Olympic Games Dataset

Kavya Elemati
kavyaele

Pallavi Thupakula
pthupaku

Ruchitha Kota
rkota2

***Abstract*—This SQL project is used for analysing the Olympic games dataset by creating schemas. We intend to identify the trends and know the history of the Olympic games by writing queries. The dataset contains information about the athletes, events, games and the countries.**

I. Problem Statement

This project will help us to know the history of the Olympic games and identify how different countries have been evolving in terms of playing and participation in different sports. It can help us in identifying the number of medals won by each country across various events conducted over the years. The Olympic games dataset consists of the information about the athletes(Like age, sex, height, weight, country), Games, events and medals. The number of events and games conducted is very huge and analysing the data from a single excel file might be difficult and time consuming. Creating the table and establishing relational database schema between them will make it easier to analyse the data and we can retrieve the data efficiently using the queries. Each table will encapsulate specific aspects of the Olympic data. By making use of the primary key and foreign key while creating the tables, we can uniquely identify the athletes, sports and events and establish the relations between multiple tables and maintain data integrity. The structured queries will enable efficient retrieval and analysis of data through the relational database schema.

II. Target Users

1. Sports Analysts: This project can be used by the sports analysts if they want to know the history about the Olympic games. It will help them to identify which countries performed well in a particular sport, which country won more number of medals, what games have evolved over time etc.

2. National Olympic Committee(NOC): This project can also be used by the National Olympic Committee(NOC) to keep track of their athletes performance so that they can analyse and make strategies to help the countries win more number of medals in the future events. This database can be helpful for athlete selection by choosing the players who had more number of medals in the previous years. Apart from the number of medals won, the age, height and weight of the athlete are also the important factors that need to be considered before making a selection. Furthermore, The database facilitates identifying which countries excel in specific sports.

3. Coaches: Sometimes, coaches might need to evaluate the performance of an athlete whom they haven't known before. In such cases, looking at the database might give them a minor idea about the athlete and how well he performed in that sport previously.
4. Sports Fans: Some fans might be very enthusiastic to know about their favourite sport or player or know how their home country has performed in the Olympic events in the previous years. Looking at the data in excel files might be confusing for them. The database will facilitate accessing this information easily.

Let us look at a real life scenario where this project will be helpful. Suppose the National Olympic Company is trying to finalise the athletes and teams for a future Olympic game. They will try to conduct training sessions according to the athletes previous performance in order to help a country win maximum number of medals possible. Looking at the database will give them an idea about the fitness of the players by having a glance at the Athlete table which contains the age, height and weight of the athlete. The number of medals won will give an idea about how well a particular player can perform in that sport. They can find out whether they lost or won in a sport with a particular country based on the medals won. It also gives them an idea about how well the rival countries perform and they can plan their athletes accordingly. If the country did not win any medals in the previous years, they can implement new strategies or try different athletes.

III. ER Diagram

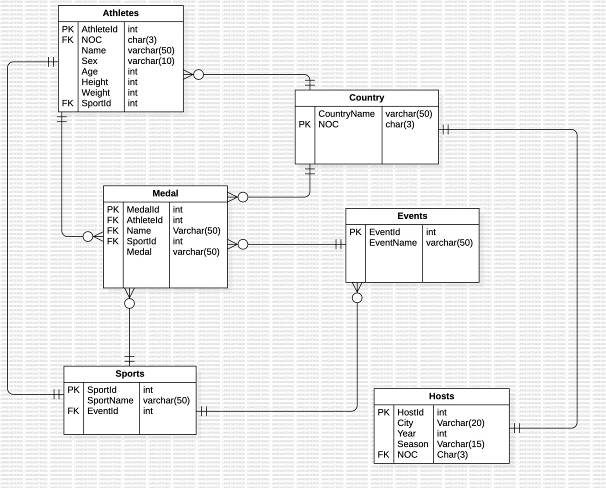


Fig. 1. ER Diagram

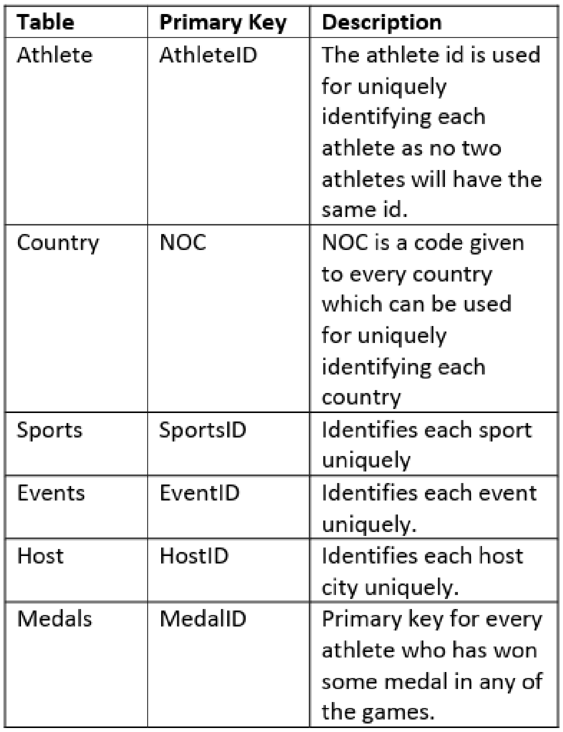


Table	Primary Key	Description
Athlete	AthleteID	The athlete id is used for uniquely identifying each athlete as no two athletes will have the same id.
Country	NOC	NOC is a code given to every country which can be used for uniquely identifying each country
Sports	SportsID	Identifies each sport uniquely
Events	EventID	Identifies each event uniquely.
Host	HostID	Identifies each host city uniquely.
Medals	MedalID	Primary key for every athlete who has won some medal in any of the games.

Fig. 2. Primary key description

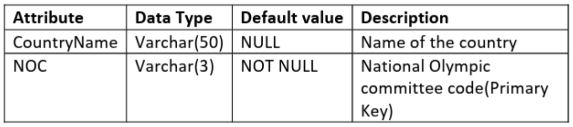
If a primary key record is deleted and if it is being referenced in any other tables, the values will be set to NULL or the default value specified while creating the tables.

IV. Database description

The original Olympic games csv file consisted of the following attributes:
ID - Unique number for each athlete
Name - Athlete's name
Sex - M or F
Age - Integer
Height - In centimeters
Weight - In kilograms
Team - Team name
NOC - National Olympic Committee 3-letter code
Games - Year and season
Year - Integer
Season - Summer or Winter
City - Host city
Sport - Sport
Event - Event
Medal - Gold, Silver, Bronze, or NA
Based on these attributes, We have created 6 tables for a better understanding of the data and retrieval using SQL.

A. Country table

The Country Table has NOC which acts as the primary key, which uniquely identifies each country. There are no foreign keys in the country table. Hence it will not reference any other tables. The purpose of this table is to store the name of the country which the athlete represents along with the NOC code associated with that country.

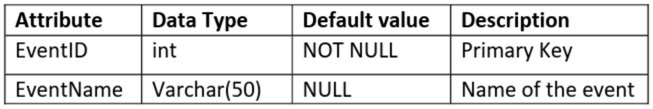


Attribute	Data Type	Default value	Description
CountryName	Varchar(50)	NULL	Name of the country
NOC	Varchar(3)	NOT NULL	National Olympic committee code(Primary Key)

Fig. 3. Country table

B. Events table

In the Events Table, EventID is the primary key. It contains information about specific events in different sports during the Olympic Games. There are no foreign keys in the events table. A particular sport might contain multiple events for the same sport. For example, the athletics sport might contains events like relay, 200 metres race, 500 metres race etc. This kind of information can be stored in the Event table.

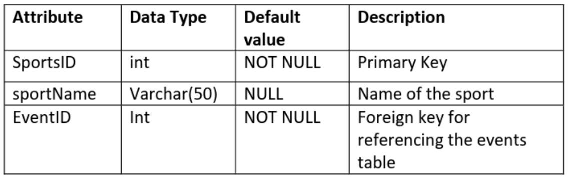


Attribute	Data Type	Default value	Description
EventID	int	NOT NULL	Primary Key
EventName	Varchar(50)	NULL	Name of the event

Fig. 4. Events Table

C. Sports table

The Sports Table has sportID, sport name and Event ID. SportID is the primary key, which uniquely identifies each sport.EventID is the foreign key. This EventID can be used to reference the Events Table to link sports with their corresponding events.

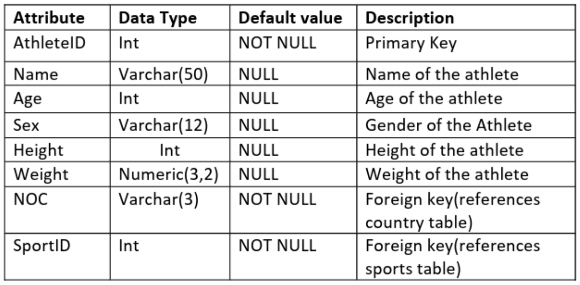


Attribute	Data Type	Default value	Description
SportsID	int	NOT NULL	Primary Key
sportName	Varchar(50)	NULL	Name of the sport
EventID	Int	NOT NULL	Foreign key for referencing the events table

Fig. 5. Sports Table

D. Athletes table

In the Athlete Table, the primary key is AthleteID. The NOC and sportsID are foreign keys. This establishes a relationship between athletes and their respective countries/teams and sports they played. Even when the related records of the NOC and sportsID are deleted from their respective tables, NOC in the Athlete's table will remain uneffected. The type of sport played by the Athlete can be known by using the foreign key sportsID and the country represented by him can be known by using the foreign key NOC.

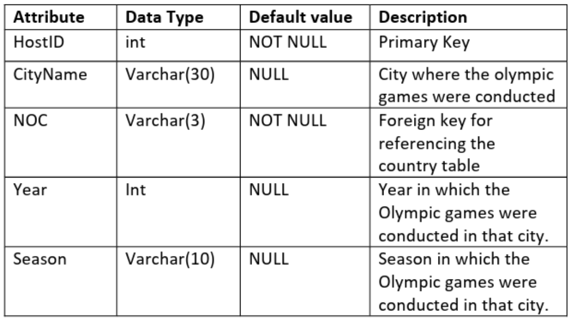


Attribute	Data Type	Default value	Description
AthleteID	Int	NOT NULL	Primary Key
Name	Varchar(50)	NULL	Name of the athlete
Age	Int	NULL	Age of the athlete
Sex	Varchar(12)	NULL	Gender of the Athlete
Height	Int	NULL	Height of the athlete
Weight	Numeric(3,2)	NULL	Weight of the athlete
NOC	Varchar(3)	NOT NULL	Foreign key(references country table)
SportID	Int	NOT NULL	Foreign key(references sports table)

Fig. 6. Athlete table

E. Host Table

In the Host Table, HostID is the primary key. It contains information about whcih city hosted the olympic games, including the year, season, and the country to which the city belongs to. NOC is the foreign key for referencing the Country Table.

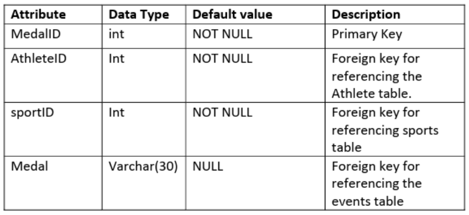


Attribute	Data Type	Default value	Description
HostID	int	NOT NULL	Primary Key
CityName	Varchar(30)	NULL	City where the olympic games were conducted
NOC	Varchar(3)	NOT NULL	Foreign key for referencing the country table
Year	Int	NULL	Year in which the Olympic games were conducted in that city.
Season	Varchar(10)	NULL	Season in which the Olympic games were conducted in that city.

Fig. 7. Host table

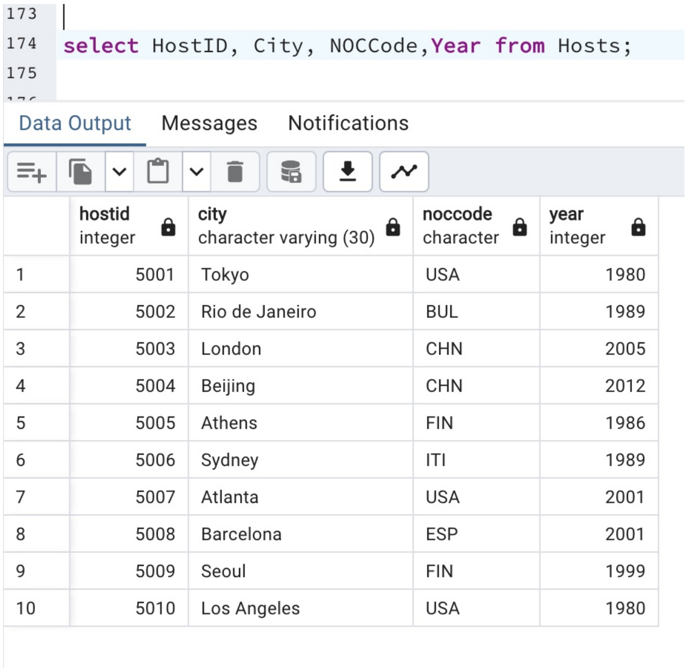
F. Medals Table

The medals table will store only the athleteID's of the athletes who have won a medal in any of the sports in some year. It also contains the SportID to get the details of the sport in which the medal was won.



Attribute	Data Type	Default value	Description
MedalID	int	NOT NULL	Primary Key
AthleteID	Int	NOT NULL	Foreign key for referencing the Athlete table.
sportID	Int	NOT NULL	Foreign key for referencing sports table
Medal	Varchar(30)	NULL	Foreign key for referencing the events table

Fig. 8. Medals table



```
select HostID, City, NOCCode,Year from Hosts;
```

Data Output Messages Notifications

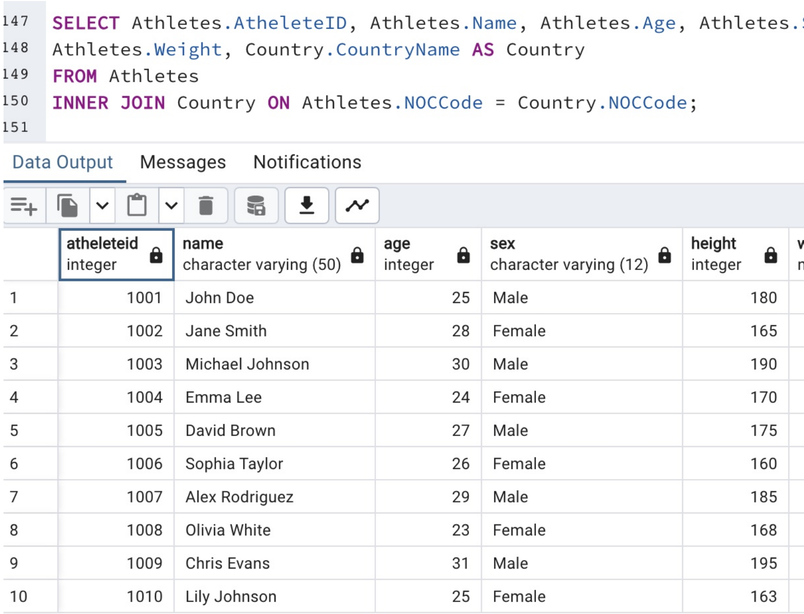
	hostid integer	city character varying (30)	noccode character	year integer
1	5001	Tokyo	USA	1980
2	5002	Rio de Janeiro	BUL	1989
3	5003	London	CHN	2005
4	5004	Beijing	CHN	2012
5	5005	Athens	FIN	1986
6	5006	Sydney	ITI	1989
7	5007	Atlanta	USA	2001
8	5008	Barcelona	ESP	2001
9	5009	Seoul	FIN	1999
10	5010	Los Angeles	USA	1980

Fig. 9. query1

V. Queries

1. Select query

2. querying the details of the athletes along with country to which they belong.



```
SELECT Athletes.AtheleteID, Athletes.Name, Athletes.Age, Athletes.
Athletes.Weight, Country.CountryName AS Country
FROM Athletes
INNER JOIN Country ON Athletes.NOCCode = Country.NOCCode;
```

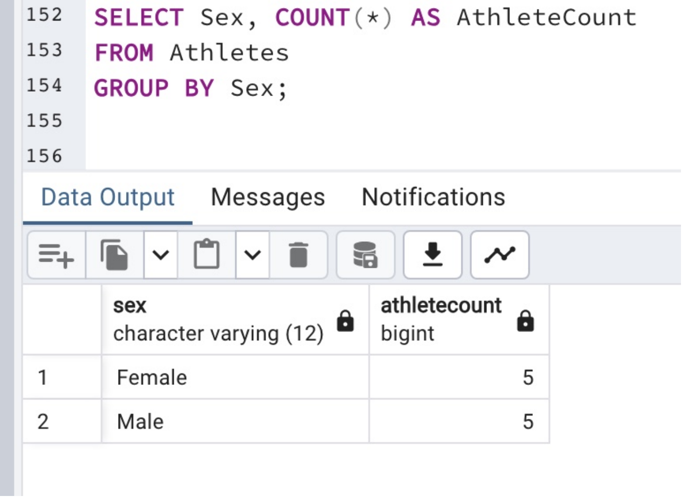
Data Output Messages Notifications

	atheleteid integer	name character varying (50)	age integer	sex character varying (12)	height integer
1	1001	John Doe	25	Male	180
2	1002	Jane Smith	28	Female	165
3	1003	Michael Johnson	30	Male	190
4	1004	Emma Lee	24	Female	170
5	1005	David Brown	27	Male	175
6	1006	Sophia Taylor	26	Female	160
7	1007	Alex Rodriguez	29	Male	185
8	1008	Olivia White	23	Female	168
9	1009	Chris Evans	31	Male	195
10	1010	Lily Johnson	25	Female	163

Fig. 10. query2

3. Selecting the count of athletes by sex using the group by clause:

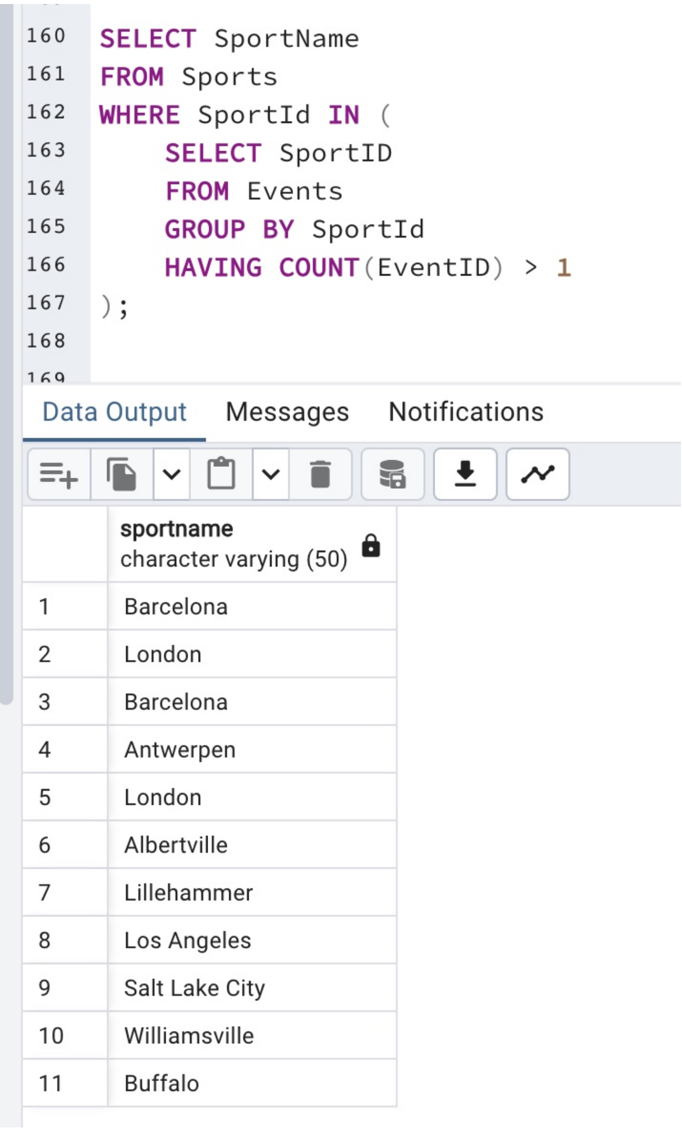
4. Finding sports with multiple events



```sql
SELECT Sex, COUNT(*) AS AthleteCount
FROM Athletes
GROUP BY Sex;
```

	sex character varying (12)	athletecount bigint
1	Female	5
2	Male	5

Fig. 11. query3



```sql
SELECT SportName
FROM Sports
WHERE SportId IN (
    SELECT SportID
    FROM Events
    GROUP BY SportId
    HAVING COUNT(EventID) > 1
);
```

	sportname character varying (50)
1	Barcelona
2	London
3	Barcelona
4	Antwerpen
5	London
6	Albertville
7	Lillehammer
8	Los Angeles
9	Salt Lake City
10	Williamsville
11	Buffalo

Fig. 12. query4

VI. Updated ER Diagram

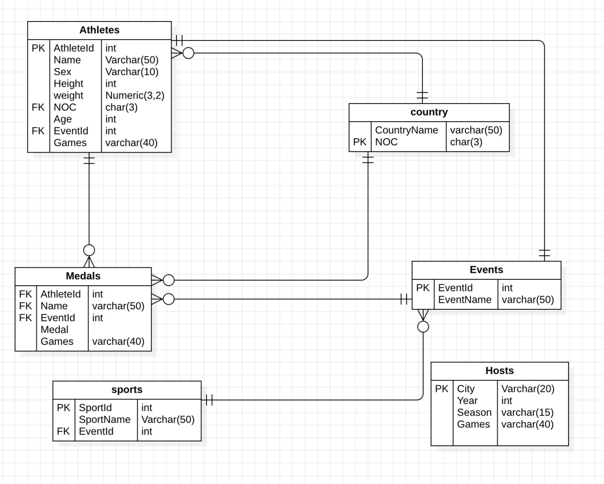


Fig. 13. ER Diagram

The following changes have been made to the previous ER diagram:
1) In the Athletes table, we have changed the SportID to EventId and added the 'Games' attribute. In the previous ER diagram, we used SportID to identify the sport an athlete participated in. However, this led to duplicate records because: A single sport can have multiple events (e.g., Basketball can have Men's Basketball and Women's Basketball). An athlete might participate in all these events for a particular sport, resulting in duplicate entries in the table. To address this issue, the SportID was replaced with EventID. This uniquely identifies the specific event (e.g., Men's Basketball or Women's Basketball) an athlete participated in, eliminating duplicate records. A new column named 'Games' was added to the table. This column specifies whether the athlete participated in the event during "Summer" or "Winter" games along with the year. This is necessary because an athlete might compete in the same sport (e.g., Basketball) in different years (Summer or Winter games). The Games column helps differentiate these participations.

2)In the Hosts table, hostID has been removed. we wanted to generate the hostID so that it would serve as a primary key. But, that is not necessary as the 'City' acts as the primary key. This is because the city itself uniquely identifies a host location for the games and hostID is not required. The column 'NOC' has been removed. We included NOC so that it will tell us the country which the host city belongs to. But, NOC can only determine the country which an athlete belongs to but not the city where the olympics were conducted. An additional column 'Games' has been added. Games just tells us the season and year in which the the host conducted the olympic games.

3)In the medals table, medalID was intended to be the primary key for the Medals table but it has been removed as it does not make any sense. AthleteID which is the foreign key referencing the Athletes table can act as the primary key as well. Removed SportsID and added the EventID instead as EventID is more specific compared to SportsID and avoids ambiguity. A sport can have multiple events (e.g., Basketball can have Men's and Women's), so EventID clarifies which specific event the medal was awarded for. We also added the 'Games' column. An athlete might win medals in the same event (e.g., 100m sprint) but in different Olympic Games. The Games column helps differentiate these achievements.

VII. Database Normalization

Normalization is a database design process used to organize a relational database in such a way that it reduces data redundancy and minimizes the potential for data anomalies, while improving data integrity and efficiency. The primary goals of normalization are to make the database structure more efficient, maintainable, and less prone to data anomalies.

Boyce-Codd Normal Form(BCNF): A relation is said to be in BCNF if the left hand sides of the functional dependencies are keys. X gives Y is an assertion about a relation R that whenever two tuples of R agree on all the attributes of X, then they must also agree on all attributes in set Y. Then there will be a functional dependency from X to Y. For the relation R to be in BCNF, X should be a key.

A. Athletes Table

Athlete id is the primary key. A primary key has to functionally determine all the attributes in the table. But, the same athlete might play in more than one event. The same athlete might also particpate in olympics acroos multiple years. For each event and year combination, you might have a separate record with the same athlete ID but different information related to that specific participation (e.g., eventid,age). In order to get rid of this redundancy, we need to decompose the Athletes table.
Example: (id, name, sex, height,weight, noc, age, eventid, games)= (5,'Christine Jacoba Aaftink',F, 185,82, "NED", 21, 45, '1988 Winter'), (5,'Christine Jacoba Aaftink',F, 185,82, "NED", 21, 50, '1988 Winter') , (5,'Christine Jacoba Aaftink',F, 185,82, "NED", 21, 45, '1992 Summer'),(5,'Christine Jacoba Aaftink',F, 185,82, "NED", 21, 50, '1988 Winter'). We can see that the id(primary key) is repeating 4 times.
There is also a violation of BCNF. The columns ID,Games and EventID functionally determine the age, sex and NOC. But (ID, Games, EventID) is not a key. Hence it is a violation of BCNF.

The Athletes table can be decomposed into 2 tables as:
1)AthletesProfile(ID, Name, Sex, Height, Weight):
Primary key: ID
All the other attributes are fully determined by the Primary key. Hence, there is no BCNF violation

2)Participation(ParticipationID, ID, Age, NOC, Games, EventID)
Primary Key: ParticipationID. It is new column created for uniquely identifying each row.
Foreign key: ID(referencing the AthletesProfile table
All the remaining attributes are all related to a specific participation(identified by particpationID) and the athlete involved(linked by the foreign key ID). They are all fully determined by the combination of participationID and the foreign key, satisfying the BCNF.

Age of the athlete is added in the Participation table instead of AthletesProfile because the same Athlete might particpate in the games in more than one year and the age will keep changing. [We are assuming that the Height and Weight of the person will remain unchanged.]

B. Country Table

There are only 2 columns and the Country Name is fully determined by the NOC. Hence, it satisfies BCNF. There is no need to check whether a table is in BCNF or not if it has only 2 columns since any table which is having only 2 columns will always satify BCNF.

C. Events Table

Events table only has 2 columns(EventID and EventName), so the BCNF condition will be satified. EventID is the primary key and it will functionally determine the EventName.

D. Sports Table

The sports table has SportID, SportName and EventID. The SportID uniquely identifies each sport and it is the primary key. However, the same sport might have multiples events which leads to the redundancies in the SportID and SportName columns. For example, "Gymnastics - Men's Horizontal Bar" and "Gymnastics - Women's Uneven Bars" would have the same SportID (Gymnastics) but different EventIDs and distinct sport names. This scenario creates a partial dependency. EventID helps determine SportName, but it's not part of the entire primary key (SportID). This violates BCNF because a non-key attribute (SportName) is not fully determined by the whole primary key.
Decomposing the Sports table into 2 tables: one table has SportID, SportName. The other table has EventID, SportID.
So the Sports table is decomposed into:
1)Sports(SportID, SportName)
2)SportsAndEvents(EventID, SportID)

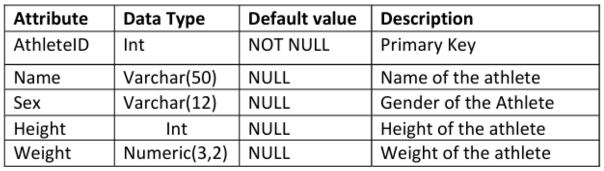
E. Hosts Table

The Hosts table has the attributes 'City','Year','Season' and 'Games'. (City, Year, Season) is the key for this relation to identify the hosting location. In some cases, a city might host multiple Olympic Games throughout history (e.g., London). In such scenarios, City alone might not be sufficient as a primary key. Hence,the combined key (City, Year, Season) is the primary key. If City is the only key, then there will be a violation of BCNF since there is a functional dependancy from season, year to games and year, season is not a key.

F. Medals Table

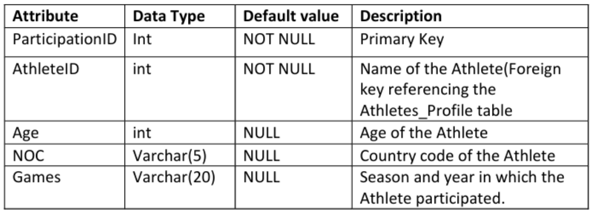
The Medals table has the columns ID, Name, Games, EventID, and Medal where ID is the foreign key referencing the Athletes table. Games, EventID, and Medal are all fully determined by the combination of the foreign key ID (referencing AthleteID) and the other two attributes (Games and EventID). Knowing an athlete's ID (linking to their full information in the Athletes table), the specific Games and EventID will uniquely identify a particular medal awarded. In this scenario, the Medal table doesn't need Name to be in BCNF because you can retrieve the athlete's name by joining the Medal table with the Athletes table using the foreign key relationship. This avoids redundancy and ensures data integrity. With ID being the foreign key, the Medals table satisfies the BCNF.

The final tables after decomposition are :



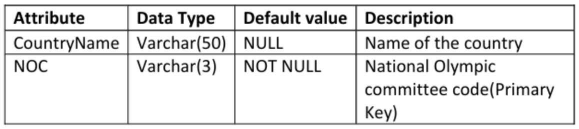
Attribute	Data Type	Default value	Description
AthleteID	Int	NOT NULL	Primary Key
Name	Varchar(50)	NULL	Name of the athlete
Sex	Varchar(12)	NULL	Gender of the Athlete
Height	Int	NULL	Height of the athlete
Weight	Numeric(3,2)	NULL	Weight of the athlete

Fig. 14. Athletes Profile Table



Attribute	Data Type	Default value	Description
ParticipationID	Int	NOT NULL	Primary Key
AthleteID	int	NOT NULL	Name of the Athlete(Foreign key referencing the Athletes_Profile table
Age	int	NULL	Age of the Athlete
NOC	Varchar(5)	NULL	Country code of the Athlete
Games	Varchar(20)	NULL	Season and year in which the Athlete participated.

Fig. 15. Participation Table



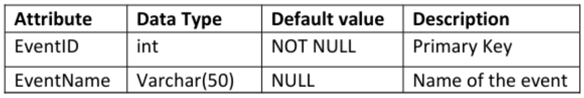
Attribute	Data Type	Default value	Description
CountryName	Varchar(50)	NULL	Name of the country
NOC	Varchar(3)	NOT NULL	National Olympic committee code(Primary Key)

Fig. 16. Country Table

G. ER diagram after decomposition

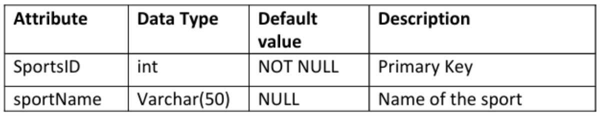
VIII. Queries

1.INSERT
2.UPDATE
3.DELETE



Attribute	Data Type	Default value	Description
EventID	int	NOT NULL	Primary Key
EventName	Varchar(50)	NULL	Name of the event

Fig. 17. Events Table



Attribute	Data Type	Default value	Description
SportsID	int	NOT NULL	Primary Key
sportName	Varchar(50)	NULL	Name of the sport

Fig. 18. Sports Table

4. Finding the name of the Event a particular athlete participated in:
5. Query to find the total number of medals won by each country in a specific year.
6. Query to find the most popular sport by number of participants in the latest Olympics
7. Query to determine the average age of medalists for each Olympic Games:
8.Query to find athletes who have participated in both the Summer and Winter Olympics:
9.Query to find the event that had the highest number of participating athletes in the last Olympic games:

IX. Indexing

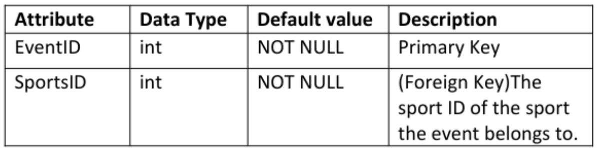
Using Indexing can optimize the performance since many complex joins and aggregations across multiple tables are being used. Indexing will help in speeding up the process of data retrieval. The amount of time the database spends scanning tables to find relevant data will be reduced and hence reduced costs.

X. Individual Contributions

1)KAVYA ELEMATI: Worked on loading the data and creating the tables. Checked whether the tables are in BCNF and decomposed them. Wrote the sql queries and did indexing to optimize the costs. Made the report in IEEE format.
2)PALLAVI THUPAKULA: Designed the ER diagrams for the database. Identified the primary keys and foreign keys in all the relations and the relationships between the tables. worked on building web application for the project and displayed query results.
3)RUCHITHA KOTA: Researched about the databases and came up with the Olympic Games dataset.worked upon cleaning the dataset.Wrote various sql queries.

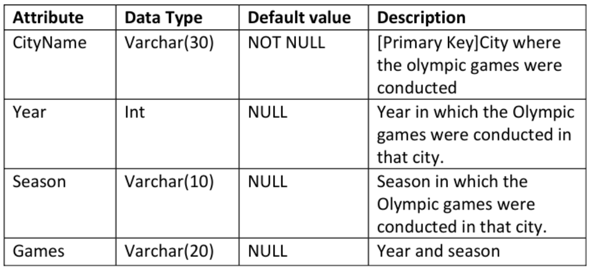
XI. references

[1]Lecture slides
[2]https://www.kaggle.com/datasets/heesoo37/120-years-of-olympic-history-athletes-and-results?resource=download
[3]https://www.geeksforgeeks.org/python-import-csv-into-postgresql
[4]https://www.tutorialspoint.com/dbms/dbms



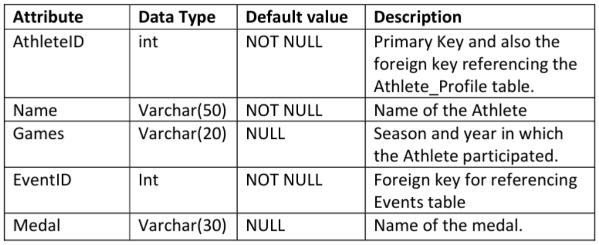
Attribute	Data Type	Default value	Description
EventID	int	NOT NULL	Primary Key
SportsID	int	NOT NULL	(Foreign Key)The sport ID of the sport the event belongs to.

Fig. 19. Events Sports Table



Attribute	Data Type	Default value	Description
CityName	Varchar(30)	NOT NULL	[Primary Key]City where the olympic games were conducted
Year	Int	NULL	Year in which the Olympic games were conducted in that city.
Season	Varchar(10)	NULL	Season in which the Olympic games were conducted in that city.
Games	Varchar(20)	NULL	Year and season

Fig. 20. Hosts Table



Attribute	Data Type	Default value	Description
AthleteID	int	NOT NULL	Primary Key and also the foreign key referencing the Athlete_Profile table.
Name	Varchar(50)	NOT NULL	Name of the Athlete
Games	Varchar(20)	NULL	Season and year in which the Athlete participated.
EventID	Int	NOT NULL	Foreign key for referencing Events table
Medal	Varchar(30)	NULL	Name of the medal.

Fig. 21. Medals Table

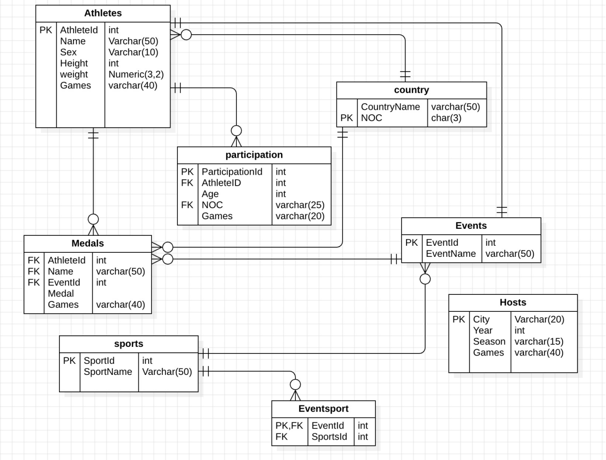
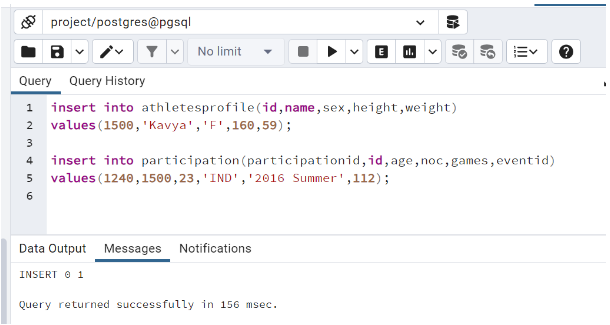


Fig. 22. Final ER Diagram



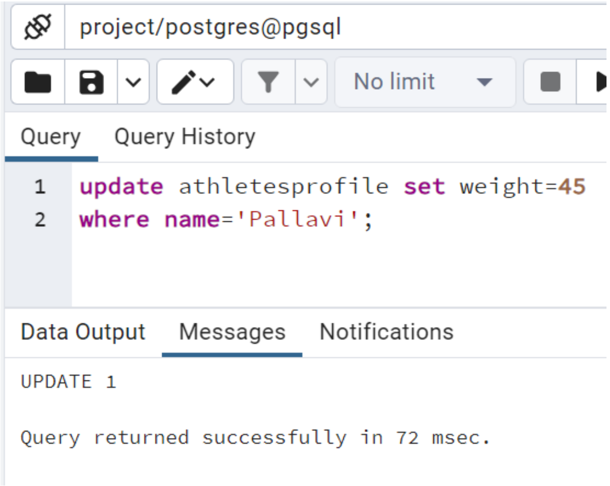
```
insert into athletesprofile(id,name,sex,height,weight)
values(1500,'Kavya','F',160,59);

insert into participation(participationid,id,age,noc,games,eventid)
values(1240,1500,23,'IND','2016 Summer',112);
```

```
INSERT 0 1

Query returned successfully in 156 msec.
```

Fig. 23. query1

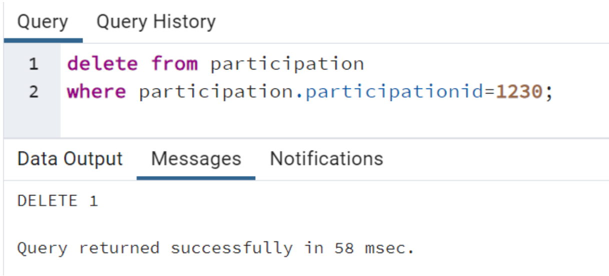


```
update athletesprofile set weight=45
where name='Pallavi';
```

```
UPDATE 1

Query returned successfully in 72 msec.
```

Fig. 24. query2

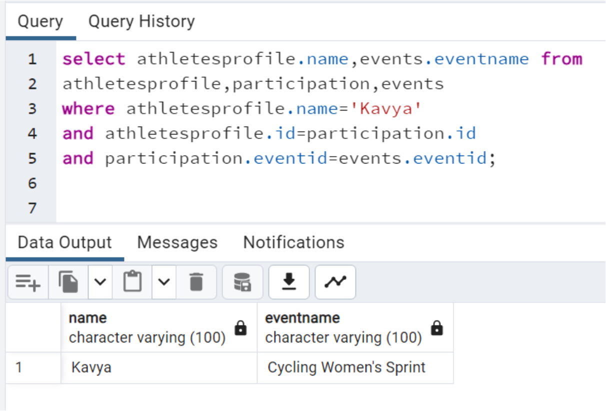


```
delete from participation
where participation.participationid=1230;
```

```
DELETE 1

Query returned successfully in 58 msec.
```

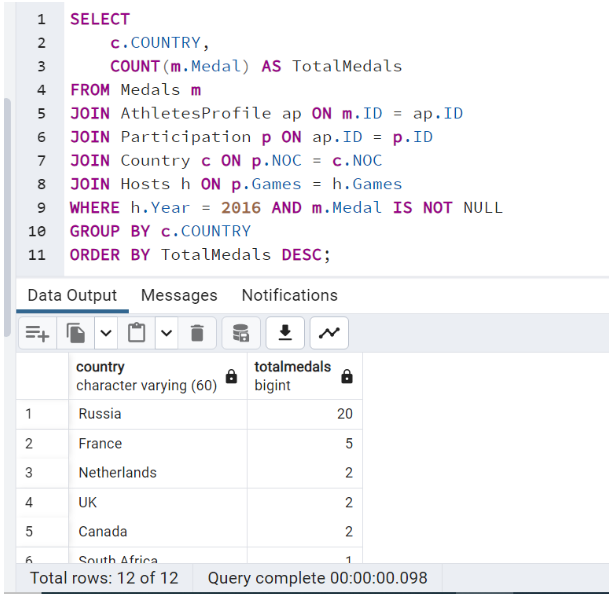
Fig. 25. query3



```sql
select athletesprofile.name,events.eventname from
athletesprofile,participation,events
where athletesprofile.name='Kavya'
and athletesprofile.id=participation.id
and participation.eventid=events.eventid;
```

	name character varying (100)	eventname character varying (100)
1	Kavya	Cycling Women's Sprint

Fig. 26. query4

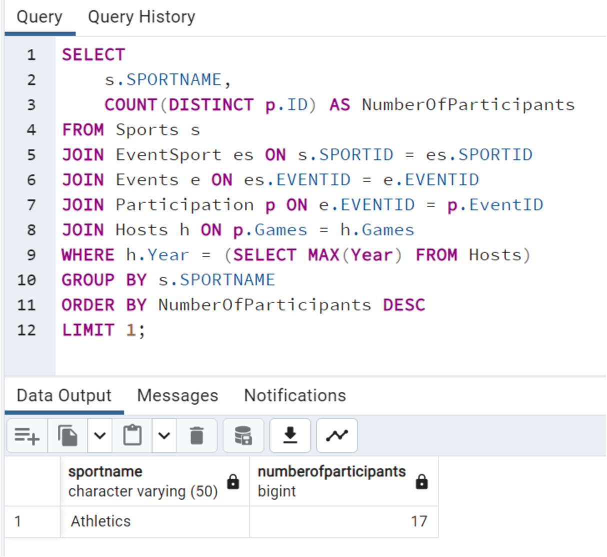


```sql
SELECT
    c.COUNTRY,
    COUNT(m.Medal) AS TotalMedals
FROM Medals m
JOIN AthletesProfile ap ON m.ID = ap.ID
JOIN Participation p ON ap.ID = p.ID
JOIN Country c ON p.NOC = c.NOC
JOIN Hosts h ON p.Games = h.Games
WHERE h.Year = 2016 AND m.Medal IS NOT NULL
GROUP BY c.COUNTRY
ORDER BY TotalMedals DESC;
```

	country character varying (60)	totalmedals bigint
1	Russia	20
2	France	5
3	Netherlands	2
4	UK	2
5	Canada	2
6	South Africa	1

Total rows: 12 of 12 Query complete 00:00:00.098

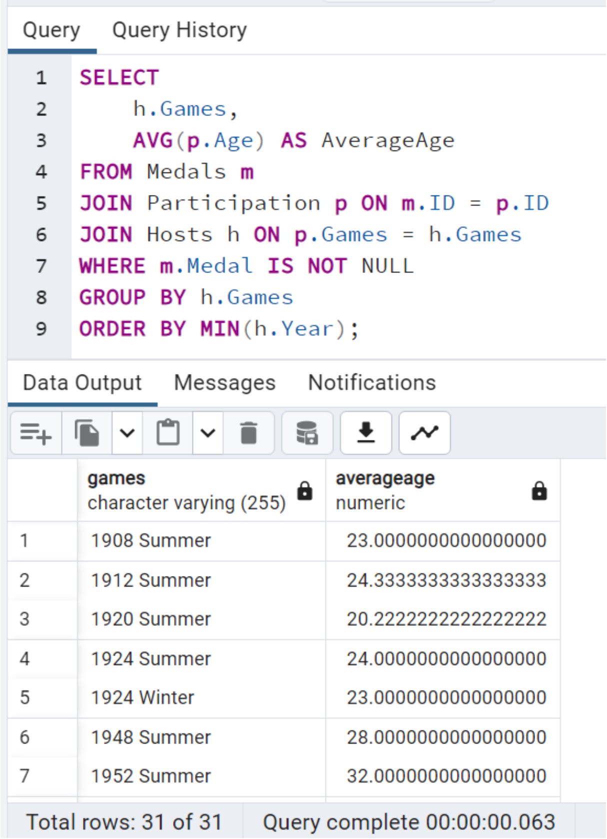
Fig. 27. query5



```sql
SELECT
    s.SPORTNAME,
    COUNT(DISTINCT p.ID) AS NumberOfParticipants
FROM Sports s
JOIN EventSport es ON s.SPORTID = es.SPORTID
JOIN Events e ON es.EVENTID = e.EVENTID
JOIN Participation p ON e.EVENTID = p.EventID
JOIN Hosts h ON p.Games = h.Games
WHERE h.Year = (SELECT MAX(Year) FROM Hosts)
GROUP BY s.SPORTNAME
ORDER BY NumberOfParticipants DESC
LIMIT 1;
```

	sportname character varying (50)	numberofparticipants bigint
1	Athletics	17

Fig. 28. query6

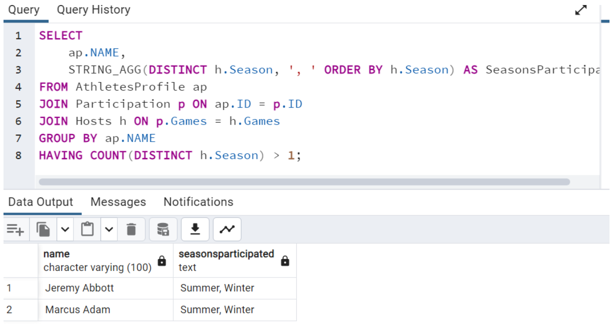


```sql
SELECT
    h.Games,
    AVG(p.Age) AS AverageAge
FROM Medals m
JOIN Participation p ON m.ID = p.ID
JOIN Hosts h ON p.Games = h.Games
WHERE m.Medal IS NOT NULL
GROUP BY h.Games
ORDER BY MIN(h.Year);
```

	games character varying (255)	averageage numeric
1	1908 Summer	23.0000000000000000
2	1912 Summer	24.3333333333333333
3	1920 Summer	20.2222222222222222
4	1924 Summer	24.0000000000000000
5	1924 Winter	23.0000000000000000
6	1948 Summer	28.0000000000000000
7	1952 Summer	32.0000000000000000

Total rows: 31 of 31 Query complete 00:00:00.063

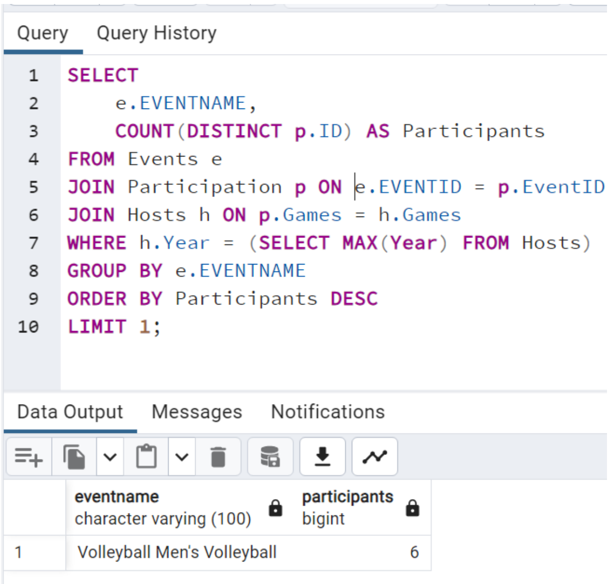
Fig. 29. query7



```
SELECT
    ap.NAME,
    STRING_AGG(DISTINCT h.Season, ', ' ORDER BY h.Season) AS SeasonsParticipa
FROM AthletesProfile ap
JOIN Participation p ON ap.ID = p.ID
JOIN Hosts h ON p.Games = h.Games
GROUP BY ap.NAME
HAVING COUNT(DISTINCT h.Season) > 1;
```

	name character varying (100)	seasonsparticipated text
1	Jeremy Abbott	Summer, Winter
2	Marcus Adam	Summer, Winter

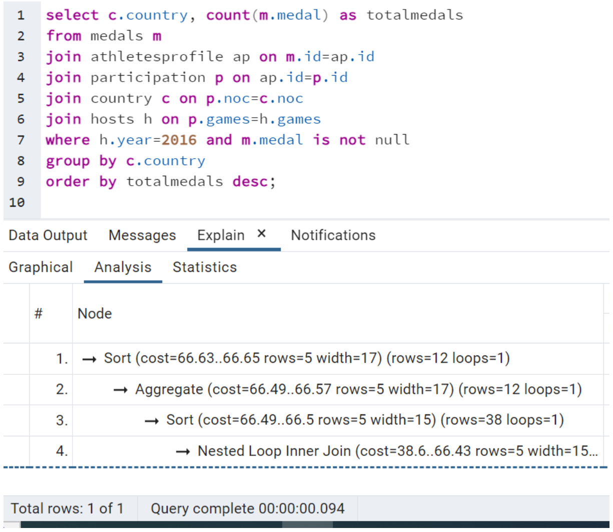
Fig. 30. query8



```
SELECT
    e.EVENTNAME,
    COUNT(DISTINCT p.ID) AS Participants
FROM Events e
JOIN Participation p ON e.EVENTID = p.EventID
JOIN Hosts h ON p.Games = h.Games
WHERE h.Year = (SELECT MAX(Year) FROM Hosts)
GROUP BY e.EVENTNAME
ORDER BY Participants DESC
LIMIT 1;
```

	eventname character varying (100)	participants bigint
1	Volleyball Men's Volleyball	6

Fig. 31. query9

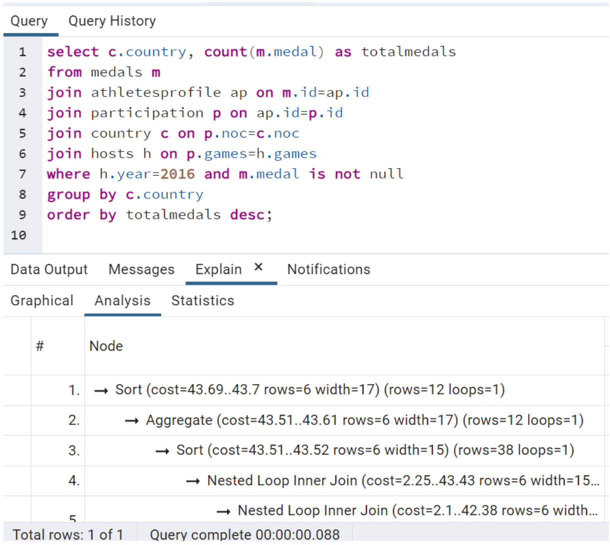


```
select c.country, count(m.medal) as totalmedals
from medals m
join athletesprofile ap on m.id=ap.id
join participation p on ap.id=p.id
join country c on p.noc=c.noc
join hosts h on p.games=h.games
where h.year=2016 and m.medal is not null
group by c.country
order by totalmedals desc;
```

#	Node
1.	→ Sort (cost=66.63..66.65 rows=5 width=17) (rows=12 loops=1)
2.	→ Aggregate (cost=66.49..66.57 rows=5 width=17) (rows=12 loops=1)
3.	→ Sort (cost=66.49..66.5 rows=5 width=15) (rows=38 loops=1)
4.	→ Nested Loop Inner Join (cost=38.6..66.43 rows=5 width=15...

Total rows: 1 of 1 Query complete 00:00:00.094

Fig. 32. query 5 before indexing

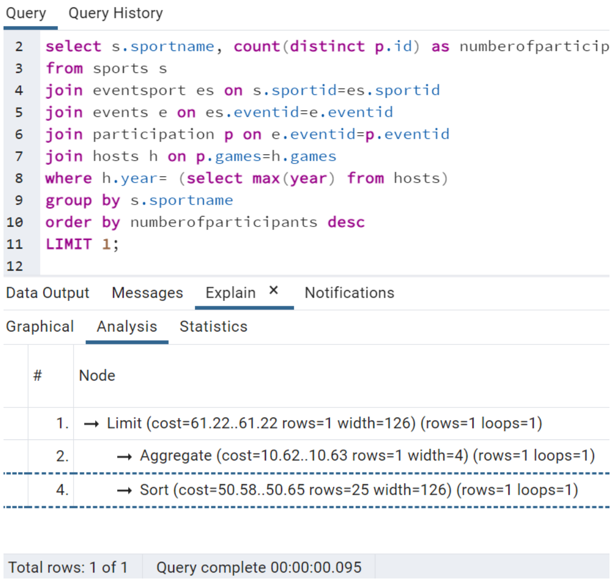


```
select c.country, count(m.medal) as totalmedals
from medals m
join athletesprofile ap on m.id=ap.id
join participation p on ap.id=p.id
join country c on p.noc=c.noc
join hosts h on p.games=h.games
where h.year=2016 and m.medal is not null
group by c.country
order by totalmedals desc;
```

#	Node
1.	→ Sort (cost=43.69..43.7 rows=6 width=17) (rows=12 loops=1)
2.	→ Aggregate (cost=43.51..43.61 rows=6 width=17) (rows=12 loops=1)
3.	→ Sort (cost=43.51..43.52 rows=6 width=15) (rows=38 loops=1)
4.	→ Nested Loop Inner Join (cost=2.25..43.43 rows=6 width=15...
5.	→ Nested Loop Inner Join (cost=2.1..42.38 rows=6 width...

Total rows: 1 of 1 Query complete 00:00:00.088

Fig. 33. query 5 after indexing

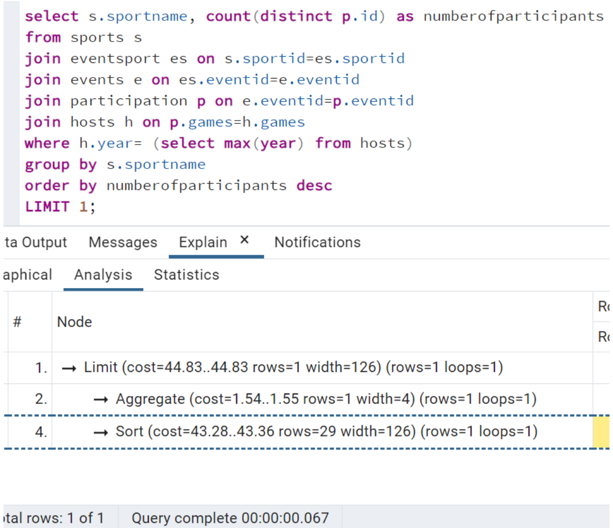


```
select s.sportname, count(distinct p.id) as numberofparticip
from sports s
join eventsport es on s.sportid=es.sportid
join events e on es.eventid=e.eventid
join participation p on e.eventid=p.eventid
join hosts h on p.games=h.games
where h.year= (select max(year) from hosts)
group by s.sportname
order by numberofparticipants desc
LIMIT 1;
```

#	Node
1.	→ Limit (cost=61.22..61.22 rows=1 width=126) (rows=1 loops=1)
2.	→ Aggregate (cost=10.62..10.63 rows=1 width=4) (rows=1 loops=1)
4.	→ Sort (cost=50.58..50.65 rows=25 width=126) (rows=1 loops=1)

Total rows: 1 of 1 Query complete 00:00:00.095

Fig. 34. query 6 before indexing

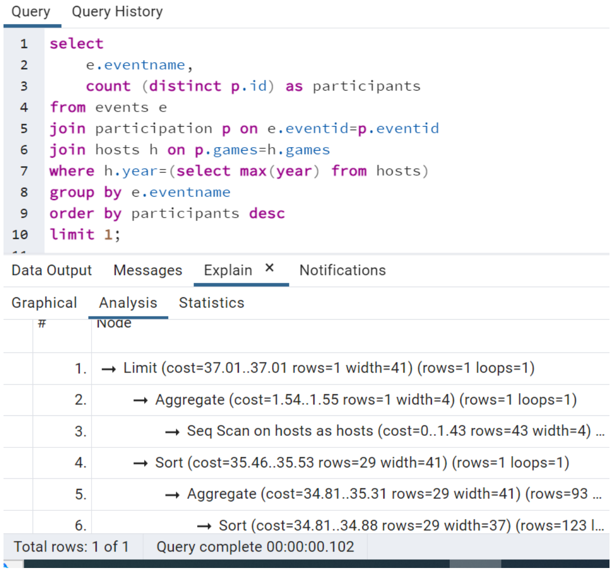


```
select s.sportname, count(distinct p.id) as numberofparticipants
from sports s
join eventsport es on s.sportid=es.sportid
join events e on es.eventid=e.eventid
join participation p on e.eventid=p.eventid
join hosts h on p.games=h.games
where h.year= (select max(year) from hosts)
group by s.sportname
order by numberofparticipants desc
LIMIT 1;
```

#	Node
1.	→ Limit (cost=44.83..44.83 rows=1 width=126) (rows=1 loops=1)
2.	→ Aggregate (cost=1.54..1.55 rows=1 width=4) (rows=1 loops=1)
4.	→ Sort (cost=43.28..43.36 rows=29 width=126) (rows=1 loops=1)

ıtal rows: 1 of 1   Query complete 00:00:00.067

Fig. 35. query 6 after indexing

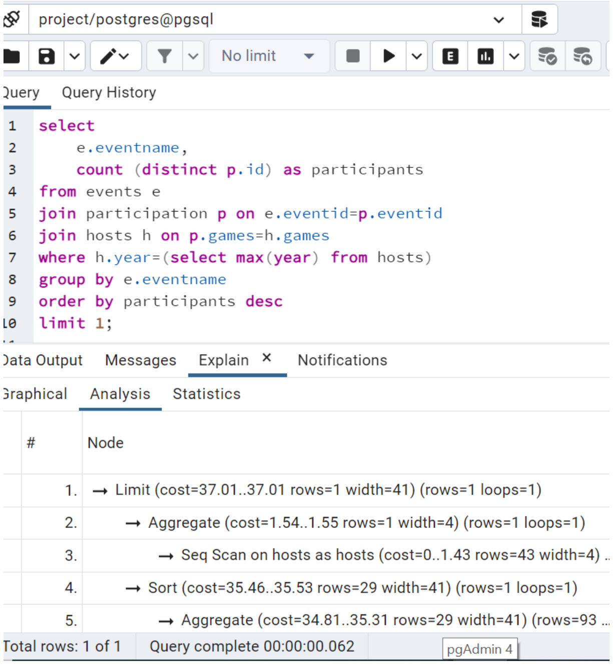


```
select
    e.eventname,
    count (distinct p.id) as participants
from events e
join participation p on e.eventid=p.eventid
join hosts h on p.games=h.games
where h.year=(select max(year) from hosts)
group by e.eventname
order by participants desc
limit 1;
```

#	Node
1.	→ Limit (cost=37.01..37.01 rows=1 width=41) (rows=1 loops=1)
2.	→ Aggregate (cost=1.54..1.55 rows=1 width=4) (rows=1 loops=1)
3.	→ Seq Scan on hosts as hosts (cost=0..1.43 rows=43 width=4) ...
4.	→ Sort (cost=35.46..35.53 rows=29 width=41) (rows=1 loops=1)
5.	→ Aggregate (cost=34.81..35.31 rows=29 width=41) (rows=93 ...
6.	→ Sort (cost=34.81..34.88 rows=29 width=37) (rows=123 l...

Total rows: 1 of 1   Query complete 00:00:00.102

Fig. 36. query 9 before indexing



```
select
    e.eventname,
    count (distinct p.id) as participants
from events e
join participation p on e.eventid=p.eventid
join hosts h on p.games=h.games
where h.year=(select max(year) from hosts)
group by e.eventname
order by participants desc
limit 1;
```

#	Node
1.	→ Limit (cost=37.01..37.01 rows=1 width=41) (rows=1 loops=1)
2.	→ Aggregate (cost=1.54..1.55 rows=1 width=4) (rows=1 loops=1)
3.	→ Seq Scan on hosts as hosts (cost=0..1.43 rows=43 width=4) ..
4.	→ Sort (cost=35.46..35.53 rows=29 width=41) (rows=1 loops=1)
5.	→ Aggregate (cost=34.81..35.31 rows=29 width=41) (rows=93 ..

Total rows: 1 of 1   Query complete 00:00:00.062

Fig. 37. query 9 after indexing