

DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

PES1201802827

Sooryanath.I.T

The IPL_TOURNAMENT data model is represented in its highest abstract form (ERD) and then converted to a Relational Database using ER mapping technique. Implemented using PostgreSQL .This data model deals with transactions related to the teams participating, players playing under those teams, the schedule of matches ,injury record of players and the wholesome set of Sponsors associated with a team.DDL queries have been implemented to ensure all 3 integrity constraints.

Few advanced queries involving quantifiers and DQL,DML etc have been executed along with a set of correlated sub-queries and aggregate queries, concluded with the outer join queries.A constraint for trigger is identified .A trigger and a procedure call have been implemented in union to prohibit violation of constraints.The output of queries have been shown as screen snips.

The DB allows the user to schedule the matches\games between participating teams at ease as there are triggers which can warn /raise exception when a violation occurs.The team-management of various teams can obtain details of their team-player's injury record, schedule with ease , hence the designed DB supports various level of views.Few future developments like inclusion of graphical visualization of order able data can be incorporated into the system.

Introduction	2
Data Model	3
FD and Normalization	4
DDL	8
Triggers	10
SQL Queries	12
Conclusion	16

Introduction

i. Mini world Description

The mini world requirements for a simple database to be created for the Indian Premier League (IPL)

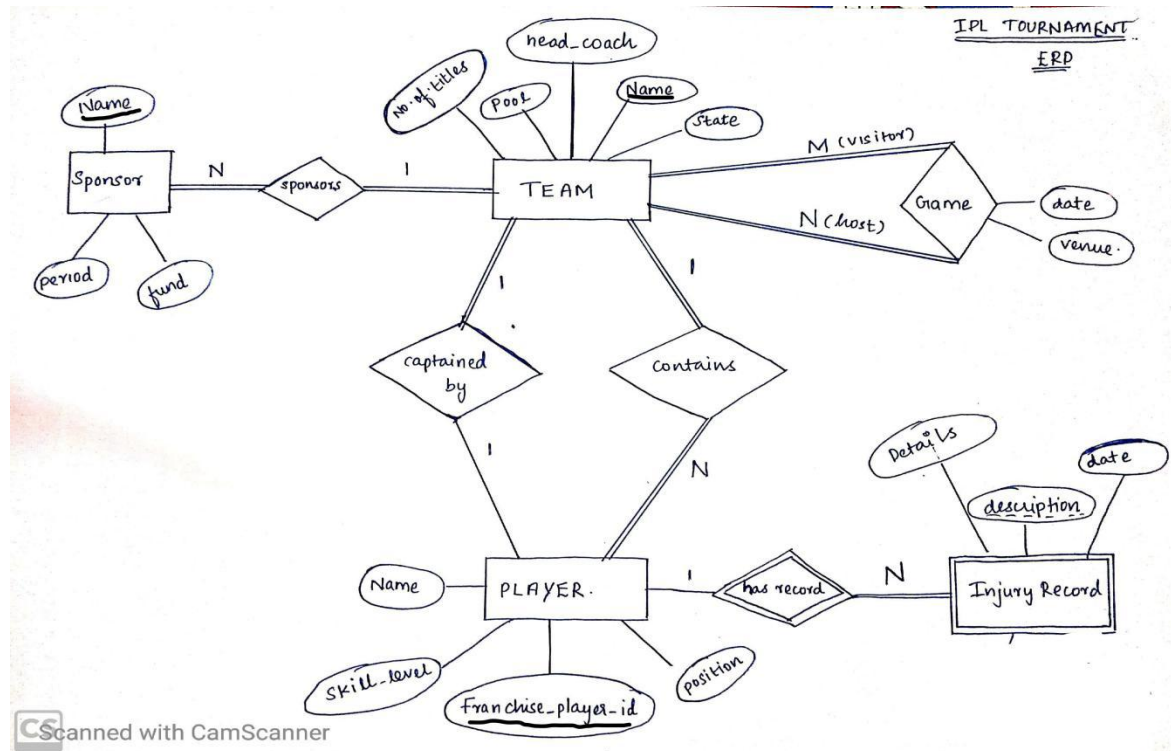
- The IPL has many teams participating each with **unique name**
- Each team has a name, represents a state , a head coach, a captain, and a set of players. There maybe more than one team representing the same state for a different city. Every team belongs to a Pool 'A' or 'B' and has won no.of.titles.
- Each player **belongs to only one** team.
- Each player has a name, a position (batsman, Bowler, All-rounder), **franchise_player ID**, a skill level.
- A player may have a log of injury records describing the injury (hamstring ,tear etc) And date of injury.
- A **player also captains a side/team** [Ignore Vice captain]
- A game is **played between two teams** (where one team hosts at home and the other is a visitor) and has a date (such as May 11th, 1999) and a venue (stadium). A team can play only one game per day. A team may be a host or a visitor.
- Every **Team has one or more** Sponsor (Some Business Unicorns/corporate) ,but a sponsor can invest funds only in one team for certain period of years.

ii. Transactions of The System

Atomicity property is ensured such that an operation is fully executed else withdrawn. A player can have an injury record only if he is a part of the players' relation else the execution is called off. Consistency is ensured as no database operation leaves the database in an incomplete phase .All operations can be executed in parallel threads without interference hence reinforcing isolation property. Any commits made are long lasting and durable.

Data Model

ER Diagram for the specified Mini world of interest.



Relational Schema

Team

<u>Team_Name</u>	State	Head_coach	No.of.titles	Captain	Pool
------------------	-------	------------	--------------	---------	------

Players

<u>Franchise_Player_id</u>	Skill	Position	Name	Team_name
----------------------------	-------	----------	------	-----------

Injury record

<u>Player_id</u>	<u>Description</u>	Date	details
------------------	--------------------	------	---------

Sponsors

<u>Corporate_name</u>	Team_name	Funds	Sponsor_period
-----------------------	-----------	-------	----------------

Game

<u>Host</u>	<u>Visitor</u>	Venue	Date
-------------	----------------	-------	------

Discussion on Choice of keys

Importance

Primary keys enforce entity integrity by uniquely identifying entity instances. Where as foreign keys enforce referential integrity constraints.

i. '**Team**' relation/table lists all the competing teams in the tournament , given in the mini world description that every team has an unique name ,we declare the Team_name attribute of the team as the primary key of identification,hence we may consider '**head_coach**' as a candidate key. Example :Super Kings,Royal Challengers etc.No teams can share the same name.Out of available candidate keys , we choose 'team_name' as primary key.

Possible super keys : {name,state},**primary key**='team_name',
foreign_key='Captain'

ii. '**Players**' relation lists the details like expertise/position/role , skill level of a player with each player assigned with an ID,the team_name may not be unique (as N players may belong to same team),player name doesn't qualify as a key (2 or more players can possess the same name).Hence we declare the **Franchise_Player_id** as the identifying key attribute.

Possible super keys : {Franchise_player_id,name}.**Primary key**='Franchise_Player_id', **Foreign Key**='Team'

iii. '**Injury_record**' relation lists the log of injuries (like hamstring ,meniscus tear) associated with a player in a season, which lets the team management to take a decision based on one's fitness record.A combination of player_id and description (partial key) forms the primary key.

Foreign_key='player_id' ,**primary composite key**={player_id,description}

iv. '**Sponsor**' relation lists the name of the fund investing companies and to which team they are sponsoring.Since a sponsor can invest fund only on one company 'corporate_name' is the primary key in accordance with the relation.

Primary key='corporate_name' ,**foreign_key**='team_name'

v. '**Game**' is the relation/table which involves the entity 'Team' in 2 different roles.One as the host and the other as the visiting team. Every team has the privilege of hosting the game.Since a venue can host only one game per day , we create a composite key '**date,venue**'.

Primary key='{team(host),team(visitor)}'

FD and Normalization

$A \rightarrow B$ is read as **A functionally determines B**, whenever two tuples have the same value for A, they *must have* the same value for B. If K is a key of a Relation S, then K functionally determines all attributes in S (because K is unique).

- Functional dependencies of the Relations in the IPL_Tournament are listed below

i. **Team_Name** \rightarrow {State,Captain,head_coach,no.of.titles,Pool} in Relation 'Team'.

ii. **Franchise_player_id** \rightarrow {Name,skill_level,position,team_name} is one among the functional dependencies for Relation 'Player'.

iii. **Corporate_name** \rightarrow {team_name,fund,period}, since every corporate can sponsor only one team, this is a functional dependency in Relation 'Sponsors'.

iv. **{host,visitor}** \rightarrow {venue,date} is a functional dependency in Relation 'Game'.

v. **{Player_id,description}** \rightarrow {date,details} is a functional dependency in Relation 'Injury_record'.

Normal Forms

The relational schema is obtained after following the steps of ERD to relational schema mapping.

First Normal Form (1NF):

All relational schemas are in 1NF as the entries in tuples (attributes of an instance) **are atomic and indivisible**, there are no group of values for an attribute.

Second Normal Form (2NF):

Relation 'Game' and 'Injury_record' possess a composite primary key.

i. {host,visitor} \rightarrow {venue,date}. Removal of host, makes the functional dependency invalid, the similar case with removal of visitor from the key. Hence it is a **full functional dependency**. Hence it is in 2NF.

ii. {Player_id,description} \rightarrow {details,date}. Removal of player_id makes the functional dependency invalid. Similarly by the removal of description it becomes impossible to track down right injury record.

Hence the above 2 Relations are undoubtedly in 2ND NF.

Example of possible Violations of 2NF:

i. **Addition of a column 'injury_id'** (which yields unique values only) in the Relation 'Injury_record' and making it the partial key may violate the 2NF. $\{player_id, injury_id\} \rightarrow \{description, date, details\}$ will be the FD. Whereas just $\{injury_id\} \rightarrow \{description, date, details\}$ alone holds good as well. Which doesn't make the FD a full functional dependency.

Third Normal Form (3NF) :

3NF is violated when **there is transitive dependency involving a non prime attribute** in R. 3NF is obeyed when there is dependency $X \rightarrow Y$ and $Y \rightarrow Z$ where in Y is a candidate key.

Example: 'head_coach' is a candidate key in Relation 'Team'

'team_name' \rightarrow 'head_coach', 'head_coach' \rightarrow 'no.of.titles' works fine, because 'head_coach' is a candidate key.

Example of possible Violations of 3NF:

i. **Addition of the column 'head_coach'** to the relation 'Players', once we add the functional dependency 'franchise_player_id' \rightarrow 'head_coach' exists, and 'head_coach' can uniquely determine the 'team_name' **but 'head_coach' is not a candidate key**, as more than one player may be mentored by a same coach. This is of the form $X \rightarrow Y$ and $Y \rightarrow Z$ where in **Y is a non-prime attribute** / not a candidate key.

Normal Form of Relations:

With above proofs and explanation, it is evident that all relations are in 3NF. Which implicitly means they qualify the rules of 1NF and 2NF.

DDL

Below set of SQL statements are the DDL statements ,**DDL commands are used to create and modify the structure of a database , enforce relationship integrity constraints and modify db objects.**

```
create database IPL_TOURNAMENT;
```

```
CREATE TABLE TEAM
(
    TEAM_NAME VARCHAR(50) PRIMARY KEY,
    STATE VARCHAR(50) NOT NULL,
    HEAD_COACH VARCHAR(50) UNIQUE NOT NULL,
    NO_OF_TITLES INTEGER NOT NULL,
    CAPTAIN CHAR(10) UNIQUE NOT NULL,
    POOL CHAR(1) NOT NULL,
    CHECK (NO_OF_TITLES >= 0 AND NO_OF_TITLES < 13),
    CHECK ( POOL IN ('A','B'))
);
```

```
ALTER TABLE TEAM
ADD CONSTRAINT "CAPTAIN_ID_FKEY" FOREIGN KEY (CAPTAIN) REFERENCES
PLAYER(FRANCHISE_PLAYER_ID);
```

```
CREATE TABLE PLAYER
(
    FRANCHISE_PLAYER_ID CHAR(10) PRIMARY KEY,
    SKILL INTEGER ,
    POSITION VARCHAR(15) NOT NULL,
    NAME VARCHAR(50) NOT NULL,
    TEAM VARCHAR(50) ,
    CHECK (SKILL > 0 AND SKILL < 11),
    CHECK (POSITION IN ('BOWLER','BATSMAN','ALL-ROUNDER','WICKET-KEEPER'))
);
```

```
ALTER TABLE PLAYER
ADD CONSTRAINT "TEAM_NAME_FKEY" FOREIGN KEY (TEAM) REFERENCES
TEAM(TEAM_NAME) ON DELETE CASCADE;
```

```
CREATE TABLE INJURY_RECORD
(
    PLAYER_ID CHAR(10),
    DESCRIPTION VARCHAR(50) NOT NULL,
    DATE_OF_INJURY DATE NOT NULL,
    DETAILS TEXT,
```



```
PRIMARY KEY(PPLAYER_ID,DESCRIPTION)
);
```

```
ALTER TABLE INJURY_RECORD
ADD CONSTRAINT "INJURED_player_id" FOREIGN KEY (PLAYER_ID) REFERENCES
PLAYER(FRANCHISE_PLAYER_ID);
```

```
CREATE TABLE SPONSOR
(
    CORPORATE_NAME VARCHAR(50) PRIMARY KEY,
    FUNDS INTEGER NOT NULL,
    TEAM VARCHAR(50) ,
    TIME_PERIOD_YEARS INTEGER NOT NULL
);
```

```
ALTER TABLE SPONSOR
ADD CONSTRAINT 'SPONSORING_TEAM' FOREIGN KEY (TEAM) REFERENCES
TEAM(TEAM_NAME);
```

```
CREATE TABLE GAME
(
    HOST VARCHAR(50) REFERENCES TEAM(TEAM_NAME),
    VISITOR VARCHAR(50) REFERENCES TEAM(TEAM_NAME),
    VENUE VARCHAR(50) NOT NULL,
    DATE_OF_GAME DATE NOT NULL,
    PRIMARY KEY(HOST,VISITOR)
);
ALTER TABLE GAME ADD CONSTRAINT "HOST_VISITOR_CHECK" CHECK (HOST != VISITOR );
```

List Of Relations created in Database 'IPL_TOURNAMENT' successfully.

```
SQL Shell (psql)
ipl_tournament=#
ipl_tournament=#
ipl_tournament=# \c ipl_tournament
You are now connected to database "ipl_tournament" as user "postgres".
ipl_tournament=# \dt
      List of relations
Schema |      Name      | Type  | Owner
-----+-----+-----+-----
public | game            | table | postgres
public | injury_record   | table | postgres
public | player          | table | postgres
public | sponsor         | table | postgres
public | team            | table | postgres
(5 rows)

ipl_tournament=#
```

Triggers

Trigger is a function invoked automatically whenever an event e.g., insert, update, or delete occurs with respect to the table of interest.

CONSTRAINT: Ensure that a team doesn't play more than one match/game on the same day.

- This constraint is to be checked **BEFORE** a record is **INSERTED**.
- The above constraint needs to be handled before INSERT and cannot be handled using a normal **CHECK** constraint.
- A procedure is written/defined to check if the teams(host ,visitor) in the current record to be inserted ,have a game to play already on the same date as in the new row/record.
- The Procedure is named as **SCHEDULE_CHECK()**, **SQL code goes below**.

```
CREATE OR REPLACE FUNCTION schedule_check()
    RETURNS trigger AS
$BODY$
BEGIN
    IF EXISTS (SELECT * FROM GAME WHERE
DATE_OF_GAME=NEW.DATE_OF_GAME AND (HOST=NEW.HOST OR
VISITOR=NEW.VISITOR OR HOST=NEW.VISITOR OR VISITOR=NEW.HOST))
    THEN
        RAISE EXCEPTION 'NO TEAM CAN PLAY 2 GAMES ON THE SAME DAY : ';
    END IF;
    RETURN NEW;
END;
$BODY$
LANGUAGE 'plpgsql';
```

- The procedure checks in the GAME relation if there exist a schedule for the teams in the insert statement, which corresponds to the same date as in the record to be inserted.
- Now a **trigger GAME_PER_DAY** is created on table GAME which calls the procedure SCHEDULE_CHECK().

```
CREATE TRIGGER GAME_PER_DAY BEFORE INSERT ON GAME FOR EACH ROW
EXECUTE PROCEDURE SCHEDULE_CHECK() ;
```

- This **Trigger gets fired whenever a new record** is to be inserted into GAME relation.**Throws Exception when constraint is violated** , else record is inserted.

OUTPUT OF TRIGGER IS SHOWN BELOW.

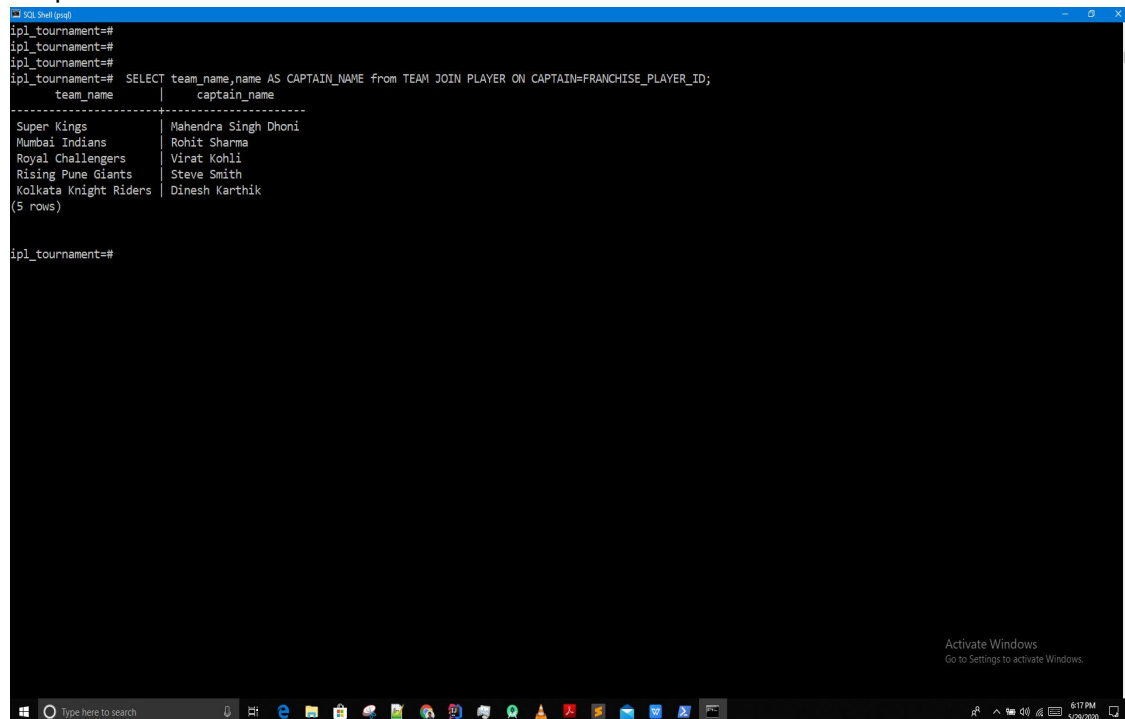
SQL Queries

I.JOIN ON CONDITION (INNER JOIN) query

English statement: List the skipper name for every corresponding team

Query_1 : `SELECT team_name,name AS CAPTAIN_NAME from TEAM JOIN PLAYER ON CAPTAIN=FRANCHISE_PLAYER_ID;`

Output



```
SQL Shell (psql)
ipl_tournament=#
ipl_tournament=#
ipl_tournament=#
ipl_tournament=# SELECT team_name,name AS CAPTAIN_NAME from TEAM JOIN PLAYER ON CAPTAIN=FRANCHISE_PLAYER_ID;
      team_name      | captain_name
-----+-----
 Super Kings        | Mahendra Singh Dhoni
 Mumbai Indians     | Rohit Sharma
 Royal Challengers   | Virat Kohli
 Rising Pune Giants  | Steve Smith
 Kolkata Knight Riders | Dinesh Karthik
(5 rows)

ipl_tournament=#
```

II.NESTED SUB QUERIES

I. English statement : Retrieve only the name and skill level of all the batsmen with skill level ≥ 8 ,playing for the team Mumbai Indians with no injury records in this season.

Query_2 : `SELECT NAME,SKILL AS SKILL_LEVEL FROM (SELECT * FROM PLAYER WHERE TEAM='Mumbai Indians' AND POSITION='batsman' AND SKILL \geq 8 AND FRANCHISE_PLAYER_ID NOT IN(SELECT PLAYER_ID FROM INJURY_RECORD)) AS FINAL_RESULT;`

```
SQL Shell (psql)
ipl_tournament=#
ipl_tournament=#
ipl_tournament=#
ipl_tournament=# SELECT NAME,SKILL AS SKILL_LEVEL FROM ( SELECT * FROM PLAYER WHERE TEAM='Mumbai Indians' AND POSITION='batsman' AND SKILL>=8 AND FRANCHISE_PLAYER_ID NOT IN
N(SELECT PLAYER_ID FROM INJURY_RECORD)) AS FINAL_RESULT;
      name      | skill_level
-----
Rohit Sharma    |          10
Quinton De Kock |           9
Surya Kumar Yadav |           8
Jos Buttler     |           9
(4 rows)

ipl_tournament=#
```

II.English statement : List the venue and the opponents of the team 'Super Kings' on all occasions when their team player was injured in this season.

Query_3 : **CREATE TABLE NEW AS SELECT** HOST,VISITOR,VENUE **FROM** GAME **WHERE** DATE_OF_GAME **IN** (SELECT DATE_OF_INJURY **FROM** PLAYER **JOIN** INJURY_RECORD **ON** TEAM='Super Kings' **AND** FRANCHISE_PLAYER_ID=PLAYER_ID) **AND** (HOST='Super Kings' **OR** VISITOR='Super Kings') ;

--temporary table is created on fly--

```
SELECT VENUE,
CASE
    WHEN VISITOR ='Super Kings' THEN HOST
    ELSE VISITOR
END AS OPPONENT
FROM NEW;
```

OUTPUT

```

ip1_tournament=#
ip1_tournament=#
ip1_tournament=#
ip1_tournament=# CREATE TABLE NEW AS SELECT HOST,VISITOR,VENUE FROM GAME WHERE DATE_OF_GAME IN (SELECT DATE_OF_INJURY FROM PLAYER JOIN INJURY_RECORD ON TEAM='Super Kings' AND FRANCHISE_PLAYER_ID=PLAYER_ID) AND (HOST='Super Kings' OR VISITOR='Super Kings');
SELECT 2
ip1_tournament=#
ip1_tournament=# --temporary table is created on fly--
ip1_tournament=#
ip1_tournament=# SELECT VENUE,
ip1_tournament=# CASE
ip1_tournament=# WHEN VISITOR ='Super Kings' THEN HOST
ip1_tournament=# ELSE VISITOR
ip1_tournament=# END AS OPPONENT
ip1_tournament=# FROM NEW;
      venue      | opponent
-----+-----
MA.Chidambaram Stadium | Royal Challengers
Eden Gardens      | Kolkata Knight Riders
(2 rows)

ip1_tournament=#

```

III.Aggregate Queries (Sum ,count ,group by)

English statement : What is the total no.of.corporate sponsoring the team 'Super Kings'?

Query_4 : `SELECT COUNT(CORPORATE_NAME) as total_sponsors FROM SPONSOR WHERE TEAM='Super Kings';`

English statement : Retrieve the total amount invested(Millions) on 'Super Kings' franchise.

Query_5 : `SELECT CAST (SUM(FUNDS) AS FLOAT)/1000000 as Funds_millions FROM SPONSOR WHERE TEAM='Super Kings';`

English statement : List the total no.of.sponsors per team along with the team_name

Query_6 : `SELECT COUNT(CORPORATE_NAME) AS NO_OF_SPONSORS ,TEAM FROM SPONSOR GROUP BY TEAM;`

OUTPUT OF ALL AGGREGATE QUERIES

```

ip1_tournament=# SELECT COUNT(CORPORATE_NAME) as total_sponsors FROM SPONSOR WHERE TEAM='Super Kings';
total_sponsors
-----
3
(1 row)

ip1_tournament=# SELECT CAST (SUM(FUNDS) AS FLOAT)/1000000 as Funds_millions FROM SPONSOR WHERE TEAM='Super Kings';
Funds_millions
-----
79.4
(1 row)

ip1_tournament=# SELECT COUNT(CORPORATE_NAME) AS NO_OF_SPONSORS ,TEAM FROM SPONSOR GROUP BY TEAM;
no_of_sponsors | team
-----+-----
1 | Royal Challengers
2 | Rising Pune Giants
2 | Mumbai Indians
1 | Kolkata Knight Riders
3 | Super Kings
(5 rows)

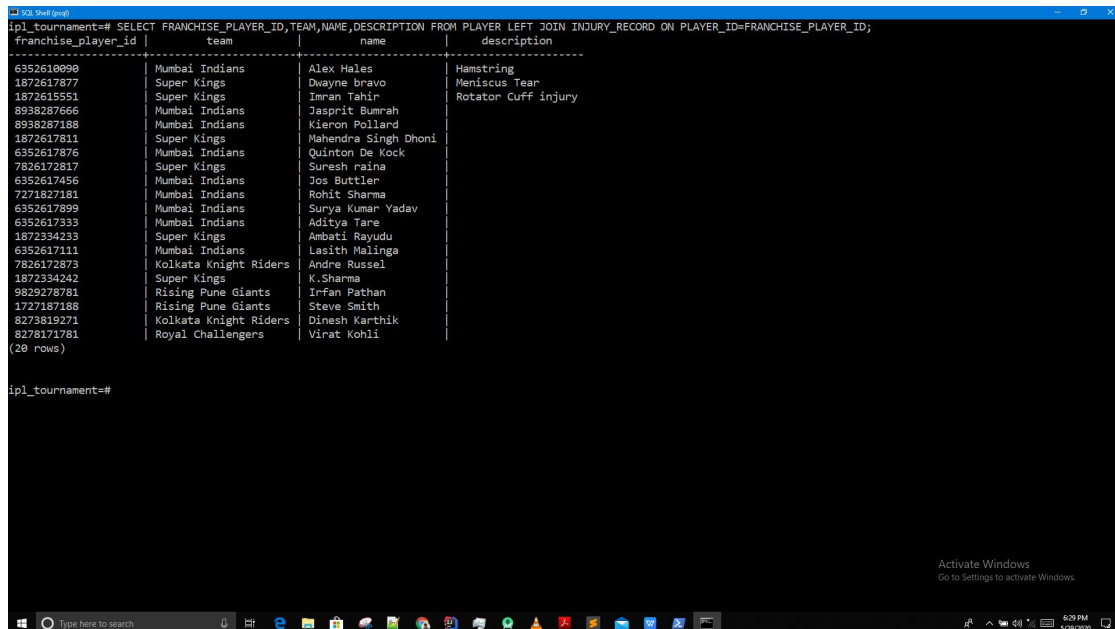
ip1_tournament=#

```

IV. Outer Join Queries

English statement : Perform a left outer join on players table and injury_record of players based on their player_id and list the name,id,team_name and injury description.

Query_7 : `SELECT FRANCHISE_PLAYER_ID,TEAM,NAME,DESCRIPTION FROM
PLAYER LEFT JOIN INJURY_RECORD ON
PLAYER_ID=FRANCHISE_PLAYER_ID;`



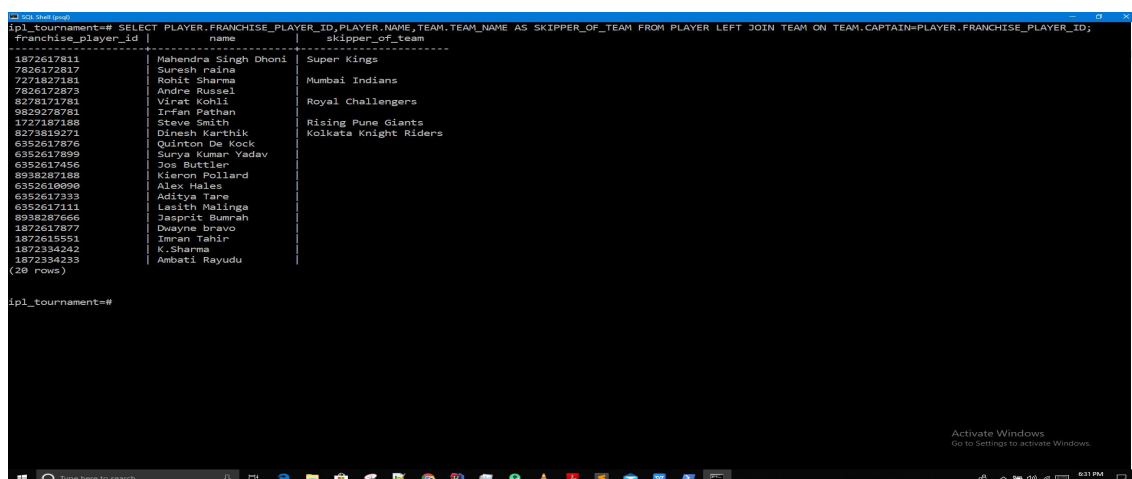
The screenshot shows a terminal window with a SQL query executed. The query is: `ipl_tournament=# SELECT FRANCHISE_PLAYER_ID,TEAM,NAME,DESCRIPTION FROM PLAYER LEFT JOIN INJURY_RECORD ON PLAYER_ID=FRANCHISE_PLAYER_ID;` The output is a table with four columns: franchise_player_id, team, name, and description. It lists 20 rows of data, showing players and their associated injury descriptions where available. For example, Alex Hales has a Hamstring injury, and Dwayne Bravo has a Meniscus Tear. Some players like Virat Kohli and Kieron Pollard have no injury records listed.

franchise_player_id	team	name	description
6352610090	Mumbai Indians	Alex Hales	Hamstring
1872617877	Super Kings	Dwayne bravo	Meniscus Tear
1872615551	Super Kings	Imran Tahir	Rotator Cuff injury
8938287666	Mumbai Indians	Jasprit Bumrah	
8938287188	Mumbai Indians	Kieron Pollard	
1872617811	Super Kings	Mahendra Singh Dhoni	
6352617876	Mumbai Indians	Quinton De Kock	
7826172817	Super Kings	Sunesh raina	
6352617456	Mumbai Indians	Jos Buttler	
7271827181	Mumbai Indians	Rohit Sharma	
6352617899	Mumbai Indians	Surya Kumar Yadav	
6352617333	Mumbai Indians	Aditya Tane	
1872334233	Super Kings	Ambati Rayudu	
6352617111	Mumbai Indians	Lasith Malinga	
7826172873	Kolkata Knight Riders	Andre Russel	
1872334242	Super Kings	K.Sharma	
9829278781	Rising Pune Giants	Irfan Pathan	
1727187188	Rising Pune Giants	Steve Smith	
8271819271	Kolkata Knight Riders	Dinesh Karthik	
8278171781	Royal Challengers	Virat Kohli	

The EMPTY FIELDS signify that not all columns on the Left side of join find a match with the relation on right side

English statement : Perform an outer join to list the players' name ,team_name for which they are the captain.

Query_8: `SELECT
PLAYER.FRANCHISE_PLAYER_ID,PLAYER.NAME,TEAM.TEAM_NAME
E AS SKIPPER_OF_TEAM FROM PLAYER LEFT JOIN TEAM ON
TEAM.CAPTAIN=PLAYER.FRANCHISE_PLAYER_ID;`



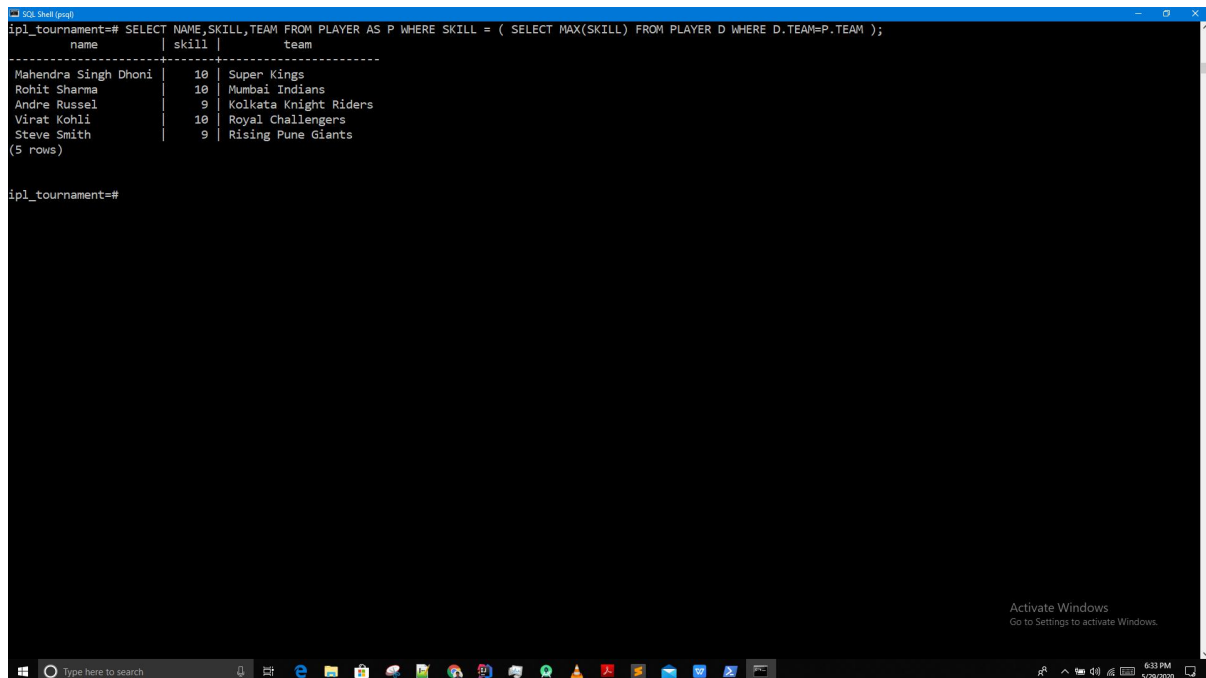
The screenshot shows a terminal window with a SQL query executed. The query is: `ipl_tournament=# SELECT PLAYER.FRANCHISE_PLAYER_ID,PLAYER.NAME,TEAM.TEAM_NAME AS SKIPPER_OF_TEAM FROM PLAYER LEFT JOIN TEAM ON TEAM.CAPTAIN=PLAYER.FRANCHISE_PLAYER_ID;` The output is a table with three columns: franchise_player_id, name, and skipper_of_team. It lists 20 rows of data, showing players and their respective teams. For example, Mahendra Singh Dhoni is the skipper of Super Kings, and Sunesh Raina is the skipper of Mumbai Indians. Some players like Virat Kohli and Kieron Pollard are not listed as captains.

franchise_player_id	name	skipper_of_team
1872617811	Mahendra Singh Dhoni	Super Kings
7826172817	Sunesh raina	Mumbai Indians
7271827181	Rohit Sharma	Mumbai Indians
7826172873	Andre Russel	Royal Challengers
8278171781	Virat Kohli	Royal Challengers
9829278781	Irfan Pathan	Rising Pune Giants
1727187188	Steve Smith	Rising Pune Giants
8271819271	Dinesh Karthik	Kolkata Knight Riders
6352617876	Quinton De Kock	Kolkata Knight Riders
6352617899	Surya Kumar Yadav	Kolkata Knight Riders
6352617456	Jos Buttler	Kolkata Knight Riders
8938287188	Kieron Pollard	Kolkata Knight Riders
1872610090	Alex Hales	Kolkata Knight Riders
6352617333	Aditya Tane	Kolkata Knight Riders
6352617111	Lasith Malinga	Kolkata Knight Riders
8938287666	Jasprit Bumrah	Kolkata Knight Riders
1872617877	Dwayne bravo	Kolkata Knight Riders
1872615551	Imran Tahir	Kolkata Knight Riders
1872334242	K.Sharma	Kolkata Knight Riders
1872334233	Ambati Rayudu	Kolkata Knight Riders

V. Correlated Sub Queries : A correlated sub-query executes once for each candidate row considered by the outer query.

English statement = Select the player with highest skill level in every Participating team along with his skill_level.

Query_9: **SELECT NAME,SKILL,TEAM FROM PLAYER AS P WHERE SKILL = (SELECT MAX(SKILL) FROM PLAYER D WHERE D.TEAM=P.TEAM);**



```
SQL Shell (psql)
ipl_tournament=# SELECT NAME,SKILL,TEAM FROM PLAYER AS P WHERE SKILL = ( SELECT MAX(SKILL) FROM PLAYER D WHERE D.TEAM=P.TEAM );
   name      | skill | team
-----|-----|-----
Mahendra Singh Dhoni |    10 | Super Kings
Rohit Sharma      |    10 | Mumbai Indians
Andre Russel      |     9 | Kolkata Knight Riders
Virat Kohli       |    10 | Royal Challengers
Steve Smith       |     9 | Rising Pune Giants
(5 rows)

ipl_tournament=#
```

Activate Windows
Go to Settings to activate Windows.

Type here to search

6:33 PM
5/29/2020

Conclusion

Capabilities of IPL_TOURNAMENT DB system

- i. The designed IPL database **ensures that data integrity is preserved** across channels/relations with viable and necessary check constraints and Triggers.
- ii. While staying **consistent and scalable**, this IPL_tournament DB lets multiple people view and work with data simultaneously.
- iii. The relations have been decomposed into the NORMAL FORMS hence **minimizing any redundancy** in the database tables.
- iv. The DB **allows the user to schedule the matches\games** between participating teams at ease as there are **triggers which can warn /raise exception when a violation occurs**.
- v. The team-management of various teams can obtain details of their team-player's injury record, schedule with ease. Hence the designed DB **supports multiple views**.

Limitations

- i. The Data types used are of fixed length in few cases or too big ,with a deeper investigation about the types of data to be recorded,maybe we can minimize the size of data type used and leverage between content and space.

Future Developments

- i. **Inclusion of data visualization** such as displaying No.of..titles secured by a team through a graph
 - ii. **Real time analysis of team's performance** using periodically updated data.
- Extending the **DB to a CLOUD BASED DB** distribution provide access in terms of services to the client.