

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	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
5	Web	15
5.1	Entities & relations	15
5.2	Query view	16
5.3	Add entity / relation	16

Entityrelationship model

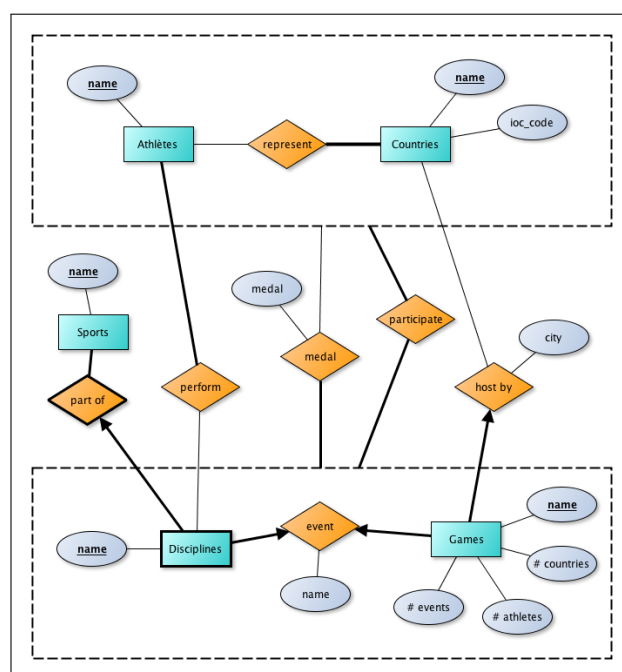


Figure 1.1: Previous ER Model.

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.

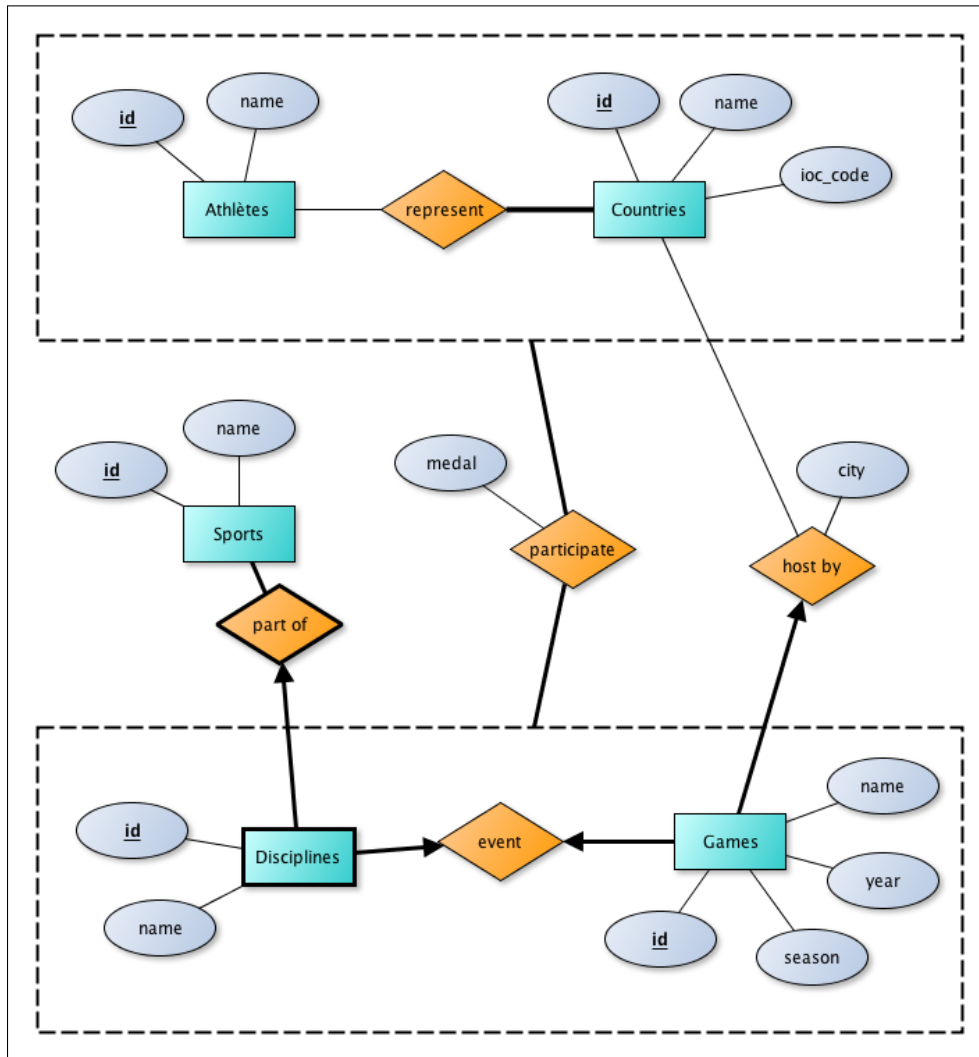


Figure 1.2: New ER Model.

Relational schema and constraints

2.1 Relational schema

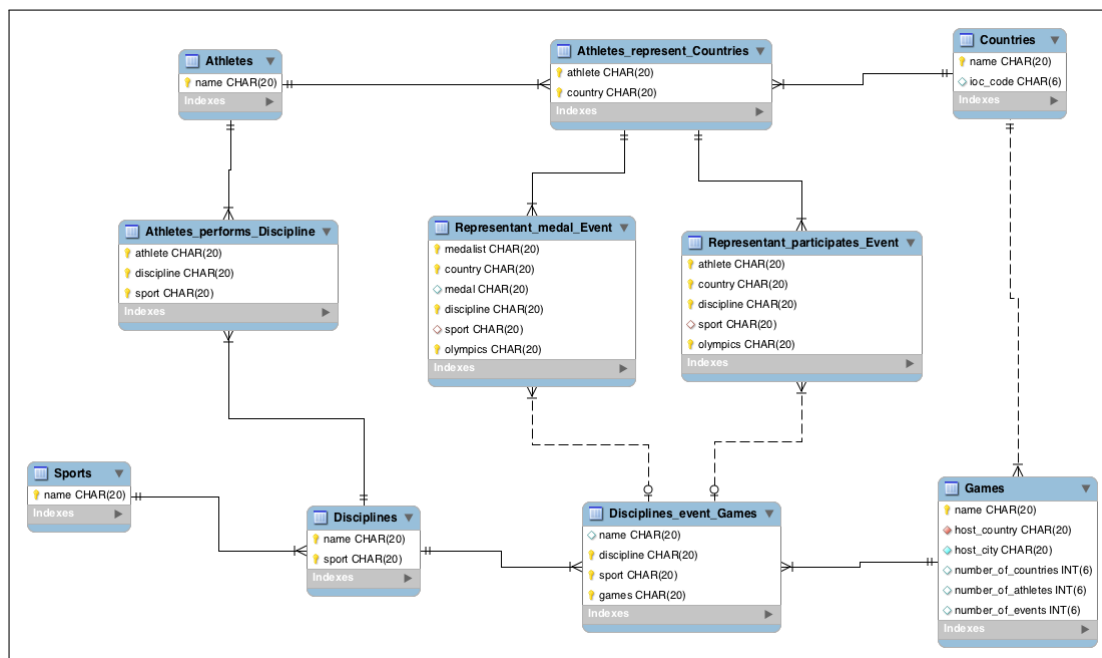


Figure 2.1: Generated EER Model from MySQL Workbench.

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

```

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

```

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 ')
52
53 # Import #1 : Athletes
54 puts "Import #1 : Athletes"
55 i = 0
56 CSV.foreach("dataset/athletes.csv") do |row|
57     db.query("INSERT INTO Athletes (name) VALUES
58     ↪ ('#{row.first.gsub("'", "''")}'") if i > 0
59     i += 1
60 end
61 puts "Done!"
62
63 # Import #2 : Countries
64 puts "Import #2 : Countries"
65 i = 0
66 CSV.foreach( "dataset/countries.csv") do |row|
67     db.query( "INSERT INTO Countries (name, ioc_code) VALUES
68     ↪ ('#{row[0]}', '#{row[1]}')") if i > 0
69     i += 1
70 end
71 puts "Done!"
72
73 # Import #3 : Sports
74 puts "Import #3 : Sports"
75 i = 0
76 CSV.foreach( "dataset/sports.csv") do |row|
77     db.query( "INSERT INTO Sports (name) VALUES ('#{row[0]}')") if i >
78     ↪ 0
79     i += 1
80 end
81 puts "Done!"
82
83 # Import #4 : Disciplines

```

```

79 puts "Import #4 : Disciplines"
80 i = 0
81 CSV.foreach( "dataset/disciplines.csv") do |row|
82   if i > 0
83     result = db.query( "SELECT id from Sports S WHERE
      ↳S.name='#{row[1]}'")
84     result.each_hash {|h| db.query( "INSERT INTO Disciplines (name,
      ↳sport_id) VALUES ('#{row[0].gsub("'", "''")}',
      ↳'#{h['id']}')") }
85   end
86   i += 1
87 end
88 puts "Done!"
89
90 # Import #5 : Games
91 puts "Import #5 : Games"
92 i = 0
93 CSV.foreach( "dataset/games.csv") do |row|
94
95   if i > 0
96     year, winter_or_summer = row[0].split( " ")
97     is_summer = winter_or_summer.downcase == "summer" ? 1 : 0
98     ↳unless winter_or_summer. nil?
99     host_city, host_country = row[4], row[5]
100
101     result = db.query( "SELECT id from Countries C WHERE
      ↳C.name='#{host_country}'")
102     result.each_hash do |h|
103       db.query( "INSERT INTO Games (year, is_summer, host_country,
      ↳host_city) VALUES ('#{year}', '#{is_summer}',
      ↳'#{h['id']}', '#{host_city.gsub("'", "''")}'")
104     end
105   end
106   i += 1
107 end
108 puts "Done!"
109
110 # Import #6 : Events
111 puts "Import #6 : Events (long one...)"
112 i = 0
113 CSV.foreach( "dataset/events.csv") do |row|
114
115   discipline = nil
116   ↳unless row[1]. nil? or row[2]. nil?
117     result = db.query( "SELECT id from Disciplines D WHERE
      ↳D.name='#{row[1].gsub("'", "''")}'")
118     result.each_hash do |h|
119       discipline = h[ 'id' ] if i > 0
120     end
121
122     year, is_summer = row[2].split( " ")
123     is_summer = is_summer.downcase == "summer" ? 1 : 0 ↳unless
      ↳is_summer. nil?
124     result = db.query( "SELECT id from Games G WHERE
      ↳G.year='#{year}' AND G.is_summer = '#{is_summer}' ")
125     result.each_hash do |h|
126       begin
127         db.query( "INSERT INTO Disciplines_event_Games

```

```

        ➤(discipline_id, games_id) VALUES ('#{discipline}',
        ➤'#{h['id']}')") if i > 0 and !discipline. nil?
127 rescue => ex
128   puts "DEBUG (events) : #{ex.message}" if @debug
129 end
130 end
131 end
132 i += 1
133 end
134 puts "Done!"
135
136 # Import #7 : Participants
137 i = 0
138 CSV.foreach( "dataset/participants.csv") do |row|
139   if i > 0 and !row[0]. nil? and !row[1]. nil? and !row[2]. nil?
140     athlete_id = nil
141     result = db.query( "SELECT id from Athletes A WHERE
        ➤A.name='#{row[0].gsub("'", "''")}'")
142     result.each_hash do |h|
143       athlete_id = h[ 'id']
144     end
145
146     year, is_summer = row[2].split( " ")
147     is_summer = is_summer.downcase == "summer" ? 1 : 0 unless
        ➤is_summer. nil?
148
149     games_id = nil
150     result = db.query( "SELECT id from Games G WHERE
        ➤G.year='#{year}' AND G.is_summer = '#{is_summer}' ")
151     result.each_hash do |h|
152       games_id = h[ 'id']
153     end
154
155     result = db.query( "SELECT id from Countries C WHERE
        ➤C.name='#{row[1]}'" )
156     result.each_hash do |h|
157
158       begin
159         # Inserts into the Representant aggregate.
160         db.query( "INSERT INTO Athletes_represent_Countries
        ➤(athlete_id, country_id) VALUES ('#{athlete_id}',
        ➤'#{h['id']}')")
161
162         # Inserts into the participants table.
163         db.query( "INSERT INTO Representant_participates_Event
        ➤(athlete_id, country_id, games_id, ranking) VALUES
        ➤('#{athlete_id}', '#{h['id']}', '#{games_id}', 0)")
164       rescue => ex
165         puts "DEBUG : (participants) : #{ex.message}" if @debug
166       end
167     end
168   end
169   i += 1
170 end
171 puts "Done!"
172
173 # Import #7 : Participants that won a medal (update of participants)
174 puts "Import #7 : Participants that won a medal (update of

```

```

175   ➡participants) ==> Long one!"
176 i = 0
177 CSV.foreach( "dataset/medals.csv") do |row|
178   if !row[0]. nil? and !row[1]. nil? and !row[2]. nil? and
179     ➡!row[3]. nil? and !row[4]. nil?
180     # Get the year and split event this way :
181     #   Split at "at the <year> <is_summer> - <Discipline>"
182
183     country, year, is_summer, discipline = row[0], row[1][/\d+/],
184       ➡row[1].downcase([ "summer"]), row[1].split( "-", 2)[1]
185     is_summer = is_summer. nil? ? 0 : 1
186
187     # Get the ranking for this line.
188     ranking = case row[2].downcase[/^\S{4,6}/]
189       when "gold" then 1
190       when "silver" then 2
191       when "bronze" then 3
192     end
193
194     # Quit if the discipline is not specified
195     unless discipline. nil?
196
197       # Fetch discipline_id
198       discipline_id = nil
199       result = db.query( "SELECT id from Disciplines D WHERE
200         ➡D.name='#{discipline.gsub("'", "''")}'")
201       result.each_hash do |h|
202         discipline_id = h[ 'id' ]
203       end
204
205       # Fetch country_id
206       country_id = nil
207       result = db.query( "SELECT id from Countries C WHERE
208         ➡C.name='#{country}'")
209       result.each_hash do |h|
210         country_id = h[ 'id' ]
211       end
212
213       # Updates each athlete
214       row[3].split( ";").each do |athlete|
215         begin
216
217           # Fetch the athlete_id
218           athlete_id = nil
219           result = db.query( "SELECT id from Athletes A WHERE
220             ➡A.name='#{athlete.gsub("'", "''")}'")
221           result.each_hash do |h|
222             athlete_id = h[ 'id' ]
223           end
224
225           # Updates the row concerned with the athlete
226           my_query = "UPDATE Representant_participates_Event RE SET
227             ➡RE.discipline_id='#{discipline_id}',
228             ➡RE.ranking='#{ranking}' WHERE
229             ➡RE.athlete_id='#{athlete_id}' AND
230             ➡RE.country_id='#{country_id}'"

```

```
223         db.query(my_query)
224
225         rescue => ex
226             puts "DEBUG : #{ex.message}" if @debug
227         end
228     end
229 end
230
231 end
232
233     i += 1
234 end
235 puts "Done!"
236
237 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;
```

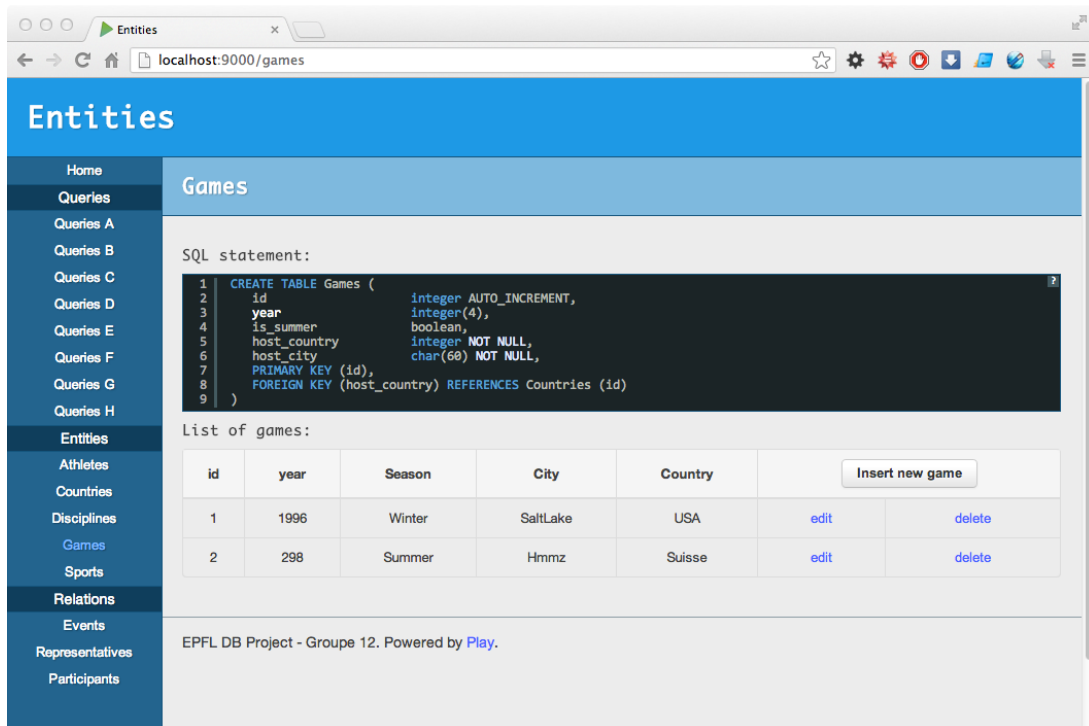
Listing 4.7: Query G

```
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

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 containing links to Home, Queries (A-H), Entities, Athletes, Countries, Disciplines (Games, Sports), Relations, Events, Representatives, and Participants. The 'Entities' section is active, showing the 'Games' entity. The main content area displays the SQL statement for creating the 'Games' table and a list of games.

SQL statement:

```
1 CREATE TABLE Games (
2   id                integer AUTO_INCREMENT,
3   year              integer(4),
4   is_summer         boolean,
5   host_country      integer NOT NULL,
6   host_city         char(60) NOT NULL,
7   PRIMARY KEY (id),
8   FOREIGN KEY (host_country) REFERENCES Countries (id)
9 )
```

List of games:

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

Insert new game

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.

The screenshot shows a web browser window with the address bar at `localhost:9000/queries/5`. The application has a blue header with the title 'Queries' and a sidebar menu on the left. The sidebar menu includes links for Home, Queries, Queries A through H, Entities (Athletes, Countries, Disciplines, Games, Sports), Relations (Events, Representatives, Participants), and a footer note: 'EPFL DB Project - Groupe 12. Powered by Play.'

The main content area is titled 'Query F' and contains the following information:

- Description:** List names of all athletes who competed for more than one country.
- SQL statement:**

```
1 SELECT A.name as name
2   FROM Athletes A
3   WHERE (
4     SELECT COUNT(AC.country_id)
5     FROM Athletes represent_Countries AC
6     WHERE A.id = AC.athlete_id
7   ) > 1
```
- Result:** A table with one column 'name' and one row containing the value 'Abc'.

name
Abc

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 address bar displaying 'localhost:9000/games/new'. The page title is 'Entities'. On the left, there is a sidebar menu with the following items: Home, Queries, Queries A, Queries B, Queries C, Queries D, Queries E, Queries F, Queries G, Queries H, Entities (highlighted), Athletes, Countries, Disciplines, Games, Sports, Relations, Events, Representatives, and Participants. The main content area is titled 'Add Games' and contains a form with the following fields: 'Year' (a text input with a 'Numeric' label), 'Season' (a dropdown menu with 'Summer' selected), 'City' (a text input with a 'Required' label), and 'Country' (a dropdown menu with 'Suisse' selected). Below the form are two buttons: 'Insert' and 'Cancel', separated by the word 'or'. At the bottom of the page, there is a footer that reads 'EPFL DB Project - Groupe 12. Powered by Play.'

Figure 5.3: Add entity form