tr><tr><td>CAPTAIN_NAME</td><td></td><td>VARCHAR2(30)</td></tr><tr><td>YEAR_OF_CAPTAINCY</td><td></td><td>NUMBER(2)</td></tr><tr><td>NO_OF_WINS</td><td></td><td>NUMBER(4)</td></tr></table>	Name	Null?	Type	-----			CAPTAIN_ID	NOT NULL	VARCHAR2(30)	CAPTAIN_NAME		VARCHAR2(30)	YEAR_OF_CAPTAINCY		NUMBER(2)	NO_OF_WINS		NUMBER(4)
Name	Null?	Type																	
-----																			
CAPTAIN_ID	NOT NULL	VARCHAR2(30)																	
CAPTAIN_NAME		VARCHAR2(30)																	
YEAR_OF_CAPTAINCY		NUMBER(2)																	
NO_OF_WINS		NUMBER(4)																	



## Second Table: TeamCaptain2NF

```
SQL> CREATE TABLE TeamCaptain2NF (  
2     CAPTAIN_ID VARCHAR2(30),  
3     TEAM_ID VARCHAR2(30),  
4     PLAYER_ID VARCHAR2(30),  
5     PRIMARY KEY (CAPTAIN_ID, TEAM_ID), -- Composite primary key  
6     FOREIGN KEY (CAPTAIN_ID) REFERENCES Captain2NF(CAPTAIN_ID),  
7     FOREIGN KEY (TEAM_ID) REFERENCES Team(Team_ID),  
8     FOREIGN KEY (PLAYER_ID) REFERENCES Player(PLAYER_ID)  
9 );
```

Table created.

```
SQL> desc TeamCaptain2NF;
```

Name	Null?	Type
CAPTAIN_ID	NOT NULL	VARCHAR2(30)
TEAM_ID	NOT NULL	VARCHAR2(30)
PLAYER_ID		VARCHAR2(30)

This decomposition helps in reducing data redundancy and ensures that each table represents a distinct entity without partial dependencies.

Now Analysing the Functional Dependencies for the table TeamCaptain2NF

- A. TEAM\_ID -> PLAYER\_ID
- B. CAPTAIN\_ID -> PLAYER\_ID

The combination of these dependencies create a **transitive dependency**

- CAPTAIN\_ID -> TEAM\_ID -> PLAYER\_ID

This means that a captain's ID indirectly determines a player's ID through the team they lead.

Since TRANSITIVE DEPENDENCY exist we are undergoing 3NF and splitting the relation as follows

### First Table: TeamCaptain3NF

```
SQL> CREATE TABLE TeamCaptain3NF (  
  2     CAPTAIN_ID VARCHAR2(30),  
  3     TEAM_ID VARCHAR2(30),  
  4     PRIMARY KEY (CAPTAIN_ID, TEAM_ID), -- Composite primary key  
  5     FOREIGN KEY (CAPTAIN_ID) REFERENCES Captain(CAPTAIN_ID),  
  6     FOREIGN KEY (TEAM_ID) REFERENCES Team(Team_ID)  
  7 );
```

Table created.

```
SQL> desc TeamCaptain3NF;
```

Name	Null?	Type
-----	-----	-----
CAPTAIN_ID	NOT NULL	VARCHAR2(30)
TEAM_ID	NOT NULL	VARCHAR2(30)

### Second Table: TeamPlayers3NF

```
SQL> CREATE TABLE TeamPlayers3NF (  
  2     TEAM_ID VARCHAR2(30) PRIMARY KEY,  
  3     PLAYER_ID VARCHAR2(30),  
  4     FOREIGN KEY (PLAYER_ID) REFERENCES Player(PLAYER_ID)  
  5 );
```

Table created.

```
SQL> desc TeamPlayers3NF;
```

Name	Null?	Type
-----	-----	-----
TEAM_ID	NOT NULL	VARCHAR2(30)
PLAYER_ID		VARCHAR2(30)

This decomposition ensures that each table represents a distinct entity with no partial or transitive dependencies within them, following the rules of 2NF and 3NF.

Hence the Table Captain is **NORMALIZED**

## CHAPTER 13

### WORKING WITH THE TRANSACTIONS

Let's consider five transactions where Transaction T1 reads the player's name for ID 'PLR33889,' T2 updates their team to 'RCB03,' T3 re-reads the player's name, T4 updates their team to 'MI01,' and T5 reads their name again. Each step reflects operations like reading player data and updating team assignments, typical in sports management systems for tracking player movements and information.

T1:

```
BEGIN TRANSACTION
READ player_name FROM player WHERE player_id = 'PLR33889';
COMMIT TRANSACTION;
```

T2:

```
BEGIN TRANSACTION
UPDATE player SET team_id = 'RCB03' WHERE player_id = 'PLR33889';
COMMIT TRANSACTION;
```

T3:

```
BEGIN TRANSACTION
READ player_name FROM player WHERE player_id = 'PLR33889';
COMMIT TRANSACTION;
```

T4:

```
BEGIN TRANSACTION
UPDATE player SET team_id = 'MI01' WHERE player_id = 'PLR33889';
COMMIT TRANSACTION;
```

T5:

```
BEGIN TRANSACTION
READ player_name FROM player WHERE player_id = 'PLR33889';
COMMIT TRANSACTION;
```

In the above example, T1 reads the player\_name of a player with player\_id 'PLR33889'. T2 then updates the team\_id of the same player, and T3 reads the player\_name again. These transactions demonstrate concurrency with a read followed by a write operation on the same data by different transactions.

## Concurrency Control

1. Transaction T1:
  - Begin Transaction
  - Read player\_name FROM player WHERE player\_id = 'PLR33889';
  - Commit Transaction
2. Transaction T2:
  - Begin Transaction
  - Read player\_name FROM player WHERE player\_id = 'PLR33889';
  - If the record is locked by T1, wait or retry later
  - Otherwise, update player SET team\_id = 'RCB03' WHERE player\_id = 'PLR33889';
  - Commit Transaction
3. Transaction T3:
  - Begin Transaction
  - Read player\_name FROM player WHERE player\_id = 'PLR33889';
  - If the record is locked by T1 or T2, wait or retry later
  - Otherwise, proceed with further actions or queries
  - Commit Transaction
4. Transaction T4:
  - Begin Transaction
  - Read player\_name FROM player WHERE player\_id = 'PLR33889';
  - If the record is locked by T1, T2, or T3, wait or retry later
  - Otherwise, update player SET team\_id = 'MI01' WHERE player\_id = 'PLR33889';
  - Commit Transaction
5. Transaction T5:
  - Begin Transaction
  - Read player\_name FROM player WHERE player\_id = 'PLR33889';
  - If the record is locked by T1, T2, T3, or T4, wait or retry later
  - Otherwise, proceed with further actions or queries
  - Commit Transaction

In this concurrency control scheme:

- T1 can read the data without any issue because it's the first transaction.
- T2 needs to wait if T1 is still reading the same record because it intends to update it.
- T3 waits if T1 or T2 is modifying the record but can proceed if they are only reading.
- T4 waits for all previous transactions (T1, T2, T3) to finish before attempting to update.
- T5 waits until all previous transactions have completed before performing its actions.

## **CHAPTER 14**

### **CONCLUSION**

In conclusion, the project centered around managing player information in a sports league demonstrates key aspects of database management and concurrency control. By analyzing transactions that read and update player data along with their team affiliations, we gain insights into the complexities of real-world systems.

The transactions highlighted the importance of concurrency control mechanisms to ensure data consistency and integrity. In a sports management system, where multiple users or processes may access and modify data concurrently, implementing robust concurrency control strategies such as locking mechanisms or transaction isolation levels becomes crucial. These strategies help prevent issues like data anomalies, conflicting updates, or lost updates, ensuring that the database remains accurate and reliable.

Furthermore, the project emphasized the need for careful transaction management, including proper transaction boundaries, commit points, and rollback mechanisms. These aspects play a vital role in maintaining data consistency and ensuring that transactions are executed reliably, even in the presence of failures or concurrent operations.

Overall, the project serves as a practical exploration of database concepts like transactions, concurrency control, and data consistency within the context of sports management systems. It underscores the importance of applying these principles effectively to build robust and dependable database systems for managing dynamic and constantly evolving data scenarios.

## REFERENCES

1. Sharma, S., & Gupta, P. (2022). Development of a Cricket Match Statistics Database Using MongoDB and Express.js. In Proceedings of the International Conference on Data Engineering and Information Systems.
2. Patel, N., & Shah, M. (2023). An Integrated Approach to Cricket Player Performance Analysis and Database Management. *Journal of Sports Analytics*, 7(2), 112-125.
3. Chatterjee, D., & Das, S. (2024). Design and Implementation of a Real-Time Scoreboard System for Cricket Tournaments Using Apache Kafka and Microservices Architecture. In IEEE International Conference on Cloud Computing.
4. Kumar, V., & Singh, R. (2023). A Comprehensive Cricket Database Schema for Performance Evaluation and Statistical Analysis. *Journal of Database Management*, 35(4), 78-92.
5. Joshi, A., & Desai, R. (2022). Building a Scalable and Reliable Cricket Tournament Database Using Amazon Web Services (AWS) and DynamoDB. In Proceedings of the ACM Symposium on Cloud Computing.
6. Mishra, S., & Verma, R. (2023). An AI-Driven Approach to Predictive Modeling in Cricket: Data Integration and Analysis from Multiple Sources. *International Journal of Artificial Intelligence in Sports*, 10(1), 45-58.
7. Gupta, A., & Sharma, V. (2024). Blockchain-Based Data Integrity and Security Framework for Cricket Match Results and Player Statistics. In Proceedings of the International Conference on Blockchain Technology.
8. Reddy, S., & Kumar, A. (2023). A Comparative Study of SQL and NoSQL Databases for Storing and Querying Cricket Match Data. *Journal of Information Technology Research*, 16(3), 45-60.
9. Patel, K., & Singh, M. (2022). Design and Development of a Mobile-First Cricket Tournament Management System Using React Native and Firebase. In Proceedings of the International Conference on Mobile Computing.
10. Agarwal, R., & Jain, S. (2023). Data Warehousing and Business Intelligence for Cricket Analytics: A Case Study of IPL Matches. *International Journal of Data Warehousing and Mining*, 19(2), 78-92.

# ANNEXURE-1

## NORMALIZATION

### DATABASE SCHEMA:

Emp-id	int (5), primary Key
Emp-name	varchar (10)
Emp-salary	int (10)
Emp-dept	varchar (10)
Emp-deptid	int (10)
Emp-address	varchar (20)
Emp-phone	int (10)
Emp-DOJ	date

### FUNCTIONAL DEPENDENCY:

Emp-id  $\rightarrow$  { Emp-name, Emp-salary, Emp-dept, Emp-address, Emp-contactno, Emp-DOJ }

Emp-deptid  $\rightarrow$  { Emp-dept-name }

Emp-dept-name  $\rightarrow$  { Emp-salary }

CLOSURE

### 2NF

Emp-deptid	int (10)
Emp-dept-name	varchar (10)

Since Partial dependency exists, it undergoes 2NF.

Emp-id	int (5)
Emp-name	varchar (10)
Emp-salary	int (10)
Emp-address	varchar (10)
Emp-phone	int (10)
Emp-DOJ	date

$\Rightarrow$  The above table is in 1NF.

Emp-name	varchar (10)
Emp-phone	int (10)
Emp-address	varchar (10)

Emp-id	int (5)
Emp-name	varchar (10)
Emp-salary	int (10)
Emp-DOJ	date

### 3NF

Emp-id  $\rightarrow$  { Emp-name }

Emp-name  $\rightarrow$  { Emp-phone }

Since transitive dependency exists, it undergoes 3NF.

DONE BY:

SURAT S KONGUR

(BA22110320100038)

SURYA SIVARAJ K

(BA22110320100043)

YAFFIN S

(BA22110320100053)

DARSHAN KUMAR M

(BA22110320100060)

MITUN M

(BA22110320100090)



## ANNEXURE-2

Ex.No. 1	CREATING DATABASE TABLE	Date :3/5/24
----------	-------------------------	--------------

### CREATE TABLE

**Q1)** Create the tables DEPT and EMP as described below

#### DEPT

<i>Column name</i>	<i>Data type</i>	<i>Description</i>
DEPTNO	Number	Department number
DNAME	Varchar	Department name
LOC	Varchar	Department Location

#### EMP

<i>Column name</i>	<i>Data type</i>	<i>Description</i>
EMPNO	Number	Employee number
ENAME	Varchar	Employee name
JOB	Char	Designation
MGR	Number	Manager's EMP.No.
HIREDATE	Date	Date of joining
SAL	Number	Basic Salary
COMM	Number	Commission
DEPTNO	Number	Department Number

#### SQL for DEPT table>

```
CREATE TABLE DEPT (  
    DEPTNO NUMBER PRIMARY KEY,  
    DNAME VARCHAR(50),  
    LOC VARCHAR(50)
```

);

**SQL for EMP table>**

```
CREATE TABLE EMP (  
    EMPNO NUMBER PRIMARY KEY,  
    ENAME VARCHAR(50),  
    JOB CHAR(10),  
    MGR NUMBER,  
    HIREDATE DATE,  
    SAL NUMBER,  
    COMM NUMBER,  
    DEPTNO NUMBER,  
    FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)  
);
```

**Q2)** Confirm table creation

```
SQL> SHOW TABLES;
```

**Q3)** List name of the tables created by the user

```
SQL> SHOW TABLES;
```

**Q4)** Describe tables owned by the user

```
SQL> DESC DEPT;
```

```
DESC EMP;
```

**Q5)** View distinct object types owned by the user

**SQL>**

```
SELECT DISTINCT table_type
FROM information_schema.tables
WHERE table_name IN ('DEPT', 'EMP') AND table_schema = 'your_database_name';
```

**Q6)** View tables, views, synonyms, and sequences owned by the user

**SQL>**

```
SELECT table_name, table_type
FROM information_schema.tables
WHERE table_name IN ('DEPT', 'EMP') AND table_schema = 'your_database_name';
```

**Q7)** Add new columns COMNT and MISCEL in DEPT table of character type.

**SQL >**

```
ALTER TABLE DEPT
ADD COLUMN COMNT VARCHAR(255),
ADD COLUMN MISCEL VARCHAR(255);
```

**Q8)** Modify the size of column LOC by 15 in the DEPT table

**SQL >**

```
ALTER TABLE DEPT
MODIFY COLUMN LOC VARCHAR(15);
```

**Q9)** Set MISCEL column in the DEPT table as unused **SQL >**

```
ALTER TABLE DEPT
SET UNUSED COLUMN MISCEL;
```

**Q10)** Drop the column COMNT from the table DEPT

**SQL >**

```
ALTER TABLE DEPT  
DROP COLUMN COMNT;
```

**Q11)** Drop unused columns in DEPT table

**SQL >**

```
ALTER TABLE DEPT  
DROP UNUSED COLUMNS;
```

**Q12)** Rename the table DEPT to DEPT12

**SQL >**

```
ALTER TABLE DEPT RENAME TO DEPT12;
```

**Q13)** Remove all the rows in the table DEPT12 (Presently no records in DEPT12)

**SQL >**

```
DELETE FROM DEPT12;
```

**Q14)** Add some comment to the table DEPT12 and also confirm the inclusion of comment

**SQL >**

```
ALTER TABLE DEPT12  
COMMENT 'This table stores department information.';
```

**Q15)** Delete the table DEPT12 from the database.

**SQL >**

```
DROP TABLE DEPT12;
```

**Q16)** Confirm the removal of table DEPT12 from the database.

SQL >

```
SELECT table_name  
FROM information_schema.tables  
WHERE table_schema = 'your_database_name' AND table_name = 'DEPT12';
```

<b>Ex.No. 2</b>	<b>Working with Data Manipulation commands</b>	<b>Date : 3/5/2024</b>
-----------------	--	------------------------

**Data for EMP table**

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

**Data for DEPT table**

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

**Q1)** Insert the rows of DEPT table using syntax (i)

```
SQL> INSERT INTO DEPT (DEPTNO, DNAME, LOC) VALUES (10,  
'ACCOUNTING', 'NEW YORK');
```

```
INSERT INTO DEPT (DEPTNO, DNAME, LOC) VALUES (20,  
'RESEARCH', 'DALLAS');
```

```
INSERT INTO DEPT (DEPTNO, DNAME, LOC) VALUES (30, 'SALES',  
'CHICAGO');
```

```
INSERT INTO DEPT (DEPTNO, DNAME, LOC) VALUES (40,  
'OPERATIONS', 'BOSTON');
```

**Q2)** Insert first & second rows of EMP table using syntax (ii)

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE,  
SAL, COMM, DEPTNO)
```

```
VALUES (7369, 'SMITH', 'CLERK', 7902, TO_DATE('17-DEC-80',  
'DD-MON-YY'), 800, NULL, 20);
```

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL,  
COMM, DEPTNO)
```

```
VALUES (7499, 'ALLEN', 'SALESMAN', 7698, TO_DATE('20-FEB-81',  
'DD-MON-YY'), 1600, 300, 30);
```

**Q3)** Insert the remaining rows of EMP table using syntax (iii).

```
INSERT INTO EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL,  
COMM, DEPTNO)
```

```
VALUES (7521, 'WARD', 'SALESMAN', 7698, TO_DATE('22-FEB-81', 'DD-  
MON-YY'), 1250, 500, 30);
```

**Q4)** Create a table MANAGER with the columns *mgr-id*, *name*, *salary* and *hiredate*

```
CREATE TABLE MANAGER (  
    MGR_ID NUMBER PRIMARY KEY,  
    NAME VARCHAR2(50),  
    SALARY NUMBER,  
    HIREDATE DATE  
);
```

- Q5)** Insert values into the table MANAGER by copying the values from EMP table where the designation of the employee is 'MANAGER'

```
INSERT INTO MANAGER (MGR_ID,  
    NAME, SALARY, HIREDATE)  
SELECT EMPNO, ENAME, SAL,  
    HIREDATE  
FROM EMP  
WHERE JOB = 'MANAGER';
```

- Q6)** Change the LOC of all rows of DEPT table by 'NEW YORK'  
UPDATE DEPT SET LOC = 'NEW YORK';

- Q7)** Change the LOC='DALLAS' for deptno=20 in DEPT table.  
UPDATE DEPT SET LOC = 'DALLAS' WHERE DEPTNO = 20;

- Q8)** Delete the rows from EMP table whose employee name = 'PAUL'  
DELETE FROM EMP WHERE ENAME = 'PAUL';

- Q9)** List all the columns and rows of the table DEPT  
SELECT \* FROM DEPT;

- Q10)** List the name of the employee and salary of EMP table  
SELECT ENAME AS NAME, SAL AS SALARY FROM EMP;

- Q11)** Without duplication, list all names of the department of DEPT table.



```
SELECT DISTINCT DNAME FROM DEPT;
```


**Q12)** Find out the name of an employee whose EMPNO is 7788. SELECT  
ENAME FROM EMP WHERE EMPNO = 7788;

**Q13)** As a copy of DEPT table, create DEPT1 table using select command.  
CREATE TABLE DEPT1 AS SELECT \* FROM DEPT;

**Q14)** List ename and sal of EMP table with the column headings NAME and  
SALARY  
SELECT ENAME AS NAME, SAL AS SALARY FROM EMP;

## ANNEXURE-3

### 1. S. YAFFIN [RA2211032010053]



Elite

# NPTEL Online Certification


(Funded by the MoE, Govt. of India)


This certificate is awarded to  
**YAFFIN SASI**  
for successfully completing the course  
**Data Base Management System**  
with a consolidated score of **65** %

Online Assignments	23.13/25	Proctored Exam	42/75
--------------------	----------	----------------	-------


Total number of candidates certified in this course: 6225

Jan-Mar 2024  
(8 week course)






Indian Institute of Technology Kharagpur



FREE ONLINE EDUCATION  
swayam  
PREFACE: WISDOM, SHARAD: WISDOM

Roll No: NPTEL24CS21S544101399

To verify the certificate



No. of credits recommended: 2 or 3

### 2.SAKINA RIZVI [RA2211032010073]

## CERTIFICATE OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO



#### Sakina Rizvi

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**  
Following are the learning items, which are covered in this tutorial

74 Video Tutorials 16 Modules 16 Challenges

03 May 2024

  
Anshuman Singh  
Co-founder **SCALER**



### 3.MITUN M [RA2211032010090]



#### MITUN MAHENDRAN (RA2211032010090)

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

74 Video Tutorials 16 Modules 16 Challenges

25 April 2024



Anshuman Singh

Co-founder **SCALER**

