We start the design process by defining the requirements for the database MiniNet:

Users:

- Each user can have only one subscription (HD or UHD).
- For each user, you need to store the user's city, country, and age.
- Each user can only have one list of favorite movies (with user-assigned scores).

Subscriptions:

- There are two subscription types: HD (£12) and UHD (£18).

Movies:

- Each movie belongs to a category, e.g. comedy, drama, etc.
- The content is organized by type, such as TV shows and movies.

Actors:

- Actors should have an age (also name, city of residence, date of birth).
- Actors are associated with more than one movie.

Reviews:

- Each user can give only one review per video.
- Each review can be any number between 0 to 5.

Special requirements:

- There should be at least one movie starting with the keyword 'The', e.g., 'The Lord of the Rings'.

Task 1

a) Entities according to the use case (tables):
- Users
- Subscriptions
- Movies
- Actors
- Reviews
- FavoriteMovies

b) Relationships between entities.
- Each user selects one subscription, but one subscription can be selected by many users
- Each user has only one list of favorite movies (several movies), and the list belongs to only one user
- Each user can post multiple reviews, but each review can be written by only one user
- Each review is written to one movie, but each movie can have multiple reviews
- Each actor can work in different movies, and each movie have several actors

c) Cardinality of relationships.
- Users and Subscriptions: many-to-one

- Users and FavoriteMovies: one-to-many
- Users and Reviews: one-to-many
- Reviews and Movies: many-to-one
- Movies and Actors: many-to many

d) Attribute identification and their data types are shown in Task 2 (Fig. 2)

e) Create Entity Relational diagram:

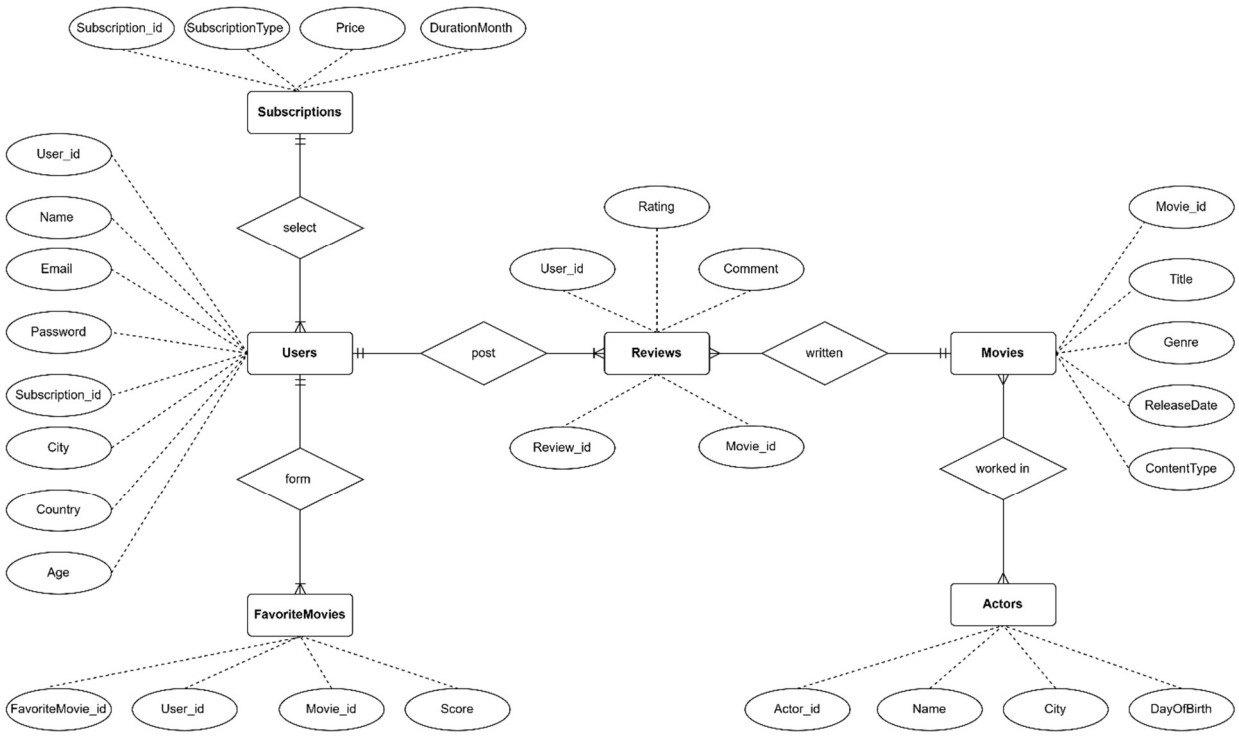

Fig. 1 ER model

Task 2, Task 3

To identify the appropriate relationships using primary and foreign key, we need to link tables.

The middle table MovieActors should be created.

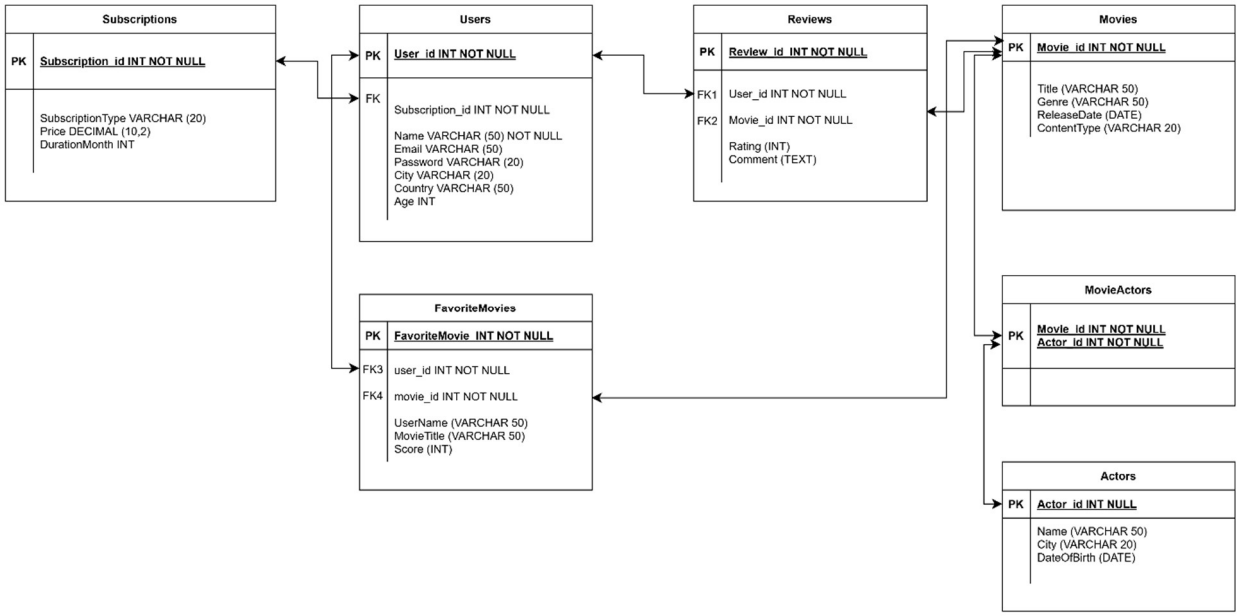

Fig2. Database model for the MiniNet system

SQL scripts for creating ang inserting data into the database tables:

```
CREATE TABLE Subscriptions (
Subscription_id INT NOT NULL,
SubscriptionType VARCHAR (20),
Price DECIMAL(10,2),
DurationMonths INT,
PRIMARY KEY (Subscription_id)
);
INSERT INTO Subscriptions (Subscription_id, SubscriptionType, Price, DurationMonths)
VALUES
(1, 'HD', 12, 1),
(2, 'UHD', 18, 3);

CREATE TABLE Users (
User_id INT NOT NULL,
Name VARCHAR (50),
Email VARCHAR (50),
Password VARCHAR (20),
Subscription_id INT NOT NULL,
City VARCHAR (20),
Country VARCHAR (50),
Age INT,
PRIMARY KEY (User_id),
CONSTRAINT FK FOREIGN KEY (Subscription_id) REFERENCES
Subscriptions(Subscription_id),
UNIQUE (Email)
);
```

```
INSERT INTO Users (User_id, Name, Email, Password, Subscription_id, City, Country, Age)
VALUES
(1, 'john_doe', 'john@example.com', 'password1', 1, 'New York', 'USA', 28),
(2, 'alice_smith', 'alice@example.com', 'password2', 1, 'Los Angeles', 'USA', 34),
(3, 'jane_doe', 'jane@example.com', 'password3', 2, 'Chicago', 'USA', 21),
(4, 'bob_jones', 'bob@example.com', 'password4', 2, 'Boston', 'USA', 30),
(5, 'emma_johnson', 'emma@example.com', 'password5', 1, 'San Francisco', 'USA', 25),
(6, 'mary_james', 'mary@example.com', 'password6', 1, 'Toronto', 'Canada', 27);

CREATE TABLE Movies (
Movie_id INT NOT NULL,
Title VARCHAR (50),
Genre VARCHAR (50),
ReleaseDate DATE,
ContentType VARCHAR (20),
PRIMARY KEY (Movie_id)
);
INSERT INTO Movies (Movie_id, Title, Genre, ReleaseDate, ContentType) VALUES
(1, 'Stranger Things', 'Sci-Fi', '2016-07-15', 'TV show'),
(2, 'Breaking Bad', 'Drama', '2008-01-20', 'Movie'),
(3, 'The Office', 'Comedy', '2005-03-24', 'TV show'),
(4, 'Parks and Recreation', 'Comedy', '2009-04-09', 'Movie'),
(5, 'The Godfather', 'Crime', '1972-03-24', 'Movie');


CREATE TABLE Actors (
Actor_id INT NOT NULL,
Name VARCHAR (50),
City VARCHAR (20),
DateOfBirth DATE,
PRIMARY KEY (Actor_id)
);
INSERT INTO Actors (Actor_id, Name, City, DateOfBirth) VALUES
(1, 'Millie Bobby Brown', 'Los Angeles', '2004-02-19'),
(2, 'Bryan Cranston', 'Hollywood', '1956-03-07'),
(3, 'Winona Ryder', 'New York', '1971-10-29'),
(4, 'Aaron Paul', 'Boise', '1979-08-27'),
(5, 'David Harbour', 'White Plains', '1975-04-10');

CREATE TABLE MovieActors (
Movie_id INT NOT NULL,
Actor_id INT NOT NULL,
PRIMARY KEY (Movie_id, Actor_id),
CONSTRAINT FK5 FOREIGN KEY (Movie_id) REFERENCES Movies(Movie_id),
CONSTRAINT FK6 FOREIGN KEY (Actor_id) REFERENCES Actors(Actor_id),
UNIQUE (Movie_id, Actor_id)
);
INSERT INTO MovieActors (Movie_id, Actor_id) VALUES
(1, 1),
(2, 2),
```

```
(1, 3),
(2, 4),
(1, 5);

CREATE TABLE Reviews (
Review_id INT NOT NULL,
User_id INT NOT NULL,
Movie_id INT NOT NULL,
Rating INT CHECK (Rating>=0 AND Rating<=5),
Comment TEXT,
PRIMARY KEY (Review_id),
CONSTRAINT FK1 FOREIGN KEY (User_id) REFERENCES Users(User_id),
CONSTRAINT FK2 FOREIGN KEY (Movie_id) REFERENCES Movies(Movie_id),
UNIQUE (User_id, Movie_id)
);
INSERT INTO Reviews (Review_id, User_id, Movie_id, Rating, Comment) VALUES
(1, 1, 1, 5, 'Amazing show!'),
(2, 2, 2, 3, 'Good show'),
(3, 3, 3, 4, 'Funny and smart'),
(4, 4, 4, 2, 'Not my taste'),
(5, 5, 5, 5, 'A classic!'),
(6, 6, 3, 1, 'Boring');

CREATE TABLE FavoriteMovies (
FavoriteMovie_id INT NOT NULL,
User_id INT NOT NULL,
Movie_id INT NOT NULL,
Score INT,
PRIMARY KEY (FavoriteMovie_id),
CONSTRAINT FK3 FOREIGN KEY (User_id) REFERENCES Users(User_id),
CONSTRAINT FK4 FOREIGN KEY (Movie_id) REFERENCES Movies(Movie_id),
UNIQUE (User_id, Movie_id)
);
INSERT INTO FavoriteMovies (FavoriteMovie_id, User_id, Movie_id, Score) VALUES
(1, 1, 3, 5),
(2, 1, 4, 4),
(3, 2, 5, 3),
(4, 3, 1, 5),
(5, 4, 2, 4);
```

Task 4

The queries below have been adapted to produce more meaningful results, taking into account the structure and data of the original MiniNet database tables.

4.1. Export all data about users in the HD subscriptions

```
SELECT U.User_id, U.Name, U.Email, U.Password, S.SubscriptionType, U.City, U.Country
FROM Users AS U, Subscriptions AS S
WHERE U.Subscription_id = S.Subscription_id
```

```
AND S.SubscriptionType = 'HD';
```

4.2 Export all data about actors and their associated movies

```
SELECT A. Actor_id, A.Name, A.City, A.DateOfBirth, M. Movie_id, M.Title, M.Genre,
M.ReleaseDate, M.ContentType
 FROM Actors AS A, Movies AS M, MovieActors AS MA
WHERE MA.Actor_id = A.Actor_id
AND MA.Movie_id = M.Movie_id;
```

4.3 Number of users who left reviews for each movie and average rating for each movie.

```
SELECT M.Title AS MovieTitle,
COUNT(R.Movie_id) AS NumberOfUsers,
ROUND (AVG (R.Rating), 1) AS AverageRating
FROM Reviews R
JOIN Movies M ON R.Movie_id = M.Movie_id
GROUP BY M.Title;
```

4.4 Export all data to show the favorite comedy movies for a specific user.

```
SELECT Users.Name, Movies.Movie_id, Movies.Title, Movies.Genre, Movies.ReleaseDate,
Movies.ContentType FROM Users, Movies, FavoriteMovies
WHERE FavoriteMovies.Movie_id=Movies.Movie_id
AND FavoriteMovies.User_id=Users.User_id
AND Movies.Genre = 'Comedy';
```

4.5 Export all data to count how many subscriptions and their costs are in the database per country.

```
SELECT Country, COUNT(*) AS SubscriptionCount,
SUM(Subscriptions.Price) AS TotalSubscriptionCost
FROM Users
JOIN Subscriptions ON Users.Subscription_id = Subscriptions.Subscription_id
GROUP BY Country;
```

4.6 Export all data to find the movies that start with the keyword ‘The’.

```
SELECT * FROM Movies
WHERE Title LIKE '%The_%';
```

4.7 Export data to find the number of subscriptions per movie category.

```
SELECT M.Genre, COUNT(DISTINCT U.User_id) AS NumberOfSubscriptions
FROM Movies AS M
JOIN Reviews AS R ON M.Movie_id=R.Movie_id
```

```
JOIN Users AS U ON U.User_id=R.User_id
JOIN Subscriptions AS S ON U.Subscription_id = S.Subscription_id
GROUP BY M.Genre;
```

4.8 Export data to find the username and the city of the youngest customer in the UHD subscription category.

```
SELECT U.User_id, U.Name, U.Email, U.Password, S.SubscriptionType, U.City, U.Country, U.Age
FROM Users AS U
JOIN Subscriptions AS S ON U.Subscription_id = S.Subscription_id
WHERE U.Age = (SELECT MIN(Age) FROM Users)
AND S.SubscriptionType = 'UHD';
```

4.9 Export data to find username and subscription type for users between 22 - 30 years old (including 22 and 30).

```
SELECT U.Name, U.Age, S.SubscriptionType
FROM Users AS U, Subscriptions AS S
WHERE U.Subscription_id = S.Subscription_id
AND U.Age >=22 and U.Age <=30;
```

4.10 Export data to find the average age of users with high score reviews (more than 3). Group your data for users under 25, 25-30, and 31 and over.

```
SELECT
CASE
WHEN U.Age < 25 THEN 'Under 25'
WHEN U.Age BETWEEN 25 AND 30 THEN '25-30'
ELSE '31 and over'
END AS AgeGroup,
COUNT(*) AS NumberOfUsers,
ROUND(AVG(U.Age),1) AS AverageAge
FROM Users AS U
JOIN Reviews AS R ON U.User_id = R.User_id
WHERE R.Rating > 3
GROUP BY AgeGroup;
```

Task 5

Python scripts to run the Task 4 queries 1-5 with request user input (file containing the complete script for running queries 4.1-4.5 mininet_queries.ipynb attached).

4.1 Export all data about the specific user in the HD subscriptions.

```
def showUserInformation():
    UserName = input("What is your name (e.g. 'mary_james')?")
    query ="""
    SELECT U.User_id, U.Name, U.Email, U.Password, S.SubscriptionType, U.City,
U.Country
    FROM Users AS U, Subscriptions AS S
    WHERE U.Subscription_id = S.Subscription_id
    AND S.SubscriptionType = 'HD'
    AND U.Name = %s;
    """
    cursor= connection.cursor()
    cursor.execute(query, (UserName,))
    results = cursor.fetchall()

    if results:
        print("User Information:")
        for result in results:
            print(result)
    else:
        print("No user found with given name and HD subscription")

    cursor.close()
showUserInformation()
```

Input:
john_doe
Output:
User Information:
(1, 'john_doe', 'john@example.com', 'password1', 'HD', 'New York', 'USA')

4.2 Export all data about actors and their associated movies

```
def showActorMovieInformation():
    ActorName = input("What is actor's name (e.g. 'Millie Bobby Brown')?")
    query ="""
    SELECT A. Actor_id, A.Name, A.City, A.DateOfBirth, M. Movie_id, M.Title,
M.Genre, M.ReleaseDate, M.ContentType
    FROM Actors AS A, Movies AS M, MovieActors AS MA
    WHERE MA.Actor_id = A.Actor_id
    AND MA.Movie_id = M.Movie_id
    AND A.Name = %s;
    """
    cursor= connection.cursor()
    cursor.execute(query, (ActorName,))
    results = cursor.fetchall()

    if results:
        print("Actor Information:")
        for result in results:
            print(result)
    else:
        print("No actor found with given name")

    cursor.close()
showActorMovieInformation()
```

Input:
Millie Bobby Brown
Output:
(1, 'Millie Bobby Brown', 'Los Angeles', datetime.date(2004, 2, 19), 1, 'Stranger Things', 'Sci-Fi', datetime.date(2016, 7, 15), 'TV show')

4.3 Export all data to group actors from a specific city, showing also the average age (per city)

```
def showNumberOfActorsCity():
    City = input("What is city name (e.g. 'Hollywood')?")
    query ="""
    SELECT City, COUNT(City) AS NumberOfActors, ROUND (AVG
(DATEDIFF(CURRENT_DATE, DateOfBirth) / 365.25), 1) AS AverageAge
    FROM Actors
    WHERE City = %s
    GROUP BY City;
    """
    cursor= connection.cursor()
    cursor.execute(query, (City,))
    results = cursor.fetchall()

    if results:
        print("Number of actors from a specific city with average age:")
        for result in results:
            print(result)
    else:
        print("No city found with given name")
    cursor.close()
showNumberOfActorsCity()
```

Input:
Hollywood
Output:
Number of actors from a specific city with average age:
('Hollywood', 1, Decimal('68.3'))

4.4 Export all data to show the favorite genre movies for a specific user

```
def showUserGenreMovies():
    UserName = input("What is user's name (e.g. 'mary_james')?")
    Genre = input("What is your favorite genre?")
    query ="""
    SELECT Users.Name, Movies.Movie_id, Movies.Title, Movies.Genre,
Movies.ReleaseDate, Movies.ContentType FROM Users, Movies, FavoriteMovies
    WHERE FavoriteMovies.Movie_id=Movies.Movie_id
    AND FavoriteMovies.User_id=Users.User_id
    AND Users.Name = %s
    AND Genre = %s;
    """
    cursor= connection.cursor()
    cursor.execute(query, (UserName, Genre))
    results = cursor.fetchall()

    if results:
        print("Movies in the user's favorite genre:")
```

```
        for result in results:
            print(result)
    else:
        print("No users found with given username")

    cursor.close()
showUserGenreMovies()
```

Input:

```
john_doe

Comedy
```

Output:

```
Movies in the user's favorite genre:
('john_doe', 3, 'The Office', 'Comedy', datetime.date(2005, 3, 24), 'TV show')
('john_doe', 4, 'Parks and Recreation', 'Comedy', datetime.date(2009, 4, 9),
'Movie')
```

4.5 Export all data to count how many subscriptions are in the database per country

```
def showSubscriptionsPerCountry():
    Country = input("What is country's name (e.g. 'USA')?")
    query ="""
    SELECT Country, COUNT(*) AS SubscriptionCount
    FROM Users
    JOIN Subscriptions ON Users.Subscription_id = Subscriptions.Subscription_id
    WHERE Country = %s
    GROUP BY Country;
    """
    cursor= connection.cursor()
    cursor.execute(query, (Country,))
    results = cursor.fetchall()

    if results:
        print("Number of subscriptions for the specific country:")
        for result in results:
            print(result)
    else:
        print("No country found with given name")
    cursor.close()
showSubscriptionsPerCountry()
```

Input:
USA
Output:

```
Number of subscriptions for the specific country:
('USA', 5)
```

Task 6.1

Create the users table in Apache Cassandra and generate the following queries in CQL:

6.1 Provide the CREATE statement in CQL.

```
CREATE KEYSPACE mininet_db
WITH REPLICATION = {
'class' : 'SimpleStrategy',
'replication_factor' : 3
};
USE mininet_db;
CREATE TABLE mininet_db.Users (
User_id UUID,
Name text,
Email text,
Password text,
City text,
Country text,
Age int,
PRIMARY KEY (User_id));

INSERT INTO mininet_db.Users (User_id, Name, Email, Password, City, Country, Age)
VALUES
(uuid(), 'john_doe', 'john@example.com', 'password1', 'New York', 'USA', 28);
INSERT INTO mininet_db.Users (User_id, Name, Email, Password, City, Country, Age)
VALUES
(uuid(), 'alice_smith', 'alice@example.com', 'password2', 'Los Angeles', 'USA', 34);
INSERT INTO mininet_db.Users (User_id, Name, Email, Password, City, Country, Age)
VALUES
(uuid(), 'jane_doe', 'jane@example.com', 'password3', 'Chicago', 'USA', 21);
INSERT INTO mininet_db.Users (User_id, Name, Email, Password, City, Country, Age)
VALUES
(uuid(), 'bob_jones', 'bob@example.com', 'password4', 'Boston', 'USA', 30);
INSERT INTO mininet_db.Users (User_id, Name, Email, Password, City, Country, Age)
VALUES
(uuid(), 'emma_johnson', 'emma@example.com', 'password5', 'San Francisco', 'USA', 25);
INSERT INTO mininet_db.Users (User_id, Name, Email, Password, City, Country, Age)
VALUES
(uuid(), 'mary_james', 'mary@example.com', 'password6', 'Toronto', 'Canada', 27);
```

6.2 Export all users.

```
SELECT* FROM mininet_db.Users;
```

6.3 Export all users from a specific country (e.g. 'Canada').

```
CREATE INDEX ON mininet_db.Users (Country);
SELECT * FROM mininet_db.Users WHERE Country = 'Canada';
```

6.4 Export data to find users between 22-30 years old (including 22 and 30).

```
CREATE INDEX ON mininet_db.Users (Age);
SELECT * FROM mininet_db.Users WHERE Age >= 22 AND Age <= 30 ALLOW
FILTERING;
```

6.5 Count how many users exist per specific city (e.g. City = 'Toronto'):

```
CREATE INDEX ON mininet_db.Users (City);
SELECT Count(*) FROM mininet_db.Users
WHERE City = 'Toronto' ALLOW FILTERING;
```

Task 6.2

Generate the queries 6.2 -6.5 and Python.

```
# Connect to mininet_db
session = cluster.connect('mininet_db')
session.set_keyspace('mininet_db')

# Use the preffered keyspace
session.execute('USE mininet_db')

# Run a query “Export all users”
rows = session.execute('SELECT * FROM mininet_db .Users')

# Iterate and show the query response
print("Export all users:")
for i in rows:
    print(i)

# Run a query “Export all users from a specific country”
rows = session.execute("SELECT * FROM mininet_db.Users WHERE Country = 'Canada'")

# Iterate and show the query response
print("Export all users from a specific country (e.g. 'Canada'):")
for i in rows:
    print(i)

# Run a query “Export data to find users between 22-30 years old (including 22 and 30)”
rows = session.execute('SELECT * FROM mininet_db.Users  WHERE Age >= 22 AND Age <=
30 ALLOW FILTERING')

# Iterate and show the query response
print("Export data to find users between 22-30 years old (including 22 and 30)")
for i in rows:
```

```
    print(i)

# Run a query “Count how many users exist per specific city (e.g. City = 'Toronto')”
rows = session.execute("SELECT COUNT(*) FROM mininet_db.Users WHERE City =
'Toronto' ALLOW FILTERING")

# Iterate and show the query response
print("Count how many users exist per specific city (e.g. City = 'Toronto')")
for i in rows:
    print(i)
```