**CIS550 Project Writeup**

by group 21

**Group members: Hongru Du, Jiapo Tai, Jiawei Liu and Yuechen luo**

**Idea and Motivation:**

In addition to the basic functions of an Olympic Games database, we would like to conduct some interesting queries such as “Is there a limitation on the strength/speed of mankind”, “Is the total performance of a nation in Olympic Games dependent on its population scale” We are curious about whether the sky is the limit, and we regard these questions as a way of interpreting the so-called Olympic Spirit.

**Modules and architecture:**

**Schema Design:**

/Users/hongrudu/Downloads/ER_161123 (1).png

Based on the given .csv file, we constructed a schema with the following tables:

**Nation *(IOC, ISO Code, NName*)**

Contains the basic attributes (names, IOC code) of countries that would be used in our database instance.

(e.g. “BOL”,“BO”, “Bolivia”).

**Nation\_Owns\_Population (*IOC, Year*, *Population*)**

Includes the information of the population of countries in the Olympic years. This is a weak entity as population is a sort of attribute belongs to a nation. (we assume that the population is almost the same in the Olympic years as the closest reference years.)

(e.g. “ALB”, 1952, 1263).

**Athlete\_Belongs\_To (*IOC, AName, Gender*)**

Stores the name, gender and nationality of an athlete. This is also a weak entity since an athlete must represent a nation to participate in Olympic games.

(e.g. “USA”,“PHELPS, Michael”, “Men”).

**Olympic\_Game (*Edition, City*)**

Indicates the specific Olympic Games with specific year edition.

(e.g. 2008, “Beijing”).

**Event\_Of (*EName, DName, Edition*)**

Contains an event with its discipline name and edition information. It is a weak entity of the Olympic Games.

(e.g. “400m”, “freestyle”, 1996).

**Attend (*IOC, AName, Gender, EName, DName, Edition, Medal, Record*)**

Stores information about which athlete participated in which event of which Olympic Games and won a medal. The *Record* attribute part was moved to our NoSQL database, so most of the *Record* content is 0 in our SQL database.

(e.g. “USA”,“PHELPS, Michael”, “Men”, “100m butterfly”, “Swimming”, 2004, “Gold”, 0).

**Discipline (*DName, Sports*)**

Contains a more general way to classify the disciplines.

(e.g. “Swimming”, “Aquatics”).

**Data Cleaning:**

In order to omit dirty data insertion, there are some additional constraints of the attributes in our DDL:

CHECK (GENDER in (‘Men’, ‘Women’));

CHECK (Medal in (‘Gold’, ‘Silver’, ‘Bronze’));

CHECK (Edition >= 1896);

Also, some of the attributes are fixed in length and type:

IOC char(3),

ISOCode char(2)

Year integer,

Edition integer,

Population integer;

Any insertions that conflicts with the above constraints cannot be executed.

**Data Importing:**

For the relational part, we converted the given .csv file to SQL and inserted all the tuples into the database. We also chose alternative resources for our MongoDB. We gathered information about Olympic Records from the following website:  
<http://www.theolympicdatabase.nl/olympic/sports>.

This database contains the names, nationalities, rankings, disciplines, sports and genders of all the Olympic medal winners and their corresponding performance. This information can be used to write queries like “What is the tendency of Olympic gold records?”.

We also extracted information of GDP from Wikipedia page: <https://en.wikipedia.org/wiki/List_of_countries_by_past_population_(United_Nations,_estimates)>

This webpage contains past populations of all the nations, which can be used to answer questions like “What is the amount of gold medal of a country per capita in the year of XXXX?”

Additionally, since the source code of these websites is written in plain html, we wrote a program (refer to AutoParser.java for source code) that can automatically parse this information into formatted text files, which enables us to process data in a very efficient manner. Followed by this procedure we can choose either to parse the .txt files into DDL or JSON documents and then populate the database.

In total, we chose the following 17 disciplines, which accounts for 874 records in the MongoDB:

100m athletics (men)

10000m athletics (men/women)

100m butterfly swimming (men/women)

200m freestyle swimming (men/women)

400m freestyle swimming (men/women)

800m freestyle swimming (women)

1500m freestyle swimming (men)

Triple jump (men/women)

Long jump (men/women)

Discus throw (men/women)

We didn’t choose any competitive sports such as basketball and boxing because the corresponding performance cannot be quantified.

**Data instance use:**

The entire database is composed of both the relational part and the non-relational part. For the former we have set up an Oracle SE One instance (11.2.0.4.v9) on Amazon Web Service. The host name is cis550group21db.c56jamskv0nb.us-west-2.rds.amazonaws.com, with Port number of 1521. User name and password of this relational database are both “CIS550Project”. The SID of this database is ORCL.

For the nosql part we launched a MongoDB instance on mLAB. mLAB is a cloud service based on AWS EC2, but it’s easier to adopt as it can be directly accessed by Node.js.

The host name of this database is ds127968.mlab.com:27968/cis550project and “cis550project” is used both as the username and the password.

**Queries and Optimization:**

Based on the previous design of the schema, our application mainly provides three functional parts consisting of 10 different query scenarios:

**Part 1. Medal numbers with ranks of all countries:**

1. Over the years, the sum of medals and rank of all the countries
2. Over the years, the sum of medal and rank of all the countries in specific event of specific discipline
3. In specific edition of Olympic Games, the sum of medals and rank of all the countries
4. In specific edition of Olympic Games, the sum of medals and rank of all the countries in specific event of specific discipline

**Part 2. Medal numbers with ranks of all Olympic games**

1. Over the years, the sum of medals and rank of all Olympic games
2. Over the years, the sum of medals and rank of all Olympic games in specific discipline
3. Over the years, the sum of medals and rank of all Olympic games in specific event of specific discipline

**Part 3. Average medal numbers per million nation population**

1. In specific edition of Olympic Games, the sum of medals and rank of all the countries and its average medal numbers per million population
2. In specific edition of Olympic Games, the sum of medals and rank of all the countries in specific discipline and its average medal numbers per million population
3. In specific edition of Olympic Games, the sum of medals and rank of all the countries in specific event of specific discipline and its average medal numbers per million population

All above 10 query scenarios own a common part as the number of medals should be derived from medal attribute inside “Attend” table. However, medal data in “Attend” table have covered all medals for different athletes even though they are teammates in some group sports event (e.g. basketball). Hence, this would lead to some redundancy for medal statistics as medal could only count as one for each kind of group sports event for each country instead of each athlete. Therefore, to solve this problem, first, we use ‘’With as” to generate a new table named “Medals” by joining “Attend” table and “Nation” table. The SQL is shown as below:

|  |
| --- |
| With Medals As (  Select A.Edition, N.NName, N.IOC, A.EName, A.DName, A.Gender, A.Medal, count(distinct A.Medal) as MedalNum  From Nation N left Join Attend A on N.IOC = A.IOC  Group by A.Edition, N.NName, N.IOC, A.EName, A.DName, A.Gender, A.Medal) |

For 4 query scenarios in Part 1, first, they could be easily obtained by joining “Medals” table with “Nation” table on attribute IOC and counting number of medals as shown below:

|  |
| --- |
| Select N.NName, N.IOC, count(M.MedalNum)  From Nation N left join Medals M on N.IOC = M.IOC  Where condition  Group by N.NName, N.IOC  Order by count(M.MedalNum) desc, N.NName, N.IOC |

And then they are divided into four cases, respectively, corresponding to different user combinations of selections. The “where” conditions are shown as below:

|  |
| --- |
| where M.EName = '#Event' and M.DName = '#Discipline'  where M.Edition = '#Edition'  where M.Edition = '#Edition' and M.EName = '#Event' and M.DName = '#Discipline' |

For 3 query scenarios in Part 2, Edition (Year) is considered as mandatory choice and then count of medals could be retrieved only from “Medals” table. And it is divided into three cases, respectively, corresponding to different user combinations of selections. The SQL code and “where” conditions are shown as below:

|  |
| --- |
| select edition, count(MedalNum) from Medals  group by edition order by edition desc |

|  |
| --- |
| where M.Edition = '#Edition'  where M.Edition = '#Edition' and M.DName = '#Discipline'  where M.Edition = '#Edition' and M.EName = '#Event' and M.DName = '#Discipline' |

For 3 query scenarios in Part 3, they should first join “Medals” table with “Nation” table on attribute IOC. Moreover, it is necessary to join the “Nation\_Owns\_Population” table on attribute Edition (Year) and counting number of medals as shown below:

|  |
| --- |
| select N.NName, N.IOC, count(M.MedalNum), NP.Population  from Nation N left join Medals M on N.IOC = M.IOC  inner join Nation\_Owns\_Population NP  on NP.IOC = A.IOC and NP.Year = A.Edition and N.IOC = NP.IOC  group by N.Name, N.IOC, NP.Population  order by count(M.MedalNum) desc, N.NName, N.IOC |

And then they are divided into three cases, respectively, corresponding to different user combinations of selections. The where conditions are same as part 2’s.

For all the queries used in our application, please refer to Supporting Information part 1

All of query are based on “Medals” and joins at least two tables. In terms of results the queries given, they are user-interested and useful data extracted from both provided and external Olympic Game database.

The query optimization starts from design of schema structure, as schema fully considers the realization of functions our applications provided. So, above queries not only achieve the functional requirements, but also has great efficiency and performance in actual query time.

Data retrieval time is short and the retrieval rate is very fast, even if it dynamically renders the data to the page displaying to the user. The results shown in query time are short enough to achieve the requirements. Schema provides convenience for data retrieval, and design of query based on this schema is the biggest reason for query optimization.

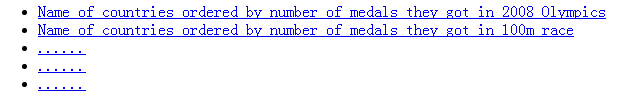
**Technical Specifications and Special Features:**

**• Bootstrap & Jquery based Front-end Pages**

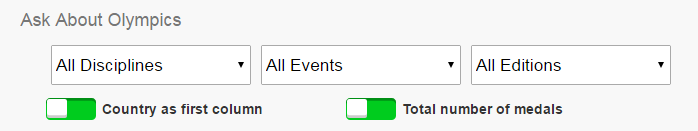
Front-end pages are developed based on bootstrap framework, along with jquery. With all of these, the website are made more concise, more efficient and most importantly, more user-friendly. The website can apply to numbers of scenarios while, at the same time, stay simple and easier to use. Users can get well-presented data that they want in three or two, even one click. The use of jquery also made some dynamic effect possible. Not fancy but improve the user experience.

**• Ajax Based Data Request**

With ajax from jquery, front pages can exchange data with server and update part of the page content without reloading the whole page. Data presenting process become far more smooth and dynamic select box become possible. As shown in the figure X.X, instead of giving the query choices separately, the pages are capable of combining dozens of queries in several select box.



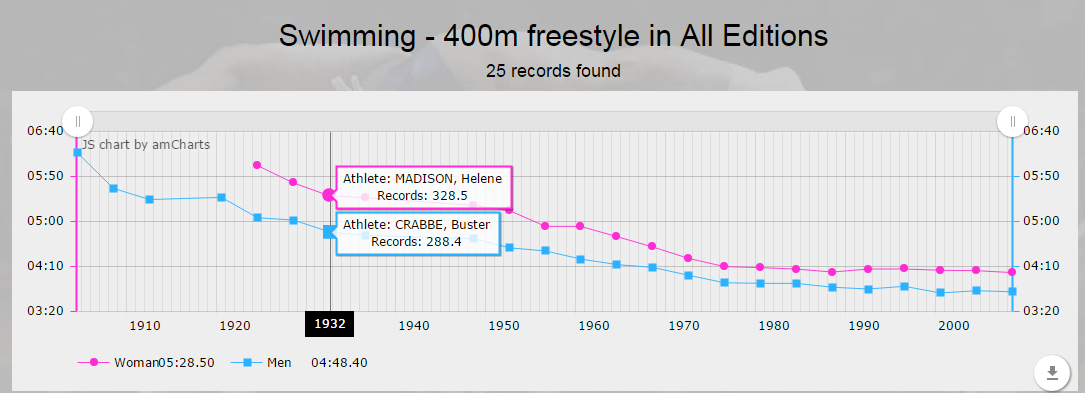
List of query choices



Dynamic select boxes

**• Graphical Presentation of Data Based on Amchart**

Javascript graph application amchart make the page capable of presenting the result data in a nice and clear graph. In this case, the best record of some certain disciplines can be visualized. For example: 400m freestyle swimming (Shown in the following figure).

****

Graph made by amchart

**Division of work:**

Jiawei Liu: Query handling, Node.js technical support (22% workload)

Jiapo Tai: Database setup, data parsing and import (22% workload)

Hongru Du: Schema design, data insertion (22% workload)

Yuechen Luo: Node.js design and coding (34% workload)