Besant Technologies

BEASANT TECHNOLOGIES

SQL PRESENTATION

SPORTS MANAGEMENT SYSTEM

BY

HARIPRASATH P

ECE 4 YEAR

PROJECT VIEW

DESCRIPTION: Create a system to manage sports teams, players, matches, and scores. Implement features for tracking player statistics, match results, and team standings.

KEY FEATURES:

1. Team & Player Management

Track team details, assign players to teams, and store player profiles including position, age, and stats.

2. Match Scheduling & Results

Record match dates, venues, participating teams, and final scores with full performance breakdown.

3. Player Statistics

Monitor goals, assists, cards, and match-by-match contributions to identify top performers.

4. Team Standings & Rankings

Calculate team points based on match outcomes and rank them by points and goal difference.

5. Analytical SQL Queries

Use JOIN, GROUP BY, and aggregate functions to get insights like top scorers, head-to-head records, or win streaks.

DATABASE SCHEMA

Lets write query to create following tables:

1. LEAGUES

```
CREATE TABLE Leagues
(    league_id INT PRIMARY KEY,
 league_name VARCHAR(100) NOT NULL,
   level VARCHAR(50));

INSERT INTO Leagues VALUES
(1, 'Premier League', 'Top Division’),
(2, 'Championship', 'Second Division');
```

2. TEAMS

```
CREATE TABLE Teams (    team_id INT PRIMARY KEY,    team_name VARCHAR(100) NOT
NULL,    coach_name VARCHAR(100),    league_id INT,    FOREIGN KEY (league_id)
REFERENCES Leagues(league_id));

INSERT INTO Teams VALUES
(101, 'Chennai Hawks', 'K. Suresh', 1),(102, 'Mumbai Warriors', 'R. Sharma', 1),(103,
'Delhi Dynamos', 'A. Iyer', 1),(104, 'Bangalore Bulls', 'V. Kohli', 2),(105, 'Hyderabad
Strikers', 'B. Kumar', 2);
```

3. PLAYERS

```
CREATE TABLE Players (
 player_id INT PRIMARY KEY,
 player_name VARCHAR(100) NOT NULL,
 position VARCHAR(50),
 team_id INT,
FOREIGN KEY (team_id) REFERENCES Teams(team_id));
```

```
INSERT INTO Players VALUES
-- Team 101: Chennai Hawks
(201, 'Arjun Reddy', 'Striker', 101),
(202, 'Rahul Mehra', 'Midfielder', 101),
(203, 'Karthik Das', 'Defender', 101),
(204, 'Siddhu Nair', 'Goalkeeper', 101),
(205, 'Jeevan Lal', 'Winger', 101),

-- Team 102: Mumbai Warriors
(206, 'Neeraj Shah', 'Striker', 102),
(207, 'Imran Khan', 'Midfielder', 102),
(208, 'Lakshman Rao', 'Defender', 102),
(209, 'Amit Kumar', 'Goalkeeper', 102),
(210, 'Harsh Patel', 'Winger', 102),
```

-- Team 103: Delhi Dynamos
(211, 'Vijay Prakash', 'Striker', 103),
(212, 'Sanjay Rao', 'Midfielder', 103),
(213, 'Dinesh Roy', 'Defender', 103),
(214, 'Ravi Varma', 'Goalkeeper', 103),
(215, 'Manoj Shetty', 'Winger', 103),

-- Team 104: Bangalore Bulls
(216, 'Ajay Menon', 'Striker', 104),
(217, 'Ravi Teja', 'Midfielder', 104),
(218, 'Pranav Joshi', 'Defender', 104),
(219, 'Kiran Dev', 'Goalkeeper', 104),
(220, 'Sahil Bhatia', 'Winger', 104),

-- Team 105: Hyderabad Strikers
(221, 'Siddharth Krishnan', 'Striker', 105),
(222, 'Vikram Naidu', 'Midfielder', 105),
(223, 'Karan Bedi', 'Defender', 105),
(224, 'Naveen Raj', 'Goalkeeper', 105),
(225, 'Anand Iqbal', 'Winger', 105);

4. MATCHES

```
CREATE TABLE Matches (
match_id INT PRIMARY KEY,
match_date DATE NOT NULL,
team1_id INT,
team2_id INT,
winner_team_id INT,
FOREIGN KEY (team1_id) REFERENCES Teams(team_id),
FOREIGN KEY (team2_id) REFERENCES Teams(team_id),
FOREIGN KEY (winner_team_id) REFERENCES
Teams(team_id));

INSERT INTO Matches VALUES
(301, '2025-07-01', 101, 102, 101),
(302, '2025-07-03', 103, 104, 104),
(303, '2025-07-05', 102, 105, 105),
(304, '2025-07-07', 101, 103, 101);
```

5. SCORES

```
CREATE TABLE Scores (
 score_id INT PRIMARY KEY,
 match_id INT,
 player_id INT,
 points_scored INT,
 FOREIGN KEY (match_id) REFERENCES Matches(match_id),
 FOREIGN KEY (player_id) REFERENCES Players(player_id));

INSERT INTO Scores VALUES
-- Match 301: Chennai Hawks vs Mumbai Warriors
(401, 301, 201, 2),
 -- Arjun Reddy (Striker, Chennai)
(402, 301, 202, 1),
 -- Rahul Mehra (Midfielder, Chennai)
(403, 301, 204, 1),
-- Neeraj Shah (Striker, Mumbai)
(404, 301, 207, 1),
-- Imran Khan (Midfielder, Mumbai)
-- Match 302: Delhi Dynamos vs Bangalore Bulls
(405, 302, 211, 3),
-- Vijay Prakash (Striker, Delhi)
(406, 302, 212, 1),
 -- Sanjay Rao (Midfielder, Delhi)
(407, 302, 216, 2),
 -- Ajay Menon (Striker, Bangalore)(
408, 302, 217, 1),
 -- Ravi Teja (Midfielder, Bangalore)
```

-- Match 303: Mumbai Warriors vs Hyderabad Strikers
(409, 303, 206, 1),
-- Neeraj Shah (Striker, Mumbai)
(410, 303, 210, 2),
-- Harsh Patel (Winger, Mumbai)
(411, 303, 221, 2),
-- Siddharth Krishnan (Striker, Hyderabad)
(412, 303, 224, 1),
-- Naveen Raj (Goalkeeper, Hyderabad)

-- Match 304: Chennai Hawks vs Delhi Dynamos
(413, 304, 205, 1),
-- Jeevan Lal (Winger, Chennai)
(414, 304, 214, 2),
-- Ravi Varma (Goalkeeper, Delhi)
(415, 304, 213, 1);
-- Dinesh Roy (Defender, Delhi)

DDL(DATA DEFINITION LANGUAGE)

- DDL (Data Definition Language) is a subset of SQL used to define and manage database structures like tables, schemas, and indexes. It includes commands such as CREATE, ALTER, DROP, and TRUNCATE to shape how data is stored.

  - Drop table matches;
  - truncate table matches;
  - ALTER TABLE matches ADD COLUMN match_time enum('mrng','afternoon' ,'evening');
  - describe matches ;
  - alter table matches drop column match_time;alter table players add column age int check(age>18);
  - alter table players drop column age;

DML

- DML (Data Manipulation Language) is a part of SQL used to interact with the data inside tables. It includes commands like SELECT, INSERT, UPDATE, and DELETE to retrieve or modify records.

- insert into teams (team_id ,team_name ,coach_name ,league_id) values (6,'goa hulks','P.hari',2);
- delete from teams where team_id=6;
- update teams set coach_name ='p.hari' where coach_name ='k. suresh';

WHERE AND HAVING

- WHERE Clause filters rows before any grouping or aggregation happens. It's used to select specific records based on conditions like Age > 18 or City = 'Chennai'.
- HAVING Clause filters groups after aggregation. It’s used with functions like SUM, COUNT, or AVG to refine grouped results, such as HAVING COUNT(*) > 5.

- SELECT *FROM players p JOIN scores s ON p.player_id = s.player_id WHERE s.player_id =201;

- SELECT l.league_name, COUNT(t.league_id) AS teams FROM Leagues l JOIN Teams t ON l.league_id = t.league_id GROUP BY l.league_name HAVING COUNT(t.league_id) = 2;

DQL

- DQL (Data Query Language) is a subset of SQL used to retrieve data from a database. Its primary command is SELECT, which lets you extract specific information from tables based on conditions or filters.

  - select *from players;
  - select*from scores;
  - select*from teams;
  - select* from leagues;
  - select*from matches;
  - SELECT l.league_name, COUNT(t.league_id) FROM leagues l JOIN teams t ON l.league_id = t.league_id GROUP BY l.league_name HAVING COUNT(t.league_id) > 1;
  - SELECT l.league_name, COUNT(t.league_id) as teams FROM leagues l JOIN teams t ON l.league_id = t.league_id GROUP BY l.league_name having count(l.league_id)<2;

TCL

- TCL (Transaction Control Language) is a subset of SQL used to manage transactions and ensure data integrity in relational databases. It helps control how changes are saved or undone during multi-step operations.
- Here are the key TCL commands:
- - COMMIT: Permanently saves all changes made during the transaction.
- - ROLLBACK: Reverts changes if something goes wrong, restoring the previous state.
- - SAVEPOINT: Sets a checkpoint within a transaction to roll back to if needed.
- These commands are essential when you're working with critical data—like updating player scores or match results—where partial updates could cause inconsistencies.

```
SELECT * FROM scores s JOIN matches m ON s.match_id = m.match_id JOIN players p ON
s.player_id = p.player_id;
  savepoint sp1;
ALTER TABLE leagues MODIFY COLUMN level text ;
select*from leagues;describe leagues;
savepoint sp2;

ALTER TABLE leaguesADD CONSTRAINT l_league_name UNIQUE (league_name);
alter table leagues drop constraint l_league_name;
savepoint sp3;

rollback ;
commit;
```

JOINS

- SQL Joins are used to combine rows from two or more tables based on a related column between them—usually a foreign key.
- Here's a quick breakdown:
- - INNER JOIN: Returns only matching rows from both tables.
- - LEFT JOIN: Returns all rows from the left table, and matching rows from the right.
- - RIGHT JOIN: Returns all rows from the right table, and matching rows from the left.
- - FULL JOIN: Returns all rows when there's a match in either table.

```
 -- right join
SELECT * FROM scores s right JOIN players p ON s.player_id= p.player_id WHERE
s.points_scored ;
 -- left join
 SELECT * FROM scores s left JOIN players p ON s.player_id= p.player_id WHERE
s.points_scored ;
   -- cross join
select * from teams t cross join players p order by t.team_id asc  ;
      -- inner join
select * from teams t inner join players p on t.team_id =p.team_id;
     -- full join
select* from  players
 UNION
 select*from teams;
```

LIMIT AND OFFSET

```
SELECT * FROM  teams ORDER BY team_name DESC   LIMIT 1 offset 1 ;
```

GROUP BY AND ORDER BY

```
SELECT l.league_name, COUNT(t.league_id) as teams FROM leagues l left JOIN teams t
ON l.league_id = t.league_id GROUP BY l.league_name ;

SELECT player_name FROM players ORDER BY player_name DESC LIMIT 1;
```

STORED PROCEDURE

```
 DELIMITER //
CREATE PROCEDURE gethari()
BEGIN

  SELECT * FROM players p JOIN scores s ON p.player_id = s.player_id     WHERE
s.player_id = 201;

END //
DELIMITER ;

Call gethari();
```

STRING FUNCTIONS

- select team_name ,upper(team_name) from teams ;
- select team_name,lower(team_name) from teams;
- select length(team_name) from teams ;
- select substring(team_name ,1 , 3) from teams ;
- select substring(team_name from 1 for 3) from teams;
- select left(team_name,5) from teams ;
- select right (team_name,3) from teams ;
- select trim(concat(' ',team_name,' '))from teams;
- select concat(team_name,'-','team') from teams ;
- select reverse(team_name) from teams ;
- select locate('bulls',team_name)from teams ;

VIEW

```
CREATE VIEW hari as
 SELECT l.league_name, COUNT(t.league_id) as teams FROM leagues l left
JOIN teams t ON l.league_id = t.league_id GROUP BY l.league_name ;

 select* from hari;
 drop view hari;
 SHOW CREATE VIEW  hari;
```

AGGREGATE FUNCTIONS

```
 select count(*) from teams;

 select sum(points_scored) from scores;

 select min(points_scored) from scores;

 select max(points_scored) from scores;
```

TRIGGERS

- Triggers are automated actions in SQL that fire when specific events like INSERT, UPDATE, or DELETE occur on a table.
- They help enforce rules, log changes, or update related data without manual intervention.

```
create table deleted_players(player_id int,player_name varchar(100),deleted_date
datetime);

DELIMITER //
CREATE TRIGGER after_player_delete
AFTER DELETE ON Players
FOR EACH ROW
 BEGIN
 INSERT INTO deleted_players (player_id, player_name, deleted_date)
VALUES (OLD.player_id, OLD.player_name, NOW());
END //
DELIMITER ;

select* from deleted_players;
```

SUBQUERY

```
select p.player_id,p.player_name, s.points_scored from players p join scores s on p.player_id =
s.player_id where s.points_scored > (select avg(points_scored) from scores);
```

INDEX

- **Index** in SQL is a performance-boosting structure that helps speed up data retrieval by creating a quick lookup reference for table columns.
- It works like a book's index—pointing directly to rows that match a query condition, reducing the need to scan the entire table.

```
create index  idex_hari on  teams (team_name);

select * from teams t join players p on t.team_id =p.team_id;
```

THANK YOU