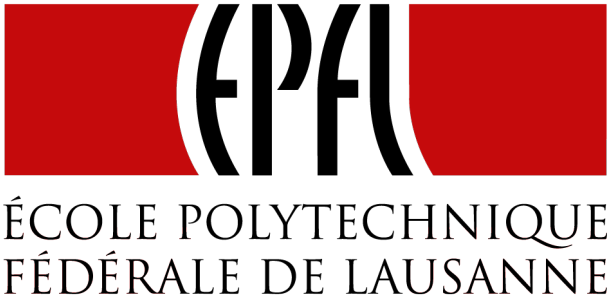
Introduction to Database Systems

Olympic Games



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

Academic years 2012-2013
(June 2, 2013)

Contents

1 Entityrelationship model

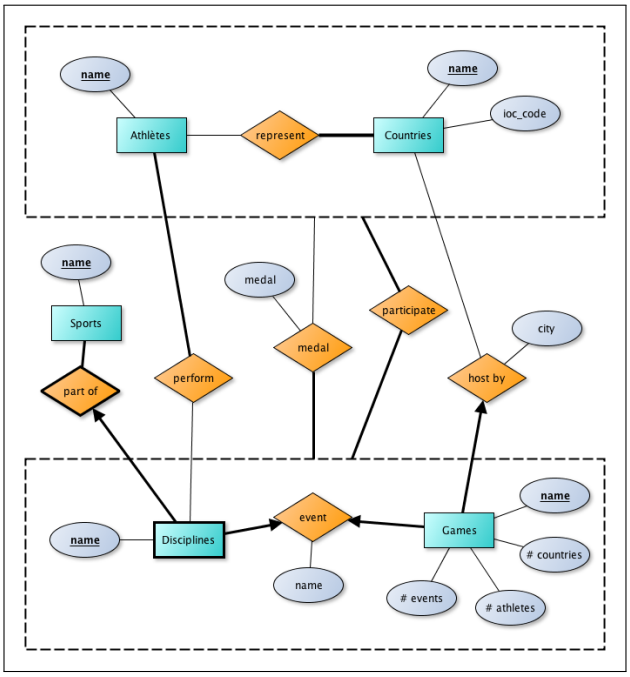


Figure 1.1: Previous ER Model.

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.

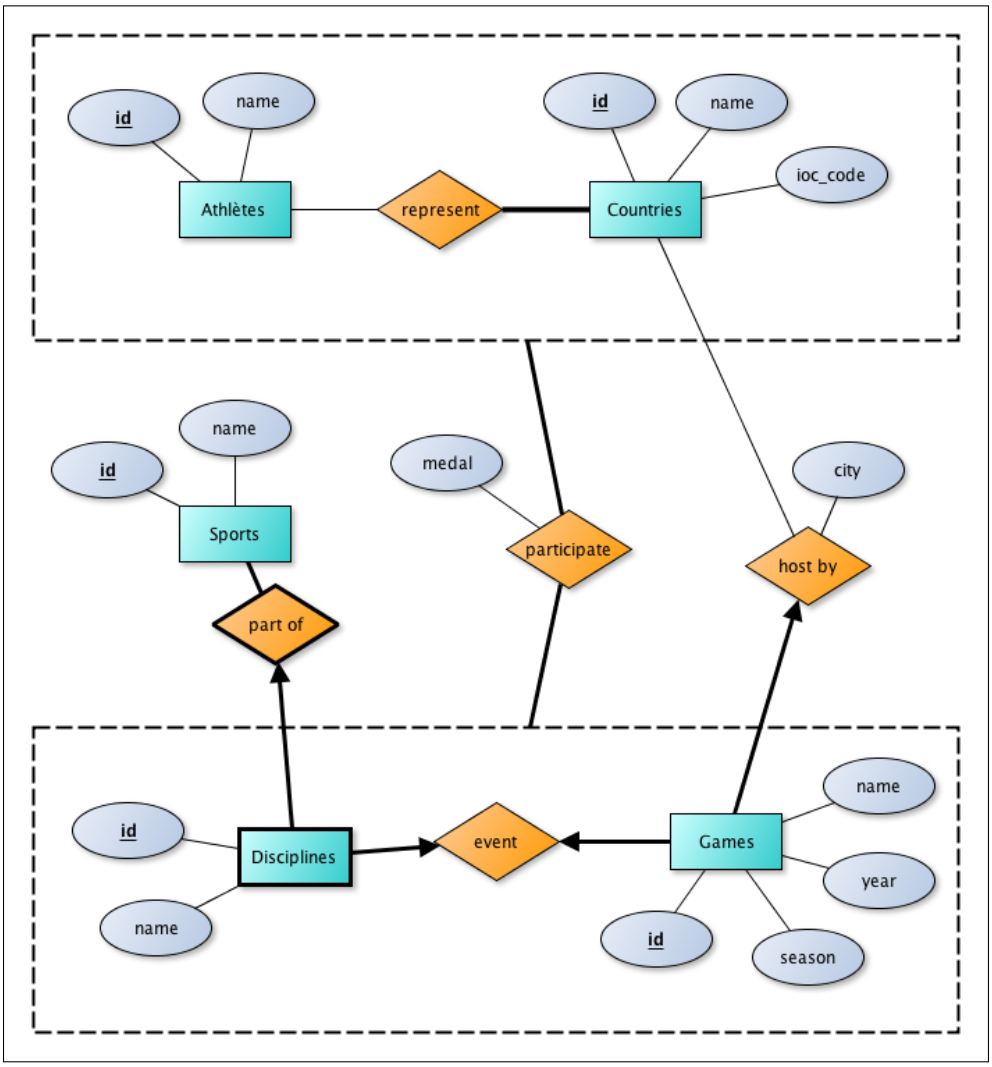


Figure 1.2: New ER Model.

2 Relational schema and constraints

2.1 Relational schema

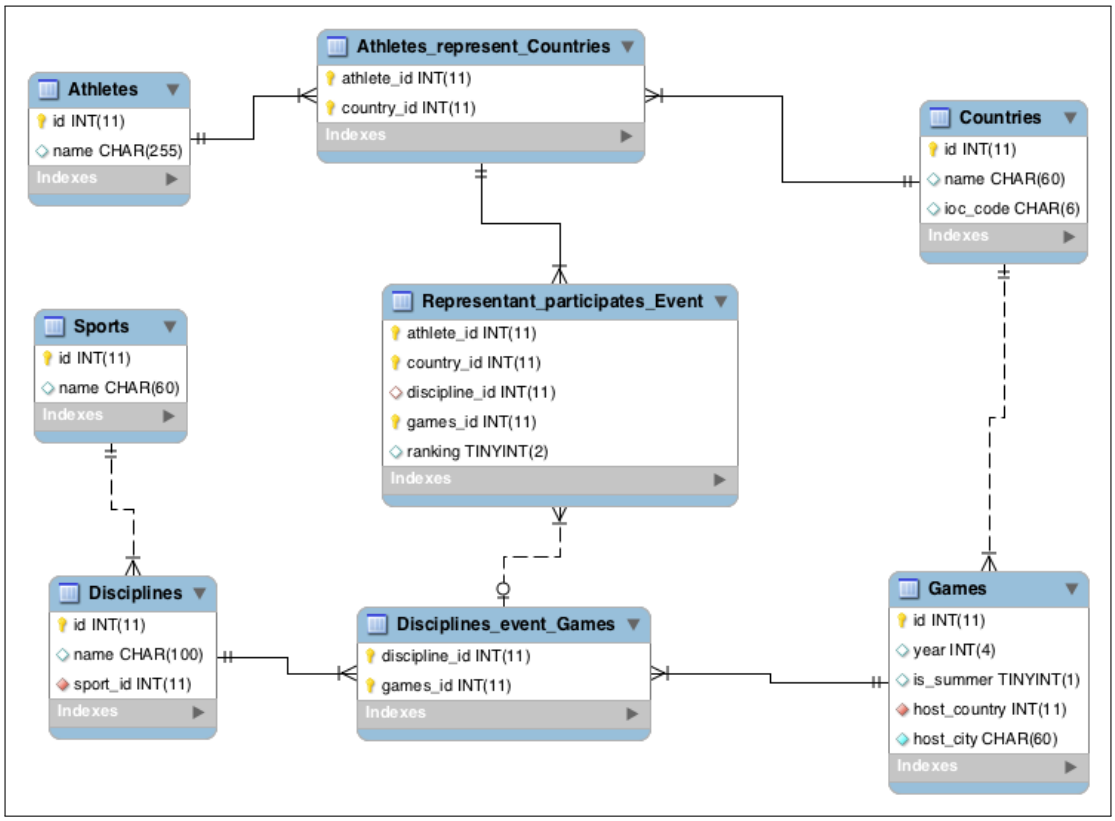


Figure 2.1: Generated EER Model from MySQL Workbench.

After implementing the DDL from Section 2.2, we generated the scheme in Figure 5.4 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.

We changed the DDL adding the unique constraints because we had to much duplicate during the import.

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

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

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

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),
    UNIQUE (year),
    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),
    UNIQUE (name),
    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 :
```

```
        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_new   ')

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

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

# Import #3 : Sports
puts "Import #3 : Sports"
i = 0
```

```
CSV.foreach( "dataset/sports.csv") do |row|
  begin
    db.query( "INSERT INTO Sports (name) VALUES ('#{row[0]}')") if i > 0
  rescue => ex
    puts "DEBUG (Sports) : #{ex.message}" if @debug
  end
  i += 1
end
puts "Done!"

# Import #4 : Disciplines
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 do |h|
      begin
        db.query( "INSERT INTO Disciplines (name, sport_id) VALUES ('#{row[0].gsub("'", "\\\\'")}', '#{h['id']}')")
      rescue => ex
        puts "DEBUG (Disciplines) : #{ex.message}" if @debug
      end
    end
  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|
        begin
          db.query( "INSERT INTO Games (year, is_summer, host_country, host_city) VALUES ('#{year}', '#{is_summer}', '#{h['id']}', '#{host_city.gsub("'", "\\\\'")}')")
        rescue => ex
          puts "DEBUG (Games) : #{ex.message}" if @debug
        end
      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}'"
          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

```
-- 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

```
-- 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

```
-- 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

```
-- Compute medal table for the specific Olympic Games supplied by
    ➥the user. Medal table should contain countrys IOC code
    ➥followed by the number of gold, silver, bronze and total
    ➥medals. It should first be sorted by the number of gold, then
    ➥silvers and finally bronzes.

```

```sql
SET @selected_game_id =    '47';

-- Here we use a case to count the number of different medals
SELECT C.ioc_code,  COUNT(case P.ranking  when 1 then 1 else null end) as nb_gold,  COUNT(case P.ranking  when 2 then 1 else null end) as nb_silver,  COUNT(case P.ranking  when 3 then 1 else null end) as nb_bronze,  COUNT(case when P.ranking > 0  then 1 else null end) as total_medals
FROM Representant_participates_Event P
-- We do an inner join in order to have the IOC_code for the countries.
INNER JOIN Countries C  ON P.country_id = C.id
WHERE P.games_id = @selected_game_id
GROUP BY C.ioc_code
ORDER BY nb_gold, nb_silver, nb_bronze
```

Listing 4.9: Query I

```sql
-- For each sport, list the 3 nations which have won the most medals.

-- Sadly this query is not finished. We could not see how to do it properly...
-- Here are the guideline that we tried to follow :
-- 1 - For each discipline, we see how many medals are there
-- 2 - Then we add them to get the number of medals per sports
-- 3 - Afterwards we select only the TOP3 for each of them
--
-- SELECT S.id, COUNT(*) as first,COUNT(*) as second,COUNT(*) as third
-- FROM Sports S
--
-- SELECT S.id as sport_id, SUM(P.nb_medals) as nb_medals_per_sport
-- FROM Sports S, (
--   SELECT P1.discipline_id, COUNT(P1.ranking) as nb_medals
--   FROM Representant_participates_Event P1
--   INNER JOIN Disciplines D ON P1.discipline_id = D.id
--   GROUP BY P1.discipline_id
-- ) P, Disciplines D
-- WHERE P.discipline_id = D.id AND S.id = D.sport_id
-- GROUP BY S.id
```

Listing 4.10: Query J

```sql
-- Compute which country in which Olympics has benefited the most from playing in front of the home crowd.

-- We did not find a way to do this one.
```

Listing 4.11: Query K

```sql
-- List top 10 nations according to their success in team sports.

-- Compute for each country the number of medals and the number of medalists. The list of countries is then sorted by the quotient (number of medals over number of medalists)
```

```sql

SELECT medalists.country_name
FROM (
  -- Gets the number of medalists
    SELECT COUNT(*) as number_of_medalists, c.name    as country_name, c.id as country_id
    FROM Representant_participates_Event p, Countries c, Disciplines d
    WHERE p.country_id = c.id   AND p.discipline_id = d.id   AND p.ranking != 0
    GROUP BY c.id
  ) medalists,
    (
  -- Gets the number of medals
    SELECT COUNT(DISTINCT d.id) as number_of_medals, c.name    as country_name, c.id   as country_id
    FROM Representant_participates_Event p, Countries c, Disciplines d
    WHERE p.country_id = c.id   AND p.discipline_id = d.id   AND p.ranking != 0
    GROUP BY c.id
  ) medals
WHERE medalists.country_id = medals.country_id
ORDER BY (medalists.number_of_medalists/medals.number_of_medals)
LIMIT 0, 10
```

Listing 4.12: Query L

```sql
-- List all Olympians who won medals for multiple nations.

-- Look if there is a participant who has a medal for two different countries.

SELECT DISTINCT a.name, c1.country_id, c1.country_name, c2.country_id, c2.country_name
FROM athletes a,
  (
    SELECT p.athlete_id  as medalist_id, c.id   as country_id, c.name as country_name
    FROM representant_participates_event p, countries c
    WHERE p.ranking != 0  AND p.country_id = c.id
  ) c1,
  (
    SELECT p.athlete_id  as medalist_id, c.id   as country_id, c.name as country_name
    FROM representant_participates_event p, countries c
    WHERE p.ranking != 0  AND p.country_id = c.id
  ) c2
WHERE c1.medalist_id = c2.medalist_id    AND a.id = c1.medalist_id   AND c1.country_id < c2.country_id
```

Listing 4.13: Query M

```sql
-- List all nations whose first medal was gold, all nations whose first medal was silver and all nations whose first medal was bronze
```

```sql

-- For each country, the query is searching for the first medal.

SELECT c.id as country_id, c.name as country_name, g. year, p.ranking
FROM representant_participates_event p
INNER JOIN Countries c ON c.id = p.country_id
INNER JOIN Games g ON g.id = p.games_id
WHERE g.year = (
    SELECT MIN(g1.year)
    FROM Games g1
    INNER JOIN representant_participates_event p1 ON p1.games_id = g1.id
    WHERE p1.country_id = g.id AND p1.ranking != 0
  )
GROUP BY c.id
ORDER BY p.ranking
```

Listing 4.14: Query N

```sql
-- For all disciplines, compute the country which waited the most between two successive medals.

-- The View is the time that a country has waited between 2 medals for each discipline. The following query is only giving the country according to the maximum time and the discipline.

CREATE VIEW DelayByCountryByDiscipline AS (
  SELECT p1.discipline_id as discipline_id, g1. year-g2.year as time_waited, p1.country_id as country_id
  FROM representant_participates_event p1, representant_participates_event p2, games g1, games g2
  WHERE p1.country_id = p2.country_id AND p1.games_id = g1.id AND p2.games_id = g2.id AND g1.year > g2.year
  AND p1.ranking != 0 AND p2.ranking != 0 AND p1.discipline_id = p2.discipline_id
  GROUP BY p1.discipline_id
);

SELECT d.name as discipline, c.name as country, join2.max_delay as number_of_years_waited
FROM DelayByCountryByDiscipline join1, Disciplines d, Countries c, (
  SELECT MAX(time_waited) as max_delay, discipline_id
  FROM DelayByCountryByDiscipline
  GROUP BY discipline_id
) join2
WHERE join1.discipline_id = join2.discipline_id AND join1.time_waited = join2.max_delay AND join1.discipline_id = d.id
AND join1.country_id = c.id
```

Listing 4.15: Query O

```sql
-- List all events for which all medals are won by athletes from the same country.

SELECT d.id as discipline_id, d.name as discipline_name, c.name as country_name
```

```sql
FROM representant_participates_event p
INNER JOIN disciplines d   ON p.discipline_id = d.id
INNER JOIN countries c     ON p.country_id    = c.id
GROUP BY d.id
HAVING COUNT(DISTINCT p.ranking != 0) = (
  -- We get all the "medal entries" for a given discipline
  SELECT COUNT(*)
  FROM representant_participates_event p1
  WHERE p1.discipline_id = d.id
)
```

Listing 4.16: Query P

```sql
-- For each Olympic Games, list the name of the country which scored the largest percentage of the medals.

-- We compute views to select the number of medals per country and per games and to select the number of medals per games. The third one is to give the percentage of medals by country and by games. The query is only using the last one in order to get the name of the country which corresponds to the maximum for each games.

-- number of medals per country per games
CREATE VIEW NbMedalsByCountryByGames
AS (SELECT g.id as games_id, c.id as country_id, COUNT(DISTINCT d.id) as number_of_medals_per_country
    FROM Games g, representant_participates_event p, disciplines d, Countries c
    WHERE p.games_id = g.id AND p.discipline_id = d.id AND p.country_id = c.id AND p.ranking != 0
    GROUP BY g.id, c.id);

-- number of total medals in games
CREATE VIEW NbMedalsByGames
AS (SELECT g.id as games_id, COUNT(DISTINCT d.id) as number_of_medals_in_games
    FROM Games g, representant_participates_event p, disciplines d
    WHERE p.games_id = g.id AND p.discipline_id = d.id AND p.ranking != 0
    GROUP BY g.id);

-- Percentage by country by games
CREATE VIEW Percentage
AS (SELECT nmg.games_id as games_id, c.name as country, (100*nmpc.number_of_medals_per_country/nmg.number_of_medals_in_games) as percentage_of_medals
    FROM NbMedalsByCountryByGames nmpc, NbMedalsByGames nmg, Countries c
    WHERE nmg.games_id = nmpc.games_id AND nmpc.country_id = c.id
    );

SELECT join1.games_id, join1.country as country, join1.percentage_of_medals as percentage_of_medals
FROM MaxPercentageByGames join1, (
    SELECT games_id, MAX(percentage_of_medals) as max_percentage_of_medals
```

```
    FROM MaxPercentageByGames
    GROUP BY games_id
) join2
WHERE join1.games_id = join2.games_id     AND
    join1.percentage_of_medals = join2.max_percentage_of_medals
```

Listing 4.17: Query Q

```
-- For all individual sports, compute the most top 10 countries according to their success score. Success
-- score of a country is sum of success points of all its medalists: gold medal is worth 3 points, silver 2
-- points, and bronze 1 point. Shared medal is worth half the points of the non-shared medal.

-- We again used a case structure to simulate the score function. Note that, in our DB, 1 is gold, 2 is silver and 3 bronze.
SELECT P.discipline_id,   COUNT(case P.ranking  when 1 then 3 when 2 then 2 when 3 then 1 else null end) AS score
FROM Representant_participates_Event P
WHERE P.discipline_id IS   NOT NULL AND P.athlete_id   IN (
  -- Gets all the athletes that are doing an individual sport
  SELECT DISTINCT(a.id)  as athlete_id
  FROM representant_participates_event p, athletes a, disciplines d, games g
  WHERE p.athlete_id = a.id   AND p.discipline_id = d.id   AND p.games_id = g.id   AND p.ranking != 0
  GROUP BY d.id, g.id, p.country_id, p.ranking
  HAVING COUNT(*) = 1
)
GROUP BY P.discipline_id
ORDER BY score  DESC
```

Listing 4.18: Query R

```
-- List names of all athletes who won medals both in individual and team sports.

-- We used the same idea fir this query as request T.

SELECT a.id  as athlete_id, a.name   as athlete_name
FROM (
      SELECT DISTINCT(a.id)  as athlete_id
      FROM representant_participates_event p, athletes a, disciplines d, games g
      WHERE p.athlete_id = a.id   AND p.discipline_id = d.id   AND p.games_id = g.id   AND p.ranking != 0
      GROUP BY d.id, g.id, p.country_id, p.ranking
      HAVING COUNT(*) = 1
     ) indidual_medalist,
    (
      SELECT DISTINCT(a.id)  as athlete_id
      FROM representant_participates_event p, athletes a, disciplines d, games g
      WHERE p.athlete_id = a.id   AND p.discipline_id = d.id   AND p.games_id = g.id   AND p.ranking != 0
```

```
        GROUP BY d.id, g.id, p.country_id, p.ranking
        HAVING COUNT(*) > 1
    ) team_medalist,
athletes a
WHERE indidual_medalist.athlete_id = team_medalist.athlete_id        AND
    ➥indidual_medalist.athlete_id = a.id
```

Listing 4.19: Query S

```
-- List names of all athletes who won gold in team sports, but only
    ➥won silvers or bronzes individually.

-- The query is searching for individual medalist and team medalist
    ➥(were there are several medals of the same value for the same
    ➥event).

SELECT a.id as athlete_id, a.name   as athlete_name
FROM (
    SELECT DISTINCT(a.id) as athlete_id
    FROM representant_participates_event p, athletes a, disciplines
        ➥d, games g
    WHERE p.athlete_id = a.id   AND p.discipline_id = d.id   AND
        ➥p.games_id = g.id   AND (p.ranking = 2   OR p.ranking = 3)
    GROUP BY d.id, g.id, p.country_id, p.ranking
    HAVING COUNT(*) = 1) indidual_medalist,
    (SELECT DISTINCT(a.id) as athlete_id
    FROM representant_participates_event p, athletes a, disciplines
        ➥d, games g
    WHERE p.athlete_id = a.id   AND p.discipline_id = d.id   AND
        ➥p.games_id = g.id   AND p.ranking != 1
    GROUP BY d.id, g.id, p.country_id, p.ranking
    HAVING COUNT(*) > 1) team_medalist,
athletes a
WHERE indidual_medalist.athlete_id = team_medalist.athlete_id        AND
    ➥indidual_medalist.athlete_id = a.id
```

Listing 4.20: Query T

```
-- List names of all events and Olympic Games for which the
    ➥individual or team has defended a title from the previous
    ➥games.
-- Forum : check only if country defened title (not athlete)

-- Checks if a country has defended it's title from the previous
    ➥game.

SELECT d1.name as discipline, c1.name   as winner_country,
    ➥g2.host_city   as first_games, g2.  year, g1.host_city   as
    ➥second_games, g1.  year
FROM representant_participates_event p1, disciplines d1, countries
    ➥c1, games g1,
representant_participates_event p2, disciplines d2, countries c2,
    ➥games g2
WHERE p1.discipline_id = d1.id   AND p1.country_id = c1.id   AND
    ➥p1.games_id = g1.id   AND p1.ranking = 1   AND
```

```
p2.discipline_id = d2.id    AND p2.country_id = c2.id    AND p2.games_id
    ➥= g2.id  AND p2.ranking = 1   AND d1.id = d2.id   AND g1.id !=
    ➥g2.id  AND NOT EXISTS (
  SELECT *
  FROM games g
  WHERE g.year < g1.year AND g.year > g2.year
)
GROUP BY d1.id
```

Listing 4.21: Query U

```
-- List top 10 countries according to their success on the events
    ➥which appear at the Olympics for the first
-- time. Present the list in the form of the medal table (as
    ➥described for query I).

SELECT e1.discipline_id, e1.games_id
FROM disciplines_event_games e1, games g1, (
    SELECT e2.discipline_id   as discipline_id,   MIN(g2.year) as
        ➥min_year
    FROM disciplines_event_games e2, games g2
    WHERE e2.games_id = g2.id
    GROUP BY e2.discipline_id
  ) min_by_dis
WHERE e1.games_id = g1.id   AND min_by_dis.discipline_id =
    ➥e1.discipline_id   AND g1.year = min_by_dis.min_year
```

Listing 4.22: Query V

4.1 Indexes

While trying to add an index, we discovered that **MySQL** already generated indexes for all entities and relations. It was then useless for us to insert a new index, here is the code we would have used.

```
CREATE INDEX myIndex
ON Representant_participates_Event (country_id, games_id)
```

Listing 4.23: Index creation

4.2 Performances

When launching the queries, we can see the time spent by the system to execute the query.

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.

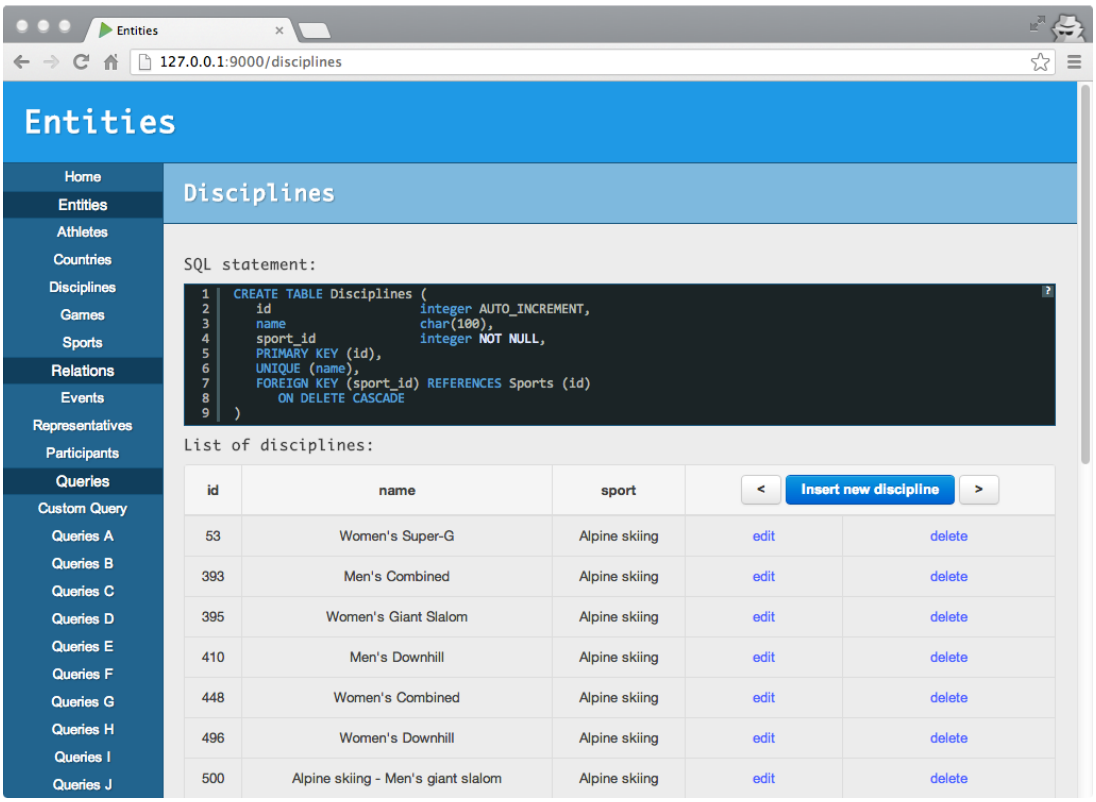


Figure 5.1: Entities listing

5.2 Query view

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

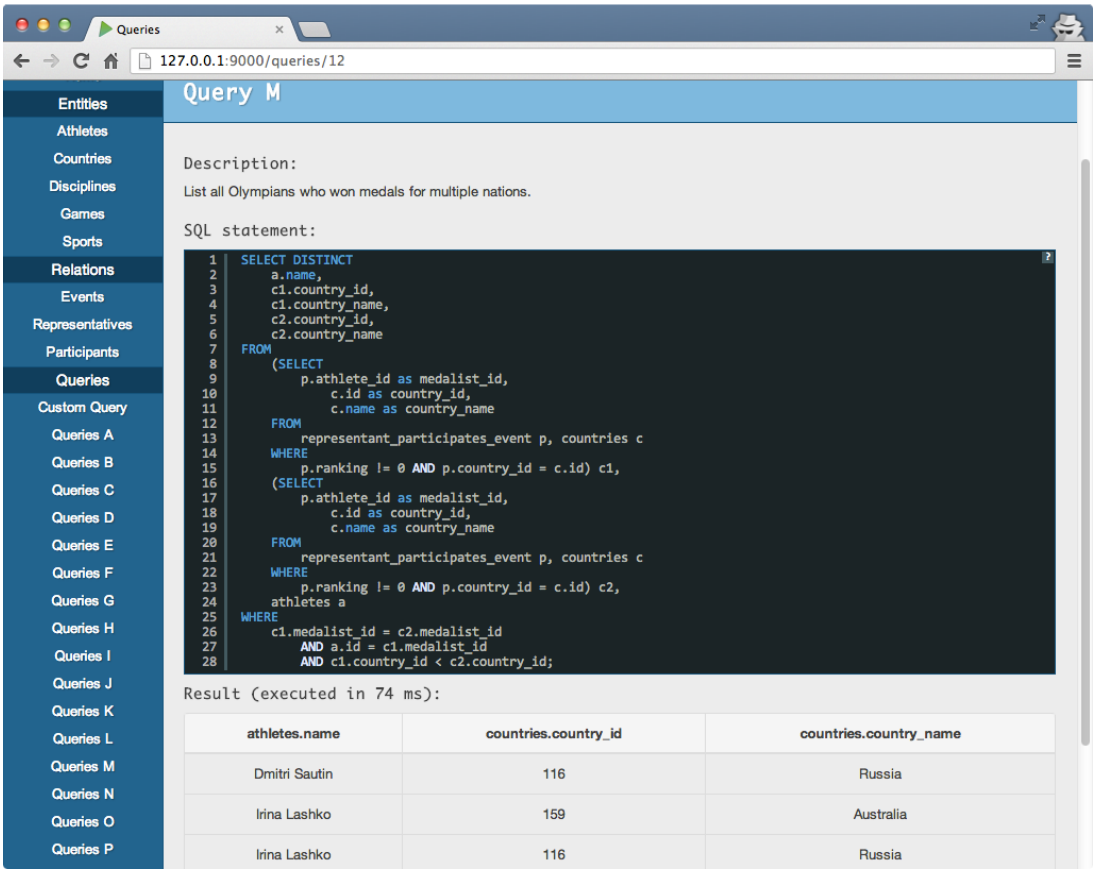


Figure 5.2: Query view

5.3 Add entity / relation

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

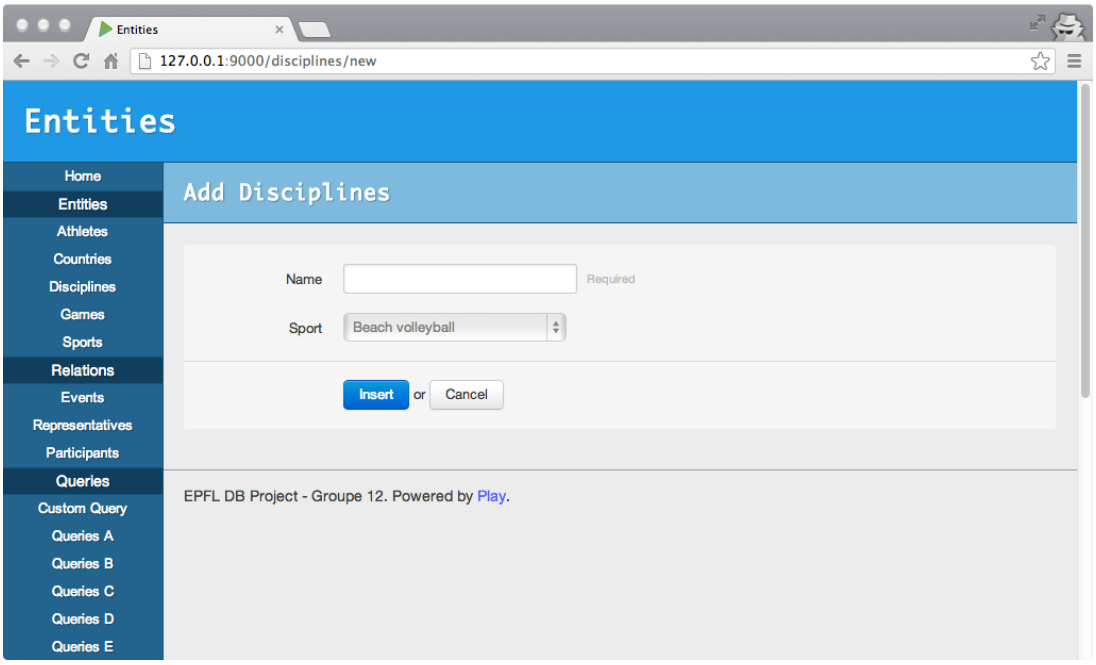


Figure 5.3: Add entity form

5.4 Custom Query

To perform custom query, we implement a small interface to insert SQL Code and then query the DB and print the result. If an error occurs a message is displayed.

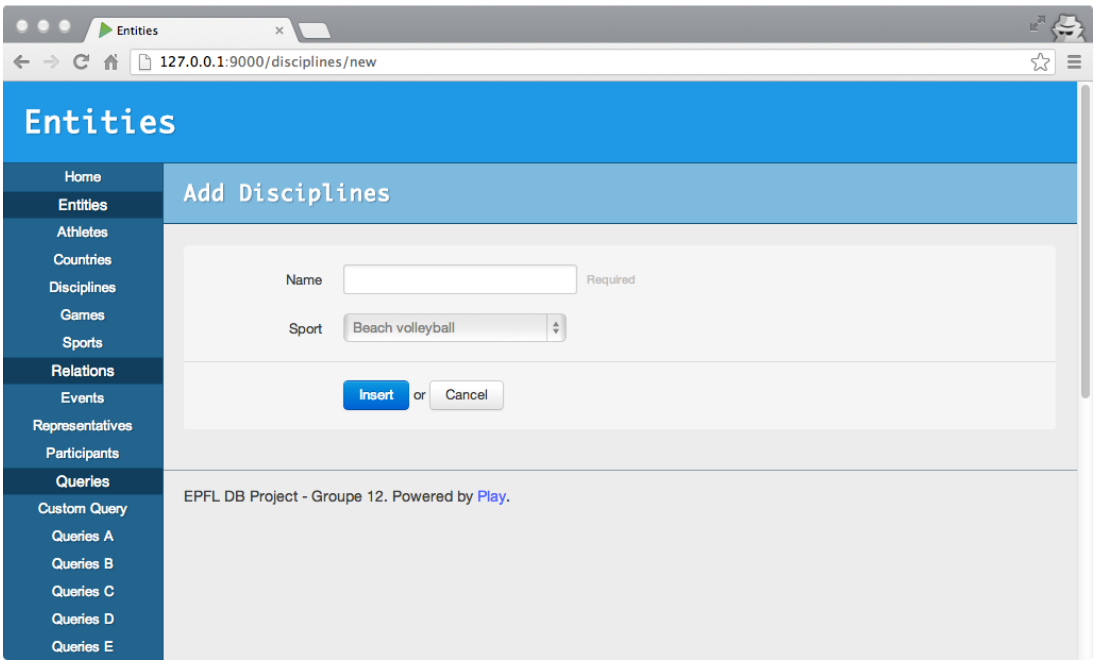


Figure 5.4: Add entity form