# 2020 Olympics Database
# User Guide

Author: Lachlan Gordon Murray
ID: 19769390
Lab Group: 2pm Friday
Date Written: 20/10/2021

# About the Database

This guide will help you to set up a functional Database using MySQL. The purpose of the database will be to contain a large amount of data relating to the proceedings of the 2020 Tokyo Olympic Games. In particular, Countries participating, their Coaches and Athletes, the Disciplines and Events that ran and the results of the Games in the form of gold/silver/bronze medallion count. The next section will explain how to use the provided files to set up the database, and the section following that will detail some interesting queries that can be performed.

# Setting Things Up

Assuming you're able to use the same method of connecting to the MySQL database as what is described in the ISYS1000 practicals, type the following in your terminal:

**> mysql -u me -p**

**> myUserPassword**

Now an appropriately named database should be created, for example:

**> create database 2020_Olympics_19769390;**

Move into this new database in order to implement it:

**> use 2020_Olympics_19769390;**

Firstly, the tables for storing the Olympics data need to be created. All of the commands for this are contained within the file named "create_tables.sql". The statements are accompanied by comments to help you understand what each field is for and why the certain datatype was chosen for them.

Simply type the following in terminal to run the given file:

**\. create_tables.sql**

This will create five tables for storing our Olympic Games related data.

"Countries" for the names of each country that participated and their results with winning medals.

"Disciplines" for the names of the various sports and activities that the Olympics Games host, as well as the total number of participants for each, also divided into male and female participants.

"Events" for the sub-categories of the disciplines that have them, for example the various relay distances in the Athletics and Swimming disciplines.

"Coaches" for the names, country of origin and disciplines/events that the coaches specialise in.

"Athletes" for the names, country of origin and disciplines that the athletes participate in.

To check that all of the tables were in fact created correctly, you may want to type the command:

**> show tables;**

And for each table, in order to show its structure:

**> describe table_name;**

Now, for the insertion of the 2020 Olympics data, obtained from the .xlsx and .csv files available at these links:

https://www.kaggle.com/arjunprasadsarkhel/2021-olympics-in-tokyo

https://www.kaggle.com/piterfm/tokyo-2020-olympics?select=medals.csv

For your convenience, the data sets were converted to a .csv format, so that the “load data infile” command that MySQL offers could be used for some of the tables instead of individual insert statements for each record.

Much like with the method for creating the tables, the commands for inserting the data are provided for you in the file “import_tables.sql” and the files that it calls, such as “import_athletes.sql”, “import_coaches.sql”, “import_events.sql”. The only one that needs to be run however, is the import_tables.sql, which can be done as follows:

**> \. import_tables.sql**

In order to check if the data was read in correctly, you can run a select all statement for each of the tables, as follows:

**> select * from table_name;**

# Querying the Database

Now that you've successfully set up the Database, it's now time to do some querying/improvement. What follows are some queries you may want to do or know the answer to and the accompanying SQL statement needed to implement the query, or a feature you may want to implement and how to implement it. Feel free to give any of these a try!

1. Select all the names of Coaches from the Country 'Australia'. Order them by their name descending:

   **> select coachName from Coaches where countryName = 'Australia' order by coachName desc;**

2. Select all the names of Athletes participating in the Discipline 'Judo' who are over the age of 35:

   **> select a.athleteName from Athletes a where a.disciplineName = 'Judo' and DATEDIFF(SYSDATE(), a.dob)/365 > 35;**

3. Select the names of all of the discipline names, only using the Athletes table:

   > **select disciplineName from Athletes group by disciplineName;**

4. Select all the names of Athletes who won at least 1 Gold Medal at the Olympics. Order them by their Discipline:

   **> select r.athleteName from Results r where r.medalType = 'Gold Medal' order by r.disciplineName;**

5. Select the name of all Disciplines with a higher ratio of females to males, and display the ratio in the format "<femaleParticipants>/<maleParticipants> = <result>", with an appropriate name for the column:

   **> select disciplineName, CONCAT(femaleParticipants, '/', maleParticipants, ' = ', ROUND(femaleParticipants/maleParticipants, 2)) as 'genderRatio' from Disciplines where femaleParticipants/maleParticipants > 1.0;**

6. Finding all of the realistic possible combinations between Coaches and Athletes:

   **> Select athleteName, coachName from Athletes natural join Coaches;**

7. Selecting all Athletes with the countries they represented as a sentence in the form "athleteName representing countryName" that participated in some form of Cycling:

   **> select CONCAT(athleteName, ' representing ', countryName) as 'candidate', disciplineName as 'sport' from Athletes where disciplineName like 'Cycling%';**

8. Adding a weight attribute to the Athletes table and then sorting them into weight categories as a union between subqueries:

   For each Athlete, give it a random decimal value between 50.00 and 90.00 for it's weight, using the stored procedure in "add_weight.sql":

   **> \. add_weight.sql**

   **> call addWeight();**

   Now do the subqueries to simply sort the weights:

   **> (select athleteName, athleteWeight, 'lightweight' as weightCategory from Athletes where athleteWeight < 75.00) UNION (select athleteName, athleteWeight, 'heavyweight' as weightCategory from Athletes where athleteWeight >= 75.00);**

9. Setting up a trigger to update the total, female or male number of participants in the Disciplines table when an Athlete is deleted from the Athletes table:

   Set up the trigger in "gender_trigger.sql"

   **> \. gender_trigger.sql**

   Display the total number of participants of a given Discipline from:

   **> select * from Disciplines where disciplineName = 'XXXXXX';**

   Remove an Athlete who participated in that Discipline;

   **> delete from Athletes where athleteName = 'YYYYYY';**

   Display the total number of participants of the Discipline again and check that it has changed:

   **> select * from Disciplines where disciplineName = 'XXXXXX';**