

Introduction to Database Systems

Olympic Games



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

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

Academic years 2012-2013
(June 2, 2013)



Contents

1	Entityrelationship model	1
2	Relational schema and constraints	3
2.1	Relational schema	3
2.2	SQL Data definition language statements	4
3	Data importation	6
4	Queries	12
4.1	Indexes	21
4.2	Performances	21
5	Web	22
5.1	Entities & relations	22
5.2	Query view	23
5.3	Add entity / relation	24
5.4	Custom Query	24

Entityrelationship model

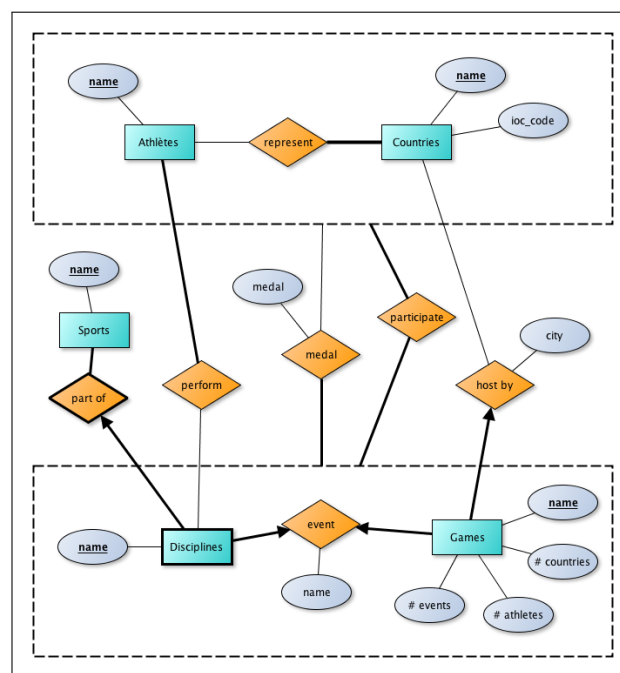


Figure 1.1: Previous ER Model.

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.

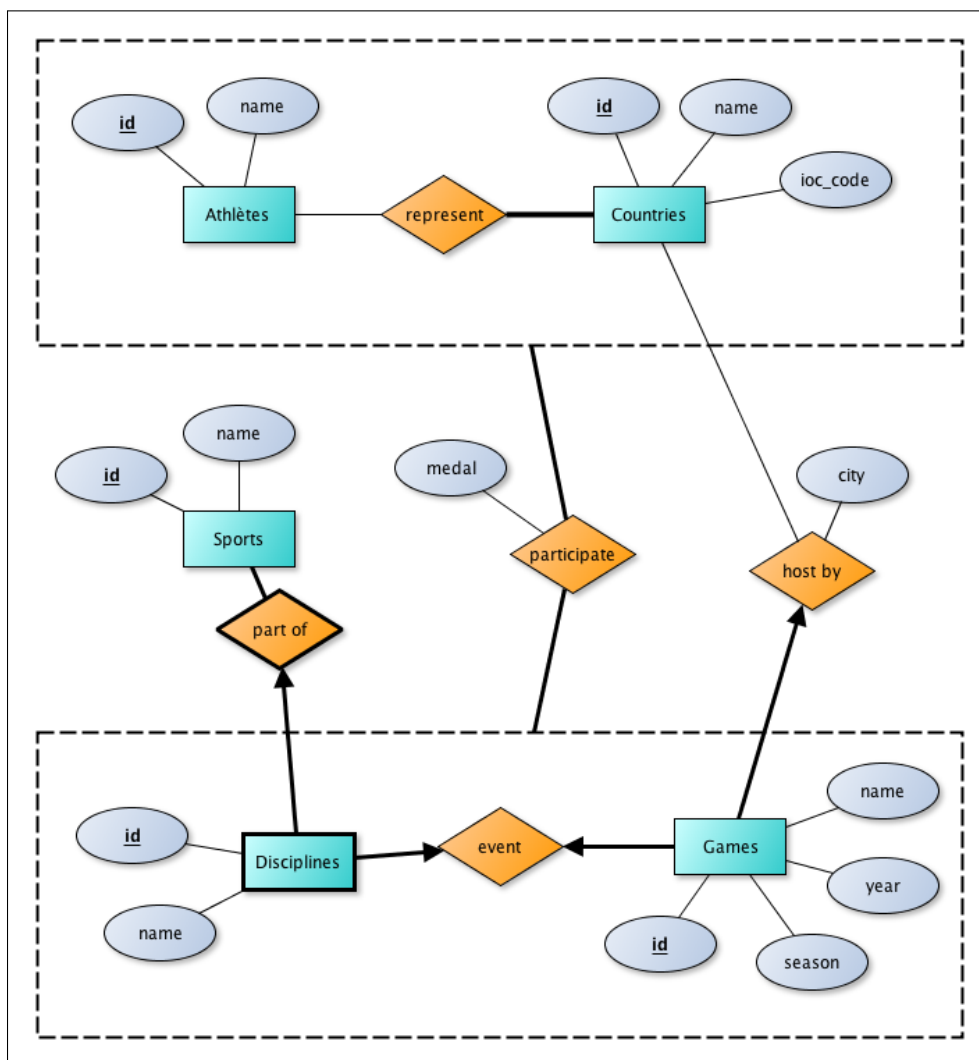


Figure 1.2: New ER Model.

Relational schema and constraints

2.1 Relational schema

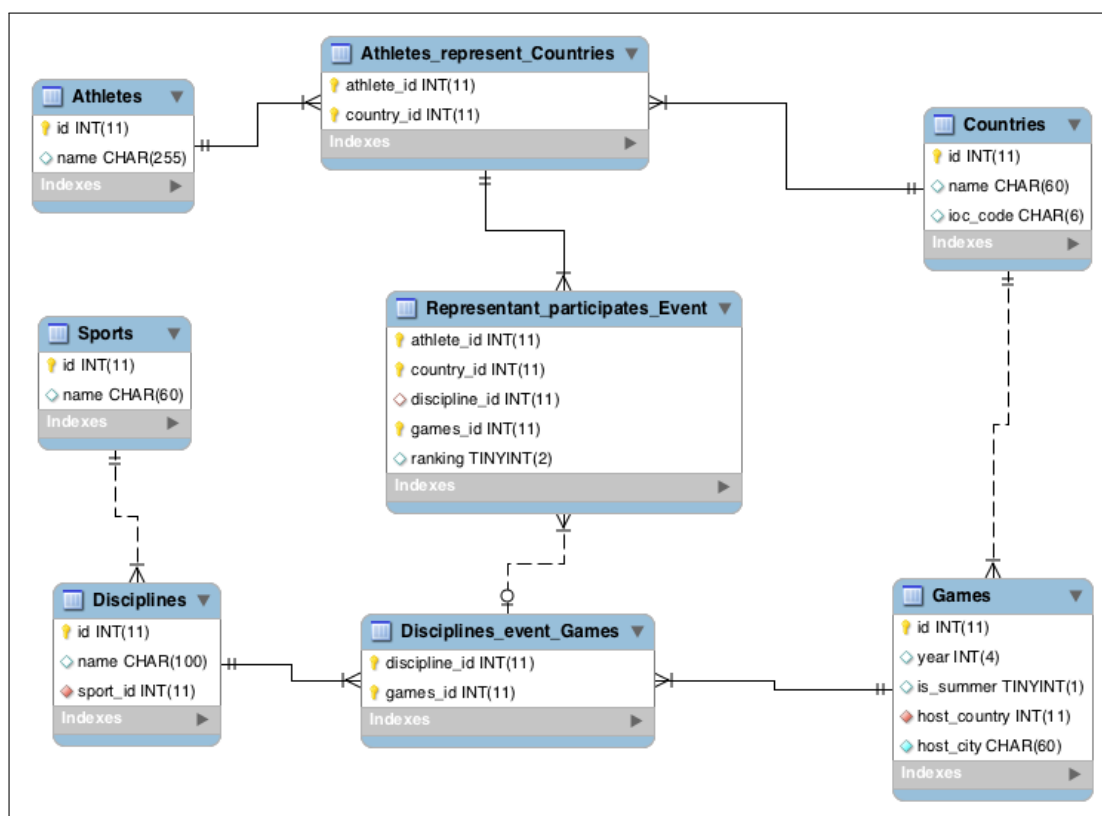


Figure 2.1: Generated EER Model from MySQL Workbench.

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.

```

1 CREATE TABLE Athletes (
2     id                integer AUTO_INCREMENT,
3     name              char(255),
4     PRIMARY KEY (id),
5     UNIQUE (name)
6 );
7
8 CREATE TABLE Countries (
9     id                integer AUTO_INCREMENT,
10    name              char(60),
11    ioc_code          char(6),
12    PRIMARY KEY (id),
13    UNIQUE (name)
14 );
15
16 CREATE TABLE Sports (
17     id                integer AUTO_INCREMENT,
18     name              char(60),
19     PRIMARY KEY (id),
20     UNIQUE (name)
21 );
22
23 CREATE TABLE Games (
24     id                integer AUTO_INCREMENT,
25     year              integer(4),
26     is_summer         boolean,
27     host_country      integer NOT NULL,
28     host_city         char(60) NOT NULL,
29     PRIMARY KEY (id),
30     UNIQUE (year),
31     FOREIGN KEY (host_country) REFERENCES Countries (id)
32 );
33
34 CREATE TABLE Disciplines (
35     id                integer AUTO_INCREMENT,
36     name              char(100),
37     sport_id          integer NOT NULL,
38     PRIMARY KEY (id),
39     UNIQUE (name),
40     FOREIGN KEY (sport_id) REFERENCES Sports (id)
41         ON DELETE CASCADE
42 );

```

Listing 2.1: DDL Entities

```

1 CREATE TABLE Athletes_represent_Countries (
2     athlete_id          integer,
3     country_id          integer,
4     PRIMARY KEY (athlete_id, country_id),
5     FOREIGN KEY (athlete_id) REFERENCES Athletes (id),
6     FOREIGN KEY (country_id) REFERENCES Countries (id)
7 );
8
9 CREATE TABLE Disciplines_event_Games (
10    discipline_id        integer,
11    games_id             integer,
12    PRIMARY KEY (discipline_id, games_id),
13    FOREIGN KEY (discipline_id) REFERENCES Disciplines (id),
14    FOREIGN KEY (games_id) REFERENCES Games (id)
15 );
16
17 -- Here Event is a shortcut to table Disciplines_event_Games
18
19 CREATE TABLE Representant_participates_Event (
20     athlete_id          integer,
21     country_id          integer,
22     discipline_id        integer,
23     games_id            integer,
24     ranking             tinyint(2),
25     PRIMARY KEY (athlete_id, country_id, games_id),
26     FOREIGN KEY (athlete_id, country_id) REFERENCES
27         ↳Athletes_represent_Countries (athlete_id, country_id),
28     FOREIGN KEY (discipline_id, games_id) REFERENCES
29         ↳Disciplines_event_Games (discipline_id, games_id)
30 );

```

Listing 2.2: DDL Relations

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.

```

1  #!/usr/bin/env ruby
2
3  =begin
4
5      Group 12
6      203267 Bastien Antoine
7      183785 Denoreaz Thomas
8      185078 Dieulivol David
9
10     This script will import data from CSV files given      in folder
11         ↳ "dataset" to our database.
12
13     ##### Known problems #####
14
15     1 - Some participants have a weird name. This is due to the
16         ↳ encoding Done! above. Hopefully it is seldom the      case.
17
18     2 - In the DDL file, we have to delete discipline_id as a primary
19         ↳ key because during Import      #6 it caused a problem while we
20         ↳ added the participants (they all have a      NULL
21         ↳ discipline_id).
22
23     3 - In the      end, all the athletes that will      not have any medals
24         ↳ will      not have a discipline_id, since this information is      not
25         ↳ given      in the dataset. On all the athletes, only 682 of them
26         ↳ will have a medal (that      's the data I have with the current
27         ↳ dataset at least) :
28         "SELECT * FROM Representant_participates_Event RE WHERE
29         ↳ discipline_id IS NOT NULL"
30
31     ##### End of Known problems #####
32
33     ##### Notes #####
34
35     2 - If you have a problem with the original CSV files, do this
36         ↳ command to convert them :
```



```

26     iconv -t UTF8 -f LATIN1 < athletes_old.csv > athletes.csv
27
28     3 - The default params for the database connection are as follows
29     ↪:
30     # db = Mysql.new('host ', 'username ', 'password ',
31     ↪ 'your_table ')
32
33     ##### End of Notes #####
34
35 =end
36
37 def usage
38     puts "Usage: ./import_script.rb start [debug]"
39     exit
40 end
41
42 case ARGV.size
43     when 1 then usage if ARGV[0].downcase != "start"
44     when 2 then @debug = ARGV[1].downcase == "debug" ? true : usage
45     else usage
46 end
47
48 require 'mysql'
49 require 'CSV'
50
51 db = Mysql.new('localhost ', 'root ', '', 'db_project_group_12_new ')
52
53 # Import #1 : Athletes
54 puts "Import #1 : Athletes"
55 i = 0
56 CSV.foreach("dataset/athletes.csv") do |row|
57     begin
58         db.query("INSERT INTO Athletes (name) VALUES
59             ↪ ('#{row.first.gsub("'", "''")}'") if i > 0
60     rescue => ex
61         puts "DEBUG (Athletes) : #{ex.message}" if @debug
62     end
63     i += 1
64 end
65 puts "Done!"
66
67 # Import #2 : Countries
68 puts "Import #2 : Countries"
69 i = 0
70 CSV.foreach( "dataset/countries.csv") do |row|
71     begin
72         db.query( "INSERT INTO Countries (name, ioc_code) VALUES
73             ↪ ('#{row[0]}', '#{row[1]}')" if i > 0
74     rescue => ex
75         puts "DEBUG (Countries) : #{ex.message}" if @debug
76     end
77     i += 1
78 end
79 puts "Done!"
80
81 # Import #3 : Sports
82 puts "Import #3 : Sports"
83 i = 0

```

```

80 CSV.foreach( "dataset/sports.csv") do |row|
81   begin
82     db.query( "INSERT INTO Sports (name) VALUES ('#{row[0]}')" ) if i
      ➡ > 0
83   rescue => ex
84     puts "DEBUG (Sports) : #{ex.message}" if @debug
85   end
86   i += 1
87 end
88 puts "Done!"
89
90 # Import #4 : Disciplines
91 puts "Import #4 : Disciplines"
92 i = 0
93 CSV.foreach( "dataset/disciplines.csv") do |row|
94   if i > 0
95     result = db.query( "SELECT id from Sports S WHERE
      ➡ S.name='#{row[1]}'" )
96     result.each_hash do |h|
97       begin
98         db.query( "INSERT INTO Disciplines (name, sport_id) VALUES
          ➡ ('#{row[0].gsub("'", "''")}', '#{h['id']}')" )
99       rescue => ex
100         puts "DEBUG (Disciplines) : #{ex.message}" if @debug
101       end
102     end
103   end
104   i += 1
105 end
106 puts "Done!"
107
108 # Import #5 : Games
109 puts "Import #5 : Games"
110 i = 0
111 CSV.foreach( "dataset/games.csv") do |row|
112
113   if i > 0
114     year, winter_or_summer = row[0].split( " ")
115     is_summer = winter_or_summer.downcase == "summer" ? 1 : 0
      ➡ unless winter_or_summer. nil?
116     host_city, host_country = row[4], row[5]
117
118     result = db.query( "SELECT id from Countries C WHERE
      ➡ C.name='#{host_country}'" )
119     result.each_hash do |h|
120       begin
121         db.query( "INSERT INTO Games (year, is_summer,
          ➡ host_country, host_city) VALUES ('#{year}',
          ➡ '#{is_summer}', '#{h['id']}', '#{host_city.gsub("'",
          ➡ "''")}'" )
122       rescue => ex
123         puts "DEBUG (Games) : #{ex.message}" if @debug
124       end
125     end
126   end
127   i += 1
128
129 end

```

```

130 puts "Done!"
131
132 # Import #6 : Events
133 puts "Import #6 : Events (long one...)"
134 i = 0
135 CSV.foreach( "dataset/events.csv" ) do |row|
136
137     discipline = nil
138     unless row[1]. nil? or row[2]. nil?
139         result = db.query( "SELECT id from Disciplines D WHERE
140             ↳D.name=#{row[1].gsub("'", "''")}" )
141         result.each_hash do |h|
142             discipline = h[ 'id' ] if i > 0
143         end
144
145         year, is_summer = row[2].split( " " )
146         is_summer = is_summer.downcase == "summer" ? 1 : 0 unless
147             ↳is_summer. nil?
148         result = db.query( "SELECT id from Games G WHERE
149             ↳G.year=#{year}' AND G.is_summer = '#{is_summer}' " )
150         result.each_hash do |h|
151             begin
152                 db.query( "INSERT INTO Disciplines_event_Games
153                     ↳(discipline_id, games_id) VALUES ('#{discipline}',
154                     ↳'#{h['id']}')" ) if i > 0 and !discipline. nil?
155             rescue => ex
156                 puts "DEBUG (events) : #{ex.message}" if @debug
157             end
158         end
159     end
160     i += 1
161 end
162 puts "Done!"
163
164 # Import #7 : Participants
165 i = 0
166 CSV.foreach( "dataset/participants.csv" ) do |row|
167     if i > 0 and !row[0]. nil? and !row[1]. nil? and !row[2]. nil?
168         athlete_id = nil
169         result = db.query( "SELECT id from Athletes A WHERE
170             ↳A.name=#{row[0].gsub("'", "''")}" )
171         result.each_hash do |h|
172             athlete_id = h[ 'id' ]
173         end
174
175         year, is_summer = row[2].split( " " )
176         is_summer = is_summer.downcase == "summer" ? 1 : 0 unless
177             ↳is_summer. nil?
178
179         games_id = nil
180         result = db.query( "SELECT id from Games G WHERE
181             ↳G.year=#{year}' AND G.is_summer = '#{is_summer}' " )
182         result.each_hash do |h|
183             games_id = h[ 'id' ]
184         end
185
186         result = db.query( "SELECT id from Countries C WHERE
187             ↳C.name=#{row[1]}'" )

```

```

179 result.each_hash do |h|
180
181   begin
182     # Inserts into the Representant aggregate.
183     db.query( "INSERT INTO Athletes_represent_Countries
               ↳(athlete_id, country_id) VALUES ('#{athlete_id}',
               ↳'#{h['id']}')" )
184
185     # Inserts into the participants table.
186     db.query( "INSERT INTO Representant_participates_Event
               ↳(athlete_id, country_id, games_id, ranking) VALUES
               ↳('#{athlete_id}', '#{h['id']}', '#{games_id}', 0)" )
187   rescue => ex
188     puts "DEBUG : (participants) : #{ex.message}" if @debug
189   end
190 end
191 end
192 i += 1
193 end
194 puts "Done!"
195
196 # Import #7 : Participants that won a medal (update of participants)
197 puts "Import #7 : Participants that won a medal (update of
      ↳participants) ==> Long one!"
198 i = 0
199 CSV.foreach( "dataset/medals.csv" ) do |row|
200
201   if !row[0]. nil? and !row[1]. nil? and !row[2]. nil? and
      ↳!row[3]. nil? and !row[4]. nil?
202
203     # Get the year and split event this way :
204     #   Split at "at the <year> <is_summer> - <Discipline>"
205
206     country, year, is_summer, discipline = row[0], row[1][/\d+/,
      ↳row[1].downcase([ "summer" ]), row[1].split( "-", 2)[1]
207     is_summer = is_summer. nil? ? 0 : 1
208
209     # Get the ranking for this line.
210     ranking = case row[2].downcase[/^\[S]{4,6}/]
211               when "gold" then 1
212               when "silver" then 2
213               when "bronze" then 3
214             end
215
216     # Quit if the discipline is not specified
217     unless discipline. nil?
218
219       # Fetch discipline_id
220       discipline_id = nil
221       result = db.query( "SELECT id from Disciplines D WHERE
               ↳D.name='#{discipline.gsub("'", "\\\'").strip}'" )
222       result.each_hash do |h|
223         discipline_id = h[ 'id' ]
224       end
225
226       # Fetch country_id
227       country_id = nil
228       result = db.query( "SELECT id from Countries C WHERE

```

```

229     ➡C.name='#{country}'")
230     result.each_hash do |h|
231         country_id = h[ 'id' ]
232     end
233     # Updates each athlete
234     row[3].split( ";" ).each do |athlete|
235         begin
236             # Fetch the athlete_id
237             athlete_id = nil
238             result = db.query( "SELECT id from Athletes A WHERE
239                               ➡A.name='#{athlete.gsub("'", "''")}'" )
240             result.each_hash do |h|
241                 athlete_id = h[ 'id' ]
242             end
243             # Updates the row concerned with the athlete
244             my_query = "UPDATE Representant_participates_Event RE SET
245                       ➡RE.discipline_id='#{discipline_id}',
246                       ➡RE.ranking='#{ranking}' WHERE
247                       ➡RE.athlete_id='#{athlete_id}' AND
248                       ➡RE.country_id='#{country_id}'"
249             db.query(my_query)
250         rescue => ex
251             puts "DEBUG : #{ex.message}" if @debug
252         end
253     end
254 end
255
256 i += 1
257 end
258 puts "Done!"
259
260 db.close

```

Listing 3.1: Ruby importation script

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.

```

1  --Names of athletes who won medals at both summer and winter
   ↳Olympics.
2
3  SELECT a.name
4  FROM (
5      SELECT p.athlete_id  as medalist_id
6      FROM representant_participates_event p, games g
7      WHERE p.ranking != 0  AND p.games_id = g.id  AND g.is_summer = 0)
       ↳m1, (
8      SELECT p.athlete_id  as medalist_id
9      FROM representant_participates_event p, games g
10     WHERE p.ranking != 0  AND p.games_id = g.id  AND g.is_summer =
        ↳1) m2, athletes a
11 WHERE m1.medalist_id = m2.medalist_id  AND a.id = m1.medalist_id;

```

Listing 4.1: Query A

```

1  --Names of gold medalists in sports which appeared only once at the
   ↳Olympics.
2
3  SELECT a.name as athlete, d.name as sport
4  FROM athletes a, representant_participates_event p, disciplines d
5  WHERE a.id = p.athlete_id  AND p.ranking = 1  AND d.id =
       ↳p.discipline_id  AND d.sport IN (
6      SELECT d.sport
7      FROM disciplines d, disciplines_event_games e
8      WHERE d.id = e.discipline_id
9      GROUP BY d.sport
10     HAVING COUNT(*) = 1);

```

Listing 4.2: Query B

```

1  --For each country, print the place where it won its first medal.
2
3  SELECT cl.name as country, g.host_city, g. year
4  FROM games g, countries cl, representant_participates_event p
5  WHERE g.id = p.games_id AND cl.id = p.country_id AND year = (
6      SELECT MIN(g.year)
7      FROM games g, representant_participates_event p
8      WHERE g.id = p.games_id AND p.ranking != 0)
9  GROUP BY p.country_id;

```

Listing 4.3: Query C

```

1  --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.
2
3  SELECT c.name
4  FROM countries c
5  WHERE c.id IN (
6      SELECT p.country_id
7      FROM representant_participates_event p, games g
8      WHERE p.games_id = g.id AND g.is_summer = 0 AND p.ranking != 0
9      GROUP BY p.country_id
10     HAVING COUNT(*) >= ALL (
11         SELECT COUNT(*)
12         FROM representant_participates_event p1, games g1
13         WHERE p1.games_id = g1.id AND g1.is_summer = 0 AND p1.ranking
14             ↳!= 0
15         GROUP BY p1.country_id)
16     UNION
17     SELECT p.country_id
18     FROM representant_participates_event p, games g
19     WHERE p.games_id = g.id AND g.is_summer = 1 AND p.ranking != 0
20     GROUP BY p.country_id
21     HAVING COUNT(*) >= ALL (
22         SELECT COUNT(*)
23         FROM representant_participates_event p1, games g1
24         WHERE p1.games_id = g1.id AND g1.is_summer = 1 AND p1.ranking
25             ↳!= 0
26         GROUP BY p1.country_id));

```

Listing 4.4: Query D

```

1  -- List all cities which hosted the Olympics more than once.
2
3  SELECT DISTINCT G.host_city
4  FROM Games G
5  WHERE EXISTS (
6      SELECT *
7      FROM Games G2
8      WHERE G.id != G2.id AND G.host_city = G2.host_city

```

```
9 );
```

Listing 4.5: Query E

```
1  -- List names of all athletes who competed for more than one
   -- country.
2
3  SELECT A.name
4  FROM Athletes A
5  WHERE (
6      SELECT COUNT(AC.country_id)
7      FROM Athletes_represent_Countries AC
8      WHERE A.id = AC.athlete_id
9  ) > 1;
```

Listing 4.6: Query F

```
1  -- For each Olympic Games print the name of the country with the
   -- most participants.
2
3  SELECT G.year, C.name
4  FROM Games G, Countries C
5  WHERE C.id = (
6      SELECT RE.country_id
7      FROM Representant_participates_Event RE
8      WHERE RE.games_id = G.id
9      GROUP BY RE.country_id
10     ORDER BY COUNT(RE.country_id) DESC LIMIT 1)
11 GROUP BY G.id;
```

Listing 4.7: Query G

```
1  -- List all countries which didnt ever win a medal.
2
3  SELECT C.name
4  FROM Countries C
5  WHERE (
6      SELECT SUM(RE.ranking)
7      FROM Representant_participates_Event RE
8      WHERE RE.country_id = C.id
9  ) IS NULL OR (
10     SELECT SUM(RE.ranking)
11     FROM Representant_participates_Event RE
12     WHERE RE.country_id = C.id
13 ) = 0;
```

Listing 4.8: Query H

```
1  -- 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.
2
```



```

3 SET @selected_game_id = '47';
4
5 -- Here we use a case to count the number of different medals
6 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
7 FROM Representant_participates_Event P
8 -- We do an inner join in order to have the IOC_code for the
   ➤countries.
9 INNER JOIN Countries C ON P.country_id = C.id
10 WHERE P.games_id = @selected_game_id
11 GROUP BY C.ioc_code
12 ORDER BY nb_gold, nb_silver, nb_bronze

```

Listing 4.9: Query I

```

1 -- For each sport, list the 3 nations which have won the most
   ➤medals.
2
3 -- Sadly this query is not finished. We couldnot see how to do it
   ➤properly...
4 -- Here are the guideline that we tried to follow :
5 -- 1 - For each discipline, we see how many medals are there
6 -- 2 - Then we add them to get the number of medals per sports
7 -- 3 - Afterwards we select only the TOP3 for each of them
8 --
9 -- SELECT S.id, COUNT(*) as first, COUNT(*) as second, COUNT(*) as
   ➤third
10 -- FROM Sports S
11 --
12 -- SELECT S.id as sport_id, SUM(P.nb_medals) as nb_medals_per_sport
13 -- FROM Sports S, (
14 --   SELECT P1.discipline_id, COUNT(P1.ranking) as nb_medals
15 --   FROM Representant_participates_Event P1
16 --   INNER JOIN Disciplines D ON P1.discipline_id = D.id
17 --   GROUP BY P1.discipline_id
18 -- ) P, Disciplines D
19 -- WHERE P.discipline_id = D.id AND S.id = D.sport_id
20 -- GROUP BY S.id

```

Listing 4.10: Query J

```

1 -- Compute which country in which Olympics has benefited the most
   ➤from playing in front of the home crowd.
2
3 -- We did not find a way to do this one.

```

Listing 4.11: Query K

```

1 -- List top 10 nations according to their success in team sports.
2
3 -- 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)

```

```

4
5 SELECT medalists.country_name
6 FROM (
7     -- Gets the number of medalists
8     SELECT COUNT(*) as number_of_medalists, c.name as country_name,
9           ↳c.id as country_id
10    FROM Representant_participates_Event p, Countries c,
11           ↳Disciplines d
12   WHERE p.country_id = c.id AND p.discipline_id = d.id AND
13           ↳p.ranking != 0
14   GROUP BY c.id
15 ) medalists,
16 (
17     -- Gets the number of medals
18     SELECT COUNT(DISTINCT d.id) as number_of_medals, c.name as
19           ↳country_name, c.id as country_id
20    FROM Representant_participates_Event p, Countries c,
21           ↳Disciplines d
22   WHERE p.country_id = c.id AND p.discipline_id = d.id AND
23           ↳p.ranking != 0
24   GROUP BY c.id
25 ) medals
26 WHERE medalists.country_id = medals.country_id
27 ORDER BY (medalists.number_of_medalists/medals.number_of_medals)
28 LIMIT 0, 10

```

Listing 4.12: Query L

```

1 -- List all Olympians who won medals for multiple nations.
2
3 -- Look if there is a participant who has a medal for two different
4 ↳countries.
5 SELECT DISTINCT a.name, c1.country_id, c1.country_name,
6           ↳c2.country_id, c2.country_name
7 FROM athletes a,
8 (
9     SELECT p.athlete_id as medalist_id, c.id as country_id, c.name
10           ↳as country_name
11    FROM representant_participates_event p, countries c
12   WHERE p.ranking != 0 AND p.country_id = c.id
13 ) c1,
14 (
15     SELECT p.athlete_id as medalist_id, c.id as country_id, c.name
16           ↳as country_name
17    FROM representant_participates_event p, countries c
18   WHERE p.ranking != 0 AND p.country_id = c.id
19 ) c2
20 WHERE c1.medalist_id = c2.medalist_id AND a.id = c1.medalist_id AND
21       ↳c1.country_id < c2.country_id

```

Listing 4.13: Query M

```

1 -- List all nations whose first medal was gold, all nations whose
2 ↳first medal was silver and all nations whose first medal was
3 ↳bronze

```

```

2
3  -- For each country, the query is searching for the first medal.
4
5  SELECT c.id as country_id, c.name as country_name, g. year, p.ranking
6  FROM representant_participates_event p
7  INNER JOIN Countries c ON c.id = p.country_id
8  INNER JOIN Games g ON g.id = p.games_id
9  WHERE g.year = (
10     SELECT MIN(g1.year)
11     FROM Games g1
12     INNER JOIN representant_participates_event p1 ON p1.games_id =
13         ↳g1.id
14     WHERE p1.country_id = g.id AND p1.ranking != 0
15 )
16 GROUP BY c.id
17 ORDER BY p.ranking

```

Listing 4.14: Query N

```

1  -- For all disciplines, compute the country which waited the most
2  ↳between two successive medals.
3
4  -- The View is the time that a country has waited between 2 medals
5  ↳for each discipline. The following query is only giving the
6  ↳country according to the maximum time and the discipline.
7
8  CREATE VIEW DelayByCountryByDiscipline AS (
9  SELECT p1.discipline_id as discipline_id, g1. year-g2.year as
10     ↳time_waited, p1.country_id as country_id
11  FROM representant_participates_event p1,
12     ↳representant_participates_event p2, games g1, games g2
13  WHERE p1.country_id = p2.country_id AND p1.games_id = g1.id AND
14     ↳p2.games_id = g2.id AND g1.year > g2.year
15  AND p1.ranking != 0 AND p2.ranking != 0 AND p1.discipline_id =
16     ↳p2.discipline_id
17  GROUP BY p1.discipline_id
18 );
19
20 SELECT d.name as discipline, c.name as country, join2.max_delay as
21     ↳number_of_years_waited
22 FROM DelayByCountryByDiscipline join1, Disciplines d, Countries c, (
23     SELECT MAX(time_waited) as max_delay, discipline_id
24     FROM DelayByCountryByDiscipline
25     GROUP BY discipline_id
26 ) join2
27 WHERE join1.discipline_id = join2.discipline_id AND
28     ↳join1.time_waited = join2.max_delay AND join1.discipline_id =
29     ↳d.id
30 AND join1.country_id = c.id

```

Listing 4.15: Query O

```

1  -- List all events for which all medals are won by athletes from
2  ↳the same country.
3
4  SELECT d.id as discipline_id, d.name as discipline_name, c.name as
5     ↳country_name

```

```

4 FROM representant_participates_event p
5 INNER JOIN disciplines d ON p.discipline_id = d.id
6 INNER JOIN countries c ON p.country_id = c.id
7 GROUP BY d.id
8 HAVING COUNT(DISTINCT p.ranking != 0) = (
9     -- We get all the "medal entries" for a given discipline
10    SELECT COUNT(*)
11    FROM representant_participates_event pl
12    WHERE pl.discipline_id = d.id
13 )

```

Listing 4.16: Query P

```

1  -- For each Olympic Games, list the name of the country which
2     ↳ scored the largest percentage of the medals.
3
4  -- We compute views to select the number of medals per country and
5     ↳ per games and to select the number of medals per games. The
6     ↳ third one is to give the percentage of medals by country and
7     ↳ by games. The query is only using the last one in order to get
8     ↳ the name of the country which corresponds to the maximum for
9     ↳ each games.
10
11 -- number of medals per country per games
12 CREATE VIEW NbMedalsByCountryByGames
13 AS (SELECT g.id as games_id, c.id as country_id, COUNT(DISTINCT
14     ↳ d.id) as number_of_medals_per_country
15     FROM Games g, representant_participates_event p, disciplines d,
16     ↳ Countries c
17     WHERE p.games_id = g.id AND p.discipline_id = d.id AND
18     ↳ p.country_id = c.id AND p.ranking != 0
19     GROUP BY g.id, c.id);
20
21 -- number of total medals in games
22 CREATE VIEW NbMedalsByGames
23 AS (SELECT g.id as games_id, COUNT(DISTINCT d.id) as
24     ↳ number_of_medals_in_games
25     FROM Games g, representant_participates_event p, disciplines d
26     WHERE p.games_id = g.id AND p.discipline_id = d.id AND
27     ↳ p.ranking != 0
28     GROUP BY g.id);
29
30 -- Percentage by country by games
31 CREATE VIEW Percentage
32 AS (SELECT nmg.games_id as games_id, c.name as country,
33     ↳ (100*nmpc.number_of_medals_per_country/nmg.number_of_medals_in_games)
34     ↳ as percentage_of_medals
35     FROM NbMedalsByCountryByGames nmpc, NbMedalsByGames nmg,
36     ↳ Countries c
37     WHERE nmg.games_id = nmpc.games_id AND nmpc.country_id = c.id
38     );
39
40 SELECT join1.games_id, join1.country as country,
41     ↳ join1.percentage_of_medals as percentage_of_medals
42 FROM MaxPercentageByGames join1, (
43     SELECT games_id, MAX(percentage_of_medals) as
44     ↳ max_percentage_of_medals

```

```

29     FROM MaxPercentageByGames
30     GROUP BY games_id
31 ) join2
32 WHERE join1.games_id = join2.games_id AND
      join1.percentage_of_medals = join2.max_percentage_of_medals

```

Listing 4.17: Query Q

```

1  -- For all individual sports, compute the most top 10 countries
   -- according to their success score. Success
2  -- score of a country is sum of success points of all its
   -- medalists: gold medal is worth 3 points, silver 2
3  -- points, and bronze 1 point. Shared medal is worth half the
   -- points of the non-shared medal.
4
5  -- 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.
6  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
7  FROM Representant_participates_Event P
8  WHERE P.discipline_id IS NOT NULL AND P.athlete_id IN (
9     -- Gets all the athletes that are doing an individual sport
10    SELECT DISTINCT(a.id) as athlete_id
11    FROM representant_participates_event p, athletes a, disciplines
       -- d, games g
12    WHERE p.athlete_id = a.id AND p.discipline_id = d.id AND
       -- p.games_id = g.id AND p.ranking != 0
13    GROUP BY d.id, g.id, p.country_id, p.ranking
14    HAVING COUNT(*) = 1
15 )
16 GROUP BY P.discipline_id
17 ORDER BY score DESC

```

Listing 4.18: Query R

```

1  -- List names of all athletes who won medals both in individual and
   -- team sports.
2
3  -- We used the same idea fir this query as request T.
4
5  SELECT a.id as athlete_id, a.name as athlete_name
6  FROM (
7     SELECT DISTINCT(a.id) as athlete_id
8     FROM representant_participates_event p, athletes a,
       -- disciplines d, games g
9     WHERE p.athlete_id = a.id AND p.discipline_id = d.id AND
       -- p.games_id = g.id AND p.ranking != 0
10    GROUP BY d.id, g.id, p.country_id, p.ranking
11    HAVING COUNT(*) = 1
12 ) individual_medalist,
13 (
14    SELECT DISTINCT(a.id) as athlete_id
15    FROM representant_participates_event p, athletes a,
       -- disciplines d, games g
16    WHERE p.athlete_id = a.id AND p.discipline_id = d.id AND
       -- p.games_id = g.id AND p.ranking != 0

```

```

17      GROUP BY d.id, g.id, p.country_id, p.ranking
18      HAVING COUNT(*) > 1
19    ) team_medalist,
20    athletes a
21 WHERE individual_medalist.athlete_id = team_medalist.athlete_id      AND
      individual_medalist.athlete_id = a.id

```

Listing 4.19: Query S

```

1  -- List names of all athletes who won gold in team sports, but only
   -- won silvers or bronzes individually.
2
3  -- The query is searching for individual medalist and team medalist
   -- (were there are several medals of the same value for the same
   -- event).
4
5  SELECT a.id as athlete_id, a.name as athlete_name
6  FROM (
7      SELECT DISTINCT(a.id) as athlete_id
8      FROM representant_participates_event p, athletes a, disciplines
          d, games g
9      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)
10     GROUP BY d.id, g.id, p.country_id, p.ranking
11     HAVING COUNT(*) = 1) individual_medalist,
12     (SELECT DISTINCT(a.id) as athlete_id
13     FROM representant_participates_event p, athletes a, disciplines
          d, games g
14     WHERE p.athlete_id = a.id AND p.discipline_id = d.id AND
          p.games_id = g.id AND p.ranking != 1
15     GROUP BY d.id, g.id, p.country_id, p.ranking
16     HAVING COUNT(*) > 1) team_medalist,
17    athletes a
18 WHERE individual_medalist.athlete_id = team_medalist.athlete_id      AND
      individual_medalist.athlete_id = a.id

```

Listing 4.20: Query T

```

1  -- List names of all events and Olympic Games for which the
   -- individual or team has defended a title from the previous
   -- games.
2  -- Forum : check only if country defended title (not athlete)
3
4  -- Checks if a country has defended it's title from the previous
   -- game.
5
6  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
7  FROM representant_participates_event p1, disciplines d1, countries
          c1, games g1,
8  representant_participates_event p2, disciplines d2, countries c2,
          games g2
9  WHERE p1.discipline_id = d1.id AND p1.country_id = c1.id AND
      p1.games_id = g1.id AND p1.ranking = 1 AND

```

```

10 | 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 (
11 |     SELECT *
12 |     FROM games g
13 |     WHERE g.year < g1.year AND g.year > g2.year
14 | )
15 | GROUP BY d1.id

```

Listing 4.21: Query U

```

1 | -- List top 10 countries according to their success on the events
  |   which appear at the Olympics for the first
2 | -- time. Present the list in the form of the medal table (as
  |   described for query I).
3 |
4 | SELECT e1.discipline_id, e1.games_id
5 | FROM disciplines_event_games e1, games g1, (
6 |     SELECT e2.discipline_id  as discipline_id,  MIN(g2.year) as
  |           min_year
7 |     FROM disciplines_event_games e2, games g2
8 |     WHERE e2.games_id = g2.id
9 |     GROUP BY e2.discipline_id
10 | ) min_by_dis
11 | 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.

```

1 | CREATE INDEX myIndex
2 | ON Representant_participates_Event (country_id, games_id)

```

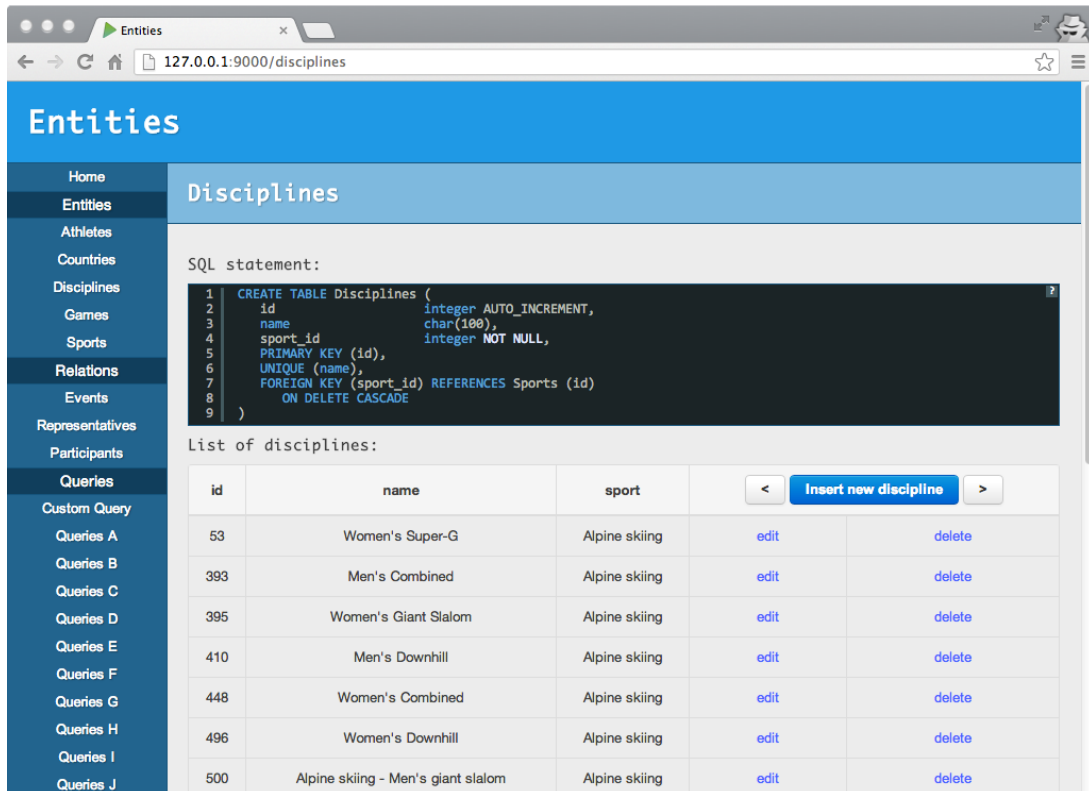
Listing 4.23: Index creation

4.2 Performances

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

5.1 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.



The screenshot shows a web application titled "Entities" with a sidebar menu containing: Home, Entities, Athletes, Countries, Disciplines, Games, Sports, Relations, Events, Representatives, Participants, Queries, Custom Query, Queries A, Queries B, Queries C, Queries D, Queries E, Queries F, Queries G, Queries H, Queries I, and Queries J. The main content area is titled "Disciplines" and displays the following SQL statement:

```

1 CREATE TABLE Disciplines (
2   id          integer AUTO_INCREMENT,
3   name        char(100),
4   sport_id    integer NOT NULL,
5   PRIMARY KEY (id),
6   UNIQUE (name),
7   FOREIGN KEY (sport_id) REFERENCES Sports (id)
8   ON DELETE CASCADE
9 )

```

Below the SQL statement is a table titled "List of disciplines:" with the following data:

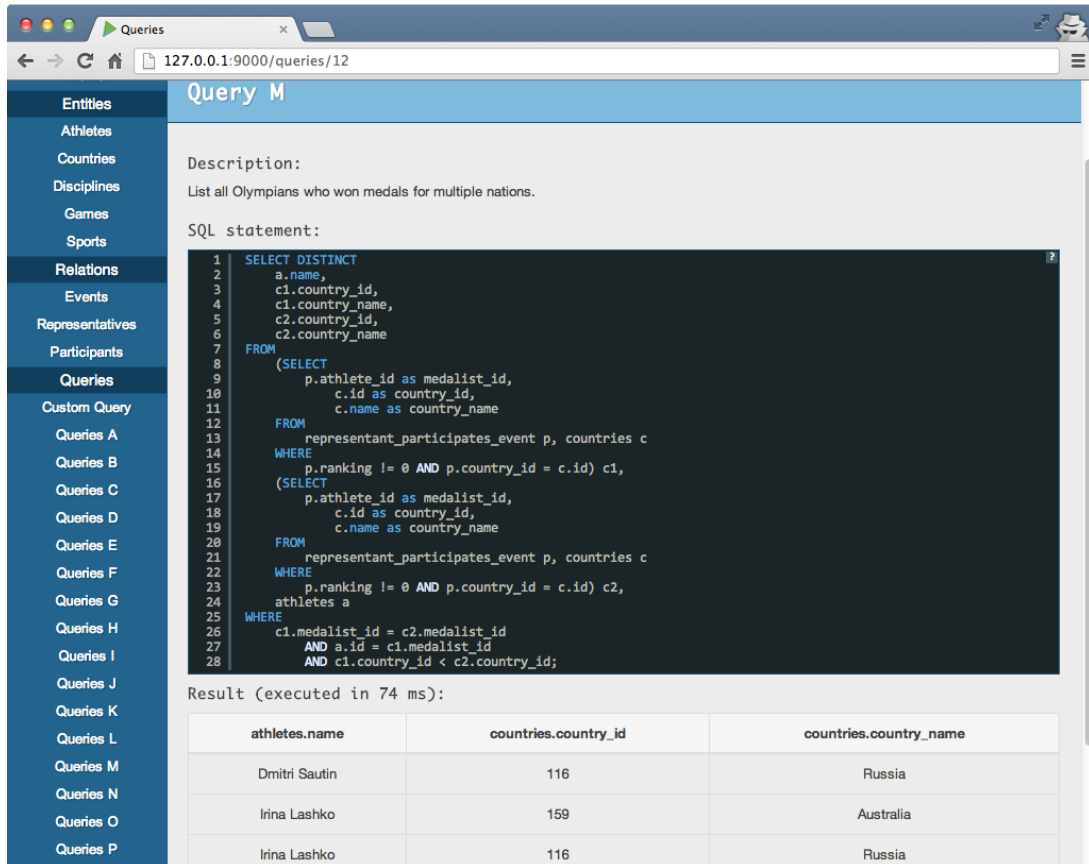
id	name	sport		
53	Women's Super-G	Alpine skiing	edit	delete
393	Men's Combined	Alpine skiing	edit	delete
395	Women's Giant Slalom	Alpine skiing	edit	delete
410	Men's Downhill	Alpine skiing	edit	delete
448	Women's Combined	Alpine skiing	edit	delete
496	Women's Downhill	Alpine skiing	edit	delete
500	Alpine skiing - Men's giant slalom	Alpine skiing	edit	delete

At the top of the table, there is a navigation bar with a left arrow, a button labeled "Insert new discipline", and a right arrow.

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 M

Description:
List all Olympians who won medals for multiple nations.

SQL statement:

```

1  SELECT DISTINCT
2    a.name,
3    c1.country_id,
4    c1.country_name,
5    c2.country_id,
6    c2.country_name
7  FROM
8    (SELECT
9      p.athlete_id as medalist_id,
10     c.id as country_id,
11     c.name as country_name
12   FROM
13     representant_participates_event p, countries c
14   WHERE
15     p.ranking != 0 AND p.country_id = c.id) c1,
16   (SELECT
17     p.athlete_id as medalist_id,
18     c.id as country_id,
19     c.name as country_name
20   FROM
21     representant_participates_event p, countries c
22   WHERE
23     p.ranking != 0 AND p.country_id = c.id) c2,
24   athletes a
25 WHERE
26   c1.medalist_id = c2.medalist_id
27   AND a.id = c1.medalist_id
28   AND c1.country_id < c2.country_id;

```

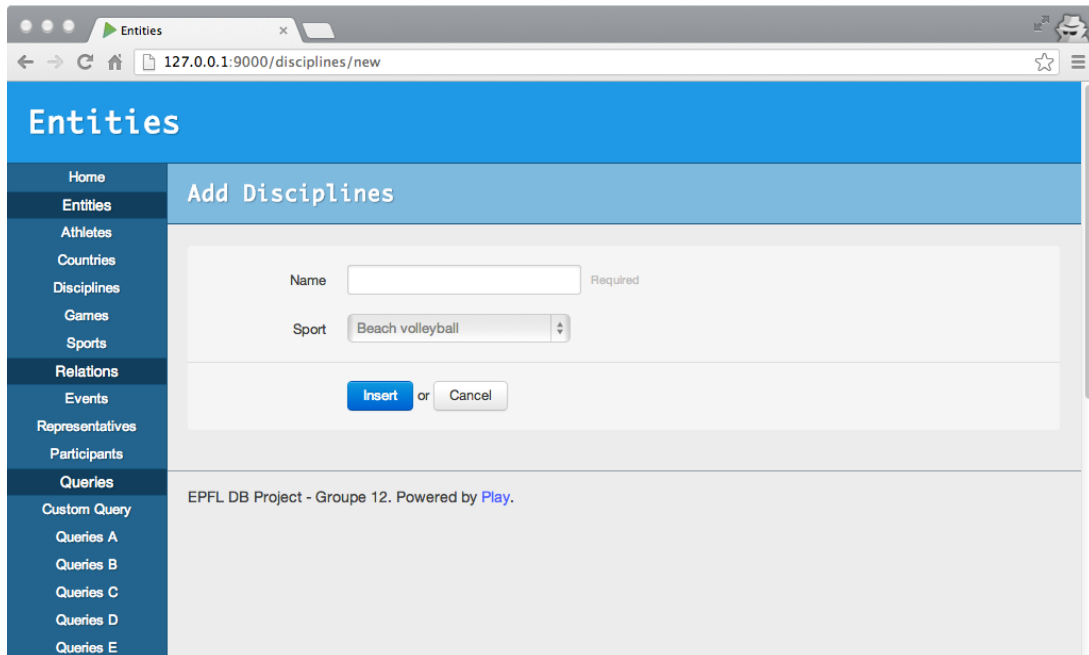
Result (executed in 74 ms):

athletes.name	countries.country_id	countries.country_name
Dmitri Sautin	116	Russia
Irina Lashko	159	Australia
Irina Lashko	116	Russia

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.

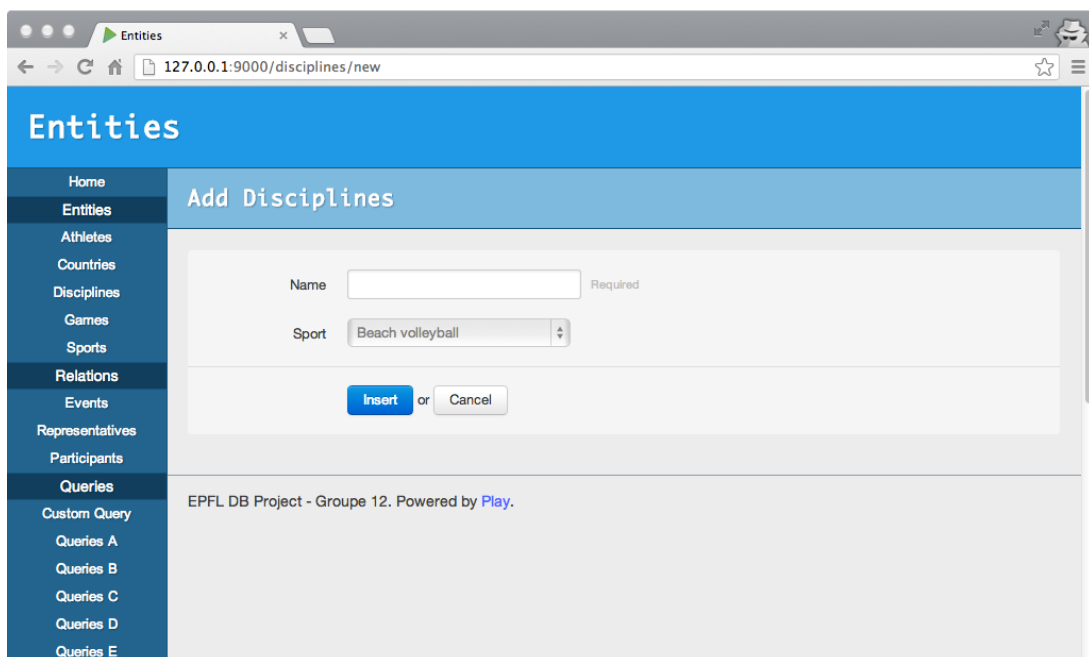


The screenshot shows a web browser window with the URL `127.0.0.1:9000/disciplines/new`. The page has a blue header with the title 'Entities'. On the left is a dark blue sidebar menu with the following items: Home, Entities, Athletes, Countries, Disciplines, Games, Sports, Relations, Events, Representatives, Participants, Queries, Custom Query, Queries A, Queries B, Queries C, Queries D, and Queries E. The main content area has a light blue header with the title 'Add Disciplines'. Below this is a form with two input fields: 'Name' (with a 'Required' label) and 'Sport' (a dropdown menu currently showing 'Beach volleyball'). At the bottom of the form are two buttons: 'Insert' and 'Cancel', separated by the word 'or'. Below the form, there is a footer text: 'EPFL DB Project - Groupe 12. Powered by Play.'

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.



This screenshot is identical to the one in Figure 5.3, showing the 'Entities' web application with the 'Add Disciplines' form. The browser window, sidebar menu, and main form content are all the same as in the previous figure.

Figure 5.4: Add entity form