Introduction to Database Systems: Exercise 2

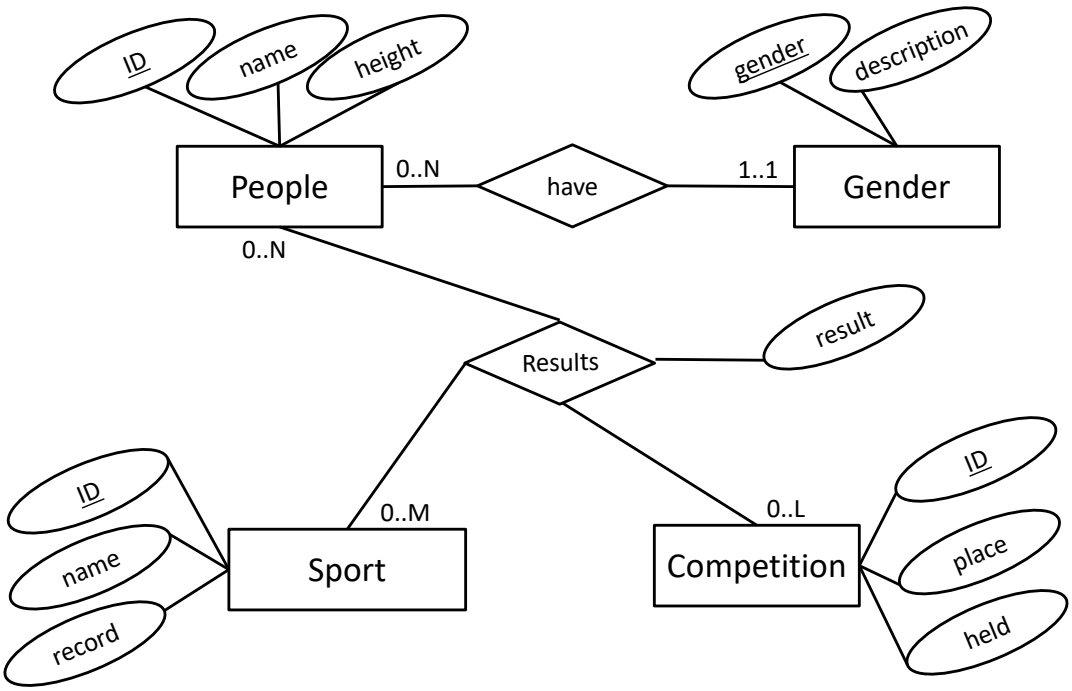
Jorge Quiané

Preparation

Run the Sports.sql script against your PostgreSQL server, to create the exercise database. As mentioned in the first lecture, we advise you to practice running SQL scripts from the command prompt using `psql`.

Database Description

The database holds results for track and field athletes from a sports club. For simplicity there are only seven sports, and a number of other simplifying assumptions are also made. The database has 254 athletes of both genders and more than 10K results from 200 competitions. The data is purely fictional and randomly generated, except that the places where competitions have been held actually exist (although spelling is "anglicized" to avoid non-ASCII characters). Note that the records represent "national records" so the best result in the database may not match the record. The ER diagram below represents the database; check also the CREATE TABLE statements in the Sports.sql script.



The Exercise

Write SQL commands to retrieve the information requested below from your PostgreSQL database. Each piece of information should be retrieved by a single query. You should make the queries as simple as you possibly can:

The output should, for example, only have the columns and rows asked for, only select from the tables required, ordering should only be used if required, and so on. Generally, the output should contain unique rows; yet, the keyword DISTINCT should not be used unless needed.

The construct ORDER BY … LIMIT 1 should not be used to find highest or lowest values; this is not a standard construct and will result in deduction in the exam.

Note also that when "gender" is requested in the queries, this refers to the description of the gender ("Male" or "Female") from the Gender table (this is to train you in joining tables).

The queries should also be well and consistently formatted, and as readable as possible, as SQL queries are generally part of your code base. We recommend that you create a script with all the queries.

Note: For many queries, we have given hints (in italics) about how to write them and what the results should be. In many cases, however, some incorrect variants of some queries could return identical results. You should take care to insert additional data to test the queries adequately. Query 2 is a prime example: if there is no sport without a result, then listing all sports will produce the correct output; adding a new sport with no result will test this aspect of the query.

Basic Queries

The first 13 queries can be solved using the SQL syntax covered in Week 2.

	name character varying (50)	record double precision
1	Discus	50.41
2	High Jump	2.11
3	Javelin	60.46
4	Long Jump	6.78
5	Pole Vault	5.52
6	Shot Put	16.66
7	Triple Jump	13.15

1. The name and record of all sports sorted by name.

   This single-table query should give the results on the right.

	name character varying (50)
1	Triple Jump
2	High Jump
3	Long Jump
4	Pole Vault
5	Discus
6	Javelin
7	Shot Put

2. The name of all sports with at least one result.

   For this query, you need to join the Results table to ensure existence of results, and the Sports table to retrieve the name of the sport. On this database instance, all 7 sports are returned. You can add one sport to test your query.

3. The number of athletes who have competed in at least one sport.

   This query requires only one table since you don't need any information about the people. It should give the result 251.

4. The ID and name of athletes who have at least twenty results.

   This query requires to JOIN two tables, GROUP BY the athlete IDs and use HAVING to check for number of rows. It should return 194 rows; for now you can simply see this in pgAdmin.

5. The ID, name and gender description of all athletes that currently hold a record.

   This query requires joining four tables (People, Gender, Results, and Sports), as well as one WHERE condition. It should return 33 rows.

6. For each sport, where some athlete holds the record, the name of the sport and the number of athletes that hold a record in that sport; the last column should be named "numathletes".

   This query only requires joining two tables but adds a GROUP BY. The results are shown right. For example, Long Jump has a total of 22 people that have equaled the record, while one sport does not yield a result row.

	name character varying (50)	numathletes bigint
1	High Jump	7
2	Long Jump	22
3	Triple Jump	2
4	Shot Put	3
5	Pole Vault	3
6	Javelin	1

   Question: How can the total number of people that have equaled a record in some sport, which is 38, be larger than the 33 from the previous query?

7. The ID and name of each athlete that has at least twenty results in the triple jump, their best result, along with the difference between the record and their best result. The second-to-last column should be named "best" while the last column should be named "difference". The last column should always contain non-negative values and should preferably be formatted to show at least one digit before the decimal point and exactly two digits after the decimal point.

   This query requires a three-way JOIN, along with WHERE, GROUP BY and HAVING. You can use the PostgreSQL-specific code `to_char(S.record-max(R.result), '0D99')` *to format the text. This query should return 7 rows. Why is it that, if you do no formatting, the difference has so many digits?*

8. The ID, name and gender description of all athletes who participated in the competition held in Hvide Sande in 2009.

   You will need to join four tables to find the result here. You can use the PostgreSQL-specific code `extract(year from C.held)` *to find the year. This query should return 84 rows.*

9. The name and gender description of all people with a last name that starts with a "J" and ends with a "sen" (e.g., Jensen, Jansen, Johansen).

   This query should return 82 rows.

10. For each result, the name of the athlete, the name of the sport, and the percentage of the record achieved by the result (a result that is a record should therefore appear as 100; this column should be named "percentage"). Preferably, format the last column to show only whole numbers, as well as the % sign, you can use CASE to detect when the result is NULL and when not.

    This query should return one row for each row in results. You can convince yourself of that by counting the rows in Results (with an SQL query, of course) and comparing the number of rows in the two queries.

11. The number of athletes with some incomplete result registrations.

    This single-table query should return a single row with a single column: the number 75.

12. For each sport, show the ID and name of the sport and the best performance over all athletes and competitions. This last column should be

called ‘maxres’ and should be formatted to show at least one digit before the decimal point and exactly two digits after the decimal point. The query result should be ordered by the sport ID.

The result should be as follows:

0, High Jump, 2.11
1, Long Jump, 6.78
2, Triple Jump, 13.15
3, Shot Put, 16.66
4, Pole Vault, 5.52
5, Javelin, 60.46s
6, Discus, 25.2

13. Show the ID and name of athletes who hold a record in at least two sports, along with the total number of their record-setting or record-equaling results.

    This query requires joining three tables, along with WHERE, GROUP BY and HAVING clauses. It should return a single row: (25, Jens Jansen, 21).

Advanced Queries

The following 7 queries require (or are easier with) SQL syntax that we will cover in Week 3. They are included a) so that you can think about what would be needed to solve them, and b) so that you can use them for practice later.

14. Show the ID, name, and height of athletes who have achieved the best result in each sport, along with the result, the name of the sport, and a column called ‘record?’ which contains 1 if the result is a record or 0 if the result is not a record. Note that for each sport there may be many athletes who share the best result, in this case all the rows should be in the result. If you can print ‘yes’ and ‘no’ instead of 1 and 0, all the better.

    This query can use the `= (SELECT MAX …)` *pattern from Week 1 (slide 64). Note that this version of this query will (most likely) execute quite slowly, while using an* `IN` *sub-query it will run much faster. Next week, you can test both versions to make the performance comparison.*

    This query should return 39 rows, with 6 columns each. Two example rows:

    204, Inge Jensen, 1.7, 6.78, Long Jump, Yes
    221, Jan Hansen, 1.75, 25.2, Discus, No

15. The ID and name of all athletes who have not participated in any competitions.

    This query should return three rows: (249, Jens Hvidt), (250, Inge Lauridsen), and (251, Peter Laudruup).

16. The ID and name of athletes who have competed in June of 2002 or hold a record in High Jump.

    This query should return 53 rows with two columns each.

17. The ID and name of all athletes who hold a record in some sport but have never competed in any other sport.

This query returns a single row: (119, Peter Jansen).

18. The number of athletes who have competed in at least ten different 'places'. Note that it does not matter how many sports or competitions they competed in, just how many places they have competed in.

    This query should return the number 206.

19. Show the ID and name of all athletes who hold a record in all sports.

    This is a fairly basic division query. In this dataset, however, there is no such athlete. You should add records to produce one or more such athletes to test your query.

20. Show the ID, name, record and worst result of all sports that have at least one result from every 'place' where a competition has ever been held.

    This is also a division query, but more complex. It should return information for five sports:

    0, High Jump, 2.11, 1.06
    1, Long Jump, 6.78, 3.39
    2, Triple Jump, 13.15, 6.58
    3, Shot Put, 16.66, 8.34
    4, Pole Vault, 5.52, 2.76

Deliverables

Recall that this exercise is purely for training purposes. No deliverables are needed.