Introduction to Database Systems

Olympic Games

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Bastien Antoine (203267)
Denoréaz Thomas (183785)
Dieulivol David (185078)

Academic years 2012-2013
(April 21, 2013)

Contents

1 Entityrelationship model

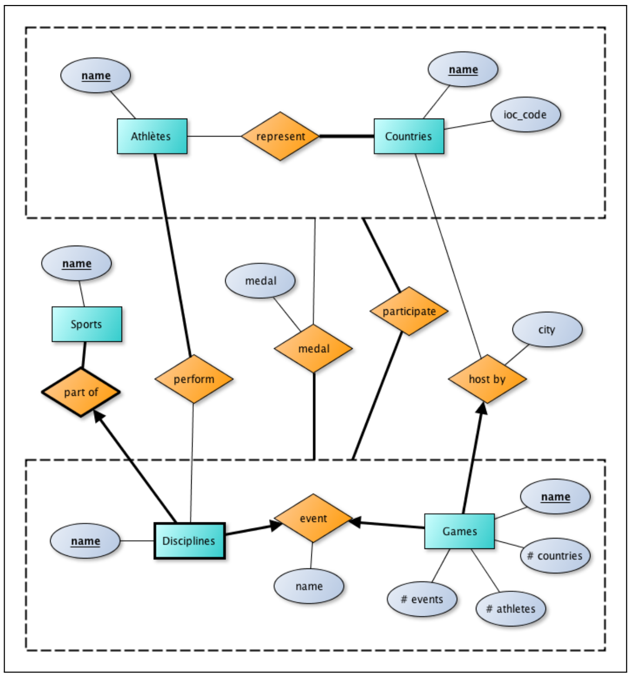


Figure 1.1: Previous ER Model.

From the analysis of the Dataset, here are our assumptions:

- An **Athlete** is always performing a **Discipline** instead of just a **Sport**.
- An **Athlete** can represent only a **Country** for a **Game**. However, he can represent another **Country** for another **Game**.
- A **Game** can only be hosted by one and only one **Country**, but this **Country** can host several **Games**.
- Each **Discipline** is defined by its **Sport**.
- An *Event* is characterized by only a **Game** and only a **Discipline**.
- A *Medal* is obtained for a *Representant* during an *Event*.
- A *Participant* is formed by both a *Representant* and an *Event*.

Changes since deliverable 1

After the first deliverable, we have made some simplifications to our model. There are still two aggregations standing for a representative (**athlete** and **country**) and an event (**discipline** and **games**). These aggregations are bonded by the relation *Representant_participates_Event* which models the participation from a representative to a **discipline**. We have removed the other relations between them because there is only redundant information and we can put the medal attribute in the participation relation.

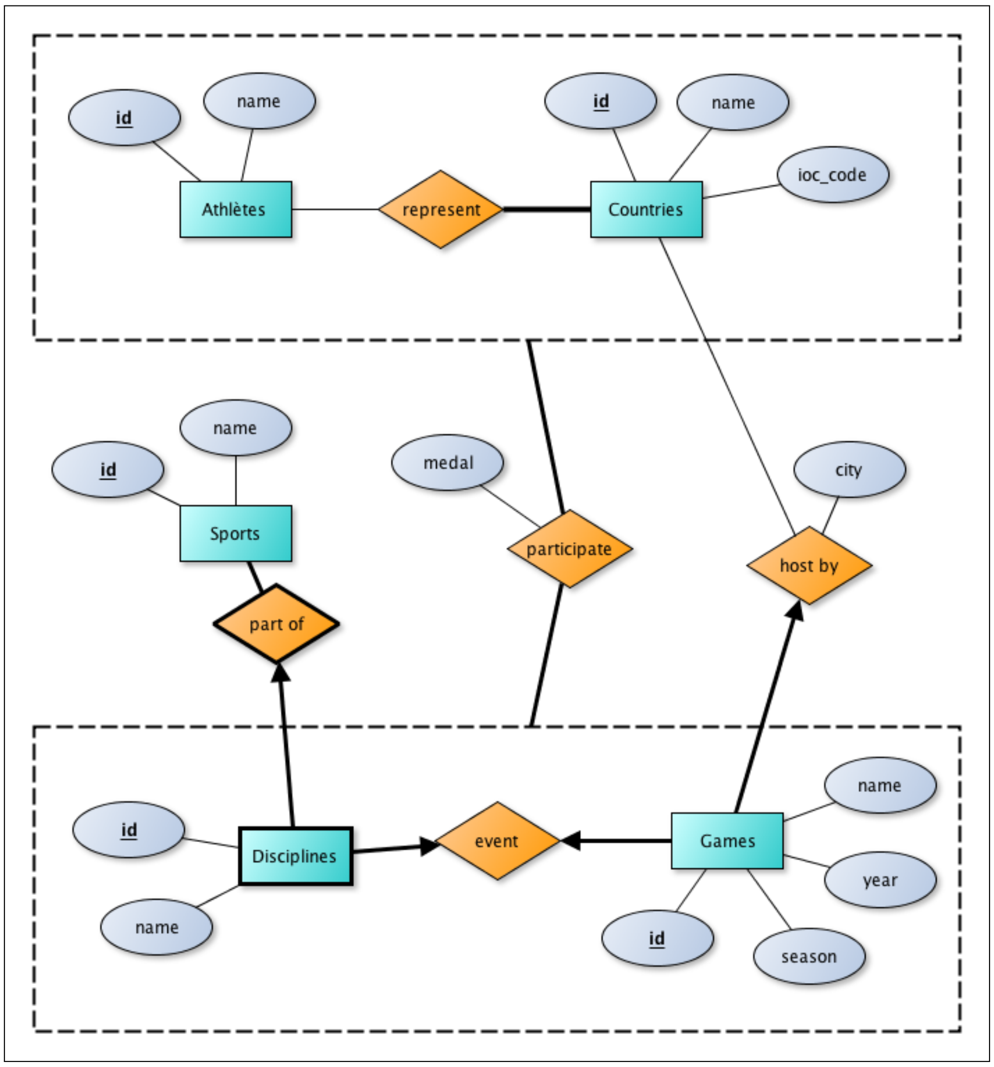


Figure 1.2: New ER Model.

2 Relational schema and constraints

2.1 Relational schema

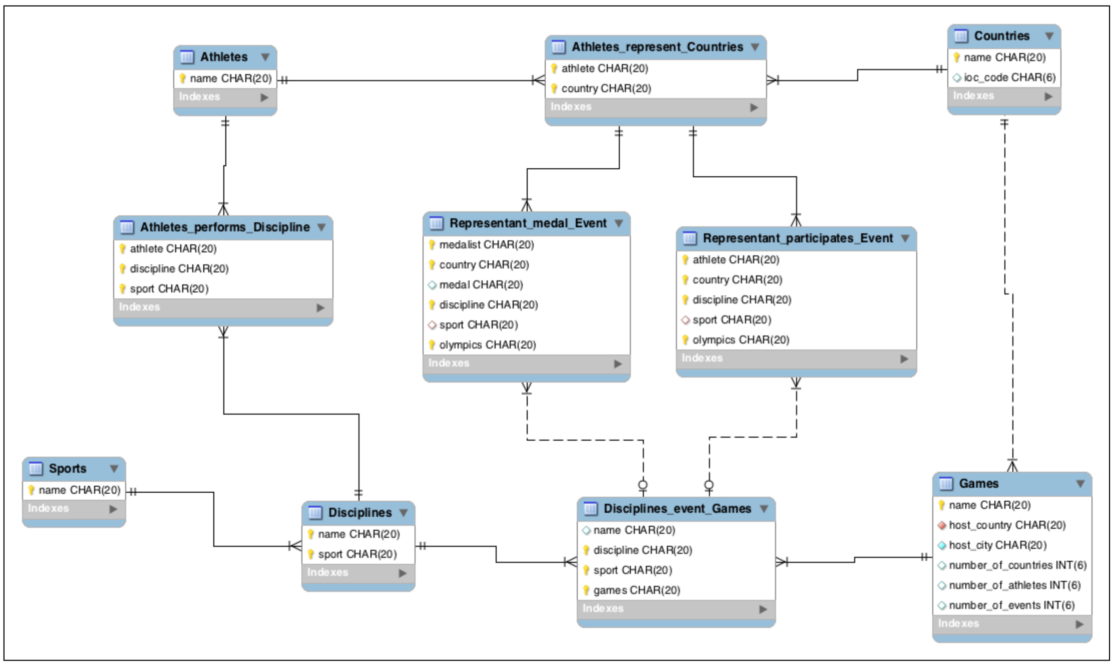


Figure 2.1: Generated EER Model from MySQL Workbench.

After implementing the DDL from Section 2.2, we generated the scheme in Figure 5.3 using MySQL Workbench.

2.2 SQL Data definition language statements

We decided to implement our project, using the Oracle MySQL database management system. Following is the listing of our entities and relations.

```sql
CREATE TABLE Athletes (
    id                          integer AUTO_INCREMENT,
    name                        char(255),
    PRIMARY KEY (id)
);

CREATE TABLE Countries (
    id                          integer AUTO_INCREMENT,
    name                        char(60),
    ioc_code                    char(6),
    PRIMARY KEY (id)
);

CREATE TABLE Sports (
    id                          integer AUTO_INCREMENT,
    name                        char(60),
    PRIMARY KEY (id)

);

CREATE TABLE Games (
    id                          integer AUTO_INCREMENT,
    year                        integer(4),
    is_summer                   boolean,
    host_country                integer NOT NULL,
    host_city                   char(60)  NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (host_country) REFERENCES Countries (id)
);

CREATE TABLE Disciplines (
    id                          integer AUTO_INCREMENT,
    name                        char(100),
    sport_id                        integer NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (sport_id) REFERENCES Sports (id)
        ON DELETE CASCADE
);
```

Listing 2.1: DDL Entities

```sql
CREATE TABLE Athletes_represent_Countries (
    athlete_id                    integer,
    country_id                    integer,
    PRIMARY KEY (athlete_id, country_id),
    FOREIGN KEY (athlete_id) REFERENCES Athletes (id),
    FOREIGN KEY (country_id) REFERENCES Countries (id)
);

CREATE TABLE Disciplines_event_Games (
    discipline_id                 integer,
    games_id                      integer,
    PRIMARY KEY (discipline_id, games_id),
    FOREIGN KEY (discipline_id) REFERENCES Disciplines (id),
    FOREIGN KEY (games_id) REFERENCES Games (id)
);

-- Here Event is a shortcut to table Disciplines_event_Games

CREATE TABLE Representant_participates_Event (
    athlete_id                  integer,
    country_id                  integer,
    discipline_id               integer,
    games_id                    integer,
    ranking                     tinyint(2),
        PRIMARY KEY (athlete_id, country_id, games_id),
    FOREIGN KEY (athlete_id, country_id) REFERENCES
        ➥Athletes_represent_Countries (athlete_id, country_id),
    FOREIGN KEY (discipline_id, games_id) REFERENCES
        ➥Disciplines_event_Games (discipline_id, games_id)
);
```

Listing 2.2: DDL Relations

3

Data importation

The main issue that appeared while importing the data is that we cannot have the discipline for each athlete. This is problematical for the relation *Representant_participates_Event*. There is information in the csv files about the sport that practices each athlete but the sport cannot define an event. So we have decided not to set the discipline as a primary key so that all non- medalists can still be stored in the database.

```
#!/usr/bin/env ruby

=begin

  Group 12
  203267 Bastien Antoine
  183785 Denoreaz Thomas
  185078 Dieulivol David

  This script will import data from CSV files given      in folder
     ➥"dataset" to our database.

  ################### Known problems ###################

  1 - Some participants have a weird name. This is due to the
     ➥encoding Done! above. Hopefully it is seldom the      case.

  2 - In the DDL file, we have to delete discipline_id as a primary
     ➥key because during Import    #6 it caused a problem while we
     ➥added the participants (they all have a       NULL
     ➥discipline_id).

  3 - In the  end, all the athletes that will    not have any medals
     ➥will not have a discipline_id, since this information is      not
     ➥given in the dataset. On all the athletes, only 682 of them
     ➥will have a medal (that  's the data I have with the current
     ➥dataset at least) :
      "SELECT * FROM Representant_participates_Event RE WHERE
         ➥discipline_id IS NOT NULL"

  ############### End of Known problems ################

  ###################### Notes ########################

  2 - If you have a problem with the original CSV files, do this
     ➥command to convert them :
```

```ruby
        iconv -t UTF8 -f LATIN1 < athletes_old.csv > athletes.csv

   3 - The default params for the database connection are as follows:
        # db = Mysql.new('host', 'username', 'password', 'your_table')

   ################## End of Notes ####################

=end

def usage
  puts "Usage: ./import_script.rb start [debug]"
  exit
end

case ARGV.size
  when 1 then usage if ARGV[0].downcase != "start"
  when 2 then @debug = ARGV[1].downcase == "debug" ? true : usage
  else usage
end

require 'mysql'
require 'CSV'

db = Mysql.new('localhost', 'root', '', 'db_project_group_12')

# Import #1 : Athletes
puts "Import #1 : Athletes"
i = 0
CSV.foreach("dataset/athletes.csv") do |row|
  db.query("INSERT INTO Athletes (name) VALUES ('#{row.first.gsub("'", "\\\\'")}')") if i > 0
  i += 1
end
puts "Done!"

# Import #2 : Countries
puts "Import #2 : Countries"
i = 0
CSV.foreach("dataset/countries.csv") do |row|
  db.query("INSERT INTO Countries (name, ioc_code) VALUES ('#{row[0]}', '#{row[1]}')") if i > 0
  i += 1
end
puts "Done!"

# Import #3 : Sports
puts "Import #3 : Sports"
i = 0
CSV.foreach("dataset/sports.csv") do |row|
  db.query("INSERT INTO Sports (name) VALUES ('#{row[0]}')") if i > 0
  i += 1
end
puts "Done!"

# Import #4 : Disciplines
```

```ruby
puts "Import #4 : Disciplines"
i = 0
CSV.foreach("dataset/disciplines.csv") do |row|
  if i > 0
    result = db.query("SELECT id from Sports S WHERE
       ➥S.name='#{row[1]}'")
    result.each_hash {|h| db.query("INSERT INTO Disciplines (name,
       ➥sport_id) VALUES ('#{row[0].gsub("'", "\\\\'")}',
       ➥'#{h['id']}')")}
  end
  i += 1
end
puts "Done!"

# Import #5 : Games
puts "Import #5 : Games"
i = 0
CSV.foreach("dataset/games.csv") do |row|

  if i > 0
    year, winter_or_summer = row[0].split(" ")
    is_summer = winter_or_summer.downcase == "summer" ? 1 : 0
       ➥unless winter_or_summer.nil?
    host_city, host_country = row[4], row[5]

    result = db.query("SELECT id from Countries C WHERE
       ➥C.name='#{host_country}'")
    result.each_hash do |h|
      db.query("INSERT INTO Games (year, is_summer, host_country,
         ➥host_city) VALUES ('#{year}', '#{is_summer}',
         ➥'#{h['id']}', '#{host_city.gsub("'", "\\\\'")}')")
    end
  end
  i += 1
end
puts "Done!"

# Import #6 : Events
puts "Import #6 : Events (long one...)"
i = 0
CSV.foreach("dataset/events.csv") do |row|

  discipline = nil
  unless row[1].nil? or row[2].nil?
    result = db.query("SELECT id from Disciplines D WHERE
       ➥D.name='#{row[1].gsub("'", "\\\\'")}'")
    result.each_hash do |h|
      discipline = h['id'] if i > 0
    end

    year, is_summer = row[2].split(" ")
    is_summer = is_summer.downcase == "summer" ? 1 : 0 unless
       ➥is_summer.nil?
    result = db.query("SELECT id from Games G WHERE
       ➥G.year='#{year}' AND G.is_summer = '#{is_summer}' ")
    result.each_hash do |h|
      begin
        db.query("INSERT INTO Disciplines_event_Games
           ➥(discipline_id, games_id) VALUES ('#{discipline}',
           ➥'#{h['id']}')") if i > 0 and !discipline. nil?
      rescue => ex
        puts "DEBUG (events) : #{ex.message}" if @debug
      end
    end
  end
  i += 1
end
puts "Done!"

# Import #7 : Participants
i = 0
CSV.foreach( "dataset/participants.csv") do |row|
  if i > 0 and !row[0]. nil? and !row[1]. nil? and !row[2]. nil?
    athlete_id = nil
    result = db.query( "SELECT id from Athletes A WHERE
       ➥A.name='#{row[0].gsub("'", "\\\\'")}'")
    result.each_hash do |h|
      athlete_id = h[ 'id']
    end

    year, is_summer = row[2].split( " ")
    is_summer = is_summer.downcase == "summer" ? 1 : 0 unless
       ➥is_summer. nil?

    games_id = nil
    result = db.query( "SELECT id from Games G WHERE
       ➥G.year='#{year}' AND G.is_summer = '#{is_summer}' ")
    result.each_hash do |h|
      games_id = h[ 'id']
    end

    result = db.query( "SELECT id from Countries C WHERE
       ➥C.name='#{row[1]}'")
    result.each_hash do |h|

      begin
        # Inserts into the Representant aggregate.
        db.query( "INSERT INTO Athletes_represent_Countries
           ➥(athlete_id, country_id) VALUES ('#{athlete_id}',
           ➥'#{h['id']}')")

        # Inserts into the participants table.
        db.query( "INSERT INTO Representant_participates_Event
           ➥(athlete_id, country_id, games_id, ranking) VALUES
           ➥('#{athlete_id}', '#{h['id']}', '#{games_id}', 0)")
      rescue => ex
        puts "DEBUG : (participants) : #{ex.message}" if @debug
      end
    end
  end
  i += 1
end
puts "Done!"

# Import #7 : Participants that won a medal (update of participants)
puts "Import #7 : Participants that won a medal (update of
    ➥participants) ==> Long one!"
i = 0
CSV.foreach( "dataset/medals.csv") do |row|

  if !row[0]. nil? and !row[1]. nil? and !row[2]. nil? and
      ➥!row[3]. nil? and !row[4]. nil?

    # Get the year and split event this way :
    #   Split at "at the <year> <is_summer> - <Discipline>"

    country, year, is_summer, discipline = row[0], row[1][/\d+/],
        ➥row[1].downcase[( "summer")], row[1].split( "-", 2)[1]
    is_summer = is_summer.  nil? ? 0 : 1

    # Get the ranking for this line.
    ranking =  case row[2].downcase[/^[\S]{4,6}/]
      when "gold" then 1
      when "silver" then 2
      when "bronze" then 3
    end

    # Quit if the discipline is not specified
    unless discipline. nil?

      # Fetch discipline_id
      discipline_id =  nil
      result = db.query(  "SELECT id from Disciplines D WHERE
          ➥D.name='#{discipline.gsub("'", "\\\\'").strip}'")
      result.each_hash   do |h|
        discipline_id = h[  'id']
      end

      # Fetch country_id
      country_id =  nil
      result = db.query(  "SELECT id from Countries C WHERE
          ➥C.name='#{country}'")
      result.each_hash   do |h|
        country_id = h[  'id']
      end

      # Updates each athlete
      row[3].split( ";").each  do |athlete|
        begin

          # Fetch the athlete_id
          athlete_id =  nil
          result = db.query(  "SELECT id from Athletes A WHERE
              ➥A.name='#{athlete.gsub("'", "\\\\'")}'")
          result.each_hash   do |h|
            athlete_id = h[  'id']
          end

          # Updates the row concerned with the athlete
          my_query =  "UPDATE Representant_participates_Event RE SET
              ➥RE.discipline_id='#{discipline_id}',
              ➥RE.ranking='#{ranking}' WHERE
              ➥RE.athlete_id='#{athlete_id}' AND
              ➥RE.country_id='#{country_id}'"
```

```ruby
            db.query(my_query)

          rescue => ex
            puts "DEBUG : #{ex.message}" if @debug
          end
        end
      end

    end

    i += 1
end
puts "Done!"

db.close
```

Listing 3.1: Ruby importation script

4 Queries

Here are some explanations of queries that seem difficult to understand:

- The query A is the intersection of athletes who won medals in summer and who won in winter.
- The query C is selecting the minimum year (so the first event) where a country won its first medal. It returns for each country the corresponding Olympics which mean the host city and the year.
- The query D is selecting the union of the best country (most of medals) of all of the winter Olympics and the best one of all of the summer Olympics.
- The query G is taking for each Olympics the maximum of all counts of participants in each country.

```sql
--Names of athletes who won medals at both summer and winter
    ➥Olympics.

SELECT a.name
FROM (
   SELECT p.athlete_id  as medalist_id
   FROM representant_participates_event p, games g
   WHERE p.ranking != 0  AND p.games_id = g.id  AND g.is_summer = 0)
      ➥m1, (
      SELECT p.athlete_id  as medalist_id
      FROM representant_participates_event p, games g
      WHERE p.ranking != 0  AND p.games_id = g.id  AND g.is_summer =
         ➥1) m2, athletes a
WHERE m1.medalist_id = m2.medalist_id    AND a.id = m1.medalist_id;
```

Listing 4.1: Query A

```sql
--Names of gold medalists in sports which appeared only once at the
    ➥Olympics.

SELECT a.name  as athlete, d.name  as sport
FROM athletes a, representant_participates_event p, disciplines d
WHERE a.id = p.athlete_id   AND p.ranking = 1  AND d.id =
    ➥p.discipline_id   AND d.sport  IN (
   SELECT d.sport
   FROM disciplines d, disciplines_event_games e
      WHERE d.id = e.discipline_id
       GROUP BY d.sport
      HAVING COUNT(*) = 1);
```

Listing 4.2: Query B

```sql
--For each country, print the place where it won its first medal.

SELECT c1.name as country, g.host_city, g. year
FROM games g, countries c1, representant_participates_event p
WHERE g.id = p.games_id AND c1.id = p.country_id AND year = (
    SELECT MIN(g.year)
    FROM games g, representant_participates_event p
    WHERE g.id = p.games_id AND p.ranking != 0)
GROUP BY p.country_id;
```

Listing 4.3: Query C

```sql
--Print the name of the country which won the most medals in summer
    ➥Olympics and the country which won the most medals in winter
    ➥Olympics.

SELECT c.name
FROM countries c
WHERE c.id IN (
    SELECT p.country_id
    FROM representant_participates_event p, games g
    WHERE p.games_id = g.id AND g.is_summer = 0 AND p.ranking != 0
    GROUP BY p.country_id
    HAVING COUNT(*) >= ALL (
        SELECT COUNT(*)
        FROM representant_participates_event p1, games g1
        WHERE p1.games_id = g1.id AND g1.is_summer = 0 AND p1.ranking
            ➥!= 0
        GROUP BY p1.country_id)
    UNION
    SELECT p.country_id
    FROM representant_participates_event p, games g
    WHERE p.games_id = g.id AND g.is_summer = 1 AND p.ranking != 0
    GROUP BY p.country_id
    HAVING COUNT(*) >= ALL (
        SELECT COUNT(*)
        FROM representant_participates_event p1, games g1
        WHERE p1.games_id = g1.id AND g1.is_summer = 1 AND p1.ranking
            ➥!= 0
        GROUP BY p1.country_id));
```

Listing 4.4: Query D

```sql
-- List all cities which hosted the Olympics more than once.

SELECT DISTINCT G.host_city
FROM Games G
WHERE EXISTS (
    SELECT *
    FROM Games G2
    WHERE G.id != G2.id AND G.host_city = G2.host_city
);
```

Listing 4.5: Query E

```sql
-- List names of all athletes who competed for more than one
    ➥country.

SELECT A.name
FROM Athletes A
WHERE (
    SELECT COUNT(AC.country_id)
    FROM Athletes_represent_Countries AC
    WHERE A.id = AC.athlete_id
) > 1;
```

Listing 4.6: Query F

```sql
-- For each Olympic Games print the name of the country with the
    ➥most participants.

SELECT G.year, C.name
FROM Games G, Countries C
WHERE C.id = (
    SELECT RE.country_id
    FROM Representant_participates_Event RE
    WHERE RE.games_id = G.id
    GROUP BY RE.country_id
    ORDER BY COUNT(RE.country_id)   DESC LIMIT 1)
GROUP BY G.id;
```

Listing 4.7: Query G

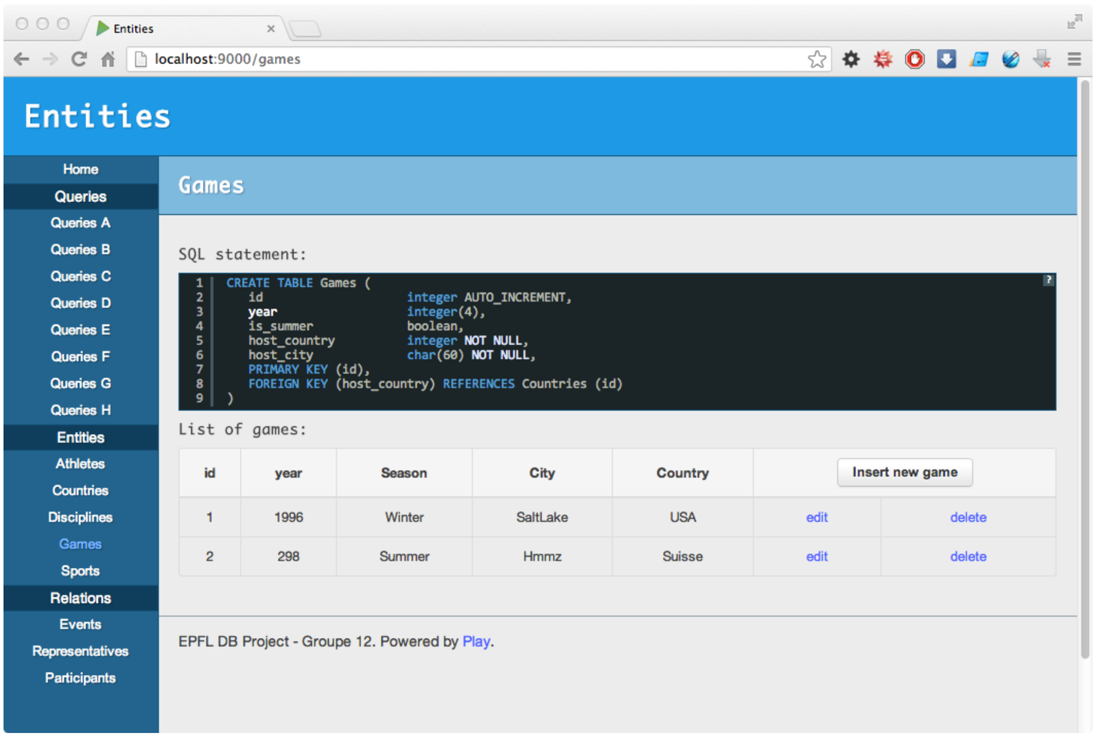
```sql
-- List all countries which didnt ever win a medal.

SELECT C.name
FROM Countries C
WHERE (
    SELECT SUM(RE.ranking)
    FROM Representant_participates_Event RE
    WHERE RE.country_id = C.id
) IS NULL OR (
    SELECT SUM(RE.ranking)
    FROM Representant_participates_Event RE
    WHERE RE.country_id = C.id
) = 0;
```

Listing 4.8: Query H

5 Web

5.1 Entities & relations

Here is a view that show the listing of an entity or a relation, we can see that the SQL statement is above the table. We can directly edit or remove an entry when clicking on the edit or delete link.



SQL statement:

```
CREATE TABLE Games (
   id                     integer AUTO_INCREMENT,
   year                   integer(4),
   is_summer              boolean,
   host_country           integer NOT NULL,
   host_city              char(60) NOT NULL,
   PRIMARY KEY (id),
   FOREIGN KEY (host_country) REFERENCES Countries (id)
)
```

List of games:

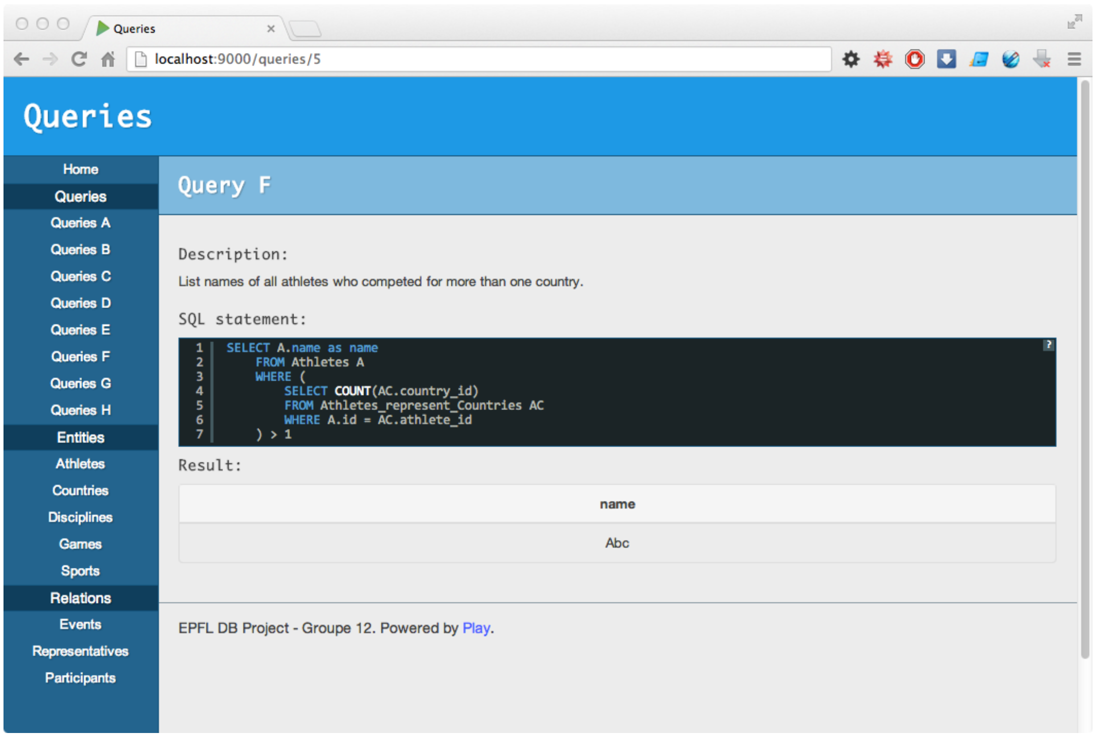
id	year	Season	City	Country	Insert new game	
1	1996	Winter	SaltLake	USA	edit	delete
2	298	Summer	Hmmz	Suisse	edit	delete

EPFL DB Project - Groupe 12. Powered by Play.

Figure 5.1: Entities listing

5.2 Query view

The result of each query is shown inside a table, a description and the SQL statement are above the results.



Query F

Description:

List names of all athletes who competed for more than one country.

SQL statement:

```sql
SELECT A.name as name
    FROM Athletes A
    WHERE (
        SELECT COUNT(AC.country_id)
        FROM Athletes_represent_Countries AC
        WHERE A.id = AC.athlete_id
    ) > 1
```

Result:

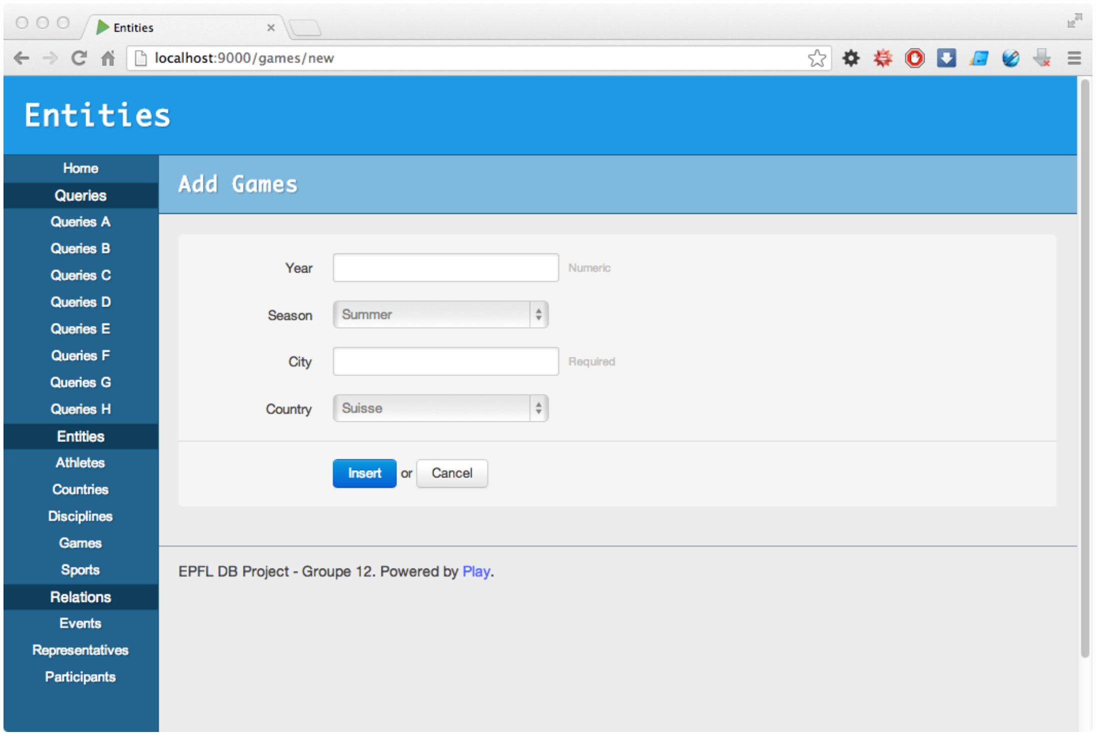
name
Abc

EPFL DB Project - Groupe 12. Powered by Play.

Figure 5.2: Query view

5.3 Add entity / relation

To change data inside the databases, we can add, edit and remove entities and relations using the WEB UI.



Add Games

Year		Numeric
Season	Summer	
City		Required
Country	Suisse	

EPFL DB Project - Groupe 12. Powered by Play.

Figure 5.3: Add entity form