FINAL ASSESSMENT REPORT

ISYS1001 – Database Systems

STUDENT ID: 19501204
LAB: MONDAY 2-4PM
DUE DATE: 27/10/2021

Contents

Introduction

The following report discusses the design, implementation, and use cases of the database to extract and analyse data detailing information on the 2020 Olympic games.

The objectives discussed in this report are as follows:

- Creating an entity relationship diagram in Chen's notation.
- Defining constraints.
- Defining the relational schema.
- Implementations of the database.
- Design and implementation of queries.
- Advanced concepts: procedures and views.
- Query results of the MySQL database in the Python3 environment.

Database Design

Assumptions

The following assumptions are made for the purpose of the design of this database:

- The database is in an archived state where all data involving the 2020 Olympic games have been inserted.

ER Diagram

The image below displays the ER diagram for the 2020 Olympics.

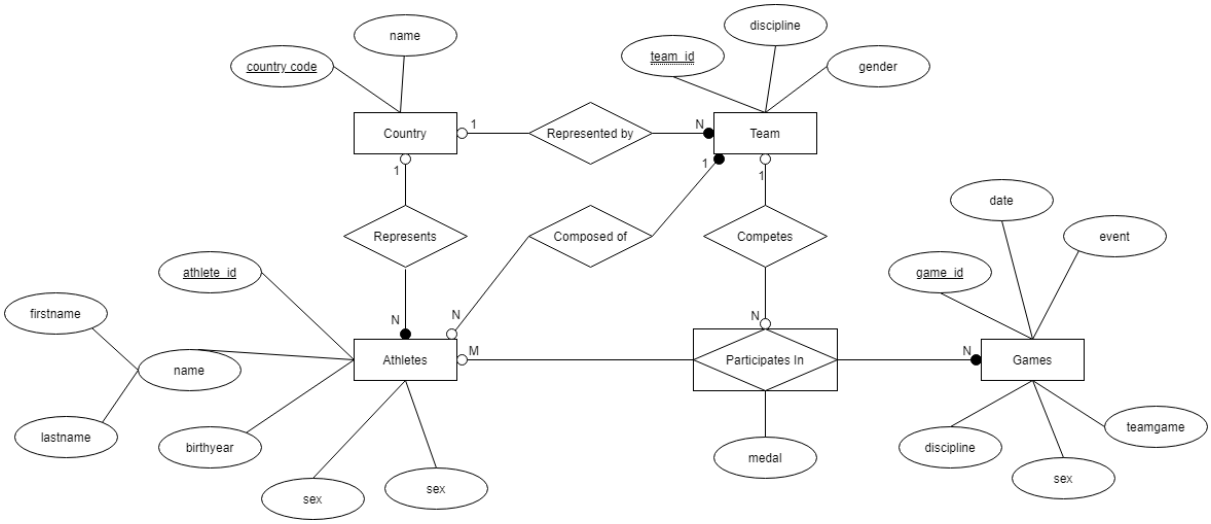


Figure 1. ER Diagram of the 2020 Olympics

For the design of this database, there are four entities: Country, Athletes, Team and Games. The relationship between Athletes and Games has a cardinality many to many restraint as Athletes can compete in more than one game and Games have multiple participating athletes. This relationship results in an additional associative entity called AthleteParticipation.

Relationships and Restraints

Table 1 describes the cardinality and participation restraints between the relationships.

Relationship set	Entity set	Cardinality	Participation
Represents	Athletes, Country	One-many (An athlete may represent one country; a country may be represented by many people)	Country-partial Athlete-total (An athlete should represent at least one country. A country may not be represented by anyone).
Represented by	Team, Country	One-many	Country-partial Team-total (A team must represent a country; a country may not be represented by a team)
Participates in	Athletes, Games	Many-many (An athlete may participate in many games; a game may be participated by many athletes	Athlete-partial Games-total (An athlete may not participate in any games; a game a should have participants).
Composed of	Athletes, Team	One-many (A team is composed of multiple athletes).	Athlete-partial Team-total (An athlete may not be part of a team, but a team needs more than one athlete)
Competes	Team, AthleteParticipation	One-many	Team-partial, AthleteParticipation-partial (A team may not participate in a game; a game may not be participated by a team)

Table 1. Entity Relations

Relational Schema

After defining the entity and relationship sets, the relational schema can be created as shown in table 2 below.

Entity Set	Key	Other Attributes
Country	Country_code (primary key)	Name
Athlete	Athlete_id (primary key), country_code (foreign key)	firstname, lastname, sex, birthyear
Games	Game_id (primary key)	event, discipline, sex, eventdate, location, teamgame
Team	Team_id (primary key), country_code (foreign key)	Discipline, country_code, sex

AthleteParticipation (associative entity set)	Athlete_id (primary key), Game_id (primary key), Team_id (foreign key)	medal

Table 2. Relational Schema

Country

The country entity has two attributes: the country_code as the primary key in the form of the ISO 3166-1 code and the name of the country. The country_code attribute is a char of size 3 because the ISO 3166-1 is a code that defines the countries which always comes in the length of 3. This entity is required due to the concept of the Olympic games which is countries around the globe competing for medals.

One detail that spectators would want to view is the number of medals a country has one in comparison of others. Medals is not a field in this case because a flat number of medals a country has would be difficult to break up into other categories such as which events these medals were won in. The number of medals can be queried into by accumulating all medals won in individual and team sports.

Fields	**Data Type**	**Size**	**Null**	**Description**
country_code (Primary Key)	CHAR	3	No	Country ISO 3166-1 code, unique
name	VARCHAR	27	No	Name of country

Table 3. Country Table

Athletes

Athletes is one of the primary entities in this database. The key details of this entity that spectators would like to know is the name of the athlete, their sex, age and the country they are representing.

A team attribute is not implemented into this table due to the fact if an athlete participates in more than one team, duplicate data will be displayed.

Sex and birthyear are the only attributes that can be set to null since an athlete may not want to disclose this information publicly when competing in the Olympics.

A discipline attribute is not included into this table in the event an athlete competes in more than one type of discipline. For this scenario, the discipline attribute is assigned to the Games table. If we want to know all the discipline the athlete participated in, we will have to join the athletes to all the games they participated in and display the disciplines.

Fields	**Data Type**	**Size**	**Null**	**Description**
athlete_id (primary key)	CHAR	6	No	Athlete number, unique
firstname	VARCHAR	12	No	First name
lastname	VARCHAR	15	No	Last name
sex	CHAR	1		M=male, F=female, O=other
birthyear	YEAR	4		Year of birth

country_code (foreign key)	CHAR	3	No	County Iso 3166-1 code

Table 4. Athletes Table

Team

The purpose of the team entity is to group the athletes together in a team game. The combination of discipline, country code and gender should be unique, i.e., America's women's 3x3 basketball team, except for in the cases when a country has multiple representative teams competing in the same discipline. Therefore, to be safe of duplicates, a team id set as the primary key for this table.

Fields	**Data Type**	**Size**	**Null**	**Reference Table**
team_id (primary key)	CHAR	6	No	Team number, unique
discipline	VARCHAR	36		Sport name
country_code (foreign key)	CHAR	3	No	County Iso 3166-1 code
gender	VARCHAR	5	No	Team's gender

Table 5. Team Table

Games

The games table contains the details of each event occurred at the 2020 Olympics, such as: the name of the event, the discipline, the sex of the participants, the date of the event and a Boolean attribute to determine if the event is a team game.

The event attribute is a subcategory of the discipline event. I.e., 400m freestyle heat is under the discipline "swimming." This allows the user to find all the events that involve a certain discipline.

The Boolean value "teamgame" shown in Table 6 allows the user to separate the individual events to the team events.

Fields	**Data Type**	**Size**	**Null**	**Reference Table**
game_id (primary key)	CHAR	6	No	Game number, unique
event	VARCHAR	40	No	Name of the event
discipline	VARCHAR	24	No	Sport type
sex	CHAR	1	No	M=Mens, W=Womens, O=Mixed
eventdate	DATE			Start date of games
teamgame	TINYINT	1		If the game is a team game; 1=true, 0= false

Table 6. Games Table

AthleteParticipation

The AthleteParticipation table is an associative entity due to the result of the many to many cardinality relationship between the Athlete and Games entities.

Considering the features of an associative entity, this entity has no singular primary key. The primary key consists of a combination of the athlete_id key from the Athletes table and the game_id key from the Games table. The team_id is an optional attribute only for team-based games.

A medal attribute is assigned to games where the athlete has won a game that awards a medal. This attribute is assigned to the AthleteParticipation table to be readily available if the user wants to view which game an athlete and/or team won a medal.

Fields	Data Type	Size	Null	Description
athlete_id (foreign key)	CHAR	6	No	Athlete number
game_id (foreign key)	CHAR	6	No	Game number
team_id (foreign key)	CHAR	6		Team identifier
medal	CHAR	1		G=gold, S=silver, B=bronze, null=no medals

Table 7. AthleteParticipation Table

Database Implementations

The database implementation assumes that the scripts used will be run in the Curtin Lab Virtual Machine environment, and the user has logged into the MySQL server in the Linux terminal. This section also assumes the user knows basic SQL and Linux commands.

The database creation relies on the user to run one script in the MySQL server: build.sql.

The build.sql is compromised of a few queries and other files.

Firstly, the script will delete the database called "olympics_19501204" if it already exists then create a new database with the same name. This allows a new database that has not been manipulated. The script will then run the command "USE olympics_19501204;" which selects the newly created database.

Table creations

Once the database has been selected, the build.sql script will run the following 5 scripts to create tables:

- createcountry.sql

```
/* createcountry.sql: MySQL table for table creation in the final assessment of ISYS1001*/

-- Create country table
DROP TABLE IF EXISTS Country;
CREATE TABLE Country(
        country_code    CHAR(3) NOT NULL,
        name            VARCHAR(24) NOT NULL,
        PRIMARY KEY(country_code)
);
```

Figure 2. createcountry.sql creates the Country table

- createteam.sql

```
/* createteam.sql: MySQL table for table creation in the final assessment of ISYS1001*/

-- Create Team table
DROP TABLE IF EXISTS Team;
CREATE TABLE Team(
        team_id         CHAR(6),
        discipline      VARCHAR(36),
        country_code    CHAR(3) NOT NULL,
        gender          VARCHAR(5) NOT NULL,
        PRIMARY KEY(team_id),
        FOREIGN KEY(country_code) REFERENCES Country(country_code) ON DELETE CASCADE
);
```

Figure 3. createteam.sql creates the Team table

- createathletes.sql

```
/* createathlete.sql: MySQL table for table creation in the final assessment of ISYS1001*/

-- Create athletes table
DROP TABLE IF EXISTS Athletes;
CREATE TABLE Athletes(
        athlete_id      CHAR(6) NOT NULL,
        firstname       VARCHAR(12) NOT NULL,
        lastname        VARCHAR(15) NOT NULL,
        sex             CHAR(1),
        birthyear       YEAR,
        country_code    CHAR(3) NOT NULL,
        PRIMARY KEY(athlete_id),
        FOREIGN KEY(country_code) REFERENCES Country(country_code)
);
```

Figure 4. createathletes.sql creates the Athletes table

- creategames.sql

```
/* creategames.sql: MySQL table for table creation in the final assessment of ISYS1001*/

-- Create games table
DROP TABLE IF EXISTS Games;
CREATE TABLE Games(
        game_id         CHAR(6) NOT NULL,
        event           VARCHAR(40) NOT NULL,
        discipline      VARCHAR(24) NOT NULL,
        sex             CHAR(1) NOT NULL,
        eventdate       DATE,
        teamgame        BOOLEAN,
        PRIMARY KEY(game_id)
);
```

Figure 5. creategames.sql creates the Games table

- createathleteparticipation.sql

```
/* createathleteparticipation.sql: MySQL table for table creation in the final assessment of
ISYS1001*/

-- Create athlete participation table
DROP TABLE IF EXISTS AthleteParticipation;
CREATE TABLE AthleteParticipation(
        athlete_id      CHAR(6),
        game_id         CHAR(6),
        team_id         CHAR(6),
        medal           CHAR(1),
        PRIMARY KEY(athlete_id, game_id),
        FOREIGN KEY(athlete_id) REFERENCES Athletes(athlete_id) ON DELETE CASCADE,
        FOREIGN KEY(game_id) REFERENCES Games(game_id) ON DELETE CASCADE,
        FOREIGN KEY(team_id) REFERENCES Team(team_id)
);
```

Figure 6. createathleteparticipation.sql creates the AthleteParticipation Table

The order in which these scripts run are important to successfully allow the foreign key restraint.
The creation of the Team and Athletes table must be after the Country table.
The games table must be created before AthleteParticipation table.
Therefore, the AthleteParticipation table must be created last.

Table Data Insertions

After the tables are created, the build.sql script runs the next 5 scripts to populate the table. The order these tables are populated must follow the same order as the table creation order.

- inscountry.sql

```
LOAD DATA LOCAL
        INFILE  'Country.csv'
        INTO TABLE Country
        FIELDS TERMINATED BY ',' ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        IGNORE 1 ROWS;
```

Figure 7. inscountry.sql inserts data from Country.csv into the Country table

- insathletes.sql

```
LOAD DATA LOCAL
        INFILE  'Athletes.csv'
        INTO TABLE Athletes
        FIELDS TERMINATED BY ',' ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        IGNORE 1 ROWS;
```

Figure 8. insathletes.sql inserts data from Athletes.csv into Athletes table

- insteam.sql

```
LOAD DATA LOCAL
        INFILE  'Team.csv'
        INTO TABLE Team
        FIELDS TERMINATED BY ',' ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        IGNORE 1 ROWS;
```

Figure 9. insteam.sql inserts data from Team.csv into Team table

- insgames.sql

```
LOAD DATA LOCAL
        INFILE  'Games.csv'
        INTO TABLE Games
        FIELDS TERMINATED BY ',' ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        IGNORE 1 ROWS;
```

Figure 10. insgames.sql inserts data from Games.csv into Games table

- insathpar.sql

```
LOAD DATA LOCAL
        INFILE  'AthleteParticipation.csv'
        INTO TABLE AthleteParticipation
        FIELDS TERMINATED BY ',' ENCLOSED BY '"'
        LINES TERMINATED BY '\n'
        IGNORE 1 ROWS;
```

Figure 11. insathpar.sql inserts data from AthleteParticipation.csv into the AthleteParticipation table

All the insertion script reads a CSV file, separate the values by a comma delimiter and insert into its respective table. Each script ignores the first row since the first row contains the column names in the CSV file.

Data Source

Refer to reference (1.) to view the source of the data in the Country.csv file to populate the Country table.

Refer to reference (2.) to view the source of the data in Games.csv file to populate the Games table.

Refer to reference (3.) to view the source of the data in the Athletes.csv and Teams.csv files used to populate the Athletes and Team tables.

Refer to reference (4.) to view the source of data containing medals used in AthleteParticipation.csv to populate the AthleteParticipation table.

Design of queries and use of the database

All queries are stored in the file 'commands.sql'.

All queries will be run if you execute the commands.sql script in the MySQL server.

Queries

Query 1

The first query will display all games where the date of the event is after 26-07-2021.

```
SELECT *
FROM Games
WHERE eventdate >= '2021-07-27';
```

Figure 12. Query 1 Script

game_id	event	discipline	sex	eventdate	teamgame
000005	Semi-final: USA vs France	3x3 Basketball	W	2021-07-28	1
000006	Semi-final: Serbia vs Russia	3x3 Basketball	M	2021-07-28	1
000007	Semi-final: Russia vs China	3x3 Basketball	W	2021-07-28	1

Figure 13. Query 1 Output

Query 2

Query 2 will display the Games table with an additional column called "DaysDifference" that shows how many days has passed since this event:

```
SELECT *,
    DATEDIFF(NOW(), eventdate) AS DaysDifference
FROM Games;
```

Figure 14. Query 2 Script

game_id	event	discipline	sex	eventdate	teamgame	DaysDifference
000001	400m freestyle heats	Swimming	M	2021-07-24	0	95
000002	400m freestyle final	Swimming	M	2021-07-25	0	94
000003	400m freestyle heats	Swimming	W	2021-07-25	0	94

Figure 15. Query 2 Output

Query 3

Query 3 joins the Athletes, AthleteParticipation and Games table to display all American athletes and the events they participated in.

```
SELECT  Athletes.athlete_id,
    CONCAT(Athletes.firstname, " ", Athletes.lastname) AS FullName,
    AthleteParticipation.game_id,
    Games.event
FROM Athletes
LEFT OUTER JOIN AthleteParticipation ON Athletes.athlete_id=AthleteParticipation.athlete_id
LEFT OUTER JOIN Games ON Games.game_id=AthleteParticipation.game_id
WHERE Athletes.country_code='USA'
ORDER BY game_id;
```

Figure 16. Query 3 Script

```
+------------+--------------------+---------+---------------------------------+
| athlete_id | FullName           | game_id | event                           |
+------------+--------------------+---------+---------------------------------+
| 100069     | Kieran Smith       | 000001  | 400m freestyle heats            |
| 100074     | Jake Mitchell      | 000001  | 400m freestyle heats            |
| 100074     | Jake Mitchell      | 000002  | 400m freestyle final            |
| 100069     | Kieran Smith       | 000002  | 400m freestyle final            |
```

Figure 17. Query 3 Output

Query 4

Query 4 displays all gold medal athletes and the total gold medals they won.

```
SELECT AthleteParticipation.athlete_id,
    Athletes.firstname,
    Athletes.lastname,
    count(medal) as total_gold_medals,
    country_code
FROM AthleteParticipation
INNER JOIN Athletes ON Athletes.athlete_id=AthleteParticipation.athlete_id
WHERE medal='G'
GROUP BY AthleteParticipation.athlete_id;
```

Figure 18. Query 4 Script

```
+------------+-----------+------------+--------------------+--------------+
| athlete_id | firstname | lastname   | total_gold_medals | country_code |
+------------+-----------+------------+--------------------+--------------+
| 100001     | Nauris    | Miezis     |                  1 | LVA          |
| 100002     | Karlis    | Lasmanis   |                  1 | LVA          |
| 100003     | Edgars    | Krumins    |                  1 | LVA          |
| 100004     | Agnis     | Cavars     |                  1 | LVA          |
```

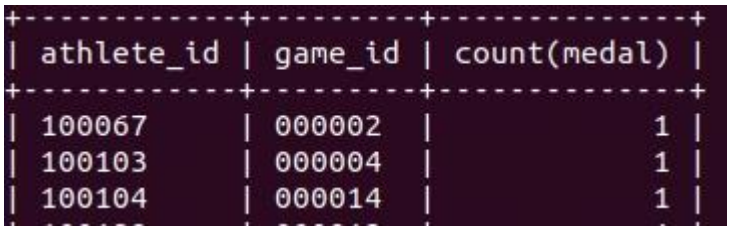
Figure 19. Query 4 Output

Query 5

Query 5 displays the athlete id of winners of individual games that awarded a gold medal.

```
SELECT athlete_id,
        game_id,
        count(medal)
FROM AthleteParticipation
WHERE medal='G' AND team_id IS NULL
group by athlete_id, game_id;
```

Figure 20. Query 5 Script



athlete_id	game_id	count(medal)
100067	000002	1
100103	000004	1
100104	000014	1

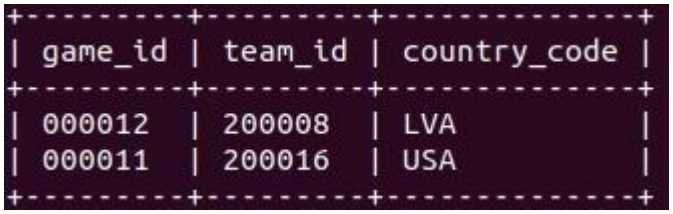
Figure 21. Query 5 Output

Query 6

Query 6 displays the team id and country code of winners of team games that awarded a gold medal

```
SELECT DISTINCT game_id,
    AthleteParticipation.team_id,
    country_code
FROM AthleteParticipation
LEFT OUTER JOIN Team ON AthleteParticipation.team_id=Team.team_id
WHERE medal='G' AND AthleteParticipation.team_id IS NOT NULL;
```

Figure 22. Query 6 Script



game_id	team_id	country_code
000012	200008	LVA
000011	200016	USA

Figure 23. Query 6 Output

Query 7

Query 7 displays the major events that award a gold medal.

```
SELECT DISTINCT AthleteParticipation.game_id,
        event,
        sex,
        eventdate
FROM AthleteParticipation
LEFT OUTER JOIN Games on AthleteParticipation.game_id=Games.game_id
WHERE medal='G'
GROUP BY game_id;
```

Figure 24. Query 7 Script

game_id	event	sex	eventdate
000002	400m freestyle final	M	2021-07-25
000004	400m freestyle final	W	2021-07-26
000011	Gold medal game: USA vs Russia	W	2021-07-28

Figure 25. Query 7 Output

Query 8

Query 8 display the amount of gold medals a country has won.

```
SELECT COUNT(DISTINCT game_id) as goldmedals,
       Athletes.country_code
FROM AthleteParticipation
INNER JOIN Athletes ON Athletes.athlete_id=AthleteParticipation.athlete_id
WHERE medal='G'
GROUP BY Athletes.country_code;
```

Figure 26. Query 8 Script

goldmedals	country_code
1	AUS
1	LVA
1	NOR
1	TUN
4	USA

Figure 27. Query 8 Output

Query 9

Query 9 displays all the medal types of athletes who won more than medal.

```
SELECT athlete_id,
    SUM(CASE WHEN medal = 'B' THEN 1 ELSE 0 END) AS BronzeMedals,
    SUM(CASE WHEN medal = 'S' THEN 1 ELSE 0 END) AS SilverMedals,
    SUM(CASE WHEN medal = 'G' THEN 1 ELSE 0 END) AS Goldmedals,
    count(medal) as TotalMedals
FROM AthleteParticipation
GROUP BY athlete_id
HAVING TotalMedals > 1;
```

Figure 28. Query 9 Script

athlete_id	BronzeMedals	SilverMedals	Goldmedals	TotalMedals
100103	0	1	1	2
100104	0	1	1	2

Figure 29. Query 9 Output

Advanced Features

Procedures

Medal Counter

The medal counter procedure allows the user to simply extract the total number of medals a country has won. This is the sum of bronze, silver and gold medals. However, this calculates the medals at the event-level meaning for a team event, even though all players will receive a medal, this medal counter procedure will count this event as one medal won.

The script is stored in the medalcounter.sql file and the code is shown in figure 30.

```
-- Procedure to count total medals of a country
DROP PROCEDURE IF EXISTS countMedals;
CREATE PROCEDURE countMedals(
        IN code CHAR(3),
        OUT total INT)
COMMENT 'Count the number of medals a country has. Team games are counted as one medal. Provided
the ISO-3166 code.'
SELECT COUNT(DISTINCT game_id)
FROM AthleteParticipation
INNER JOIN Athletes
        ON Athletes.athlete_id=AthleteParticipation.athlete_id
WHERE medal IS NOT NULL
        AND Athletes.country_code = code
GROUP BY Athletes.country_code INTO total;
```

Figure 30. Medal Counter Procedure Script

Figure 31 shows on how to call this procedure, assuming that the procedure is inserted into the database.

```
mysql> call countMedals('USA', @result);
```

Figure 31. Calling countMedals

Figure 32 shows on how to view the result.

```
mysql> select @result;
```

Figure 32. Command to view result

Figure 33 displays a sample output.

```
+----------+
| @result  |
+----------+
|        7 |
+----------+
```

Figure 33. Medal Counter Sample Result

Insert New Athlete

This procedure has the purpose of inserting a new athlete into the Athletes table. This procedure takes in the parameters of a firstname, lastname, sex, year of birth and the country code respectively and assigns a unique id to this athlete by taking the highest value existing id and incrementing it by one. Though this procedure does not have a purpose for this current implementation, its designed for

future purposes such as the following Olympics and assigning a unique id for each new Athlete participating.

The script is stored in insathproc.sql file and the code is shown in figure 34.

```
-- Procedure to insert a new Athlete
DROP PROCEDURE IF EXISTS insNewAthlete;
DELIMITER //
CREATE PROCEDURE insNewAthlete(
        f VARCHAR(12),  -- firstname
        l VARCHAR(15),  -- lastname
        s CHAR(1),              -- sex
        y YEAR,                 -- year
        c CHAR(3)               -- country_code
        )
        COMMENT 'Insert a new Athlete into Athletes table.'
        BEGIN
                DECLARE newid CHAR(6);
                SELECT MAX(athlete_id)+1 FROM Athletes INTO newid;
                INSERT INTO Athletes
                        VALUES(newid, f, l, s, y, c);
        END//
DELIMITER ;
```

Figure 34. insathproc.sql script

Figure 35 shows an sample call of the new procedure.

```
mysql> call insNewAthlete('Anthony', 'Nguyen', 'M', '1998', 'AUS');
```

Figure 35. insNewAthlete() sample call

Figure 36 shows a sample output.

```
| 100154     | Anna         | Cockrell      | F    |      1997 | USA          |
| 100155     | Anthony      | Nguyen        | M    |      1998 | AUS          |
+------------+--------------+---------------+------+-----------+--------------+
```

Figure 36. Sample output for inserting a new athlete procedure

Views

Athlete Details

This view is implemented to easily allow the extraction of data displaying all the events an athlete is participating in. Inserting this view into the database will create a view table called Athletedetails.

The script is stored in athdetview.sql and the code is shown in figure 37.

```
-- A view to display the athlete and the event they participated in.
DROP VIEW IF EXISTS Athletedetails;
CREATE VIEW Athletedetails AS
        SELECT Athletes.athlete_id,
                firstname,
                lastname,
                country_code,
                AthleteParticipation.game_id AS GameID,
                event,
                Games.sex
        FROM (Athletes JOIN AthleteParticipation
                        ON AthleteParticipation.athlete_id = Athletes.athlete_id)
        JOIN Games ON Games.game_id = AthleteParticipation.game_id;
```

Figure 37. Athlete Details View Script

A sample output is shown in figure 38.

athlete_id	firstname	lastname	country_code	GameID	event	sex
100067	Ahmed	Hafnaoui	TUN	000001	400m freestyle heats	M
100068	Jack	Mcloughlin	AUS	000001	400m freestyle heats	M
100069	Kieran	Smith	USA	000001	400m freestyle heats	M
100070	Felix	Auboeck	AUT	000001	400m freestyle heats	M

Figure 38. Athletedetails view sample output

Athlete Medals

This view allows displays a table of athletes and all the types of medals they have won. This view allows spectators to view this data without assigning 3 attributes to the Athletes table to save storage in the database as this database would contain thousands of entries on completion.

```
DROP VIEW IF EXISTS Athletemedals;
CREATE VIEW Athletemedals AS
        SELECT Athletes.athlete_id,
                firstname,
                lastname,
                SUM(CASE WHEN medal = 'B' THEN 1 ELSE 0 END) AS BronzeMedals,
                SUM(CASE WHEN medal = 'S' THEN 1 ELSE 0 END) AS SilverMedals,
                SUM(CASE WHEN medal = 'G' THEN 1 ELSE 0 END) AS Goldmedals,
                country_code
        FROM (Athletes JOIN AthleteParticipation
                        ON AthleteParticipation.athlete_id = Athletes.athlete_id)
        GROUP BY Athletes.athlete_id;
```

Figure 39. Athlete Medals View Script

athlete_id	firstname	lastname	BronzeMedals	SilverMedals	Goldmedals	country_code
100001	Nauris	Miezis	0	0	1	LVA
100002	Karlis	Lasmanis	0	0	1	LVA
100003	Edgars	Krumins	0	0	1	LVA
100004	Agnis	Cavars	0	0	1	LVA

Figure 40. Athletemedals view sample output

Database connectivity and Python implementation

This section assumes:

- The user has access to the python3 console.
- The python3 mysql connector is installed.
- The olympics_19501204 database has been built, and the tables within are created and populated with data.

Three python scripts are implemented.

Query 1

The python script is stored in query1.py and the purpose of this script is to run the same query as figure 12. The content of the script is shown in figure 41.

```
import mysql.connector
db = mysql.connector.connect(host='localhost', database='olympics_19501204', user='me',
password='myUserPassword')
cur = db.cursor()

select_query1="SELECT * FROM Games WHERE eventdate >= '2021-07-27';"
cur.execute(select_query1)
rows = cur.fetchall()
for row in rows:
        print (row)

cur.close();
```

Figure 41. query1.py script

To execute this script, enter the following command in the python3 console:

```
>>> exec(open("query1.py").read())
```

A sample output of this query is shown in figure 42.

```
('000005', 'Semi-final: USA vs France', '3x3 Basketball', 'W', datetime.date(2021, 7, 28), 1)
('000006', 'Semi-final: Serbia vs Russia', '3x3 Basketball', 'M', datetime.date(2021, 7, 28), 1)
('000007', 'Semi-final: Russia vs China', '3x3 Basketball', 'W', datetime.date(2021, 7, 28), 1)
('000008', 'Semi-final: Belgium vs Latvia', '3x3 Basketball', 'M', datetime.date(2021, 7, 28), 1)
('000009', 'Bronze medal game: France vs China', '3x3 Basketball', 'W', datetime.date(2021, 7, 28), 1)
('000010', 'Bronze medal game: Serbia vs Belgium', '3x3 Basketball', 'M', datetime.date(2021, 7, 28), 1)
('000011', 'Gold medal game: USA vs Russia', '3x3 Basketball', 'W', datetime.date(2021, 7, 28), 1)
('000012', 'Gold medal game: Russia vs Latvia', '3x3 Basketball', 'M', datetime.date(2021, 7, 28), 1)
('000013', '800m freestyle final', 'Swimming', 'M', datetime.date(2021, 7, 29), 0)
('000014', '800m freestyle final', 'Swimming', 'W', datetime.date(2021, 7, 31), 0)
('000015', '400m hurdles final', 'Athletics', 'M', datetime.date(2021, 8, 3), 0)
('000016', '400m hurdles final', 'Athletics', 'W', datetime.date(2021, 8, 4), 0)
```

Figure 42. sample output of query1.py

Using python to extract this data externally, this can be applied to future front-end projects.

Query 3

This python script is stored in query3.py and executes the same query as shown in figure 16.

To execute this script, enter the following command in the python3 console:

```
>>> exec(open("query1.py").read())
```

```
import mysql.connector
db = mysql.connector.connect(host='localhost', database='olympics_19501204', user='me',
password='myUserPassword')
cur = db.cursor()

select_query3= (
        "SELECT Athletes.athlete_id, \
        CONCAT(Athletes.firstname, ' ', Athletes.lastname) AS FullName, \
        AthleteParticipation.game_id, \
        Games.event \
        FROM Athletes \
        LEFT OUTER JOIN AthleteParticipation ON
Athletes.athlete_id=AthleteParticipation.athlete_id \
        LEFT OUTER JOIN Games ON Games.game_id=AthleteParticipation.game_id \
        WHERE Athletes.country_code='USA' \
        ORDER BY game_id;"
)
cur.execute(select_query3)
row = cur.fetchall()
for row in rows:
        print (row)

cur.close()
```

Figure 43. query3.py script

```
('000005', 'Semi-final: USA vs France', '3x3 Basketball', 'W', datetime.date(2021, 7, 28), 1)
('000006', 'Semi-final: Serbia vs Russia', '3x3 Basketball', 'M', datetime.date(2021, 7, 28), 1)
('000007', 'Semi-final: Russia vs China', '3x3 Basketball', 'W', datetime.date(2021, 7, 28), 1)
('000008', 'Semi-final: Belgium vs Latvia', '3x3 Basketball', 'M', datetime.date(2021, 7, 28), 1)
('000009', 'Bronze medal game: France vs China', '3x3 Basketball', 'W', datetime.date(2021, 7, 28), 1)
('000010', 'Bronze medal game: Serbia vs Belgium', '3x3 Basketball', 'M', datetime.date(2021, 7, 28), 1)
('000011', 'Gold medal game: USA vs Russia', '3x3 Basketball', 'W', datetime.date(2021, 7, 28), 1)
('000012', 'Gold medal game: Russia vs Latvia', '3x3 Basketball', 'M', datetime.date(2021, 7, 28), 1)
('000013', '800m freestyle final', 'Swimming', 'M', datetime.date(2021, 7, 29), 0)
('000014', '800m freestyle final', 'Swimming', 'W', datetime.date(2021, 7, 31), 0)
('000015', '400m hurdles final', 'Athletics', 'M', datetime.date(2021, 8, 3), 0)
('000016', '400m hurdles final', 'Athletics', 'W', datetime.date(2021, 8, 4), 0)
```

Figure 44. sample output of query3.py

Load AthleteParticipation

This script is stored in loadathpar.py and the purpose of this script populates the AthletesParticipation table with data from the “AthleteParticipation.csv” file. The purpose of designing this script is to allow an alternative example of populating a table within the database.

```
# Python load file to fill up the AthleteParticipation table
import mysql.connector
db = mysql.connector.connect(host='localhost', database='olympics_19501204', user='me',
password='myUserPassword')
cur = db.cursor()

insert_stmt="INSERT INTO AthleteParticipation VALUES (%s, %s, %s, %s)"

check="NULL"

with open("AthleteParticipation.csv") as f:
        # Discard the first line
        f.readline()
        for line in f.readlines():
                values = line.strip().split(",")
                if values[2] == "NULL":
                        values[2] = None
                if values[3] == "NULL":
                        values[3] = None
                obj = (values[0], values[1], values[2], values[3])
                cur.execute(insert_stmt, obj)

print("Done!")
cur.close()
db.commit()
```

Figure 45. loadathpar.py

References

1. https://en.wikipedia.org/wiki/ISO_3166-1

2. https://www.theroar.com.au/olympics/olympics-events-schedule/#prelimday-1

3. https://www.kaggle.com/arjunprasadsarkhel/2021-olympics-in-tokyo?select=Athletes.xlsx

4. https://olympics.com/en/olympic-games/tokyo-2020/medals