Database Systems Final Assignment Report

Name: Daniil Sergeev

Student ID: 21936856

Practicals Class: Thursday, 10am, 314.Superlab

Introduction

This Assignment required me to design, implement, and expand a database system. The database focused on the 2024 Paris Summer Olympic Games. It contains 7 different tables for a general-purpose database.

Database Design

The reason the following entities were chosen is that they already had their own CSV files on the source website. Most of the data is the same data from the files except the Team entity which has an Increment ID composite primary key as the original primary key got duplicated from 1NF normalization. All the data types are derived from the table and what could be abbreviated like gender fields were shortened to a single character.

Entities and Attributes

Entity Sets	**Primary Keys**	**Other Attributes**
Athlete	Athlete Code (Primary Key)	Gender Name Country Code Country Event Discipline
Event	Event Name (Primary key) Sport Code (Primary key)	Sport Type
Medals	Athlete Code (Primary key)	Medal Type Medal Date Event Type
Team	Team Code (Primary key) Increment ID (Primary Key used to fix multivalued attributes duplicating FKs)	Team Name Discipline Athlete Codes Coach Code Event
Coach	Coach Code (Primary key)	Name Gender Function
Schedule	Day (Primary key) Discipline Code (Primary key) Phase (Primary key) Event (Primary key)	Venue
Technical Officials	Code (Primary key)	Name Gender Function Organisation Country Disciplines

Relationships

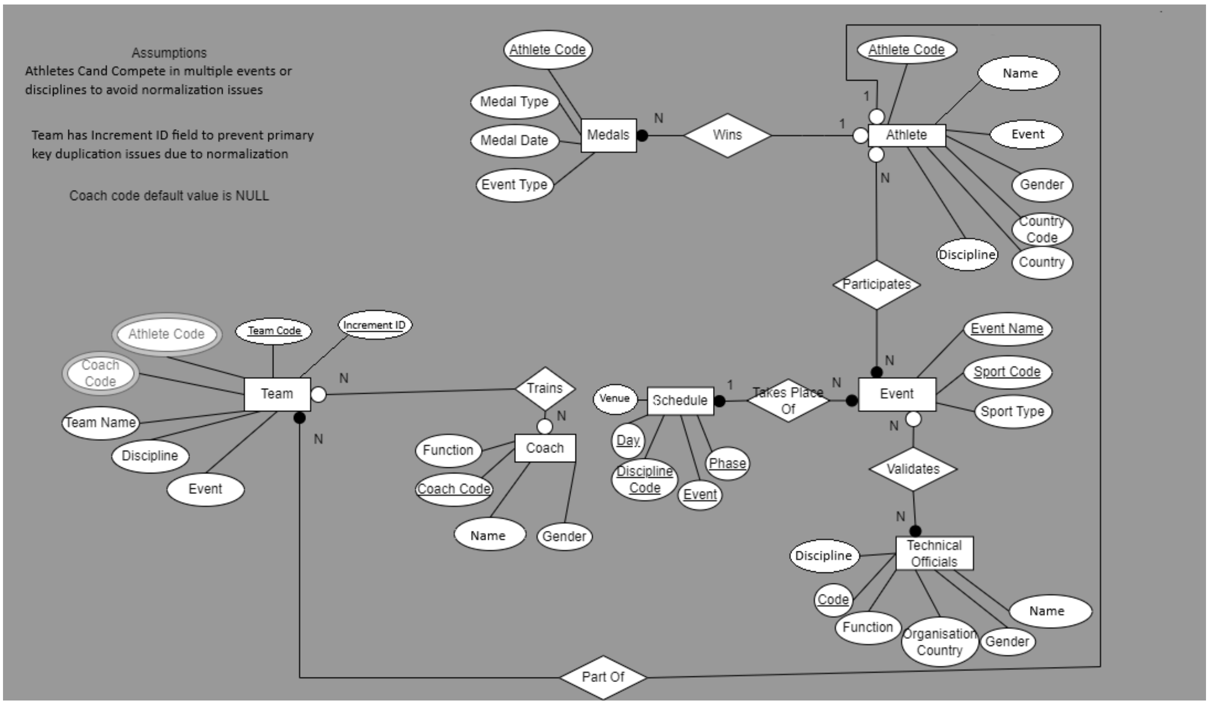
Relationship Sets	**Between Which Entity Sets**	**Attributes of Relationship (if any)**
Participates	Athlete, Event	-
Wins	Athlete, Medals	-
Part of	Athlete, Team	-
Trains	Coach, Team	-
Takes Place of	Schedule, Event	-
Validates	Technical Official, Event	-

Participation and Cardinality Constraints

Relationship Sets	**Cardinality Constraints**	**Participation/ Other Constraints**
Participates	Many-Many (Many athletes may participate in many Events) → Assumption: Group sports have many athletes	Athlete – Partial Event – Total (Athlete can exist without participating in event e.g. injury, an event needs participants to commence)
Wins	One-Many (One Athlete may win many medals)	Athlete – Partial Medals – Total (Athlete can play without winning, Medal needs to be won to be awarded)
Part of	One-Many (One athlete may be part of many teams)	Athlete – Partial Team – Total (An athlete can participate in individual sport, team has to have members to exist)
Trains	Many-Many (Many Coaches per team can train many teams)	Coach – Partial Team – Partial (Coach may exist without coaching any team, some teams don't have any coaches)
Takes Place of	One-Many (Many events can happen during one day)	Event – Total Schedule – Total (Schedule must exist for event to take place and event has to exist to be scheduled
Validates	Many-Many (Many Technical Officials validate many Events)	Technical Official – Total Event – Partial (Technical official has to validate one discipline whilst

		some events don’t have any technical officials)

Entity Relationship (ER) Diagram



Assumptions
Athletes Cand Compete in multiple events or disciplines to avoid normalization issues

Team has Increment ID field to prevent primary key duplication issues due to normalization

Coach code default value is NULL

Relational Schema

Team(Team Code, Increment ID, Athlete Code, Coach Code, Team Name, Discipline, Event)

FK Athlete Code REF Athlete(Athlete Code)

Coach(Coach Code, Name, Gender, Function)

Schedule(Day, Discipline Code, Event, Phase, Venue)

Technical Officials(Code, Name, Disciplines, Function, Organisation Country, Gender)

Event(Event Name, Sport Code, Sport Type, Day, Phase)

FK Day REF Schedule(Day)

FK Sport Code REF Schedule(Discipline Code)

FK Event Name REF Schedule(Event)

FK Phase REF Schedule(Phase)

Athlete(Athlete Code, Name, Event, Gender, Country Code, Country, Discipline)

Medals(Athlete Code, Medal Type, Medal Date, Event Type)

FK Athlete Code REF Athlete(Athlete Code)

Trains(Coach Code, Team Code, Increment ID)

FK Coach Code REF Coach(Coach Code)

FK Team Code REF Team(Team Code)

FK Increment ID REF Team(Increment ID)

Participates(Athlete Code, Event Name, Sport Code)

FK Athlete Code REF Athlete(Athlete Code)

FK Event Name REF Event(Event Name)

FK Sport Code REF Event(Sport Code)

Validates(Code, Event Name, Sport Code)

FK Code REF Technical Officials(Code)

FK Event Name REF Event(Event Name)

FK Sport Code REF Event(Sport Code)

Data Description

Table Name	**Attribute Name**	**Data Type**	**Additional Constraints**	**Description**
Athlete	Athlete Code	INTEGER(7)	PRIMARY KEY, NOT NULL	The main field that Identifies athlete
-	Name	VARCHAR(255)		Can be null because athlete can be identified through code
-	Event	VARCHAR(255)	NOT NULL	Business Rule: Athlete Can only compete in 1 Event to avoid normalization issues
-	Gender	CHAR(1)	NOT NULL	M of F
-	Country Code	CHAR(3)	NOT NULL	3 letter standard country code
-	Country	VARCHAR(64)	NOT NULL	Name of the athlete's country
-	Discipline	VARCHAR(128)	NOT NULL	Business Rule: Athlete Can only compete in 1 Discipline to avoid normalization issues

Table Name	**Attribute Name**	**Data Type**	**Additional Constraints**	**Description**
Medals	Athlete Code	INTEGER(7)	PRIMARY KEY, NOT NULL	Athlete code used as primary key to identify winner
-	Medal Type	VARCHAR(6)	NOT NULL	Gold, Silver or Bronze
-	Medal Date	DATE	NOT NULL	Date the Medal was won
-	Event Type	VARCHAR(5)	NOT NULL	Predefined Code 4-5 letters long

Table Name	**Attribute Name**	**Data Type**	**Additional Constraints**	**Description**
Event	Event Name	VARCHAR(255)	COMPOSITE PRIMARY KEY, NOT NULL	Name of the event forms composite primary key
-	Sport Code	CHAR(3)	COMPOSITE PRIMARY KEY, NOT NULL	Fixed 3 letter sport code paired with Event Name makes for complete primary key
-	Sport Type	VARCHAR(32)	NOT NULL	Name of sport the event takes place in
-	Day	DATE	NOT NULL	Foreign key for Event Date
-	Phase	VARCHAR(256)	NOT NULL	The Other part of the foreign key that completes the Schedules Primary Key, Phase FK makes this field become multivalued

Table Name	**Attribute Name**	**Data Type**	**Additional Constraints**	**Description**
Schedule	Day	DATE	COMPOSITE PRIMARY KEY, NOT NULL	This makes a part of a composite primary key and contains date
-	Discipline Code	CHAR(3)	COMPOSITE PRIMARY KEY, NOT NULL	Fixed 3 letter code on discipline, make up composite primary key
-	Event	VARCHAR(32)	COMPOSITE PRIMARY KEY, NOT NULL	Make up composite primary key
-	Phase	VARCHAR(64)	COMPOSITE PRIMARY KEY, NOT NULL	Same as above
-	Venue	VARCHAR(64)	NOT NULL	Name of location of event

Table Name	Attribute Name	Data Type	Additional Constraints	Description
Technical Officials	Code	INTERGER(7)	PRIMARY KEY, NOT NULL	Number used to identify Technical Officials
-	Disciplines	VARCHAR(255)	NOT NULL	Business Rule: Technical official can only validate 1 discipline to prevent normalization issues
-	Function	VARCHAR(32)	NOT NULL	Predefined variable length title
-	Organisation Country	VARCHAR(64)	NOT NULL	Variable length string used to help get more interesting queries
-	Name	VARCHAR(255)		Can be null as Code is all that's needed to uniquely identify official
-	Gender	CHAR(1)	NOT NULL	M or F

Table Name	Attribute Name	Data Type	Additional Constraints	Description
Team	Team Code	CHAR(17)	PRIMARY KEY, NOT NULL	Unique 17-character long code used to identify teams
-	Increment ID	INTEGER(7)	PRIMARY KEY, NOT NULL, AUTO INCREMENT	Keeps primary key integrity during normalization
-	Athlete Code	INTEGER(7)	MULTIVALUED, NOT NULL	Normalize to 1NF as there can be multiple athletes in 1 team
-	Coach Code	INTEGER(7)	MULTIVALUED	Normalize to 1NF as there can be multiple coaches in 1 team, there can also be no coaches so null is valid
-	Team Name	VARCHAR(128)	NOT NULL	Variable length string for finding team name
-	Discipline	VARCHAR(64)	NOT NULL	The Discipline the team competes in
-	Event	VARCHAR(64)		Can be null

Table Name	Attribute Name	Data Type	Additional Constraints	Description
Coach	Coach Code	INTEGER(7)	PRIMARY KEY, NOT NULL	Unique 7-digit code given to each coach

-	Name	VARCHAR(255)		Can be null as Coach Code Can Identify uniquely
-	Gender	CHAR(1)	NOT NULL	M or F
-	Function	VARCHAR(64)		Predefined String

Assumptions

Default value for coach code in Team table is NULL

Names can be NULL as a code identifies an individual

An athlete can compete in many teams

Technical Official can only validate 1 discipline

Athlete may only be in one discipline

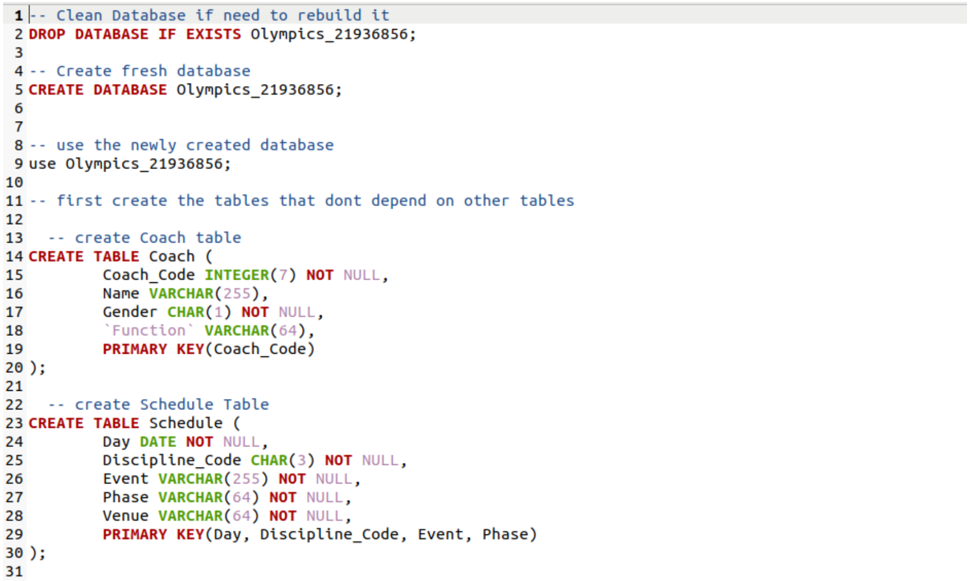
Athlete can only compete in one event to avoid duplicate entries

An athlete can only win one medal

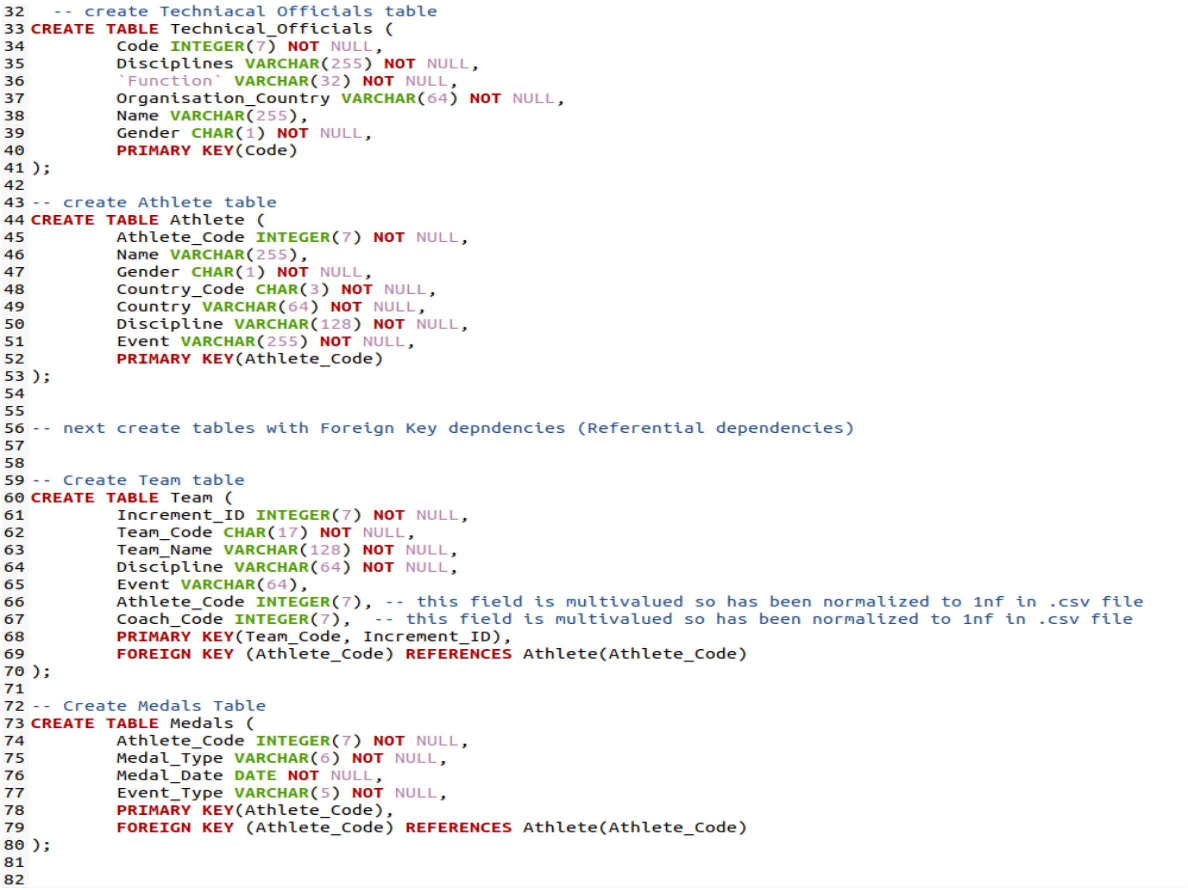
Database Implementation

The database was implemented by first following the relational schema to create an SQL file which creates the database and tables. The CSV files were filtered and analysed using Excel and later simple Python scripts to remove characters that may cause formatting issues. Next, the insert files are generated by Python scripts located in their own directories. All the data came from the given 2024 Paris Olympics website. The data insertion can be done in 2 ways with the recommended method being the automated method which just requires you to run the insert all file.

Below is a file that creates the database and tables

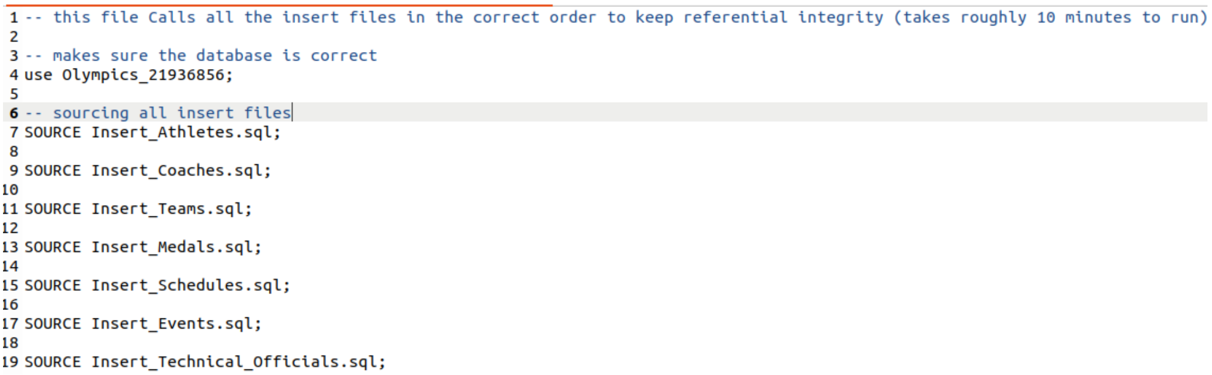


```
-- Clean Database if need to rebuild it
DROP DATABASE IF EXISTS Olympics_21936856;

-- Create fresh database
CREATE DATABASE Olympics_21936856;


-- use the newly created database
use Olympics_21936856;

-- first create the tables that dont depend on other tables

  -- create Coach table
CREATE TABLE Coach (
        Coach_Code INTEGER(7) NOT NULL,
        Name VARCHAR(255),
        Gender CHAR(1) NOT NULL,
        `Function` VARCHAR(64),
        PRIMARY KEY(Coach_Code)
);

  -- create Schedule Table
CREATE TABLE Schedule (
        Day DATE NOT NULL,
        Discipline_Code CHAR(3) NOT NULL,
        Event VARCHAR(255) NOT NULL,
        Phase VARCHAR(64) NOT NULL,
        Venue VARCHAR(64) NOT NULL,
        PRIMARY KEY(Day, Discipline_Code, Event, Phase)
);

```



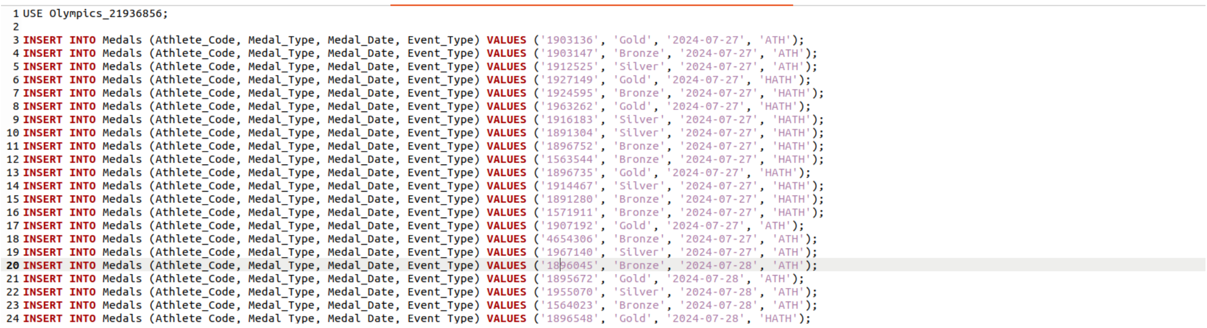
```sql
  -- create Techniacal Officials table
CREATE TABLE Technical_Officials (
        Code INTEGER(7) NOT NULL,
        Disciplines VARCHAR(255) NOT NULL,
        `Function` VARCHAR(32) NOT NULL,
        Organisation_Country VARCHAR(64) NOT NULL,
        Name VARCHAR(255),
        Gender CHAR(1) NOT NULL,
        PRIMARY KEY(Code)
);

-- create Athlete table
CREATE TABLE Athlete (
        Athlete_Code INTEGER(7) NOT NULL,
        Name VARCHAR(255),
        Gender CHAR(1) NOT NULL,
        Country_Code CHAR(3) NOT NULL,
        Country VARCHAR(64) NOT NULL,
        Discipline VARCHAR(128) NOT NULL,
        Event VARCHAR(255) NOT NULL,
        PRIMARY KEY(Athlete_Code)
);


-- next create tables with Foreign Key depndencies (Referential dependencies)


-- Create Team table
CREATE TABLE Team (
        Increment_ID INTEGER(7) NOT NULL,
        Team_Code CHAR(17) NOT NULL,
        Team_Name VARCHAR(128) NOT NULL,
        Discipline VARCHAR(64) NOT NULL,
        Event VARCHAR(64),
        Athlete_Code INTEGER(7), -- this field is multivalued so has been normalized to 1nf in .csv file
        Coach_Code INTEGER(7),  -- this field is multivalued so has been normalized to 1nf in .csv file
        PRIMARY KEY(Team_Code, Increment_ID),
        FOREIGN KEY (Athlete_Code) REFERENCES Athlete(Athlete_Code)
);

-- Create Medals Table
CREATE TABLE Medals (
        Athlete_Code INTEGER(7) NOT NULL,
        Medal_Type VARCHAR(6) NOT NULL,
        Medal_Date DATE NOT NULL,
        Event_Type VARCHAR(5) NOT NULL,
        PRIMARY KEY(Athlete_Code),
        FOREIGN KEY (Athlete_Code) REFERENCES Athlete(Athlete_Code)
);


```

Below is a file that calls all other files to insert data in the correct order

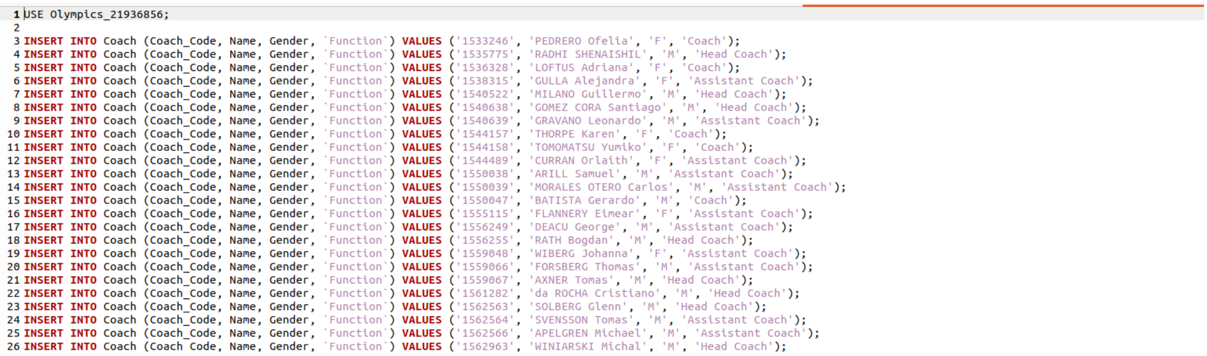


```sql
-- this file Calls all the insert files in the correct order to keep referential integrity (takes roughly 10 minutes to run)

-- makes sure the database is correct
use Olympics_21936856;

-- sourcing all insert files
SOURCE Insert_Athletes.sql;

SOURCE Insert_Coaches.sql;

SOURCE Insert_Teams.sql;

SOURCE Insert_Medals.sql;

SOURCE Insert_Schedules.sql;

SOURCE Insert_Events.sql;

SOURCE Insert_Technical_Officials.sql;
```

Below is the structure present in all insert files



```sql
USE Olympics_21936856;

INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1903136', 'Gold', '2024-07-27', 'ATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1903147', 'Bronze', '2024-07-27', 'ATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1912525', 'Silver', '2024-07-27', 'ATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1927149', 'Gold', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1924595', 'Bronze', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1963262', 'Gold', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1916183', 'Silver', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1891304', 'Silver', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1896752', 'Bronze', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1563544', 'Bronze', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1896735', 'Gold', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1914467', 'Silver', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1891280', 'Bronze', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1571911', 'Bronze', '2024-07-27', 'HATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1907192', 'Gold', '2024-07-27', 'ATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('4654306', 'Bronze', '2024-07-27', 'ATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1967140', 'Silver', '2024-07-27', 'ATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1896045', 'Bronze', '2024-07-28', 'ATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1895672', 'Gold', '2024-07-28', 'ATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1955070', 'Silver', '2024-07-28', 'ATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1564023', 'Bronze', '2024-07-28', 'ATH');
INSERT INTO Medals (Athlete_Code, Medal_Type, Medal_Date, Event_Type) VALUES ('1896548', 'Gold', '2024-07-28', 'HATH');
```

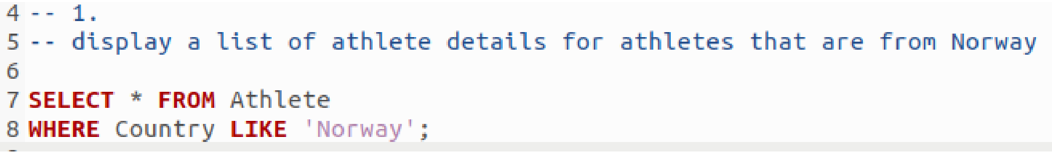


```sql
USE Olympics_21936856;

INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1533246', 'PEDRERO Ofelia', 'F', 'Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1535775', 'RADHI SHENAISHIL', 'M', 'Head Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1536328', 'LOFTUS Adriana', 'F', 'Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1538315', 'GULLA Alejandra', 'F', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1540522', 'MILANO Guillermo', 'M', 'Head Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1540638', 'GOMEZ CORA Santiago', 'M', 'Head Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1540639', 'GRAVANO Leonardo', 'M', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1544157', 'THORPE Karen', 'F', 'Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1544158', 'TOMOMATSU Yumiko', 'F', 'Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1544489', 'CURRAN Orlaith', 'F', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1550038', 'ARILL Samuel', 'M', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1550039', 'MORALES OTERO Carlos', 'M', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1550047', 'BATISTA Gerardo', 'M', 'Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1555115', 'FLANNERY Eimear', 'F', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1556249', 'DEACU George', 'M', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1556255', 'RATH Bogdan', 'M', 'Head Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1559048', 'WIBERG Johanna', 'F', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1559066', 'FORSBERG Thomas', 'M', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1559067', 'AXNER Tomas', 'M', 'Head Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1561282', 'da ROCHA Cristiano', 'M', 'Head Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1562563', 'SOLBERG Glenn', 'M', 'Head Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1562564', 'SVENSSON Tomas', 'M', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1562566', 'APELGREN Michael', 'M', 'Assistant Coach');
INSERT INTO Coach (Coach_Code, Name, Gender, `Function`) VALUES ('1562963', 'WINIARSKI Michal', 'M', 'Head Coach');
```

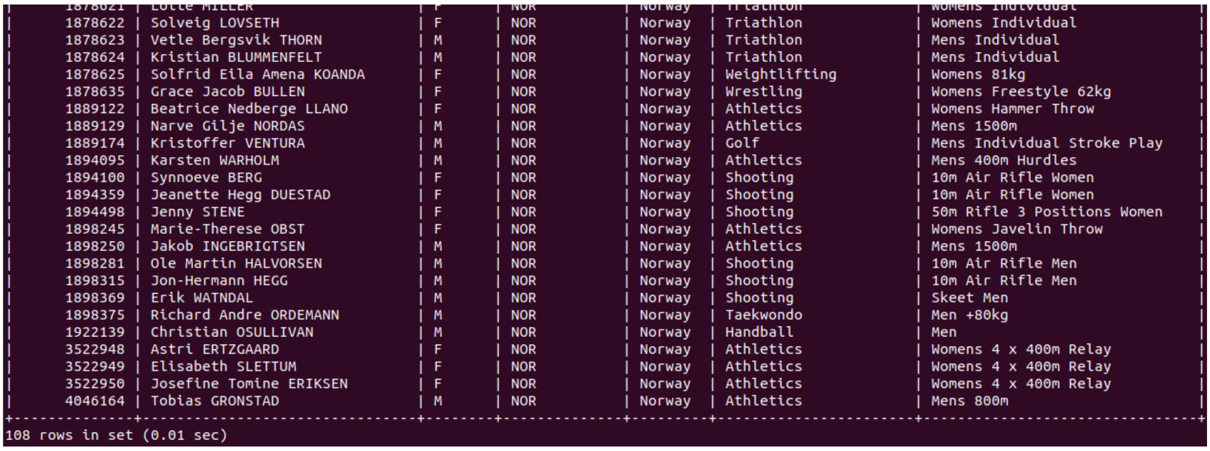
Use Of Database (Queries)

I have implemented a total of 8 queries ranging in complexity with example outputs and explanations.

The first query displays all the details for athletes from Norway. This query is useful because any country name can be used for all countries to see all the athlete details.

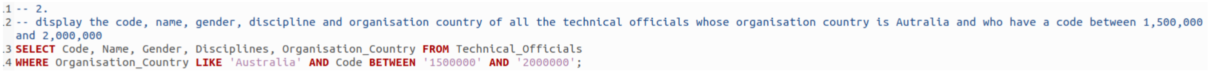


```sql
-- 1.
-- display a list of athlete details for athletes that are from Norway

SELECT * FROM Athlete
WHERE Country LIKE 'Norway';
```

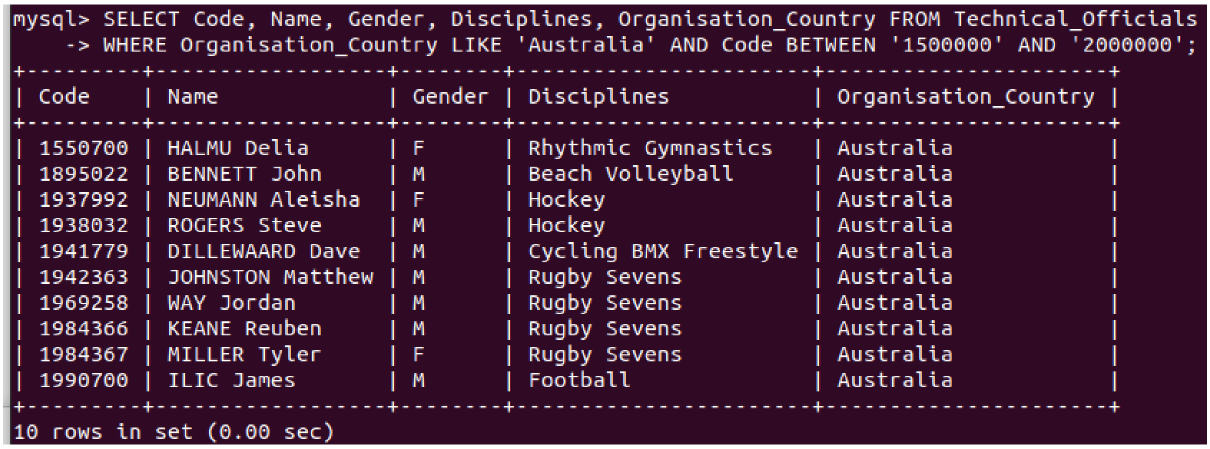


```
|      1878621 | Lotte MILLER                      | F      | NOR          | Norway  | Triathlon           | Womens Individual              |
|      1878622 | Solveig LOVSETH                   | F      | NOR          | Norway  | Triathlon           | Womens Individual              |
|      1878623 | Vetle Bergsvik THORN              | M      | NOR          | Norway  | Triathlon           | Mens Individual                |
|      1878624 | Kristian BLUMMENFELT              | M      | NOR          | Norway  | Triathlon           | Mens Individual                |
|      1878625 | Solfrid Eila Amena KOANDA         | F      | NOR          | Norway  | Weightlifting       | Womens 81kg                    |
|      1878635 | Grace Jacob BULLEN                | F      | NOR          | Norway  | Wrestling           | Womens Freestyle 62kg          |
|      1889122 | Beatrice Nedberge LLANO           | F      | NOR          | Norway  | Athletics           | Womens Hammer Throw            |
|      1889129 | Narve Gilje NORDAS                | M      | NOR          | Norway  | Athletics           | Mens 1500m                     |
|      1889174 | Kristoffer VENTURA                | M      | NOR          | Norway  | Golf                | Mens Individual Stroke Play    |
|      1894095 | Karsten WARHOLM                   | M      | NOR          | Norway  | Athletics           | Mens 400m Hurdles              |
|      1894100 | Synnoeve BERG                     | F      | NOR          | Norway  | Shooting            | 10m Air Rifle Women            |
|      1894359 | Jeanette Hegg DUESTAD             | F      | NOR          | Norway  | Shooting            | 10m Air Rifle Women            |
|      1894498 | Jenny STENE                       | F      | NOR          | Norway  | Shooting            | 50m Rifle 3 Positions Women    |
|      1898245 | Marie-Therese OBST                | F      | NOR          | Norway  | Athletics           | Womens Javelin Throw           |
|      1898250 | Jakob INGEBRIGTSEN                | M      | NOR          | Norway  | Athletics           | Mens 1500m                     |
|      1898281 | Ole Martin HALVORSEN              | M      | NOR          | Norway  | Shooting            | 10m Air Rifle Men              |
|      1898315 | Jon-Hermann HEGG                  | M      | NOR          | Norway  | Shooting            | 10m Air Rifle Men              |
|      1898369 | Erik WATNDAL                      | M      | NOR          | Norway  | Shooting            | Skeet Men                      |
|      1898375 | Richard Andre ORDEMANN            | M      | NOR          | Norway  | Taekwondo           | Men +80kg                      |
|      1922139 | Christian OSULLIVAN               | M      | NOR          | Norway  | Handball            | Men                            |
|      3522948 | Astri ERTZGAARD                   | F      | NOR          | Norway  | Athletics           | Womens 4 x 400m Relay          |
|      3522949 | Elisabeth SLETTUM                 | F      | NOR          | Norway  | Athletics           | Womens 4 x 400m Relay          |
|      3522950 | Josefine Tomine ERIKSEN           | F      | NOR          | Norway  | Athletics           | Womens 4 x 400m Relay          |
|      4046164 | Tobias GRONSTAD                   | M      | NOR          | Norway  | Athletics           | Mens 800m                      |
+-------------+-----------------------------------+--------+--------------+---------+---------------------+--------------------------------+
108 rows in set (0.01 sec)
```

The second query displays all the technical officials from Australia whose coach code is between 1.5 million and 2 million which is useful if you are looking for technical officials between specific values.

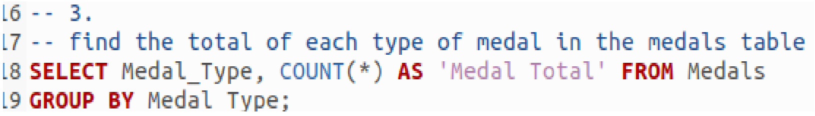


```sql
-- 2.
-- display the code, name, gender, discipline and organisation country of all the technical officials whose organisation country is Autralia and who have a code between 1,500,000
   and 2,000,000
SELECT Code, Name, Gender, Disciplines, Organisation_Country FROM Technical_Officials
WHERE Organisation_Country LIKE 'Australia' AND Code BETWEEN '1500000' AND '2000000';
```

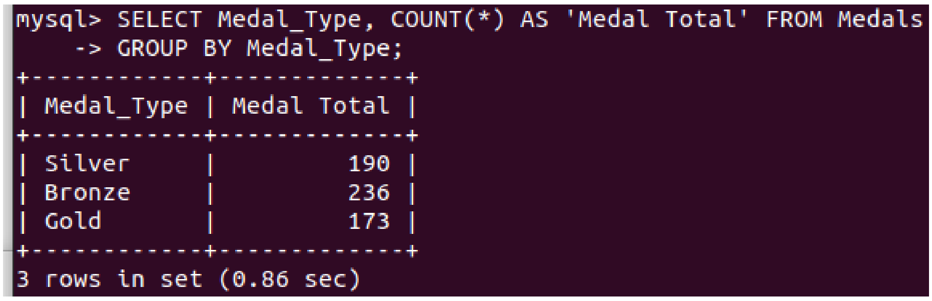


```
mysql> SELECT Code, Name, Gender, Disciplines, Organisation_Country FROM Technical_Officials
    -> WHERE Organisation_Country LIKE 'Australia' AND Code BETWEEN '1500000' AND '2000000';
+---------+------------------+--------+-----------------------+----------------------+
| Code    | Name             | Gender | Disciplines           | Organisation_Country |
+---------+------------------+--------+-----------------------+----------------------+
| 1550700 | HALMU Delia      | F      | Rhythmic Gymnastics   | Australia            |
| 1895022 | BENNETT John     | M      | Beach Volleyball      | Australia            |
| 1937992 | NEUMANN Aleisha  | F      | Hockey                | Australia            |
| 1938032 | ROGERS Steve     | M      | Hockey                | Australia            |
| 1941779 | DILLEWAARD Dave  | M      | Cycling BMX Freestyle | Australia            |
| 1942363 | JOHNSTON Matthew | M      | Rugby Sevens          | Australia            |
| 1969258 | WAY Jordan       | M      | Rugby Sevens          | Australia            |
| 1984366 | KEANE Reuben     | M      | Rugby Sevens          | Australia            |
| 1984367 | MILLER Tyler     | F      | Rugby Sevens          | Australia            |
| 1990700 | ILIC James       | M      | Football              | Australia            |
+---------+------------------+--------+-----------------------+----------------------+
10 rows in set (0.00 sec)
```

The third query displays the count for each medal this is very useful as it can be expanded to show medal count per country.

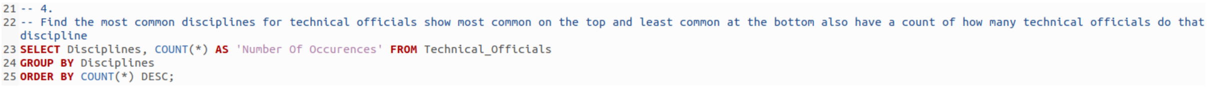


```
-- 3.
-- find the total of each type of medal in the medals table
SELECT Medal_Type, COUNT(*) AS 'Medal Total' FROM Medals
GROUP BY Medal_Type;
```

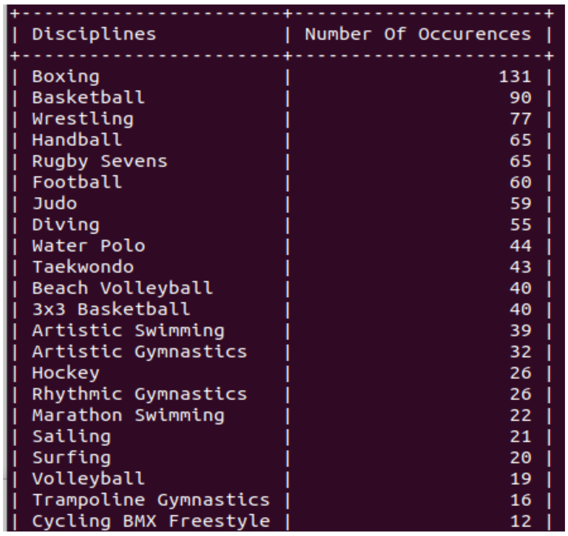


```
mysql> SELECT Medal_Type, COUNT(*) AS 'Medal Total' FROM Medals
    -> GROUP BY Medal_Type;
+------------+-------------+
| Medal_Type | Medal Total |
+------------+-------------+
| Silver     |         190 |
| Bronze     |         236 |
| Gold       |         173 |
+------------+-------------+
3 rows in set (0.86 sec)
```

The fourth query displays the most common disciplines for technical officials. This data is useful as it can be used by hiring agencies to help figure out how many technical officials are needed.

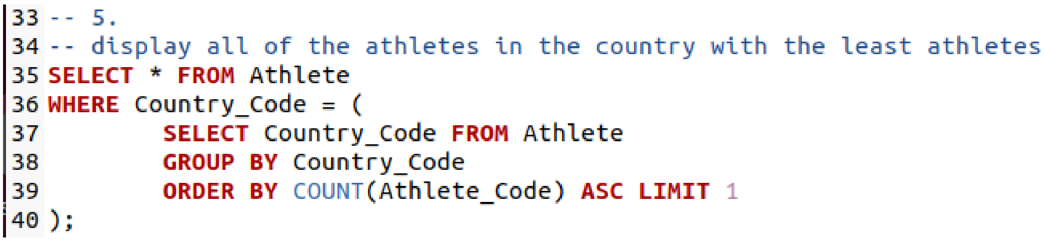


```
-- 4.
-- Find the most common disciplines for technical officials show most common on the top and least common at the bottom also have a count of how many technical officials do that discipline
SELECT Disciplines, COUNT(*) AS 'Number Of Occurences' FROM Technical_Officials
GROUP BY Disciplines
ORDER BY COUNT(*) DESC;
```

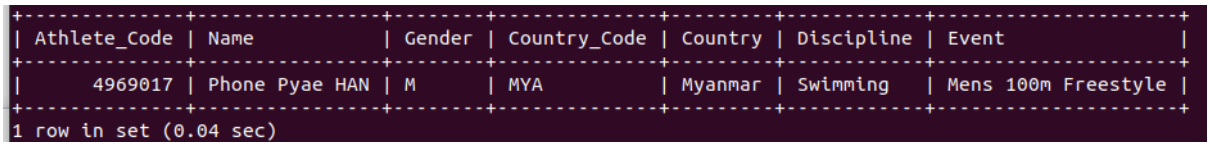


```
+-----------------------+----------------------+
| Disciplines           | Number Of Occurences |
+-----------------------+----------------------+
| Boxing                |                  131 |
| Basketball            |                   90 |
| Wrestling             |                   77 |
| Handball              |                   65 |
| Rugby Sevens          |                   65 |
| Football              |                   60 |
| Judo                  |                   59 |
| Diving                |                   55 |
| Water Polo            |                   44 |
| Taekwondo             |                   43 |
| Beach Volleyball      |                   40 |
| 3x3 Basketball        |                   40 |
| Artistic Swimming     |                   39 |
| Artistic Gymnastics   |                   32 |
| Hockey                |                   26 |
| Rhythmic Gymnastics   |                   26 |
| Marathon Swimming     |                   22 |
| Sailing               |                   21 |
| Surfing               |                   20 |
| Volleyball            |                   19 |
| Trampoline Gymnastics |                   16 |
| Cycling BMX Freestyle |                   12 |
```

Moving onto more complex queries the following example utilises subqueries to show the athletes in the country with the lowest number of athletes.

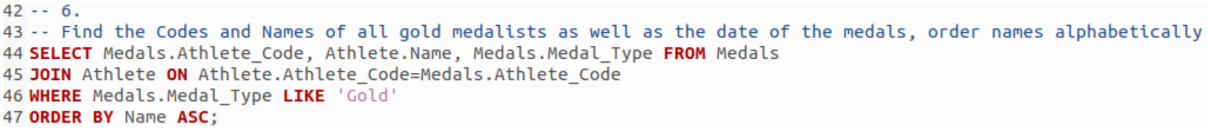


```
-- 5.
-- display all of the athletes in the country with the least athletes
SELECT * FROM Athlete
WHERE Country_Code = (
        SELECT Country_Code FROM Athlete
        GROUP BY Country_Code
        ORDER BY COUNT(Athlete_Code) ASC LIMIT 1
);
```

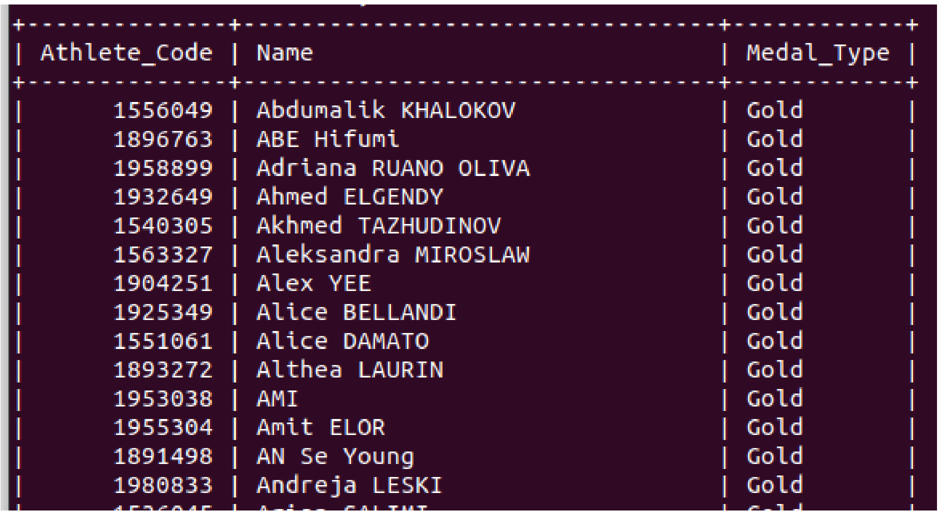


```
+--------------+----------------+--------+--------------+---------+------------+---------------------+
| Athlete_Code | Name           | Gender | Country_Code | Country | Discipline | Event               |
+--------------+----------------+--------+--------------+---------+------------+---------------------+
|      4969017 | Phone Pyae HAN | M      | MYA          | Myanmar | Swimming   | Mens 100m Freestyle |
+--------------+----------------+--------+--------------+---------+------------+---------------------+
1 row in set (0.04 sec)
```

The sixth query utilizes joins to display the names of all gold medallists in alphabetical order. This adds extra useful data into the result that is not otherwise in the medals table. This query can be expanded for all medal types and used for data visualization.

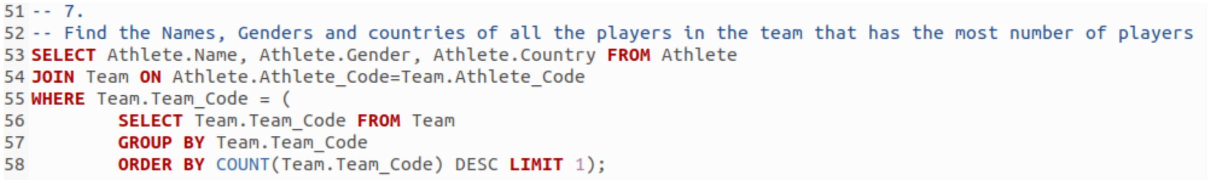


```
-- 6.
-- Find the Codes and Names of all gold medalists as well as the date of the medals, order names alphabetically
SELECT Medals.Athlete_Code, Athlete.Name, Medals.Medal_Type FROM Medals
JOIN Athlete ON Athlete.Athlete_Code=Medals.Athlete_Code
WHERE Medals.Medal_Type LIKE 'Gold'
ORDER BY Name ASC;
```

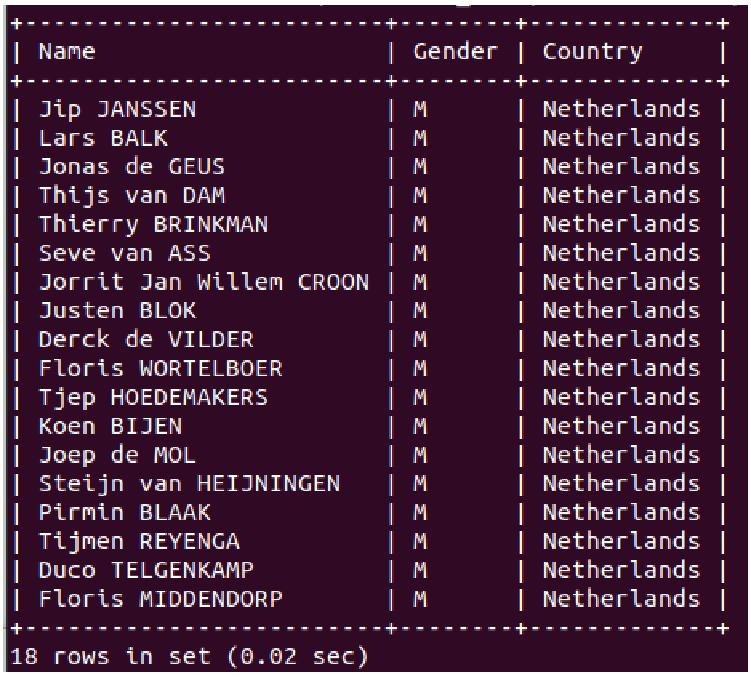


```
+--------------+--------------------------------+------------+
| Athlete_Code | Name                           | Medal_Type |
+--------------+--------------------------------+------------+
|      1556049 | Abdumalik KHALOKOV             | Gold       |
|      1896763 | ABE Hifumi                     | Gold       |
|      1958899 | Adriana RUANO OLIVA            | Gold       |
|      1932649 | Ahmed ELGENDY                  | Gold       |
|      1540305 | Akhmed TAZHUDINOV              | Gold       |
|      1563327 | Aleksandra MIROSLAW            | Gold       |
|      1904251 | Alex YEE                       | Gold       |
|      1925349 | Alice BELLANDI                 | Gold       |
|      1551061 | Alice DAMATO                   | Gold       |
|      1893272 | Althea LAURIN                  | Gold       |
|      1953038 | AMI                            | Gold       |
|      1955304 | Amit ELOR                      | Gold       |
|      1891498 | AN Se Young                    | Gold       |
|      1980833 | Andreja LESKI                  | Gold       |
```

The seventh query uses joins to display the details of athletes in the team that has the largest number of athletes. This can be useful in figuring out where more technical officials may be needed.

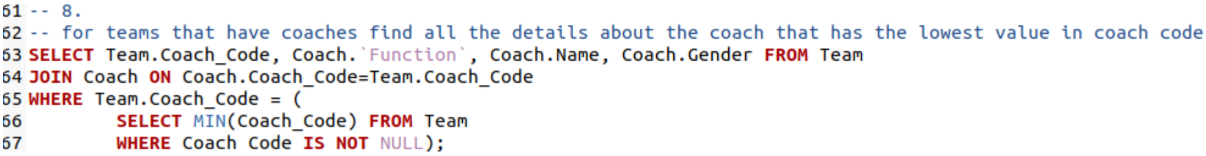


```
-- 7.
-- Find the Names, Genders and countries of all the players in the team that has the most number of players
SELECT Athlete.Name, Athlete.Gender, Athlete.Country FROM Athlete
JOIN Team ON Athlete.Athlete_Code=Team.Athlete_Code
WHERE Team.Team_Code = (
        SELECT Team.Team_Code FROM Team
        GROUP BY Team.Team_Code
        ORDER BY COUNT(Team.Team_Code) DESC LIMIT 1);
```

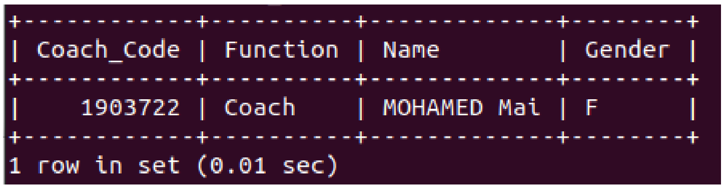


```
+-------------------------+--------+-------------+
| Name                    | Gender | Country     |
+-------------------------+--------+-------------+
| Jip JANSSEN             | M      | Netherlands |
| Lars BALK               | M      | Netherlands |
| Jonas de GEUS           | M      | Netherlands |
| Thijs van DAM           | M      | Netherlands |
| Thierry BRINKMAN        | M      | Netherlands |
| Seve van ASS            | M      | Netherlands |
| Jorrit Jan Willem CROON | M      | Netherlands |
| Justen BLOK             | M      | Netherlands |
| Derck de VILDER         | M      | Netherlands |
| Floris WORTELBOER       | M      | Netherlands |
| Tjep HOEDEMAKERS        | M      | Netherlands |
| Koen BIJEN              | M      | Netherlands |
| Joep de MOL             | M      | Netherlands |
| Steijn van HEIJNINGEN   | M      | Netherlands |
| Pirmin BLAAK            | M      | Netherlands |
| Tijmen REYENGA          | M      | Netherlands |
| Duco TELGENKAMP         | M      | Netherlands |
| Floris MIDDENDORP       | M      | Netherlands |
+-------------------------+--------+-------------+
18 rows in set (0.02 sec)
```

The final query finds the lowest coach code for a technical official who trains a team.



```
-- 8.
-- for teams that have coaches find all the details about the coach that has the lowest value in coach code
SELECT Team.Coach_Code, Coach.`Function`, Coach.Name, Coach.Gender FROM Team
JOIN Coach ON Coach.Coach_Code=Team.Coach_Code
WHERE Team.Coach_Code = (
        SELECT MIN(Coach_Code) FROM Team
        WHERE Coach Code IS NOT NULL);
```

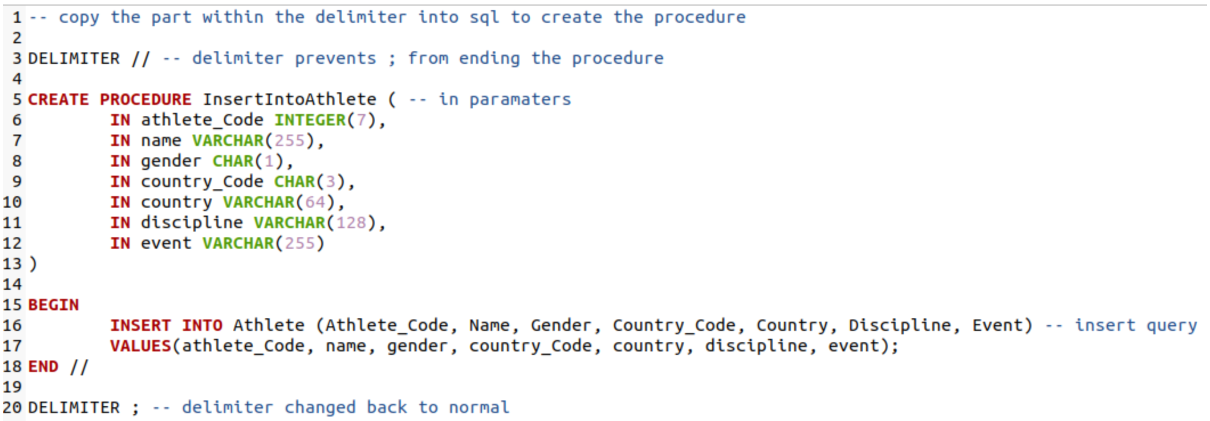


```
+------------+----------+-------------+--------+
| Coach_Code | Function | Name        | Gender |
+------------+----------+-------------+--------+
|    1903722 | Coach    | MOHAMED Mai | F      |
+------------+----------+-------------+--------+
1 row in set (0.01 sec)
```

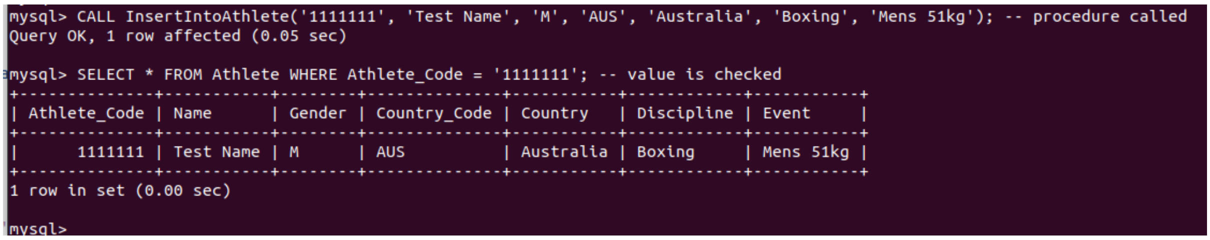
Use of Database (Advanced Features)

When expanding the database I created 2 stored procedures and 2 views.

The first procedure was a simple procedure that allowed for data insertion into the athlete table by utilizing IN parameters as well as a delimiter. This procedure is very useful and can be implemented into an athlete sign-up page on a website with slight modifications such as an auto increment athlete code assignment.

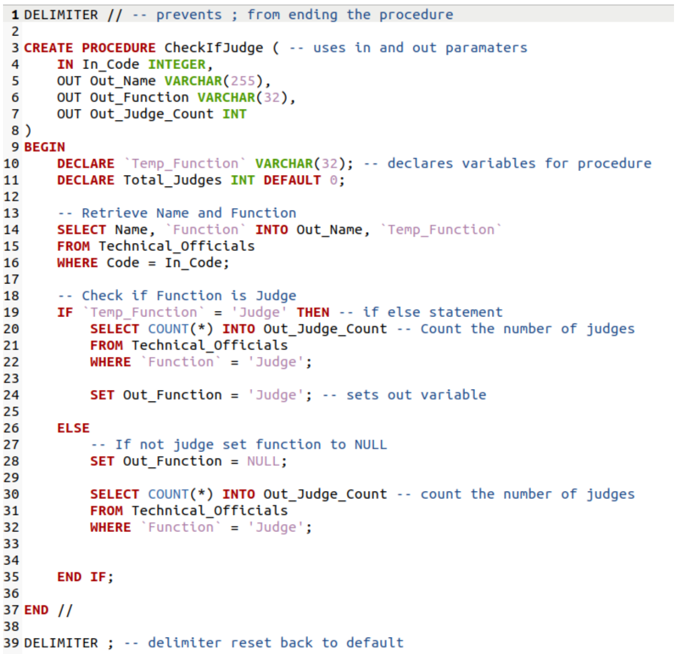


```
-- copy the part within the delimiter into sql to create the procedure

DELIMITER // -- delimiter prevents ; from ending the procedure

CREATE PROCEDURE InsertIntoAthlete ( -- in paramaters
        IN athlete_Code INTEGER(7),
        IN name VARCHAR(255),
        IN gender CHAR(1),
        IN country_Code CHAR(3),
        IN country VARCHAR(64),
        IN discipline VARCHAR(128),
        IN event VARCHAR(255)
)

BEGIN
        INSERT INTO Athlete (Athlete_Code, Name, Gender, Country_Code, Country, Discipline, Event) -- insert query
        VALUES(athlete_Code, name, gender, country_Code, country, discipline, event);
END //

DELIMITER ; -- delimiter changed back to normal
```

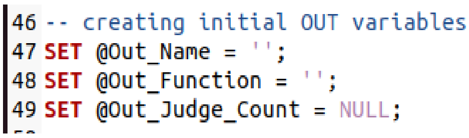


```
mysql> CALL InsertIntoAthlete('1111111', 'Test Name', 'M', 'AUS', 'Australia', 'Boxing', 'Mens 51kg'); -- procedure called
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM Athlete WHERE Athlete_Code = '1111111'; -- value is checked
+--------------+-----------+--------+--------------+-----------+------------+-----------+
| Athlete_Code | Name      | Gender | Country_Code | Country   | Discipline | Event     |
+--------------+-----------+--------+--------------+-----------+------------+-----------+
|      1111111 | Test Name | M      | AUS          | Australia | Boxing     | Mens 51kg |
+--------------+-----------+--------+--------------+-----------+------------+-----------+
1 row in set (0.00 sec)

mysql>
```

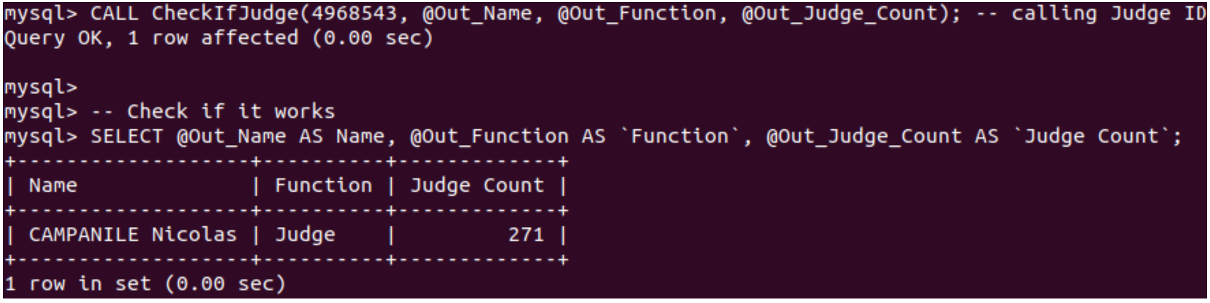
The second stored procedure is much more complex and utilises both IN and OUT parameters, variables, and if else statements. The purpose of this procedure is to take an input of the coach code, check if their function is a judge, and count the total number of judges there are. If they are not a judge the function is then displayed as NULL. This stored procedure also requires for the declaration of the out variables outside the procedure.



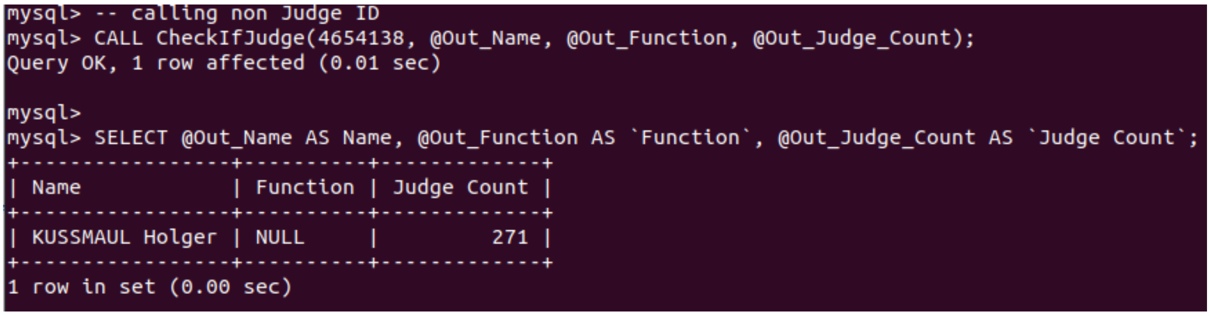
```
DELIMITER // -- prevents ; from ending the procedure

CREATE PROCEDURE CheckIfJudge ( -- uses in and out paramaters
    IN In_Code INTEGER,
    OUT Out_Name VARCHAR(255),
    OUT Out_Function VARCHAR(32),
    OUT Out_Judge_Count INT
)
BEGIN
    DECLARE `Temp_Function` VARCHAR(32); -- declares variables for procedure
    DECLARE Total_Judges INT DEFAULT 0;

    -- Retrieve Name and Function
    SELECT Name, `Function` INTO Out_Name, `Temp_Function`
    FROM Technical_Officials
    WHERE Code = In_Code;

    -- Check if Function is Judge
    IF `Temp_Function` = 'Judge' THEN -- if else statement
        SELECT COUNT(*) INTO Out_Judge_Count -- Count the number of judges
        FROM Technical_Officials
        WHERE `Function` = 'Judge';

        SET Out_Function = 'Judge'; -- sets out variable

    ELSE
        -- If not judge set function to NULL
        SET Out_Function = NULL;

        SELECT COUNT(*) INTO Out_Judge_Count -- count the number of judges
        FROM Technical_Officials
        WHERE `Function` = 'Judge';


    END IF;

END //

DELIMITER ; -- delimiter reset back to default
```



```
-- creating initial OUT variables
SET @Out_Name = '';
SET @Out_Function = '';
SET @Out_Judge_Count = NULL;
```

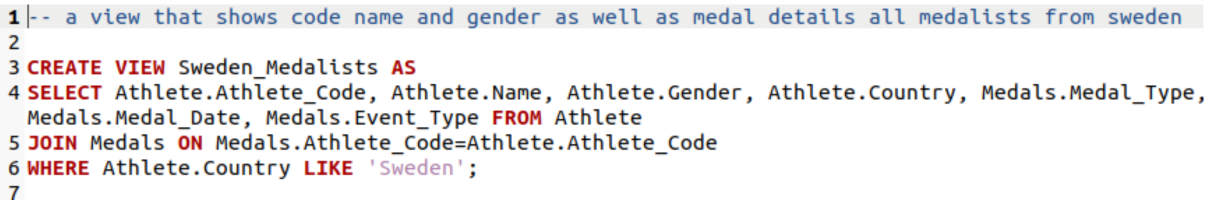


```
mysql> CALL CheckIfJudge(4968543, @Out_Name, @Out_Function, @Out_Judge_Count); -- calling Judge ID
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> -- Check if it works
mysql> SELECT @Out_Name AS Name, @Out_Function AS `Function`, @Out_Judge_Count AS `Judge Count`;
+-------------------+----------+-------------+
| Name              | Function | Judge Count |
+-------------------+----------+-------------+
| CAMPANILE Nicolas | Judge    |         271 |
+-------------------+----------+-------------+
1 row in set (0.00 sec)
```

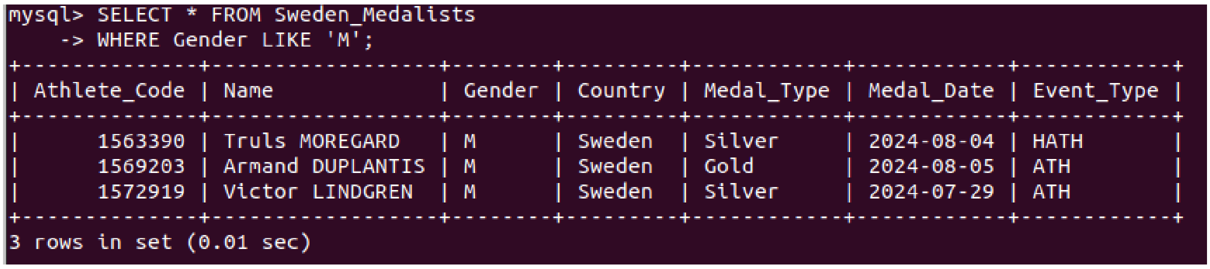


```
mysql> -- calling non Judge ID
mysql> CALL CheckIfJudge(4654138, @Out_Name, @Out_Function, @Out_Judge_Count);
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> SELECT @Out_Name AS Name, @Out_Function AS `Function`, @Out_Judge_Count AS `Judge Count`;
+-----------------+----------+-------------+
| Name            | Function | Judge Count |
+-----------------+----------+-------------+
| KUSSMAUL Holger | NULL     |         271 |
+-----------------+----------+-------------+
1 row in set (0.00 sec)
```

Views allow for simpler queries by hiding complexity. The first view shows details from all medallists from Sweden, and it can be further expanded in more queries.

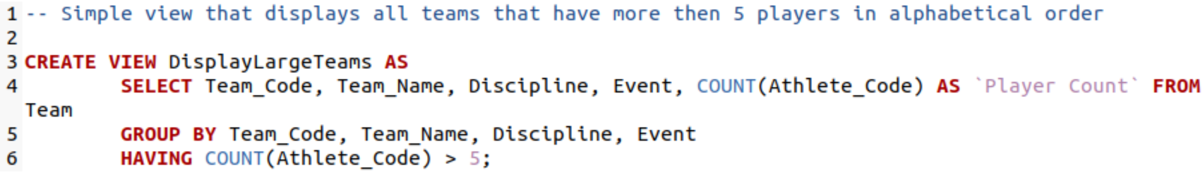


```
-- a view that shows code name and gender as well as medal details all medalists from sweden

CREATE VIEW Sweden_Medalists AS
SELECT Athlete.Athlete_Code, Athlete.Name, Athlete.Gender, Athlete.Country, Medals.Medal_Type,
Medals.Medal_Date, Medals.Event_Type FROM Athlete
JOIN Medals ON Medals.Athlete_Code=Athlete.Athlete_Code
WHERE Athlete.Country LIKE 'Sweden';

```

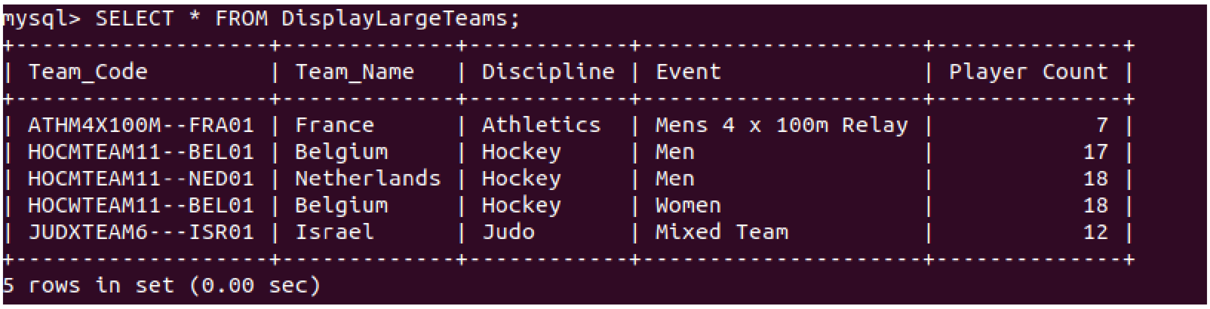


```
mysql> SELECT * FROM Sweden_Medalists
    -> WHERE Gender LIKE 'M';
+--------------+-----------------+--------+---------+------------+------------+------------+
| Athlete_Code | Name            | Gender | Country | Medal_Type | Medal_Date | Event_Type |
+--------------+-----------------+--------+---------+------------+------------+------------+
|      1563390 | Truls MOREGARD  | M      | Sweden  | Silver     | 2024-08-04 | HATH       |
|      1569203 | Armand DUPLANTIS | M     | Sweden  | Gold       | 2024-08-05 | ATH        |
|      1572919 | Victor LINDGREN | M      | Sweden  | Silver     | 2024-07-29 | ATH        |
+--------------+-----------------+--------+---------+------------+------------+------------+
3 rows in set (0.01 sec)
```

The second view displays the number of players in large teams above 5 athletes. This view can be useful when assigning coaches to teams as larger teams may need more coaches.



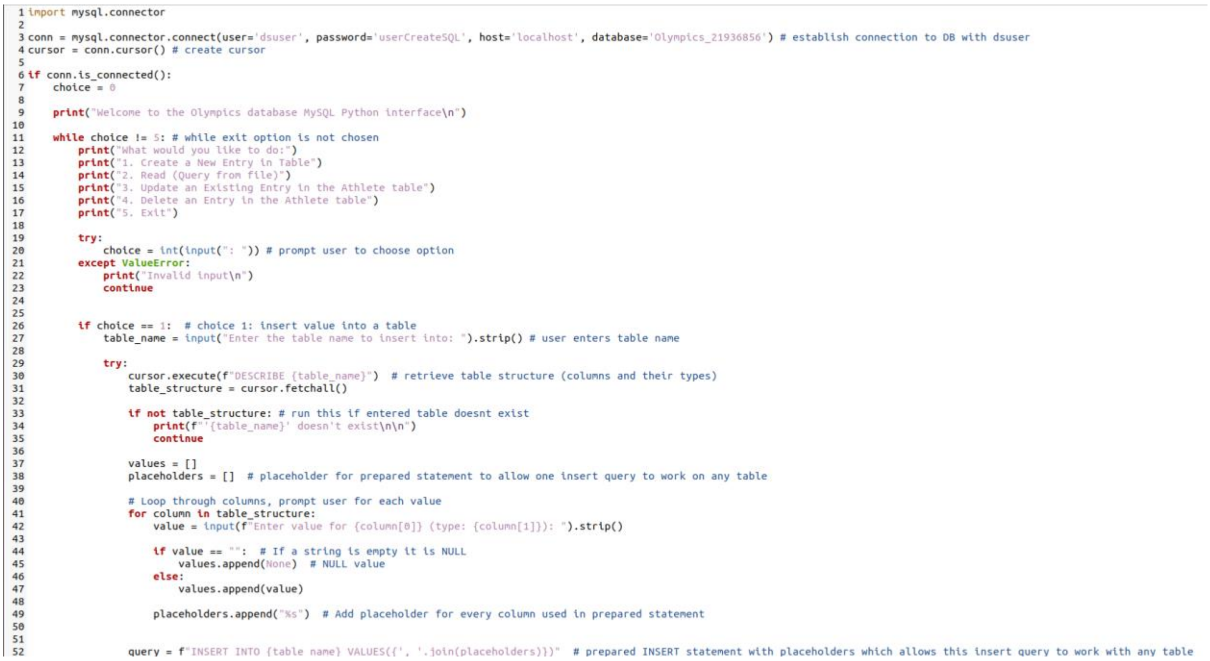
```
-- Simple view that displays all teams that have more then 5 players in alphabetical order

CREATE VIEW DisplayLargeTeams AS
        SELECT Team_Code, Team_Name, Discipline, Event, COUNT(Athlete_Code) AS `Player Count` FROM
Team
        GROUP BY Team_Code, Team_Name, Discipline, Event
        HAVING COUNT(Athlete_Code) > 5;
```



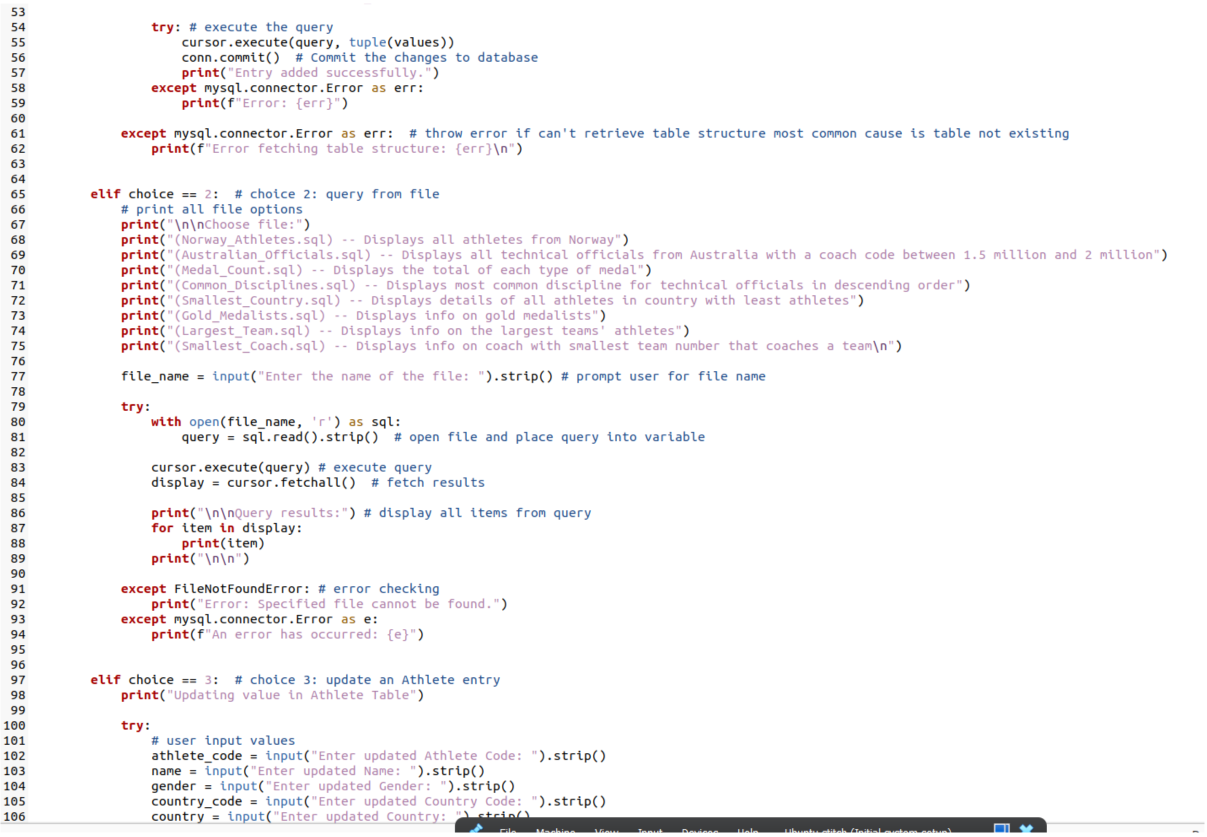
```
mysql> SELECT * FROM DisplayLargeTeams;
+---------------------+-------------+------------+-----------------------+--------------+
| Team_Code           | Team_Name   | Discipline | Event                 | Player Count |
+---------------------+-------------+------------+-----------------------+--------------+
| ATHM4X100M--FRA01   | France      | Athletics  | Mens 4 x 100m Relay   |            7 |
| HOCMTEAM11--BEL01   | Belgium     | Hockey     | Men                   |           17 |
| HOCMTEAM11--NED01   | Netherlands | Hockey     | Men                   |           18 |
| HOCWTEAM11--BEL01   | Belgium     | Hockey     | Women                 |           18 |
| JUDXTEAM6---ISR01   | Israel      | Judo       | Mixed Team            |           12 |
+---------------------+-------------+------------+-----------------------+--------------+
5 rows in set (0.00 sec)
```

Use of Database (Python Connectivity Implementation)

The database connectivity application is a simple CRUD application written in Python which first connects to the database in the same way as shown in lecture 9. A while loop runs in the entire script until the user chooses option 5 to exit. If the user chooses 1 they will be able to insert data into any table. Option 2 reads SQL files that have queries and executes them. Option 3 updates any existing entry in the athlete table and option 4 deletes any existing entry in the athlete table.

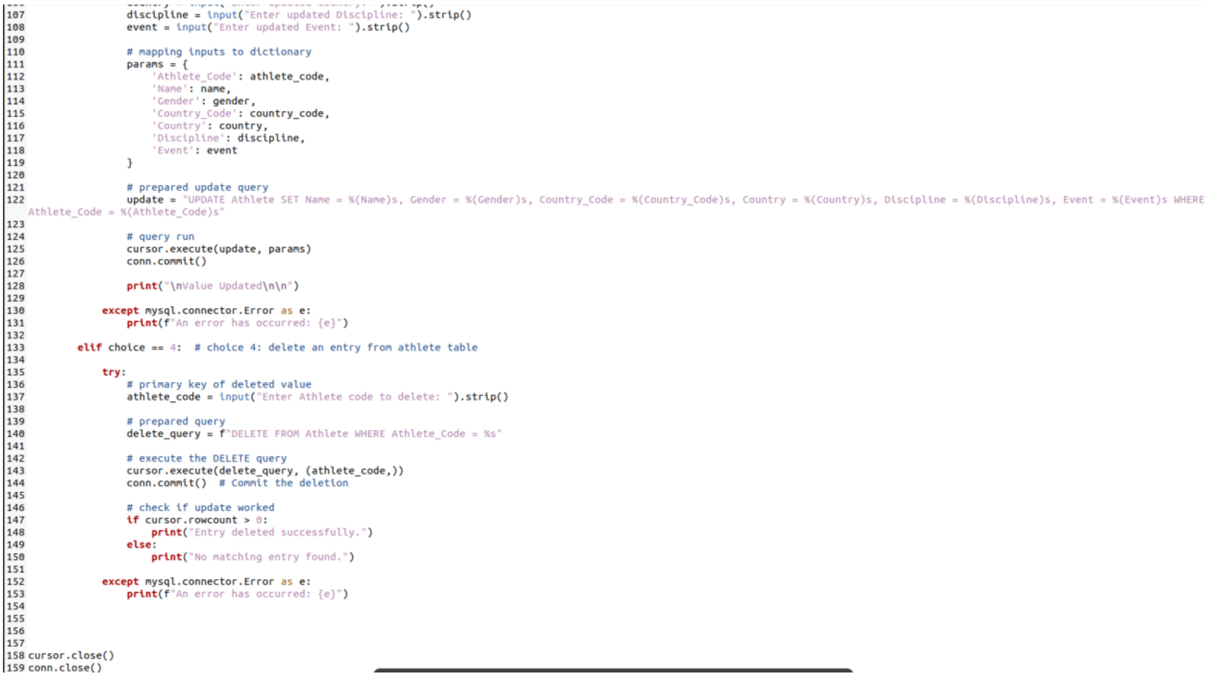


```python
import mysql.connector

conn = mysql.connector.connect(user='dsuser', password='userCreateSQL', host='localhost', database='Olympics_21936856') # establish connection to DB with dsuser
cursor = conn.cursor() # create cursor

if conn.is_connected():
    choice = 0

    print("Welcome to the Olympics database MySQL Python interface\n")

    while choice != 5: # while exit option is not chosen
        print("What would you like to do:")
        print("1. Create a New Entry in Table")
        print("2. Read (Query from file)")
        print("3. Update an Existing Entry in the Athlete table")
        print("4. Delete an Entry in the Athlete table")
        print("5. Exit")

        try:
            choice = int(input(": ")) # prompt user to choose option
        except ValueError:
            print("Invalid input\n")
            continue


        if choice == 1:  # choice 1: insert value into a table
            table_name = input("Enter the table name to insert into: ").strip() # user enters table name

            try:
                cursor.execute(f"DESCRIBE {table_name}")  # retrieve table structure (columns and their types)
                table_structure = cursor.fetchall()

                if not table_structure: # run this if entered table doesnt exist
                    print(f"'{table_name}' doesn't exist\n\n")
                    continue

                values = []
                placeholders = []  # placeholder for prepared statement to allow one insert query to work on any table

                # Loop through columns, prompt user for each value
                for column in table_structure:
                    value = input(f"Enter value for {column[0]} (type: {column[1]}): ").strip()

                    if value == "":  # If a string is empty it is NULL
                        values.append(None)  # NULL value
                    else:
                        values.append(value)

                    placeholders.append("%s")  # Add placeholder for every column used in prepared statement


                query = f"INSERT INTO {table_name} VALUES({', '.join(placeholders)})"  # prepared INSERT statement with placeholders which allows this insert query to work with any table
```



```python

                try: # execute the query
                    cursor.execute(query, tuple(values))
                    conn.commit()  # Commit the changes to database
                    print("Entry added successfully.")
                except mysql.connector.Error as err:
                    print(f"Error: {err}")

            except mysql.connector.Error as err:  # throw error if can't retrieve table structure most common cause is table not existing
                print(f"Error fetching table structure: {err}\n")


        elif choice == 2:  # choice 2: query from file
            # print all file options
            print("\n\nChoose file:")
            print("(Norway_Athletes.sql) -- Displays all athletes from Norway")
            print("(Australian_Officials.sql) -- Displays all technical officials from Australia with a coach code between 1.5 million and 2 million")
            print("(Medal_Count.sql) -- Displays the total of each type of medal")
            print("(Common_Disciplines.sql) -- Displays most common discipline for technical officials in descending order")
            print("(Smallest_Country.sql) -- Displays details of all athletes in country with least athletes")
            print("(Gold_Medalists.sql) -- Displays info on gold medalists")
            print("(Largest_Team.sql) -- Displays info on the largest teams' athletes")
            print("(Smallest_Coach.sql) -- Displays info on coach with smallest team number that coaches a team\n")

            file_name = input("Enter the name of the file: ").strip() # prompt user for file name

            try:
                with open(file_name, 'r') as sql:
                    query = sql.read().strip()  # open file and place query into variable

                cursor.execute(query) # execute query
                display = cursor.fetchall()  # fetch results

                print("\n\nQuery results:") # display all items from query
                for item in display:
                    print(item)
                print("\n\n")

            except FileNotFoundError: # error checking
                print("Error: Specified file cannot be found.")
            except mysql.connector.Error as e:
                print(f"An error has occurred: {e}")


        elif choice == 3:  # choice 3: update an Athlete entry
            print("Updating value in Athlete Table")

            try:
                # user input values
                athlete_code = input("Enter updated Athlete Code: ").strip()
                name = input("Enter updated Name: ").strip()
                gender = input("Enter updated Gender: ").strip()
                country_code = input("Enter updated Country Code: ").strip()
                country = input("Enter updated Country: ").strip()
```

File Machine View Input Devices Help Ubuntu-stitch (Initial custom setup)



```
                discipline = input("Enter updated Discipline: ").strip()
                event = input("Enter updated Event: ").strip()

                # mapping inputs to dictionary
                params = {
                    'Athlete_Code': athlete_code,
                    'Name': name,
                    'Gender': gender,
                    'Country_Code': country_code,
                    'Country': country,
                    'Discipline': discipline,
                    'Event': event
                }

                # prepared update query
                update = "UPDATE Athlete SET Name = %(Name)s, Gender = %(Gender)s, Country_Code = %(Country_Code)s, Country = %(Country)s, Discipline = %(Discipline)s, Event = %(Event)s WHERE Athlete_Code = %(Athlete_Code)s"

                # query run
                cursor.execute(update, params)
                conn.commit()

                print("\nValue Updated\n\n")

            except mysql.connector.Error as e:
                print(f"An error has occurred: {e}")

        elif choice == 4:  # choice 4: delete an entry from athlete table

            try:
                # primary key of deleted value
                athlete_code = input("Enter Athlete code to delete: ").strip()

                # prepared query
                delete_query = f"DELETE FROM Athlete WHERE Athlete_Code = %s"

                # execute the DELETE query
                cursor.execute(delete_query, (athlete_code,))
                conn.commit()  # Commit the deletion

                # check if update worked
                if cursor.rowcount > 0:
                    print("Entry deleted successfully.")
                else:
                    print("No matching entry found.")

            except mysql.connector.Error as e:
                print(f"An error has occurred: {e}")




cursor.close()
conn.close()
```

Discussion

In conclusion, I designed and implemented a database that contains a total of 7 tables. I implemented several queries, stored procedures, and views as well as connecting the database to a Python application to perform CRUD operations. During the designing and building of the database, I faced many issues and problems with not understanding how to use Excel well, so I had to rely on Python to filter some of the more complex data, normalisation was also difficult to do in Excel and I had to use Python again to normalize tables. The CSV data also had invalid characters that would cause issues in readability and in SQL syntax such as ‘’ and []. The final issue I faced was the date not being in the correct format in the csv file which took a long time to fix. One way to improve this in the future would be the Python database application as initially I implemented a way to be able to run any query, but the assignment asked for running the queries we already made which made me change the implementation. Modifying the update and delete any row from any table is another way to improve this, however, I had issues with this part, so I stayed with one table.