Please Indicate below the group number and then past the following material:

(1) the SQL code you have used to create the schema of your database (only create table and alter table statements (if any), not statements for inserting values)

(2) the SQL code of the queries (possibly with an explanation)

(3) the SQL code used for query optimization for HW2. For each query, indicate the un-optimized version and the optimized one. In case the optimization has been realized through indexes, insert the SQL code for the index creation; in case you have modified the schema (e.g., defined constraints, changed the domain of a field, created a view, constructed a new materialized table, etc.), insert the code you have used for this modification.

INTRODUCTION

The dataset we have chosen concerns the Olympic Games.

We have 11 tables collecting information about all the Olympics from 1896 to 2016.

- **Tables**

```
CREATE SCHEMA olympics;

CREATE TABLE olympics.medal (
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  medal_name VARCHAR(50) DEFAULT NULL,
  CONSTRAINT pk_medal PRIMARY KEY (id)
);


CREATE TABLE olympics.noc_region (
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  noc VARCHAR(5) DEFAULT NULL,
  region_name VARCHAR(200) DEFAULT NULL,
  CONSTRAINT pk_nocregion PRIMARY KEY (id)
);


CREATE TABLE olympics.sport (
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  sport_name VARCHAR(200) DEFAULT NULL,
  PRIMARY KEY (id)
);


CREATE TABLE olympics.city (
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  city_name VARCHAR(200) DEFAULT NULL,
  PRIMARY KEY (id)
```

```
);


CREATE TABLE olympics.event (
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  sport_id INT DEFAULT NULL,
  event_name VARCHAR(200) DEFAULT NULL,
  CONSTRAINT pk_event PRIMARY KEY (id),
  CONSTRAINT fk_ev_sp FOREIGN KEY (sport_id) REFERENCES olympics.sport
(id)
);



CREATE TABLE olympics.games (
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  games_year INT DEFAULT NULL,
  games_name VARCHAR(100) DEFAULT NULL,
  season VARCHAR(100) DEFAULT NULL,
  CONSTRAINT pk_games PRIMARY KEY (id)
);


CREATE TABLE olympics.games_city (
  games_id INT DEFAULT NULL,
  city_id INT DEFAULT NULL,
  CONSTRAINT fk_gci_city FOREIGN KEY (city_id) REFERENCES olympics.city
(id),
  CONSTRAINT fk_gci_gam FOREIGN KEY (games_id) REFERENCES
olympics.games (id)
);


CREATE TABLE olympics.person (
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  full_name VARCHAR(500) DEFAULT NULL,
  gender VARCHAR(10) DEFAULT NULL,
  height INT DEFAULT NULL,
  weight INT DEFAULT NULL,
  CONSTRAINT pk_person PRIMARY KEY (id)
);


CREATE TABLE olympics.games_competitor (
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  games_id INT DEFAULT NULL,
  person_id INT DEFAULT NULL,
  age INT DEFAULT NULL,
  CONSTRAINT pk_gamescomp PRIMARY KEY (id),
  CONSTRAINT fk_gc_gam FOREIGN KEY (games_id) REFERENCES olympics.games
(id),
  CONSTRAINT fk_gc_per FOREIGN KEY (person_id) REFERENCES
olympics.person (id)
);


CREATE TABLE olympics.person_region (
  person_id INT DEFAULT NULL,
  region_id INT DEFAULT NULL,
  CONSTRAINT fk_per_per FOREIGN KEY (person_id) REFERENCES
olympics.person (id),
  CONSTRAINT fk_per_reg FOREIGN KEY (region_id) REFERENCES
olympics.noc_region (id)
);


CREATE TABLE olympics.competitor_event (
  event_id INT DEFAULT NULL,
  competitor_id INT DEFAULT NULL,
  medal_id INT DEFAULT NULL,
  CONSTRAINT fk_ce_com FOREIGN KEY (competitor_id) REFERENCES
olympics.games_competitor (id),
  CONSTRAINT fk_ce_ev FOREIGN KEY (event_id) REFERENCES olympics.event
(id),
  CONSTRAINT fk_ce_med FOREIGN KEY (medal_id) REFERENCES olympics.medal
(id)
);
```

- **Manipulation**

With regard to the manipulation of the dataset, we made two main changes:

1) Remove the height and weight columns from the person table which has 136k rows → although the execution time was about the same, for the purposes of the homework assignment we decided to work without the height and weight columns as none of the queries required them.

```
ALTER TABLE olympics.person DROP COLUMN height, weight;
```

2) Create a new column in the cities table because we would need it for a query.

```
ALTER TABLE olympics.city ADD region varchar(255);
```

Q1) This query returns all countries that have never participated in the sport 'swimming'.

We created a view based on the table person_region, which connects the table person, where there are the names of the athletes, with the table noc_region, where there are the names of the different states, and we inserted into this view all the ids of the states to which the athletes who have never competed in swimming belong.

```
CREATE VIEW not_swimming AS
SELECT DISTINCT pr.region_id
    FROM olympics.competitor_event ce
    INNER JOIN olympics.games_competitor gc ON ce.competitor_id =
gc.id
    INNER JOIN olympics.person_region pr ON gc.person_id =
pr.person_id
    INNER JOIN olympics.event e ON ce.event_id = e.id
    INNER JOIN olympics.sport s ON e.sport_id = s.id
    WHERE s.sport_name = 'Swimming'
```

Then we wrote a query that would return the names of the countries corresponding to the ids that were previously selected in the view.

```
SELECT r.region_name
FROM olympics.noc_region r
WHERE r.id NOT IN (
     SELECT region_id FROM not_swimming
)
```

RUNNING TIME: 0.324 s

Q2) This query returns the person who won the most medals, how many medals he won, in which year and how old he was.

We initially selected the name, age, number of medals won and Olympiad in which the athlete participated.

We then made the necessary explicit joins, using two subqueries:
- the first to add the number of medals won by each person to the competitor_event table;
- the second to associate each event with the number of medals won.
and a nested query to group by event the result of the subquery.

We then sorted the number of medals in descending order and limited the output to a single row to have only the person with the most medals.

```
SELECT p.full_name, g.games_name, ce.num_medals, gc.age
FROM (
    SELECT competitor_id, event_id, COUNT(*) as num_medals
    FROM olympics.competitor_event
    GROUP BY competitor_id, event_id
) ce
INNER JOIN olympics.games_competitor gc ON gc.id = ce.competitor_id
INNER JOIN olympics.person p ON p.id = gc.person_id
INNER JOIN olympics.games g ON g.id = gc.games_id
WHERE (ce.event_id, ce.num_medals) IN (
    SELECT ce2.event_id, MAX(ce2.num_medals) as max_medals
    FROM (
        SELECT competitor_id, event_id, COUNT(*) as num_medals
        FROM olympics.competitor_event
        GROUP BY competitor_id, event_id
    ) ce2
    GROUP BY ce2.event_id
)
ORDER BY ce.num_medals DESC
LIMIT 1;
```

RUNNING TIME: 1.501 s

Q2 OPTIMIZED)

This query is one of those that we decided to optimize and to do this we created views instead of subqueries in order to make the joins more efficient and we slightly modified the code to reduce the amount of rows to compare.

In the first view, we added a descending sorting of the maximum number of medals to make it easier to compare the medals won by the athletes.

```
CREATE VIEW max_medals_per_event_and_athlete AS
SELECT competitor_id, event_id, MAX(num_medals) as max_medals
FROM (
    SELECT competitor_id, event_id, COUNT(*) as num_medals
    FROM olympics.competitor_event
    GROUP BY competitor_id, event_id
) ce
GROUP BY competitor_id, event_id
ORDER BY max_medals DESC;
```

In the second view, we added a having clause to additionally filter the number of medals by eliminating all people who had won only one medal and reduce the number of rows to compare even more.

```
CREATE VIEW competitor_medals AS
      SELECT competitor_id
    FROM max_medals_per_event_and_athlete
    GROUP BY competitor_id
    HAVING MAX(max_medals) > 1;
```

We ran the query again and we saw that the time was halved.

```
SELECT p.full_name, g.games_name, m.max_medals, gc.age
FROM olympics.games_competitor gc
INNER JOIN olympics.person p ON p.id = gc.person_id
INNER JOIN olympics.games g ON g.id = gc.games_id
INNER JOIN max_medals_per_event_and_athlete m ON m.competitor_id =
gc.id
WHERE gc.id IN (
    SELECT competitor_id
    FROM competitor_medals)
ORDER BY m.max_medals DESC
LIMIT 1;
```

RUNNING TIME: 0.695 s

Q3) This query returns in the last 3 Olympics (which are the ones we remember) who took part in 'Women's 100-meter butterfly swimming' and 'Athletics Men's discus throwing' (we considered swimming and athletics because they were the sports we played competitively) and what kind of medals they won.

We selected the columns we needed and starting with the person table we did a series of explicit joins to collect the information we needed, the last join of it being done with a subquery filtering only the last three Olympics.

We then used a where to also filter the names of the events we were interested in and imposed the condition that the selected athletes had won at least one medal.
Finally, we sorted by the year in which the Olympics took place.

```
SELECT person.full_name, games.games_name, event.event_name,
medal.medal_name
FROM olympics.person
INNER JOIN olympics.games_competitor ON person.id =
games_competitor.person_id
INNER JOIN olympics.games ON games_competitor.games_id = games.id
INNER JOIN olympics.competitor_event ON person.id =
competitor_event.competitor_id
INNER JOIN olympics.event ON competitor_event.event_id = event.id
INNER JOIN olympics.sport ON event.sport_id = sport.id
INNER JOIN olympics.medal ON competitor_event.medal_id = medal.id
INNER JOIN (
    SELECT id FROM olympics.games WHERE games_name IN ('2016 Summer',
'2012 Summer', '2008 Summer')
) AS filtered_games ON games.id = filtered_games.id
WHERE event.event_name IN ('Swimming Women''s 100 meters Butterfly',
'Athletics Men''s Discus Throw')
AND medal.medal_name <> 'NA'
ORDER BY games.games_year DESC;
```

RUNNING TIME: 0.131 s

Q4) This query returns in which year and in which city did Italians take the most medals and how many medals did they take in total and for each type.

We started by selecting the name of the city and year in which the games were held, and we used a count function to count the different types of medals.

We then used implicit join to merge the necessary tables and a nested query to filter the results so that only Italian athletes were considered.

We then grouped by year and city, sorted the total of medals in descending order and limited the output to a single row so as to have only the city in which the most medals were won by Italians.

```
SELECT c.city_name, g.games_year,
    COUNT(CASE WHEN ce.medal_id = (SELECT id FROM olympics.medal WHERE
medal_name = 'Gold') THEN 1 ELSE NULL END) AS gold_count,
    COUNT(CASE WHEN ce.medal_id = (SELECT id FROM olympics.medal WHERE
medal_name = 'Silver') THEN 1 ELSE NULL END) AS silver_count,
    COUNT(CASE WHEN ce.medal_id = (SELECT id FROM olympics.medal WHERE
medal_name = 'Bronze') THEN 1 ELSE NULL END) AS bronze_count,
    COUNT(*) AS total_count
FROM olympics.competitor_event AS ce,olympics.event AS e,olympics.games
AS g ,olympics.games_city AS gc, olympics.city AS c
WHERE ce.medal_id IN (SELECT id FROM olympics.medal WHERE medal_name IN
('Gold', 'Silver', 'Bronze'))
AND ce.competitor_id IN (
    SELECT pc.person_id
    FROM olympics.person_region AS pc
    JOIN olympics.noc_region AS n ON pc.region_id = n.id
    WHERE n.region_name = 'Italy'
) AND ce.event_id = e.id AND e.sport_id = g.id AND g.id = gc.games_id
AND gc.city_id = c.id
GROUP BY g.games_year, c.city_name
ORDER BY total_count DESC
LIMIT 1;
```

RUNNING TIME: 0.434 s

Q4 OPTIMIZED)

This is the second query we decided to optimize, and to do this we tried to combine views with indexes.

First, we created two views to replace the subqueries that selected the ids of only the Italian athletes and the ids of the medals, respectively.

```
-- ids of only the Italian athletes
CREATE VIEW italy AS
SELECT pc.person_id
FROM olympics.person_region AS pc
INNER JOIN olympics.noc_region AS n ON pc.region_id = n.id
WHERE n.region_name = 'Italy'

-- the ids of the medals
CREATE VIEW med_not_null AS
SELECT id FROM olympics.medal WHERE medal_name IN ('Gold', 'Silver',
'Bronze')
```

Then we created two indexes on the two columns which were taken into account in the different clauses used in the query and which were not primary keys, specifically on the games_year column and the city_name column.

```
CREATE INDEX idx_games_year ON olympics.games (games_year)
CREATE INDEX idx_city_name ON olympics.city (city_name)
```

Finally we converted implicit joins to explicit joins.

Now, we can see that it is a little faster even if the difference is very small.

```
SELECT c.city_name, g.games_year,
    COUNT(CASE WHEN ce.medal_id = (SELECT id FROM olympics.medal WHERE
medal_name = 'Gold') THEN 1 ELSE NULL END) AS gold_count,
    COUNT(CASE WHEN ce.medal_id = (SELECT id FROM olympics.medal WHERE
medal_name = 'Silver') THEN 1 ELSE NULL END) AS silver_count,
    COUNT(CASE WHEN ce.medal_id = (SELECT id FROM olympics.medal WHERE
medal_name = 'Bronze') THEN 1 ELSE NULL END) AS bronze_count,
    COUNT(*) AS total_count
FROM olympics.competitor_event AS ce
INNER JOIN olympics.event AS e ON ce.event_id = e.id
INNER JOIN olympics.games AS g ON e.sport_id = g.id
INNER JOIN olympics.games_city AS gc ON g.id = gc.games_id
INNER JOIN olympics.city AS c ON gc.city_id = c.id
WHERE ce.medal_id IN (SELECT mn.id FROM med_not_null mn)
AND ce.competitor_id IN (
SELECT i.person_id FROM italy i
)
GROUP BY g.games_year, c.city_name
ORDER BY total_count DESC
LIMIT 1;
```

RUNNING TIME: 0.114 s

Q5)

This query returns which was the strongest country in 'Tug-of-War' and in 'Athletics Men's Stone Throw', the total number of medals that country won in that competition and the year of the last medal.

This query is one that we thought it would be nice to make because, in analyzing all the sports that have been included in the database, we realized that, as the data had been collected since 1896, sports that no longer exist had also been considered.

We selected the necessary columns, using the max function to find the last year in which a medal was won in these sports for each country (and it is interesting to see that stone-throw has not been an Olympic sport since 1906 and tug-of-war since 1920).

We then did the appropriate explicit joins, filtered out the names of the events we were interested in, grouped by event and country and considered only those countries that had won at least one medal.

```
SELECT
    e.event_name,
    nc.region_name AS country,
    COUNT(*) AS total_medals,
    MAX(g.games_year) AS last_medal_year
FROM olympics.competitor_event ce
INNER JOIN olympics.medal m ON ce.medal_id = m.id
INNER JOIN olympics.games_competitor gc ON ce.competitor_id = gc.id
INNER JOIN olympics.event e ON ce.event_id = e.id
INNER JOIN olympics.sport s ON e.sport_id = s.id
INNER JOIN olympics.person p ON gc.person_id = p.id
INNER JOIN olympics.person_region pr ON p.id = pr.person_id
INNER JOIN olympics.noc_region nc ON pr.region_id = nc.id
INNER JOIN olympics.games g ON gc.games_id = g.id
WHERE e.event_name = 'Athletics Men''s Stone Throw' OR s.sport_name =
'Tug-Of-War'
GROUP BY e.event_name, nc.region_name
HAVING COUNT(*) > 0;
```

RUNNING TIME: 0.110 s

Q6)

This query returns the 5 countries that won the most medals at the Olympic games (summer and winter), how many medals they won in total and how many in the summer and in the winter.

We selected the country names and used the COUNT function to have the total medals for each country. Then to distinguish the total summer medals and the total winter medals we used the command SUM with the option CASE WHEN.

After that we did an explicit join on the tables we needed, we grouped by the country, sorted the output in a descending order with respect to the total of medals, and limited the output to five.

```
SELECT
      noc_region.region_name,
      COUNT(*) AS total_medals,
      SUM(CASE WHEN games.season = 'Summer' THEN 1 ELSE 0 END) AS
summer_medals,
      SUM(CASE WHEN games.season = 'Winter' THEN 1 ELSE 0 END) AS
winter_medals
FROM
    olympics.games_competitor
      INNER JOIN olympics.games ON games_competitor.games_id = games.id
      INNER JOIN olympics.games_city ON games.id = games_city.games_id
      INNER JOIN olympics.city ON games_city.city_id = city.id
      INNER JOIN olympics.person_region ON games_competitor.person_id =
person_region.person_id
      INNER JOIN olympics.noc_region ON person_region.region_id =
noc_region.id
GROUP BY
    olympics.noc_region.region_name
ORDER BY
      total_medals DESC
LIMIT 5;
```

RUNNING TIME: 0.367 s

Q7)

This query returns all the athletes who have won at least three medals, who have participated in at least 10 different events in their life and who have participated in at least 5 different Olympic Games.

We selected the name of the athlete, the sport in which he or she participated and we used the count function that we will need later in the command having, to select only the athletes who meet the conditions we want.

Then we used implicit join to merge the necessary tables, we grouped by name of the athlete and sport and we specified the desired conditions in the having function.

```
SELECT
  p.full_name,
  s.sport_name,
  COUNT(DISTINCT gc.games_id) AS num_games,
  COUNT(DISTINCT ce.event_id) AS num_events,
  COUNT(DISTINCT ce.medal_id) AS num_medals
FROM olympics.person p, olympics.games_competitor gc,
olympics.competitor_event ce, olympics.event e, olympics.sport s,
olympics.medal m
```

```
WHERE
  p.gender IN ('M', 'F') AND
  gc.person_id = p.id AND
  ce.competitor_id = gc.id AND
  e.id = ce.event_id AND
  s.id = e.sport_id AND
  ce.medal_id = m.id AND
  ce.medal_id IS NOT NULL
GROUP BY
  p.full_name,
  s.sport_name
HAVING
  COUNT(DISTINCT gc.games_id) >= 5 AND
  COUNT(DISTINCT ce.event_id) >= 10 AND
  COUNT(DISTINCT ce.medal_id) >= 3
ORDER BY
  num_medals DESC,
  num_games DESC,
  num_events DESC;
```

RUNNING TIME: 3.789 s

Q7 OPTIMIZED)

We optimized this query by creating indexes on the columns that were not primary keys and by changing the implicit join with an explicit join.

First we created the indexes.

```
CREATE INDEX person_gender_idx ON olympics.person (gender);
CREATE INDEX gc_person_id_idx ON olympics.games_competitor
(person_id);
CREATE INDEX ce_competitor_id_idx ON olympics.competitor_event
(competitor_id);
CREATE INDEX sport_id_idx ON olympics.event (sport_id);
CREATE INDEX medal_id_idx ON olympics.competitor_event (medal_id);
```

Then we converted implicit joins to explicit joins and ran the query again → we can see that the time is halved (and this was mainly due to the use of explicit joins: the insertion of indexes alone had not produced such a significant change in runtime).

```
SELECT
  p.full_name,
  s.sport_name,
  COUNT(DISTINCT gc.games_id) AS num_games,
  COUNT(DISTINCT ce.event_id) AS num_events,
  COUNT(DISTINCT ce.medal_id) AS num_medals
FROM olympics.person p
INNER JOIN olympics.games_competitor gc ON gc.person_id = p.id
INNER JOIN olympics.competitor_event ce ON ce.competitor_id = gc.id
INNER JOIN olympics.event e ON e.id = ce.event_id
INNER JOIN olympics.sport s ON s.id = e.sport_id
INNER JOIN olympics.medal m ON ce.medal_id = m.id
WHERE
  p.gender IN ('M', 'F') AND
  ce.medal_id IS NOT NULL
GROUP BY
  p.full_name,
  s.sport_name
HAVING
  COUNT(DISTINCT gc.games_id) >= 5 AND
  COUNT(DISTINCT ce.event_id) >= 10 AND
  COUNT(DISTINCT ce.medal_id) >= 3
ORDER BY
  num_medals DESC,
  num_games DESC,
  num_events DESC;
```

RUNNING TIME: 1.765 s

Q8)

This query returns the country, the year and the total number of gold medals won by athletes representing the same region where the Olympic Games were held.

To implement this query we created the column region in the table city since there were only the names of the cities where the games took place but the names of the countries were missing, so we inserted them in `city.region`.

Then we have selected the region, the year of the games and we count the total number of medals won. We made the appropriate explicit joins, we filtered the medals through the command WHERE in order to consider only the gold ones and always through the WHERE clause we made sure that the ids of the host cities corresponded with the ids of the countries of origin of the athletes.

Then we grouped by year and country and sorted by number of medals in a descending order.

```
SELECT
  city.region,
  games.games_year,
  COUNT(*) AS total_medals
FROM
  olympics.games_city
  INNER JOIN olympics.games ON games_city.games_id = games.id
  INNER JOIN olympics.games_competitor ON games.id =
games_competitor.games_id
```

```
  INNER JOIN olympics.person ON games_competitor.person_id =
person.id
  INNER JOIN olympics.competitor_event ON games_competitor.id =
competitor_event.competitor_id
  INNER JOIN olympics.medal ON competitor_event.medal_id = medal.id
  INNER JOIN olympics.city ON games_city.city_id = city.id
  INNER JOIN olympics.person_region ON person.id =
person_region.person_id
  INNER JOIN olympics.noc_region ON person_region.region_id =
noc_region.id
WHERE
  medal.medal_name IN ('Gold')
  AND city.region = noc_region.region_name
GROUP BY
  city.region,
  games.games_year
ORDER BY
  total_medals DESC,
  city.region ASC;
```

RUNNING TIME: 0.453 s

Q9)

This query returns the youngest and oldest person, and how old were both, to have won a medal in the Olympic games, in which season they participated and the sport in which they won.

For this query, we implemented two different queries and then joined them with the UNION command (we decided to use UNION and not UNION ALL to avoid duplicates).

For both, we selected the athletes, in which Olympics they participated, the sport and we made sure that only people who had won a medal were considered. Then we selected the minimum and maximum age respectively.

After that we made the necessary explicit joins, grouped by name, season and medal, and sorted by age, before in a descending order (youngest first) and then in an ascending order (oldest first).

```
 (SELECT DISTINCT person.full_name,
 games.season,
  sport.sport_name,
  competitor_event.medal_id IS NOT NULL as has_medal,
  MIN(games_competitor.age) as age
FROM
  olympics.games_competitor
  INNER JOIN olympics.games ON games.id = games_competitor.games_id
  INNER JOIN olympics.competitor_event ON
competitor_event.competitor_id = games_competitor.id
```

```
  INNER JOIN olympics.event ON event.id = competitor_event.event_id
  INNER JOIN olympics.sport ON sport.id = event.sport_id
  INNER JOIN olympics.person ON person.id =
games_competitor.person_id
GROUP BY person.full_name,
  games.season,
  sport.sport_name,
  has_medal
ORDER BY
  age DESC
LIMIT 1)

UNION

(SELECT DISTINCT person.full_name,
  games.season,
  sport.sport_name,
  competitor_event.medal_id IS NOT NULL as has_medal,
  MAX(games_competitor.age) as age
FROM
  olympics.games_competitor
  INNER JOIN olympics.games ON games.id = games_competitor.games_id
  INNER JOIN olympics.competitor_event ON
competitor_event.competitor_id = games_competitor.id
  INNER JOIN olympics.event ON event.id = competitor_event.event_id
  INNER JOIN olympics.sport ON sport.id = event.sport_id
  INNER JOIN olympics.person ON person.id =
games_competitor.person_id
GROUP BY person.full_name,
  games.season,
  sport.sport_name,
  has_medal
ORDER BY
  age ASC
LIMIT 1)
```

RUNNING TIME: 4.326 s

Q9 OPTIMIZED)

In order to optimize the query we removed the UNION clause and instead of it we used the WHERE clause by doing a nested query using the command MIN and MAX to select the youngest and the oldest person (as we can see, it was sufficient to halve the execution time).

```
SELECT
  person.full_name,
  games.season,
  sport.sport_name,
  MIN(CASE WHEN competitor_event.medal_id IS NOT NULL THEN 'true'
END) AS has_medal,
  games_competitor.age
FROM
  olympics.games_competitor
  INNER JOIN olympics.games ON games.id = games_competitor.games_id
  INNER JOIN olympics.competitor_event ON
competitor_event.competitor_id = games_competitor.id
  INNER JOIN olympics.event ON event.id = competitor_event.event_id
  INNER JOIN olympics.sport ON sport.id = event.sport_id
  INNER JOIN olympics.person ON person.id =
games_competitor.person_id
WHERE
  (games_competitor.age = (SELECT MIN(age) FROM
olympics.games_competitor))
  OR
  (games_competitor.age = (SELECT MAX(age) FROM
olympics.games_competitor))
GROUP BY
  person.full_name,
  games.season,
  sport.sport_name,
  games_competitor.age
ORDER BY
  games_competitor.age ASC;
```

RUNNING TIME: 0.228 s

Q10)

This query returns how many women in total participated in the Olympic Games before and after 1968 → we came up with this query in connection with the fact that women have not always been allowed to participate in sports events.

To implement it, we have counted all the athletes who took part in the Olympics before and after 1968, inserting the DISTINCT clause so as not to count the same athlete twice.

We then made a series of explicit joins to collect the information we needed and we imposed the WHERE clause to filter only women athletes.

We finally grouped by 'period' column, which contains the categories "Before 1968" or "After 1968".

```
SELECT
     COUNT(DISTINCT competitor_event.competitor_id) AS
num_female_athletes,
     CASE
```

```
      WHEN games.games_year < 1968 THEN 'Before 1968'
      ELSE 'After 1968'
      END AS period
FROM
      olympics.games
      INNER JOIN olympics.games_competitor ON games.id =
games_competitor.games_id
      INNER JOIN olympics.competitor_event ON games_competitor.id =
competitor_event.competitor_id
      INNER JOIN olympics.person ON games_competitor.person_id =
person.id
WHERE
      person.gender = 'F'
GROUP BY
      period;
```

RUNNING TIME: 0.235 s