Database Systems Assignment

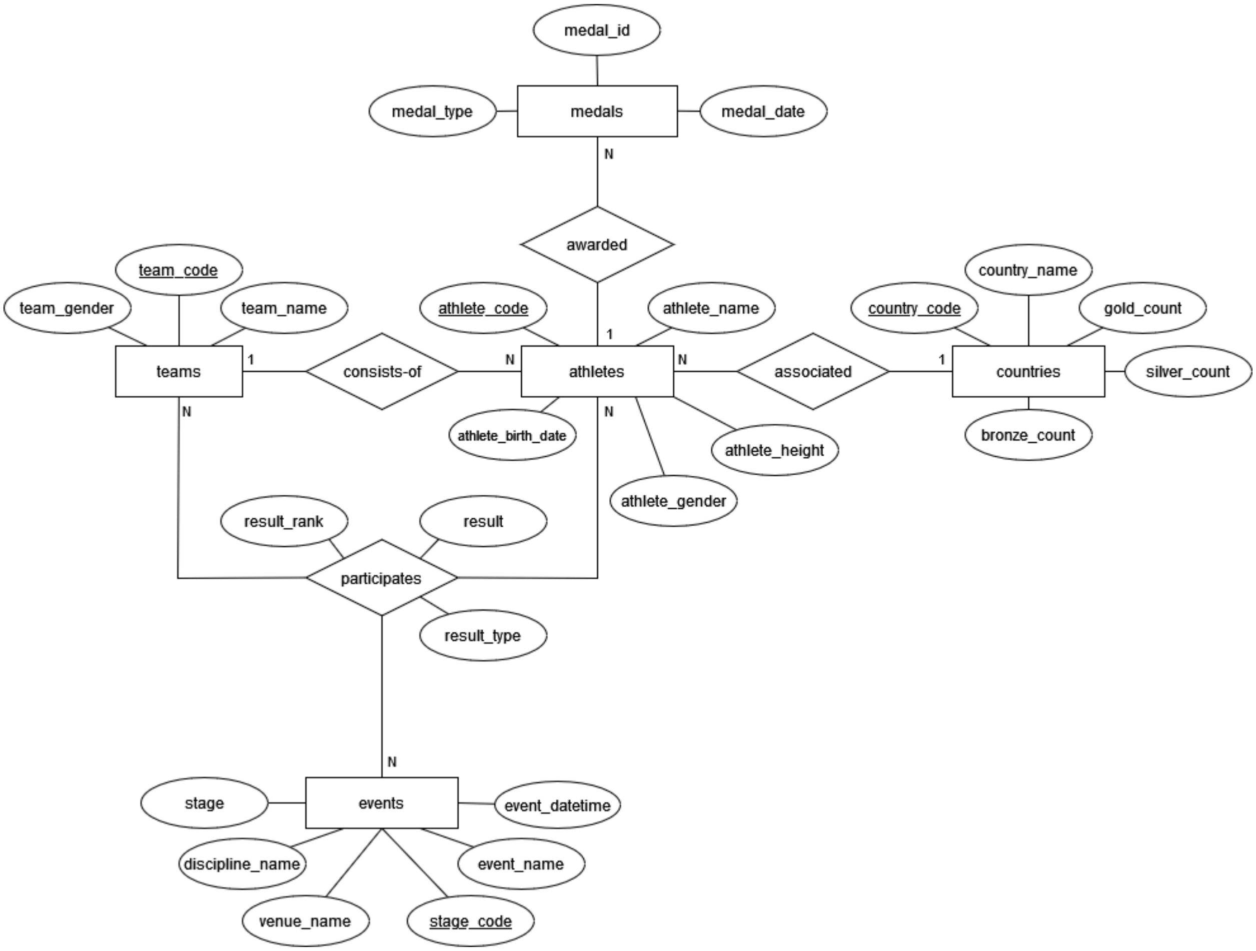
Olympics 2024 Data

Flynn Leveridge (21425477)

Wednesday 2-4pm Lab

Database Design

ER Diagram



Data Description

See tables.sql in the sql_scripts folder, remaking the tables in this document would just make it huge and also achieve nothing more than just looking at the SQL. The tables are all 3NF.

Relational Schema

Relationship Set	Entities	Cardinality
associated	countries, athletes	One-many (athlete belongs to one country and a country has many athletes
awarded	athletes, medals	One-many (a medal belongs to only one athlete but an athlete can have many medals)
participates	athletes, teams, events	Many-Many-Many (many athletes belong to many teams participate in many events and so on)
consists-of	teams, athletes	One-many (an athlete can belong to one team and a team can have many athletes)

Relationship Set	Entities	Participation
associated	countries, athletes	partial, total (an athlete in the olympics has to be from a country, country can have no athletes)
awarded	athletes, medals	partial, total (a medal has to belong to an athlete, but an athlete can have no medals)
participates	athletes, teams, events	partial, partial, partial (all entities can exist separately of this relationship)
consists-of	teams, athletes	total, partial (a team cannot exist without athletes, but athletes can exist without a team)

Database Implementation

Data Source

The data was retrieved from the supplied Kaggle link.

- athletes.csv is identical to the supplied athletes.csv except redundant columns have been removed
- countries.csv is identical to the supplied nocs.csv except redundant columns have been removed
- events.csv is all the individual sport result files from the results folder in supplied data, but it has been joined into one large file with all non-event related columns removed
- individual_participants.csv and team_participants.csv are also from the sport result files in the supplied data, where all team or individual participant rows have been separated into their respective files and all redundant columns have been removed
- medals.csv is identical to the supplied medallists.csv but redundant columns have been removed
- teams.csv is identical to the supplied teams.csv but redundant columns have been removed

Implementation Methods

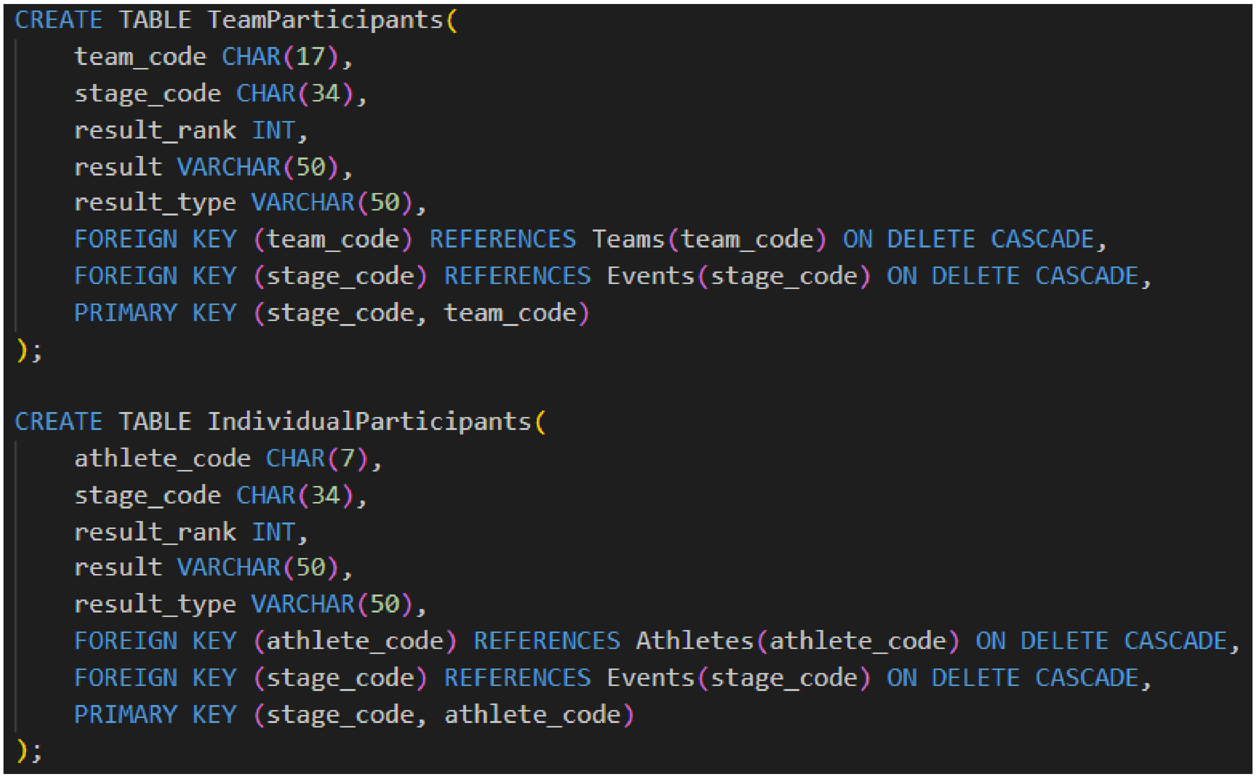
The following tables were implemented in SQL:

- Countries (countries entity from ER diagram)
- Teams (teams entity from ER diagram)
- Events (events entity from ER diagram)
- Athletes (athletes entity from ER diagram)
- Medals (medals entity from ER diagram)
- TeamParticipants (half of the implementation of participants relation from ER diagram)
- IndividualParticipants (half of the implementation of participants relation from ER diagram)

The Countries and Athletes table implement the ‘associated’ relation by having each Athletes entry contain a foreign key to a Countries entry.

The Athletes and Medals table implement the ‘awarded’ relation by having each Medals entry contain a foreign key to an Athletes entry, along with generating a unique id for each medal.

The IndividualParticipants and TeamParticipants tables implement the ternary relation between Athletes, Teams and Events. The reason for two tables instead of one was to more correctly implement primary keys for the entries, as one table caused situations where the primary key consists of three foreign keys of which could be NULL.

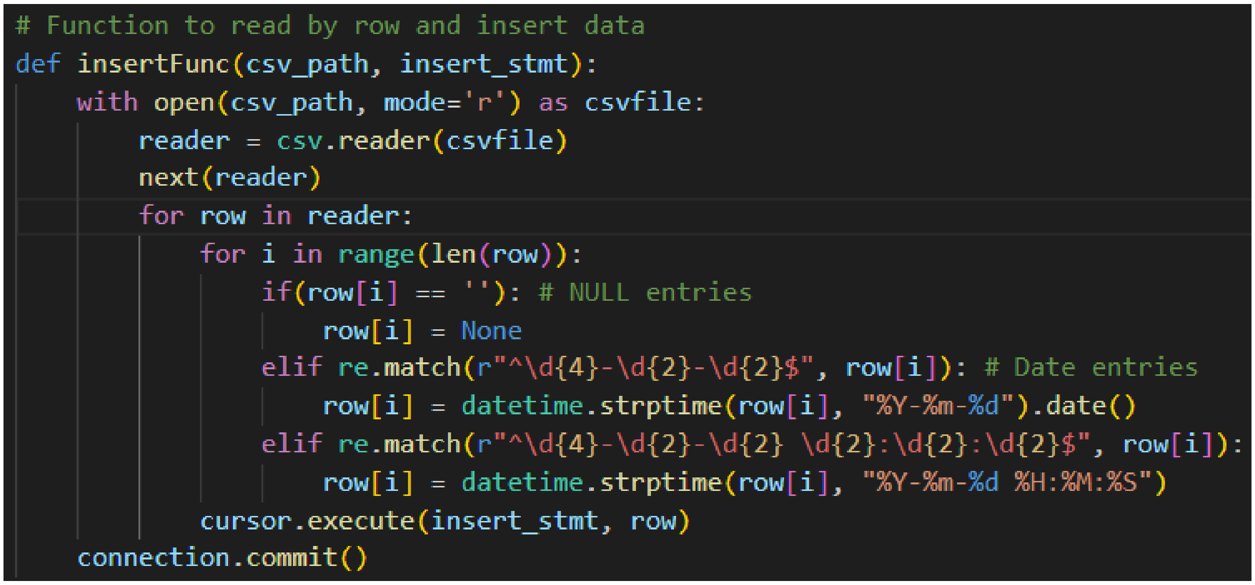


```sql
CREATE TABLE TeamParticipants(
    team_code CHAR(17),
    stage_code CHAR(34),
    result_rank INT,
    result VARCHAR(50),
    result_type VARCHAR(50),
    FOREIGN KEY (team_code) REFERENCES Teams(team_code) ON DELETE CASCADE,
    FOREIGN KEY (stage_code) REFERENCES Events(stage_code) ON DELETE CASCADE,
    PRIMARY KEY (stage_code, team_code)
);

CREATE TABLE IndividualParticipants(
    athlete_code CHAR(7),
    stage_code CHAR(34),
    result_rank INT,
    result VARCHAR(50),
    result_type VARCHAR(50),
    FOREIGN KEY (athlete_code) REFERENCES Athletes(athlete_code) ON DELETE CASCADE,
    FOREIGN KEY (stage_code) REFERENCES Events(stage_code) ON DELETE CASCADE,
    PRIMARY KEY (stage_code, athlete_code)
);
```

Data Insertion Methods

The data files were inserted into the database through the use of a python script (insert_data.py).



```python
# Function to read by row and insert data
def insertFunc(csv_path, insert_stmt):
    with open(csv_path, mode='r') as csvfile:
        reader = csv.reader(csvfile)
        next(reader)
        for row in reader:
            for i in range(len(row)):
                if(row[i] == ''): # NULL entries
                    row[i] = None
                elif re.match(r"^\d{4}-\d{2}-\d{2}$", row[i]): # Date entries
                    row[i] = datetime.strptime(row[i], "%Y-%m-%d").date()
                elif re.match(r"^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}$", row[i]):
                    row[i] = datetime.strptime(row[i], "%Y-%m-%d %H:%M:%S")
            cursor.execute(insert_stmt, row)
    connection.commit()
```

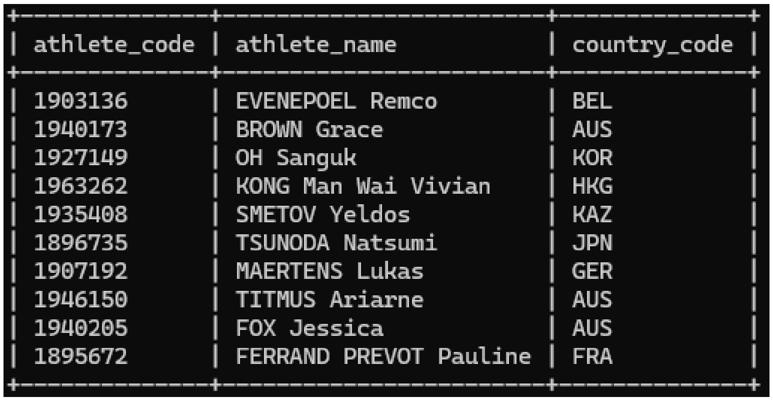
This function is called for each csv file with the file path to be inserted and the matching SQL insert statement.

Use of Database

Queries

Find all athletes that achieved a gold medal

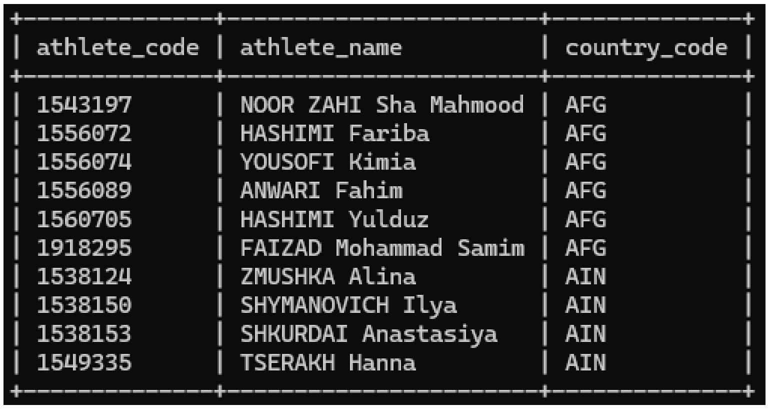
```
SELECT Athletes.athlete_code, athlete_name, country_code FROM Athletes
JOIN Medals ON Athletes.athlete_code = Medals.athlete_code
WHERE Medals.medal_type = 'Gold Medal';
```



```
+--------------+------------------------+--------------+
| athlete_code | athlete_name           | country_code |
+--------------+------------------------+--------------+
| 1903136      | EVENEPOEL Remco        | BEL          |
| 1940173      | BROWN Grace            | AUS          |
| 1927149      | OH Sanguk              | KOR          |
| 1963262      | KONG Man Wai Vivian    | HKG          |
| 1935408      | SMETOV Yeldos          | KAZ          |
| 1896735      | TSUNODA Natsumi        | JPN          |
| 1907192      | MAERTENS Lukas         | GER          |
| 1946150      | TITMUS Ariarne         | AUS          |
| 1940205      | FOX Jessica            | AUS          |
| 1895672      | FERRAND PREVOT Pauline | FRA          |
+--------------+------------------------+--------------+
```

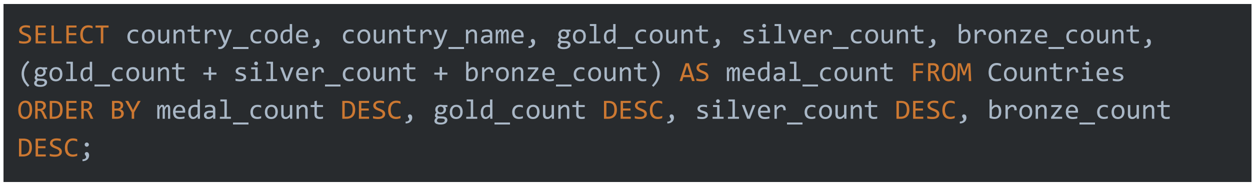
Find all athletes that didn't win any medals

```
SELECT athlete_code, athlete_name, country_code FROM Athletes
WHERE athlete_code NOT IN (SELECT athlete_code FROM Medals)
ORDER BY country_code;
```

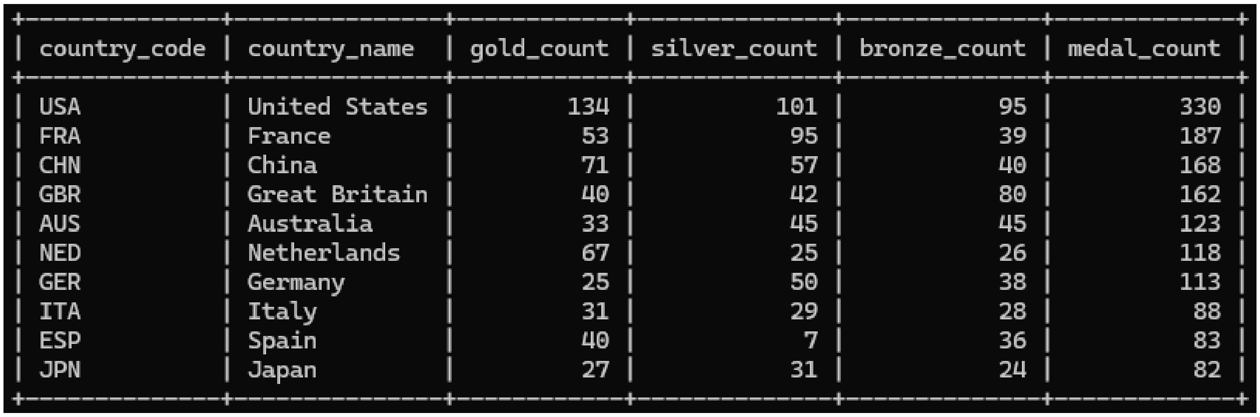


```
+--------------+-----------------------+--------------+
| athlete_code | athlete_name          | country_code |
+--------------+-----------------------+--------------+
| 1543197      | NOOR ZAHI Sha Mahmood | AFG          |
| 1556072      | HASHIMI Fariba        | AFG          |
| 1556074      | YOUSOFI Kimia         | AFG          |
| 1556089      | ANWARI Fahim          | AFG          |
| 1560705      | HASHIMI Yulduz        | AFG          |
| 1918295      | FAIZAD Mohammad Samim | AFG          |
| 1538124      | ZMUSHKA Alina         | AIN          |
| 1538150      | SHYMANOVICH Ilya      | AIN          |
| 1538153      | SHKURDAI Anastasiya   | AIN          |
| 1549335      | TSERAKH Hanna         | AIN          |
+--------------+-----------------------+--------------+
```

List all countries in descending order of medals achieved (with priorities to medals)

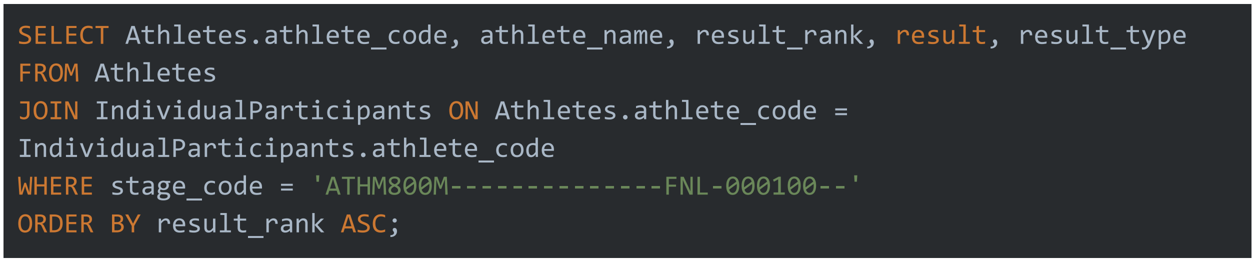


```
SELECT country_code, country_name, gold_count, silver_count, bronze_count,
(gold_count + silver_count + bronze_count) AS medal_count FROM Countries
ORDER BY medal_count DESC, gold_count DESC, silver_count DESC, bronze_count
DESC;
```

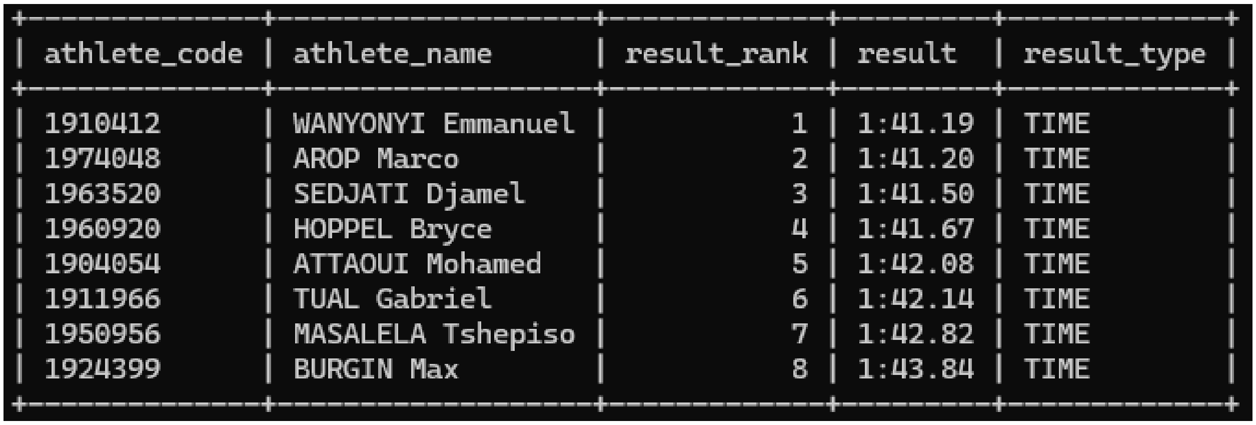


```
+--------------+---------------+------------+--------------+--------------+-------------+
| country_code | country_name  | gold_count | silver_count | bronze_count | medal_count |
+--------------+---------------+------------+--------------+--------------+-------------+
| USA          | United States |        134 |          101 |           95 |         330 |
| FRA          | France        |         53 |           95 |           39 |         187 |
| CHN          | China         |         71 |           57 |           40 |         168 |
| GBR          | Great Britain |         40 |           42 |           80 |         162 |
| AUS          | Australia     |         33 |           45 |           45 |         123 |
| NED          | Netherlands   |         67 |           25 |           26 |         118 |
| GER          | Germany       |         25 |           50 |           38 |         113 |
| ITA          | Italy         |         31 |           29 |           28 |          88 |
| ESP          | Spain         |         40 |            7 |           36 |          83 |
| JPN          | Japan         |         27 |           31 |           24 |          82 |
+--------------+---------------+------------+--------------+--------------+-------------+
```

Obtain results of all athletes competing in the Men’s 800m Final in ascending rank



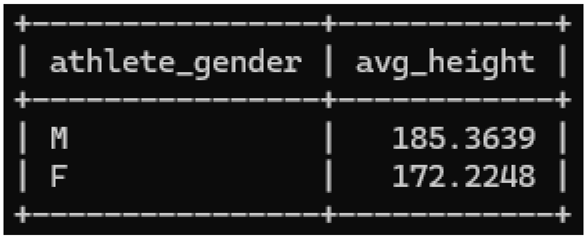
```
SELECT Athletes.athlete_code, athlete_name, result_rank, result, result_type
FROM Athletes
JOIN IndividualParticipants ON Athletes.athlete_code =
IndividualParticipants.athlete_code
WHERE stage_code = 'ATHM800M--------------FNL-000100--'
ORDER BY result_rank ASC;
```



```
+--------------+-------------------+-------------+---------+-------------+
| athlete_code | athlete_name      | result_rank | result  | result_type |
+--------------+-------------------+-------------+---------+-------------+
| 1910412      | WANYONYI Emmanuel |           1 | 1:41.19 | TIME        |
| 1974048      | AROP Marco        |           2 | 1:41.20 | TIME        |
| 1963520      | SEDJATI Djamel    |           3 | 1:41.50 | TIME        |
| 1960920      | HOPPEL Bryce      |           4 | 1:41.67 | TIME        |
| 1904054      | ATTAOUI Mohamed   |           5 | 1:42.08 | TIME        |
| 1911966      | TUAL Gabriel      |           6 | 1:42.14 | TIME        |
| 1950956      | MASALELA Tshepiso |           7 | 1:42.82 | TIME        |
| 1924399      | BURGIN Max        |           8 | 1:43.84 | TIME        |
+--------------+-------------------+-------------+---------+-------------+
```

Obtain average height of all athletes grouped by gender

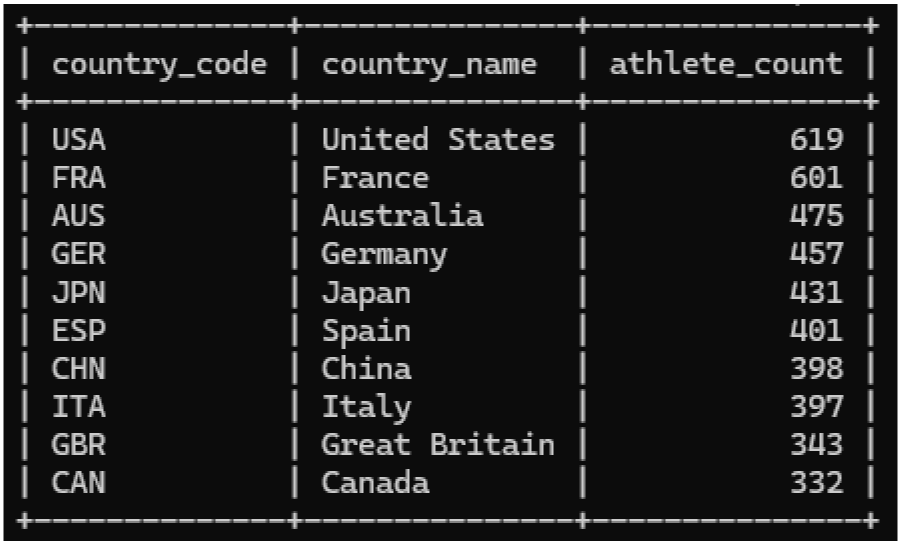
```
SELECT athlete_gender, AVG(athlete_height) as avg_height
FROM Athletes
GROUP BY athlete_gender;
```



```
+----------------+------------+
| athlete_gender | avg_height |
+----------------+------------+
| M              |   185.3639 |
| F              |   172.2248 |
+----------------+------------+
```

Obtain the number of athletes from each country in descending order

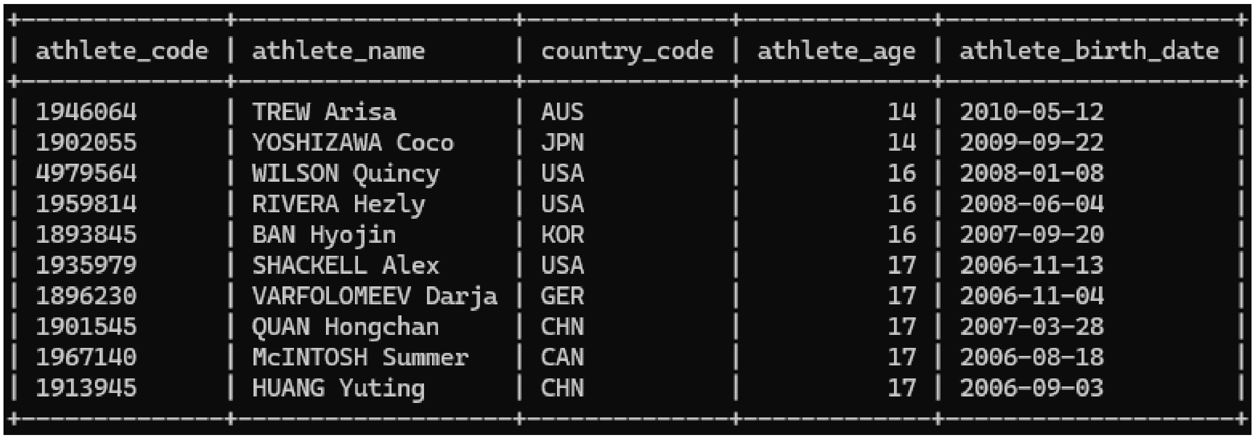
```
SELECT Countries.country_code, Countries.country_name, COUNT(athlete_code) AS
athlete_count
FROM Athletes
JOIN Countries ON Athletes.country_code = Countries.country_code
GROUP BY Countries.country_code
ORDER BY athlete_count DESC;
```



```
+--------------+---------------+---------------+
| country_code | country_name  | athlete_count |
+--------------+---------------+---------------+
| USA          | United States |           619 |
| FRA          | France        |           601 |
| AUS          | Australia     |           475 |
| GER          | Germany       |           457 |
| JPN          | Japan         |           431 |
| ESP          | Spain         |           401 |
| CHN          | China         |           398 |
| ITA          | Italy         |           397 |
| GBR          | Great Britain |           343 |
| CAN          | Canada        |           332 |
+--------------+---------------+---------------+
```

List all athletes below age 25 that achieved a gold medal in ascending order of age

```
SELECT athlete_code, athlete_name, country_code, athlete_age,
athlete_birth_date
FROM
(
    SELECT Athletes.athlete_code, athlete_name, country_code,
FLOOR(DATEDIFF('2024-08-11', athlete_birth_date) / 365) AS athlete_age,
athlete_birth_date FROM Athletes
    JOIN Medals ON Athletes.athlete_code = Medals.athlete_code
    WHERE Medals.medal_type = 'Gold Medal'
) AS gold_medallists
WHERE athlete_age < 25
GROUP BY athlete_code
ORDER BY athlete_age ASC;
```



```
+--------------+--------------------+--------------+-------------+--------------------+
| athlete_code | athlete_name       | country_code | athlete_age | athlete_birth_date |
+--------------+--------------------+--------------+-------------+--------------------+
| 1946064      | TREW Arisa         | AUS          |          14 | 2010-05-12         |
| 1902055      | YOSHIZAWA Coco     | JPN          |          14 | 2009-09-22         |
| 4979564      | WILSON Quincy      | USA          |          16 | 2008-01-08         |
| 1959814      | RIVERA Hezly       | USA          |          16 | 2008-06-04         |
| 1893845      | BAN Hyojin         | KOR          |          16 | 2007-09-20         |
| 1935979      | SHACKELL Alex      | USA          |          17 | 2006-11-13         |
| 1896230      | VARFOLOMEEV Darja  | GER          |          17 | 2006-11-04         |
| 1901545      | QUAN Hongchan      | CHN          |          17 | 2007-03-28         |
| 1967140      | McINTOSH Summer    | CAN          |          17 | 2006-08-18         |
| 1913945      | HUANG Yuting       | CHN          |          17 | 2006-09-03         |
+--------------+--------------------+--------------+-------------+--------------------+
```

As a note, all of these queries results were limited to a size of 10 for screenshots, but most return much more.

Advanced Features

Example of a stored procedure to speed up inserting medal data:

```
CREATE PROCEDURE insertMedal(
    temp_medal_date DATE,
    temp_medal_type VARCHAR(50),
    temp_athlete_code CHAR(7)
)
COMMENT 'Insert new medal into the Medals table.'
INSERT INTO Medals(medal_date, medal_type, athlete_code)
VALUES (temp_medal_date, temp_medal_type, temp_athlete_code);
```

An example of a trigger used to update the medal counts of each country when a medal entry belonging to an athlete of that country is inserted:

```
DELIMITER //
CREATE TRIGGER UpdateMedals AFTER INSERT ON Medals
    FOR EACH ROW
        BEGIN
            # Finding the country code of the athlete that won the medal
            DECLARE medal_country_code CHAR(3);
            SELECT country_code INTO medal_country_code FROM Athletes WHERE athlete_code = NEW.athlete_code;

            IF NEW.medal_type = 'Gold Medal' THEN
                UPDATE Countries SET gold_count = gold_count + 1 WHERE country_code = medal_country_code;
            ELSEIF NEW.medal_type = 'Silver Medal' THEN
                UPDATE Countries SET silver_count = silver_count + 1 WHERE country_code = medal_country_code;
            ELSEIF NEW.medal_type = 'Bronze Medal' THEN
                UPDATE Countries SET bronze_count = bronze_count + 1 WHERE country_code = medal_country_code;
            END IF;
        END//
DELIMITER ;
```

Python Usage

Examples of using SQL insert statements in python can be seen in the Data Insertion Methods section where python was used to insert all the data.

Basics examples of using delete, update and query statements in python can be seen in the py_scripts folder:

```
delete_stmt = 'DELETE FROM Medals WHERE medal_type = \'Bronze Medal\''
cursor.execute(delete_stmt)
connection.commit()
```

```
update_stmt = 'UPDATE Medals SET medal_type = \'Silver Medal\' WHERE medal_type = \'Gold Medal\''
cursor.execute(update_stmt)
connection.commit()
```

```
query_stmt = 'SELECT * FROM Athletes WHERE athlete_height > 180'
cursor.execute(query_stmt)
row = cursor.fetchone()
while row is not None:
    print(row)
    row = cursor.fetchone()
```

Discussion

The database functions well and allows helpful and complex queries on the more important parts of the Olympics 2024 data. I wished to make it a bit differently but was limited by the data I could find, having to change parts of my design just to accommodate the data that was available. I felt I had to spend more time than I should have on formatting the data to get into my database, especially considering it constitutes no marks, whilst at the same time preventing me from getting marks in other areas without it.