CM3010: Databases and Advanced Data Techniques

Midterm Assignment

Arjun Muralidharan

24th December 2021

Contents

1 Find and critique a dataset

1.1 Finding the dataset

I will use a dataset of sports matches, specifically test cricket matches from 2004 until 2021, obtained from [1].

The data describes ball-by-ball statistics and can be considered **pre-existing external data**. It is assembled with automated **data curation** to assemble the data from various sources. Some general points on the structure of the data:

- Each match's data is organised in a separate JSON file.
- Each file includes some meta data about the creation and the event that this match was part of.
- All ball-by-ball data is stored in **nested objects** within the JSON structure.
- This leads to lots of repetition of information as this follows a **tree-like** structure rather than a relational one.

Hence the data needs to be normalized and mapped to a relational model in order to use it in a relational database.

1.2 Critiquing the dataset

One advantage of the dataset is that it is very **up to date** as new matches are added every day. This makes the data set highly usable and accurate, at least starting from 2004 onwards. The dataset is also **lightweight to use** given that each match has it's own file; new data does not change or affect the size and integrity of previous data.

However, the data is distributed and hence **harder to consolidate** into a single repository, and it contains lots of repeated data as player names occur again for each individual delivery (a ball played) where the player was involved. Matches that belong to the same event or test match series are in separate files and the event name is repeated in all the files related to an event.

Event names are further not unique, e.g. "New Zealand tour of India" may have happened many times in the past 17 years, so the year of the event needs to be considered when defining what a unique event is.

1.3 Interest in the data set

I chose this data set as I personally enjoy watching Cricket as a sport, and believe the data set to be interesting for the following reasons:

- Cricket is a statistics-heady sport, with lots of data recorded officially on a ball-by-ball basis

- Generally, sports data is rarely available in an openly available format and often hidden behind paywalls or graphical interfaces, gatekept by sports companies and sports news sites. This dataset made openly available a lot of information about this fascinating sport.
- The data is structured in an intersting manner that in its current form is not suited for relational representation and many questions cannot be answered without transforming the data into a relational form.

1.4 Questions to answer

By transferring a part of the data into relational form, I aim to answer the following questions which cannot be answered with the current source structure.

1. How many test matches have the test-playing nations played since 2004?
2. Who are the 10 batsmen with the highest number of runs scored since 2004?
3. Which 10 players are the most successful bowlers in terms of wickets taken and runs conceded since 2004?
4. What is the most common type of dismissal since 2004?

1.5 Model your data

I have built a tentative ER diagram shown in Figure 1. Some notes on normalization:

1. The core entities are *players* and *matches*. A match in cricket is played through *deliveries*, or individuall balls. Hence, players are involved in deliveries, and called up to line-ups to play in matches. These relationships are modelled with linking entities that would eventually result in link tables.
2. This ER diagram is not a complete representation, as further, more granular normalization might be warrented based on the use case. For example, the entitiy **delivery role** might need an entity that stores the possible roles, from which an attribute is drawn (e.g. bowler, batter, fielder, etc.).

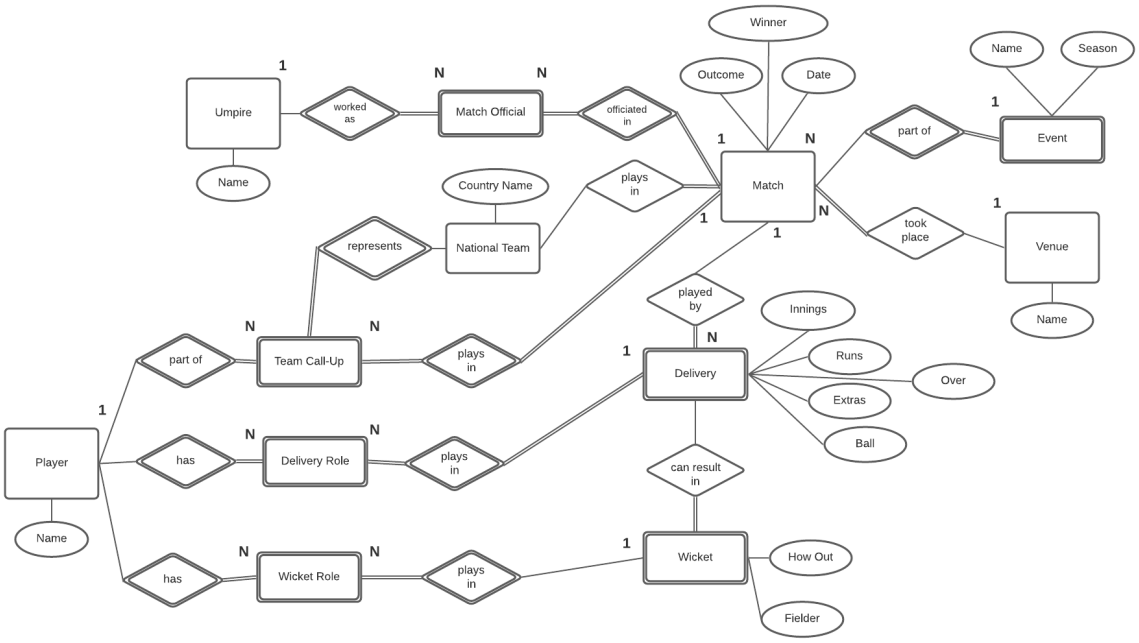

Figure 1. *ER diagram for Cricsheet data*

1.6 Create the database

The database implementation does not completely reflect the ER diagram exactly, because the scope far exceeds the current needs of answering the questions. Therefore, I took a slightly simplified approach and did not implement every link table shown above. I recognize this shortcoming but want to demonstrate that implementing additional link tables would be trivial.

Listing 1 Listing: Database Implementation

```
CREATE TABLE `player` (
  `name` VARCHAR(100),
  PRIMARY KEY (`name`)
);

CREATE TABLE `team` (
  `country_name` VARCHAR(50),
  PRIMARY KEY (`country_name`)
);

CREATE TABLE `match` (
  `match_id` INT NOT NULL AUTO_INCREMENT,
  `team_A` VARCHAR(50) NOT NULL,
  `team_B` VARCHAR(50) NOT NULL,
  `date` DATE NOT NULL,
  `winner` VARCHAR(50),
  `outcome` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`match_id`),
  FOREIGN KEY (`team_A`) REFERENCES `team`(`country_name`) ON DELETE CASCADE,
  FOREIGN KEY (`team_B`) REFERENCES `team`(`country_name`) ON DELETE CASCADE,
  FOREIGN KEY (`winner`) REFERENCES `team`(`country_name`) ON DELETE CASCADE,
  CONSTRAINT unique_match UNIQUE (`team_A`, `team_B`, `date`)
);

CREATE TABLE `delivery` (
  `match_id` INT NOT NULL,
  `innings` INT NOT NULL,
  `over` INT NOT NULL,
  `ball` INT NOT NULL,
  `runs` INT NOT NULL,
  `extras` INT NOT NULL,
  `batter` VARCHAR(50) NOT NULL,
  `bowler` VARCHAR(50) NOT NULL,
  `nonstriker` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`match_id`,`innings`,`over`,`ball`),
  FOREIGN KEY (`match_id`) REFERENCES `match`(`match_id`),
  FOREIGN KEY (`batter`) REFERENCES `player`(`name`),
  FOREIGN KEY (`bowler`) REFERENCES `player`(`name`),
  FOREIGN KEY (`nonstriker`) REFERENCES `player`(`name`)
);

CREATE TABLE `wicket`(
  `playerout` VARCHAR(50) NOT NULL,
  `fielder`VARCHAR(50),
  `kind` VARCHAR(50),
  `match_id` INT NOT NULL,
  `innings` INT,
  `over` INT,
  `ball` INT,
  PRIMARY KEY (`match_id`,`innings`,`over`,`ball`),
  FOREIGN KEY (`match_id`,`innings`,`over`,`ball`) REFERENCES `delivery`(`match_id`,`innings`,`over`,`ball`),
  FOREIGN KEY (`playerout`) REFERENCES `player`(`name`)
);
```

Reflecting on this implementation, for a more scalable version I would implement more linking tables. Also, I have not implemented tables unneeded for the questions posed, such as Umpires, Events or Venues. If I were building a more full-fledged cricket scoring app, I would implement this analogously.

1.7 Create a simple web application

The web application is hosted in the lab environment. Some notes in running it:

1. It can be started with the `npm start` command.

2. The database is already populated, but can be re-populated by running `node db/cric.js`. This instantiates a class that processes the JSON files in the `matches` folder and adds them to the database. Because this operation can be long and heavy on the database, I decided to decouple this from running the application.

3. The application requires about a minute (in the lab environemnt) to query the database, as I have not implemented any caching of query results. Because some queries need to sift through every ball every played since 2004, this takes a moment. The app presents a warning and asks you to refresh the page after waiting a minute.

The app displays all the associated SQL queries on-demand. This functionality may not work in the lab environment as the browser preview can be slow and tedious to interact with; hence I recommend installing the app locally if desired.

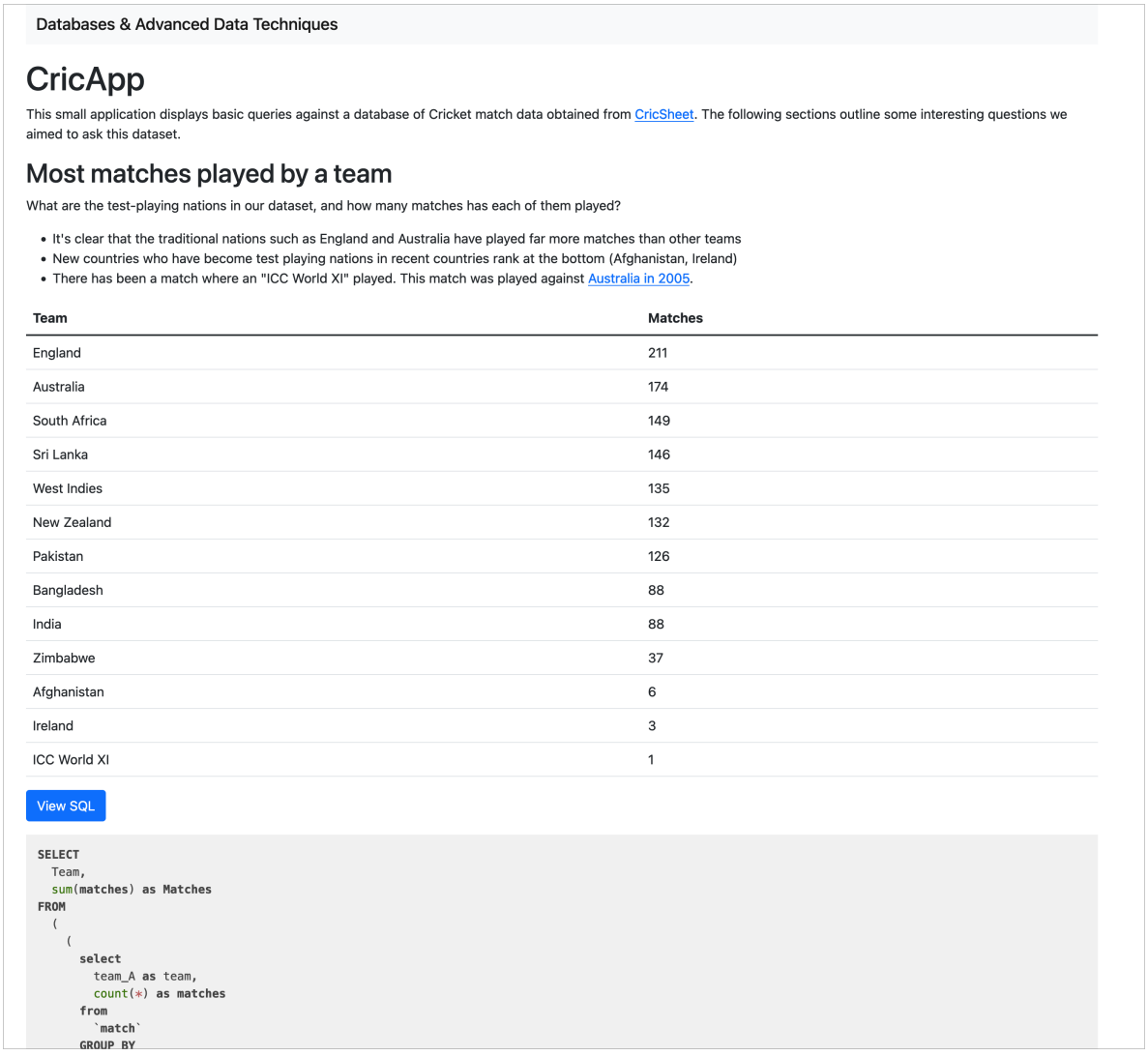

Figure 2. *Screenshot of the working app*

References

[1] S. Rushe, "Cricsheet Matches," Dec 2021. [Online]. Available: https://cricsheet.org/matches/