CSE 512- Distributed Database Systems

Group Project: Cassandrian

Part 1: Design and Implementation of a Distributed Database System

1. **Distributed Database Schema:**

Entity-relationship diagram –

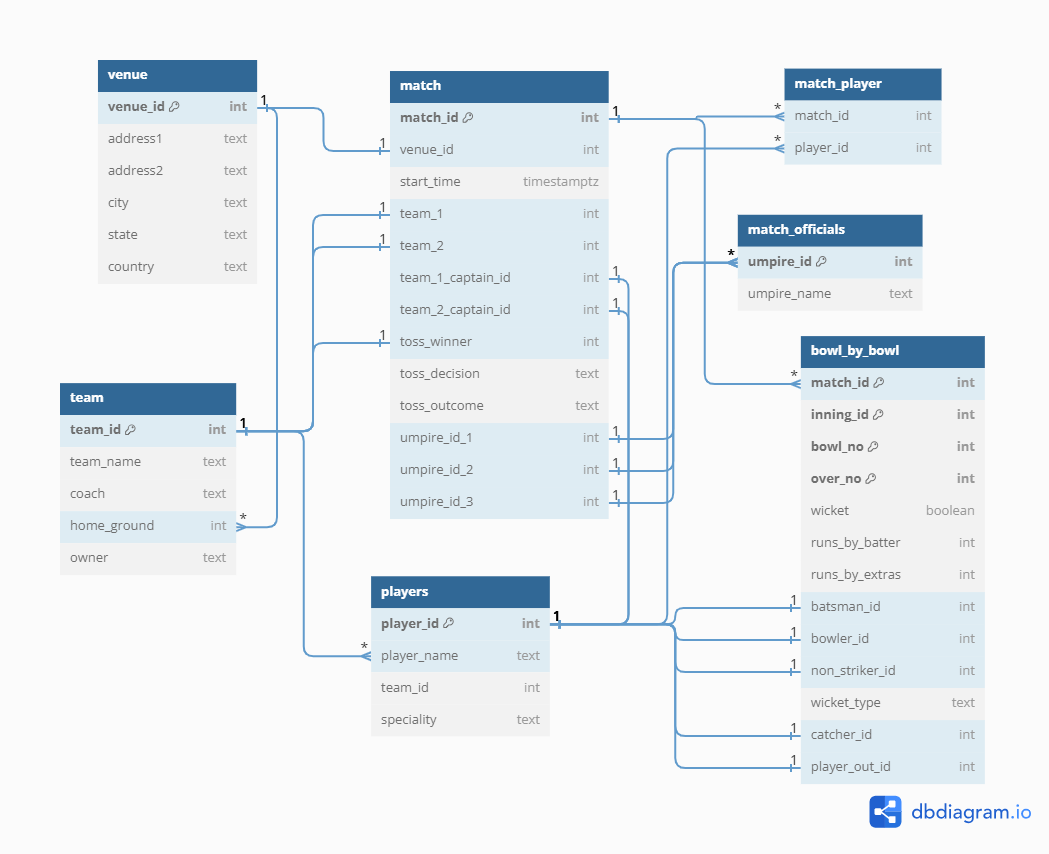


Table Definitions –

1. Venue table –

* This table contains information about the stadiums/venues for the matches.
* Attributes:

Venue\_id: Auto-incremental unique ID [Primary Key]

Address1: Address of the stadium

Address2: Landmark/Street, etc of Address

City: City of the stadium

State: State of the stadium

Country: Country of the stadium

1. Team –

* Team table contains basic information of the teams that play.
* Attributes:

Team\_id: Auto-incremental unique ID [Primary Key]

Team\_name: Name of the team

Coach: Name of the coach of the team

Home\_ground: Venue used for practice by team [Foreign Key – Venue (venue\_id)]

Owner: Owner of the team

1. Match –

* Match table contains data related to the match.
* Attributes:

Match\_id: Auto-incremental unique ID [Primary Key]

Venue\_id: Stadium where the match has/is taking place [Foreign Key – Venue (venue\_id)]

Start\_time: Starting time of the match

Team\_1: First team that’s playing the match [Foreign Key – Team (team\_id)]

Team\_2: Second team that’s playing the match [Foreign Key – Team (team\_id)]

Team\_1\_captain\_id: Captain of 1st team [Foreign Key – Players (player\_id)]

Team\_2\_captain\_id: Captain of 2nd team [Foreign Key – Players (player\_id)]

Toss\_winner: ID of team that won the toss [Foreign Key – Team (team\_id)]

Toss\_decision: Decision of the toss winning team

Toss\_outcome: Outcome of the toss

Umpire\_id\_1: ID of umpire [Foreign Key – match\_officials(umpire\_id)]

Umpire\_id\_2: ID of umpire [Foreign Key – match\_officials(umpire\_id)]

Umpire\_id\_3: ID of umpire [Foreign Key – match\_officials(umpire\_id)]

1. Match\_Player –

* This table contains information players that played for a match.
* Attributes:

Match\_id: ID of match [Foreign Key – match(match\_id)]

Player\_id: ID of player playing the match [Foreign Key – Players (player\_id)]

Primary Key [Match\_id, Player\_id] as this would always will be unique.

1. Match\_Officials –

* This table contains information of the match officials (umpires)
* Attributes:

Umpire\_id: Auto-incremental unique ID [Primary Key]

Umpire\_name: Name of the umpire

1. Players –

* This table contains basic information about the players
* Attributes:

Player\_id: Auto-incremental unique ID [Primary Key]

Player\_name: Name of the player

Team\_id: ID of the team that this player belongs to [Foreign Key – Team (team\_id)]

Specialty: Specialty of the player

1. Bowl\_by\_bowl –

* This table contains bowl-by-bowl score of the match
* Attributes:

Match\_id: ID the match [Foreign Key – match(match\_id)]

Inning\_id: Inning number of the match

Bowl\_no: Ball number for the particular match & inning

Over\_no: Over number of the match & inning

Wicket: boolean value whether wicket is taken or not

Runs\_by\_batter: runs scored by batter for the particular ball

Runs\_by\_extras: runs scored due to extras

Batsman\_id: ID of player who is batting [Foreign Key – Players (player\_id)]

Bowler\_id: ID of the player who is bowling [Foreign Key – Players (player\_id)]

Non\_striker\_id: ID of player who is non-striker for the ball [Foreign Key – Players (player\_id)]

Wicket\_type: If the wicket it taken for this ball, then type of wicket

Catcher\_id: ID of the player who caught the ball [Foreign Key – Players (player\_id)]

Player\_out\_id: ID of the player that got out because of the wicket [Foreign Key – Players (player\_id)]

**Part 2: Database table code –**

Code/Script written in part1.py

**Part 3: Data Distribution Plan –**

Since match data will be consumed by various match officials, organizations, or individual users i.e., viewers, we will be incorporating data distribution plan to minimize the latency when accessing the generated data.

Possible data distribution methods (could be changed as the project progresses):

1. Data replication:

It minimizes the time and latency when performing read operations on database tables and accessing required information. Data replication also ensures that data is always available to the end-users and eliminates issues such as Single point of failure.

Possible match tables that should be replicated:

* Match: this table contains information about the match and will be replicated
* Players: contains information about players involved in the match and their speciality
* Bowl\_by\_bowl: this table will be containing information about every bowl that takes place during the match. This information will be match\_id, bowl\_no, over\_no, among various other significant information.
* Other tables that might be replicated are venue, team, match\_officials, and match\_player. These tables will not be having continuously changing elements as compared to above 3 tables, so data replication will not result in any substantial improvement in time and latency.

Advantages of data replication are:

* Improved Read Performance: Users can access information about teams and players from a nearby node, reducing latency.
* Fault Tolerance: If one node goes down, users can still access team and player information from other nodes.
* Load Balancing: The read load is distributed among multiple nodes, improving overall system performance.

1. Horizontal fragmentation:

Since match data will be relatively huge in size, horizontal fragmentation of the table will result in efficient read operation.

For match data, tables can be fragmented based on geographical locations of the venues, or teams.

Possible fragmentations are as follows:

* Fragmentation 1: Matches involving Team India
  + Match 1: Team India vs Team Australia
  + Match 2: Team India vs Team England... so on and so forth
* Fragmentation 2: Matches involving Team Australia
  + Match 1: Australia vs Netherlands
  + Match 2: Australia vs Sri Lanka

Advantages of Horizontal fragmentations:

* Improved Query Performance: Horizontal fragmentation allows for parallel processing of queries. When queries are focused on a specific fragment, multiple fragments can be processed simultaneously, leading to improved query performance.
* Reduced Data Transfer Overhead: Since each fragment contains a subset of the data, queries that only require information from a specific fragment can minimize the amount of data transferred over the network. This reduces network traffic and improves overall system efficiency.
* Enhanced Scalability: Horizontal fragmentation facilitates the distribution of data across multiple servers or nodes. As the data grows, additional nodes can be added to the system, and the workload can be distributed, resulting in improved scalability.
* Increased Security: Horizontal fragmentation can enhance data security by restricting access to specific fragments. Access controls can be applied at the fragment level, ensuring that users or applications only have access to the data relevant to their needs.
* Customized Indexing and Optimization: Each fragment can be indexed and optimized independently based on the specific characteristics of the data it contains. This customization allows for more efficient storage structures and indexing strategies tailored to the nature of the data within each fragment.
* Support for Distributed Systems: Horizontal fragmentation aligns well with the principles of distributed systems, where data is distributed across multiple nodes. This makes it easier to design and implement distributed databases or systems that can scale horizontally.

1. Range-Based partition:

Match data can be partitioned based on various ranges as required by the database designer.

Few of the examples are as follows for match data:

* Partition 1: Matches played between 2019-2023: this partition will display all the matches that were played between the year 2019 and 2023.
* Partition 2: Matches played from 2015-2018.
* partition 3: matches played from 2010-2014

We can also partition based on runs scored by a batsman or wickets taken by a bowler. There are various other possible range partitions.

Advantages of range-based partition are as follows:

* Improved Query Performance: It can significantly enhance query performance, especially when queries involve ranges of data.
* Parallel Processing: It enables parallel processing of queries and data operations. Different partitions can be processed concurrently by different nodes or servers, improving overall system throughput, and reducing query response times.
* Scalability: As the dataset grows, range-based partitioning allows for easier scalability.
* Better Load Balancing: It is possible to achieve better load balancing across nodes. If the data is evenly distributed based on a logical criterion like date ranges, each node can handle a similar workload, preventing hotspots and ensuring a more balanced system.
* Ease of Data Retrieval for Time-Dependent Queries: It is particularly beneficial for time-dependent queries, where users are interested in data within specific time intervals. It allows for quick isolation of the relevant partition, leading to faster query response times.

**4. Data insertion mechanism:**

We used a real world dataset from cricsheet website. Specifically, we used ICC T20 Men’s World Cup 2014, 2016. It’s single CSV file which had also the details of the specific match but we can’t directly import that csv to our database. We had to write data cleaning and data extraction functions in python jupyter notebook and created multiple csv files according to our table schema to later import those csv’s into our database. After extraction, we got 15,224 ball by ball data for the 2 World Cups.

**5. Data retrieval proof:**

We wrote 5 sample queries which accesses at most all tables we had in our database and provided results for the required queries without any issues.

The Sample queries are  
1. List of matches played at a specific venue (Eg. Chinnaswamy Stadium)

2. List of matches officiated by a specific umpire (Adrian Holdstock)

3. List of matches where specific team was a team winner (Eg. Indian Cricket Team)

4. List of bowlers who have taken at least one wicket

5. List of matches where specific player was captain of the team (Eg. MS Dhoni)