
Set-exercise 1

Find my initial ERD in Appendix A

UNF

TrailID
TrailName
Length
ElevationGain
Route Type
Description
Difficulty
UserEmail
City
County
Country

1NF
To transition from UNF to 1NF I needed to:

- Eliminate repeating groups or arrays.
- Ensure that all columns contain atomic (indivisible) values.
- Choose a primary key for the new relation

In this case it meant separation out the location attributes into a separate table and link them with a Location ID (the primary key of the Location table and a foreign key in the trail table)

TrailID
TrailName
Length
ElevationGain
Route Type
Description
Difficulty
Description
UserEmail
LocationID*

LocationID
City
County
Country

2NF
To transition from 1NF to 2NF I needed to remove any part-key dependencies; however, in the 1NF form there were no part-key dependencies thus the structure remained the same.

<u>TrailID</u>
TrailName
Length
ElevationGain
Route Type
Description
Difficulty
Description
UserEmail

<u>LocationID</u>
City
County
Country

3NF
To transition from 2NF to 3NF I needed to remove any transitive dependencies; however in the 2NF form there were no transitive dependencies thus the structure remained the same. In the Trail table, all the non-key attributes (TrailName, Length, ElevationGain, RouteType, Difficulty, UserEmail) depend directly on the TrailID. There are no non-key attributes depending on other non-key attributes, thus no transitive dependencies exist. In the Location table, Country, County, and City depend only on LocationID and not on each other. Therefore, there are no transitive dependencies in the Location table either.

<u>TrailID</u>
TrailName
Length
ElevationGain
Route Type
Description
Difficulty
Description
UserEmail

<u>LocationID</u>
City
County
Country

My 3NF ERD can be shown in Appendix B

Assumptions:

- TrailID is used as the unique auto-incrementing primary key because it's assumed that it is hypothetically possible that there could be 2 trails with the same name e.g. two places with the same name in different countries
- It is assumed that every trail has a creator/owner who can be uniquely identified in this case by a UserEmail
- Length is in a consistent unit (e.g: kilometres), ensuring uniformity and removing the need for an additional attribute to identify the unit type.
- Elevation gain is in a consistent unit (e.g: metres), ensuring uniformity and removing the need for an additional attribute to identify the unit type.
- difficulty represents a standardised difficulty level (e.g: "Easy", "Moderate", "Difficult").
- LocationID is an auto-incrementing integer, uniquely identifying each location.
- Each unique combination of Country, County, and City is uniquely identifiable LocationID.
- A trail is associated with one location only e.g. the start point

---

Set-exercise 2

The Final ERD can be found in Appendix C

1. **User-to-Trail Relationship**:

One user can be the owner of multiple trails, creating a **one-to-many** relationship between User and Trail. This makes sense because, in this system, a user is responsible for creating and managing trails, but a trail can only have one creator/owner. This avoids the issues associated with many-to-many relationships, where multiple users would own the same trail, which isn't relevant for this design. Each trail belongs to a single user, and users can have multiple trails tied to their account.

2. **Location-to-Trail Relationship**:

Each location can be used by many different trails, resulting in a **one-to-many** relationship between Location and Trail. A location represents a place where a trail exists but many trails could share the same location. Multiple trails might start at the same park or area, so it's important that the location is linked to many trails, but each trail is tied to one location(**many-to-one**).

3. **Key Structure**:

The User **entity** uses UserEmail as its **primary key**, which ensures that each user is uniquely identified by their email address. This UserEmail is then used as a **foreign key** in the Trail table, connecting the trail back to its creator.
In the same way, LocationID is the **primary key** in the Location table, which uniquely identifies each location; this LocationID is also a **foreign key** in the Trail table, linking each trail to its specific location. **This structure avoids redundancy** as the Trail table references both the user and location through foreign keys, ensuring **referential integrity**.

4. **Assumptions**:

- Each trail can have only **one creator (user)**, preventing the need for a many-to-many relationship between User and Trail.
- Each trail can only have **one location**, simplifying the model.
- LocationID is **auto-incrementing**, ensuring that each location is unique.
- UserEmail is assumed to be a valid and unique identifier for each user as it is generally not allowed for the same email address to be linked to multiple accounts.

5. **Justification**:

• This design is optimal for the requirements of the micro-service because it simplifies the relationships between users, trails, and locations. By sticking to **one-to-many** relationships, it avoids unnecessary complexity and makes it easier to manage data integrity.

Set-exercise 3

Field definition grid is located in Appendix D

Set-exercise 4

THE SQL DOCUMENTATION IS IN THE COMMENTS IN THE CODE BELOW

- **[X] All the following sql queries have been successfully executed; you can see this was on the university database through Appendix D**
- **[X] Go back and fix SQL data to match my SQL in the University Microsoft Azure Database:**
  "*Note your SQL Scripts must match the implementation on the module hosted Microsoft SQL*
  Server EXACTLY."
- **Note that the screenshots are when I did this on localhost. However I later on came into the university to do the exact same queries in the DIST-6-505.uopnet.plymouth.ac.uk database.**

2001 Set Exercises

Run Cancel Disconnect Change Database: master Estimated Plan
Enable Actual Plan Parse Enable SQLCMD To Notebook

```sql
IF NOT EXISTS (SELECT * FROM sys.schemas WHERE name = 'CW1') -- if the schema doesn't alrea
BEGIN -- begin the following block of code
    EXEC('CREATE SCHEMA CW1') -- Execute schema creation CW1
END
```

Messages

1:19:04 PM Started executing query at Line 1
Commands completed successfully.
Total execution time: 00:00:00.010

Run Cancel Disconnect Change Database: master Estimated Plan Enable Actual Plan Parse Enable SQLCMD To Notebook

```sql
CREATE TABLE CW1.Location ( -- 100 == safe bet for all location columns
    LocationID INT PRIMARY KEY IDENTITY(1,1),
    City VARCHAR(100) NOT NULL,
    County VARCHAR(100) NOT NULL,
    Country VARCHAR(100) NOT NULL
);

CREATE TABLE CW1.[User] ( -- 254 == legal email maximum length; 100 == safe bet for username; 255 == safe bet for hashed password
    Email NVARCHAR(254) PRIMARY KEY,
    UserName VARCHAR(100) NOT NULL,
    Password VARCHAR(255) NOT NULL
);

CREATE TABLE CW1.Trail (
    TrailID INT PRIMARY KEY IDENTITY(1,1),
    TrailName VARCHAR(75) NOT NULL, -- can't find any names above 50 characters; 75 == safe bet
    Length FLOAT NOT NULL,
    ElevationGain INT NOT NULL,
    RouteType VARCHAR(15) NOT NULL, -- Out & Back || Loop || Point to Point
    Description VARCHAR(500), -- Average character-length of a description is around 400; thus 500 == safe bet
    Difficulty VARCHAR(10) NOT NULL, -- Easy || Moderate || Hard
    LocationID INT NOT NULL,
    UserEmail NVARCHAR(254) NOT NULL, -- 254 is legal maximum length for an email address
    FOREIGN KEY (LocationID) REFERENCES CW1.Location(LocationID),
    FOREIGN KEY (UserEmail) REFERENCES CW1.[User](Email)
);
```

Messages

2:10:54 PM Started executing query at Line 1
Commands completed successfully.
Total execution time: 00:00:00.020

2001 Set Exercises

Run Cancel Disconnect Change Database: master Estimated Plan Enable Actual Plan Parse Enable SQLCMD To Notebook

```sql
-- Insert sample data into Location table
INSERT INTO CW1.Location (City, County, Country) VALUES
('London', 'Greater London', 'United Kingdom'),
('Plymouth', 'Devon', 'United Kingdom'),
('Manchester', 'Greater Manchester', 'United Kingdom'),
('Edinburgh', 'Midlothian', 'United Kingdom'),
('Cardiff', 'South Glamorgan', 'United Kingdom');

-- Insert sample data into User table
INSERT INTO CW1.[User] (Email, UserName, Password) VALUES
('john@example.com', 'JohnDoe', 'hashedpassword1'),
('jane@example.com', 'JaneSmith', 'hashedpassword2'),
('mike@example.com', 'MikeJohnson', 'hashedpassword3'),
('sarah@example.com', 'SarahBrown', 'hashedpassword4'),
('alex@example.com', 'AlexWilson', 'hashedpassword5');

-- Insert sample data into Trail table
INSERT INTO CW1.Trail (TrailName, Length, ElevationGain, RouteType, Description, Difficulty, LocationID, UserEmail) VALUES
('Thames Path', 8.5, 50, 'Out & Back', 'Scenic walk along the River Thames in London', 'Easy', 1, 'john@example.com'),
('Dartmoor Way', 15.0, 500, 'Loop', 'Circular route showcasing Dartmoor National Park near Plymouth', 'Moderate', 2, 'jane@example.com'),
('Pennine Way', 20.0, 800, 'Point to Point', 'Part of the famous trail near Manchester', 'Difficult', 3, 'mike@example.com'),
('Arthurs Seat', 4.5, 250, 'Loop', 'Iconic hill walk in Edinburgh with panoramic views', 'Moderate', 4, 'sarah@example.com'),
('Taff Trail', 12.0, 150, 'Out & Back', 'Popular cycling and walking route in Cardiff', 'Easy', 5, 'alex@example.com');
```

Messages

2:12:00 PM Started executing query at Line 1
(5 rows affected)
(5 rows affected)
(5 rows affected)
Total execution time: 00:00:00.032

Run Cancel Disconnect Change Database: master Estimated Plan Enable Actual Plan Parse Enable SQLCMD To Notebook

```sql
SELECT * FROM CW1.Location;
SELECT * FROM CW1.[User];
SELECT * FROM CW1.Trail;
```

Results Messages

	LocationID	City	County	Country
1	1	London	Greater London	United Kingdom
2	2	Plymouth	Devon	United Kingdom
3	3	Manchester	Greater Manchester	United Kingdom
4	4	Edinburgh	Midlothian	United Kingdom
5	5	Cardiff	South Glamorgan	United Kingdom

	Email	UserName	Password
1	alex@example.com	AlexWilson	hashedpassword5
2	jane@example.com	JaneSmith	hashedpassword2
3	john@example.com	JohnDoe	hashedpassword1
4	mike@example.com	MikeJohnson	hashedpassword3
5	sarah@example.com	SarahBrown	hashedpassword4

	TrailID	TrailName	Length	ElevationGain	RouteType	Description	Difficulty	LocationID	UserEmail
1	1	Thames Path	8.5	50	Out & Back	Scenic walk along the River Thames in London	Easy	1	john@example.com
2	2	Dartmoor Way	15	500	Loop	Circular route showcasing Dartmoor National Park nea…	Moderate	2	jane@example.com
3	3	Pennine Way	20	800	Point to Point	Part of the famous trail near Manchester	Difficult	3	mike@example.com
4	4	Arthurs Seat	4.5	250	Loop	Iconic hill walk in Edinburgh with panoramic views	Moderate	4	sarah@example.com
5	5	Taff Trail	12	150	Out & Back	Popular cycling and walking route in Cardiff	Easy	5	alex@example.com

---

2001 Set Exercises

Run Cancel Disconnect Change Database: master Estimated Plan Enable Actual Plan Parse Enable SQLCMD To Notebook

```sql
-- query to show the number of trails per user -
SELECT
    u.Email,
    u.UserName,
    COUNT(t.TrailID) AS NumberOfTrails
FROM
    CW1.[User] u
    LEFT JOIN CW1.Trail t ON u.Email = t.UserEmail
GROUP BY
    u.Email, u.UserName
ORDER BY
    NumberOfTrails DESC;
```

Results Messages

	Email	UserName	NumberOfTrails
1	alex@example.com	AlexWilson	1
2	jane@example.com	JaneSmith	1
3	john@example.com	JohnDoe	1
4	mike@example.com	MikeJohnson	1
5	sarah@example.com	SarahBrown	1

Run Cancel Disconnect Change Database: master Estimated Plan Enable Actual Plan Parse Enable SQLCMD To Notebook

```sql
-- query to show the number of trails per location -
SELECT
    l.LocationID,
    l.City,
    l.County,
    l.Country,
    COUNT(t.TrailID) AS NumberOfTrails
FROM
    CW1.Location l
    LEFT JOIN CW1.Trail t ON l.LocationID = t.LocationID
GROUP BY
    l.LocationID, l.City, l.County, l.Country
ORDER BY
    NumberOfTrails DESC;
```

Results Messages

	LocationID	City	County	Country	NumberOfTrails
1	1	London	Greater London	United Kingdom	1
2	2	Plymouth	Devon	United Kingdom	1
3	3	Manchester	Greater Manchester	United Kingdom	1
4	4	Edinburgh	Midlothian	United Kingdom	1
5	5	Cardiff	South Glamorgan	United Kingdom	1

2001 Set Exercises

Run Cancel Disconnect Change Database: master Estimated Plan Enable Actual Plan Parse Enable SQLCMD To Notebook

```sql
-- query to show trails with their difficulty levels and creators -
SELECT
    t.TrailName,
    t.Difficulty,
    t.Length,
    u.UserName AS Creator,
    l.City,
    l.Country
FROM
    CW1.Trail t
    JOIN CW1.[User] u ON t.UserEmail = u.Email
    JOIN CW1.Location l ON t.LocationID = l.LocationID
ORDER BY
    t.Difficulty, t.Length DESC; -- DESC = descending order
```

Results Messages

	TrailName	Difficulty	Length	Creator	City	Country
1	Pennine Way	Difficult	20	MikeJohnson	Manchester	United Kingdom
2	Taff Trail	Easy	12	AlexWilson	Cardiff	United Kingdom
3	Thames Path	Easy	8.5	JohnDoe	London	United Kingdom
4	Dartmoor Way	Moderate	15	JaneSmith	Plymouth	United Kingdom
5	Arthurs Seat	Moderate	4.5	SarahBrown	Edinburgh	United Kingdom

---

CONNECTIONS SERVERS localhost, 1434... Databases Security Server Objects

Welcome localhost, 1434 SQLQuery_1 - (54) l...er (SA) 8

Run Cancel Disconnect Change Database: master Estimated Plan Enable Actual Plan Parse Enable SQLCMD To Notebook

```sql
-- Check current data in Trail table
SELECT * FROM CW1.Trail;

-- Verify row counts individually
SELECT COUNT(*) AS LocationCount FROM CW1.Location;
SELECT COUNT(*) AS UserCount FROM CW1.[User];
SELECT COUNT(*) AS TrailCount FROM CW1.Trail;

-- Check for NULL values
SELECT * FROM CW1.Trail WHERE TrailName IS NULL OR Length IS NULL OR Difficulty IS NULL;
```

Results Messages

	TrailID	TrailName	Length	ElevationGain	RouteType	Description	Difficulty	LocationID	UserEmail
1	1	Thames Path	8.5	50	Out & Back	Scenic walk along the River Thames in London	Easy	1	john@example.com
2	2	Dartmoor Way	15	500	Loop	Circular route showcasing Dartmoor National Park nea…	Moderate	2	jane@example.com
3	3	Pennine Way	20	800	Point to Point	Part of the famous trail near Manchester	Difficult	3	mike@example.com
4	4	Arthurs Seat	4.5	250	Loop	Iconic hill walk in Edinburgh with panoramic views	Moderate	4	sarah@example.com
5	5	Taff Trail	12	150	Out & Back	Popular cycling and walking route in Cardiff	Easy	5	alex@example.com

	LocationCount
1	5

	UserCount
1	5

	TrailCount
1	5

TrailID	TrailName	Length	ElevationGain	RouteType	Description	Difficulty	LocationID	UserEmail

AZURE

Ln 11, Col 1 Spaces: 4 UTF-8 LF 8 rows MSSQL 00:00:00 localhost, 1434 : master (54)

2001 Set Exercises

This should fail:

Run Cancel Disconnect Change Database: master Estimated Plan Enable Actual Plan Parse Enable SQLCMD To Notebook

```sql
-- This should fail
INSERT INTO CW1.Trail (TrailName, Length, ElevationGain, RouteType, Difficulty, LocationID, UserEmail)
VALUES ('Test Trail', 10, 100, 'Loop', 'Easy', 999, 'nonexistent@example.com');
```

Messages

2:53:09 PM Started executing query at Line 1

Msg 547, Level 16, State 0, Line 2

The INSERT statement conflicted with the FOREIGN KEY constraint "FK__Trail__LocationI__2F2FFC0C". The conflict occurred in database "master", table "CW1.Location", column 'LocationID'.

The statement has been terminated.

Total execution time: 00:00:00.008

.sql code: {

```sql
-- Creates the CW1 schema if it doesn't already exist
IF NOT EXISTS (SELECT * FROM sys.schemas WHERE name = 'CW1') -- if the schema
doesn't already exist in the systems catalogue; select everything from sys.schemas
where the name is CW1
BEGIN -- begin the following block of code
   EXEC('CREATE SCHEMA CW1') -- Execute schema creation CW1
END -- end if statement
--------------------------------------------------------------------------------
-----------------------------------------------
-- Create Trail table in CW1 schema
/*
   This table state TrailID as the primary key with auto-increment; TrailName,
Length, ElevationGain, RouteType,
   Difficulty, and UserEmail as NOT null constraints, and their respective data
types and lengths. Description
   is also included as a column with a maximum length. LocationID and UserEmail are
foreign keys that reference
   the Location and User table, respectively.
*/
-- Create Location table in CW1 schema
/*
   This table state LocationID as the primary key with auto-increment, City,
County, and Country as not null constraints,
   and their respective data types and lengths.
*/
CREATE TABLE CW1.Location ( -- 100 == safe bet for all location columns
   LocationID INT PRIMARY KEY IDENTITY(1,1),
   City VARCHAR(100) NOT NULL,
   County VARCHAR(100) NOT NULL,
```

```
   Country VARCHAR(100) NOT NULL
);
--------------------------------------------------------------------------------
-------------------------------------------------
-- Create User table in CW1 schema
/*
   This table sets Email as the primary key, with UserName and Password as not null
constraints,
   along with their respective data types and lengths. Although NVARCHAR allocates
twice the
   storage space compared to VARCHAR, it is used for the Email column to support
   international characters emails.
*/
CREATE TABLE CW1.[User] ( -- 254 == legal email maximum length; 100 == safe bet for
username; 255 == safe bet for hashed password
   Email NVARCHAR(254) PRIMARY KEY,
   UserName VARCHAR(100) NOT NULL,
   Password VARCHAR(255) NOT NULL
);
--------------------------------------------------------------------------------
-------
CREATE TABLE CW1.Trail (
   TrailID INT PRIMARY KEY IDENTITY(1,1),
   TrailName VARCHAR(75) NOT NULL, -- can't find any names above 50 characters; 75
== safe bet
   Length FLOAT NOT NULL,
   ElevationGain INT NOT NULL,
   RouteType VARCHAR(15) NOT NULL, -- Out & Back || Loop || Point to Point
   Description VARCHAR(500), -- Average character-length of a description is around
400; thus 500 == safe bet
   Difficulty VARCHAR(10) NOT NULL, -- Easy || Moderate || Hard
   LocationID INT NOT NULL,
   UserEmail NVARCHAR(254) NOT NULL, -- 254 is legal maximum length for an email
address
   FOREIGN KEY (LocationID) REFERENCES CW1.Location(LocationID),
   FOREIGN KEY (UserEmail) REFERENCES CW1.[User](Email)
);
--------------------------------------------------------------------------------
-------
-- Insert sample data into Location table
INSERT INTO CW1.Location (City, County, Country) VALUES
('London', 'Greater London', 'United Kingdom'),
('Plymouth', 'Devon', 'United Kingdom'),
('Manchester', 'Greater Manchester', 'United Kingdom'),
('Edinburgh', 'Midlothian', 'United Kingdom'),
```

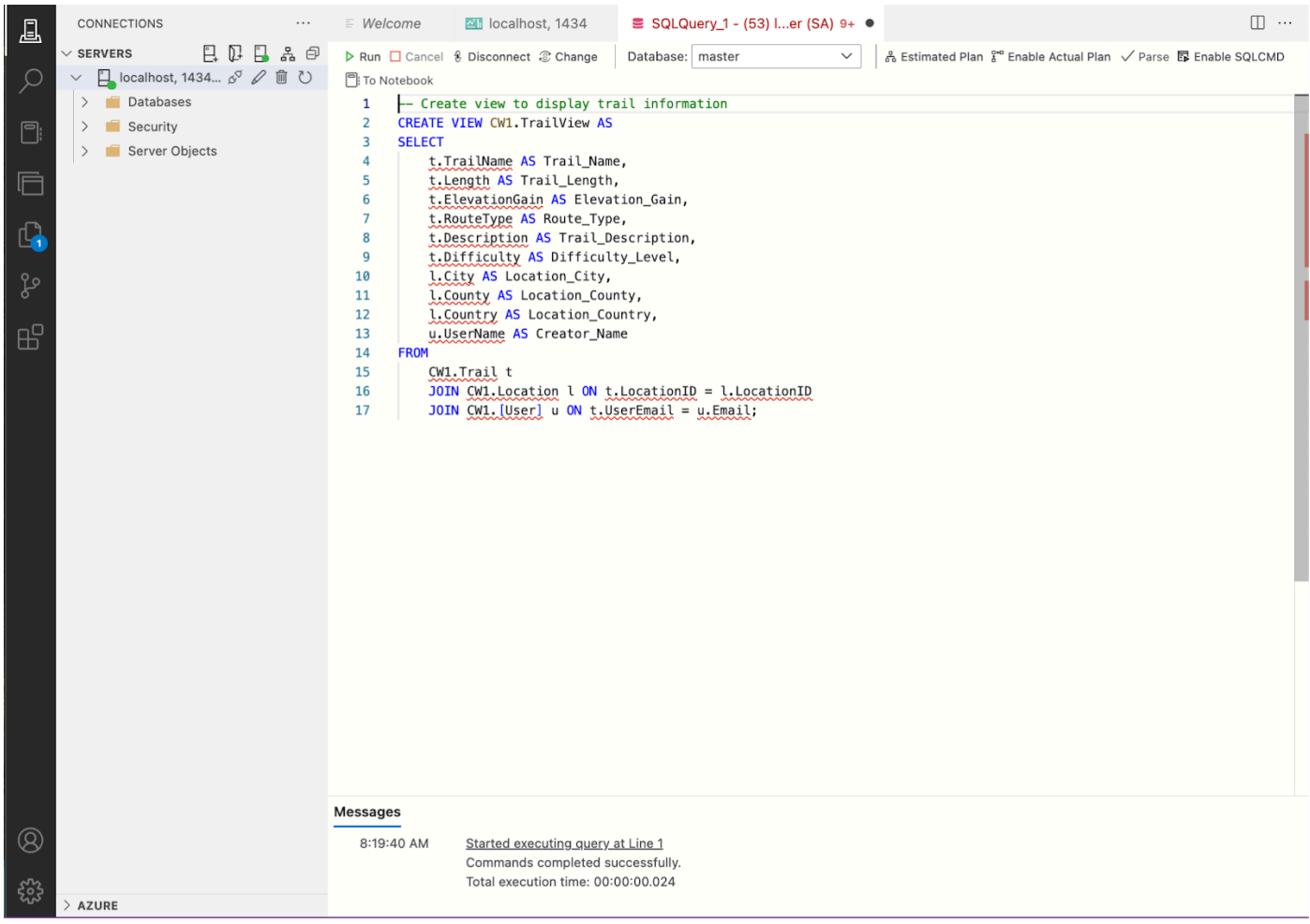
```
('Cardiff', 'South Glamorgan', 'United Kingdom');

-- Insert sample data into User table
INSERT INTO CW1.[User] (Email, UserName, Password) VALUES
('john@example.com', 'JohnDoe', 'hashedpassword1'),
('jane@example.com', 'JaneSmith', 'hashedpassword2'),
('mike@example.com', 'MikeJohnson', 'hashedpassword3'),
('sarah@example.com', 'SarahBrown', 'hashedpassword4'),
('alex@example.com', 'AlexWilson', 'hashedpassword5');

-- Insert sample data into Trail table
INSERT INTO CW1.Trail (TrailName, Length, ElevationGain, RouteType, Description,
Difficulty, LocationID, UserEmail) VALUES
('Thames Path', 8.5, 50, 'Out & Back', 'Scenic walk along the River Thames in
London', 'Easy', 1, 'john@example.com'),
('Dartmoor Way', 15.0, 500, 'Loop', 'Circular route showcasing Dartmoor National
Park near Plymouth', 'Moderate', 2, 'jane@example.com'),
('Pennine Way', 20.0, 800, 'Point to Point', 'Part of the famous trail near
Manchester', 'Difficult', 3, 'mike@example.com'),
('Arthurs Seat', 4.5, 250, 'Loop', 'Iconic hill walk in Edinburgh with panoramic
views', 'Moderate', 4, 'sarah@example.com'),
('Taff Trail', 12.0, 150, 'Out & Back', 'Popular cycling and walking route in
Cardiff', 'Easy', 5, 'alex@example.com');

-- SELECT statements to demonstrate data in tables
SELECT * FROM CW1.Location;
SELECT * FROM CW1.[User];
SELECT * FROM CW1.Trail;
-- '*' = all columns

-- Analysis queries
-------------------------------------------------------------------------------------
-----------------------------------

-- query to show the number of trails per user -
SELECT
   u.Email,
   u.UserName,
   COUNT(t.TrailID) AS NumberOfTrails
FROM
   CW1.[User] u
   LEFT JOIN CW1.Trail t ON u.Email = t.UserEmail
GROUP BY
   u.Email, u.UserName
ORDER BY
```

```
   NumberOfTrails DESC;

-- query to show the number of trails per location -
SELECT
   l.LocationID,
   l.City,
   l.County,
   l.Country,
   COUNT(t.TrailID) AS NumberOfTrails
FROM
   CW1.Location l
   LEFT JOIN CW1.Trail t ON l.LocationID = t.LocationID
GROUP BY
   l.LocationID, l.City, l.County, l.Country
ORDER BY
   NumberOfTrails DESC;

-- query to show trails with their difficulty levels and creators -
SELECT
   t.TrailName,
   t.Difficulty,
   t.Length,
   u.UserName AS Creator,
   l.City,
   l.Country
FROM
   CW1.Trail t
   JOIN CW1.[User] u ON t.UserEmail = u.Email
   JOIN CW1.Location l ON t.LocationID = l.LocationID
ORDER BY
   t.Difficulty, t.Length DESC; -- DESC = descending order

-- Testing queries
-------------------------------------------------------------------------------------
-----------------------------------

-- Check current data in Trail table
SELECT * FROM CW1.Trail;

-- Verify row counts individually
SELECT COUNT(*) AS LocationCount FROM CW1.Location;
SELECT COUNT(*) AS UserCount FROM CW1.[User];
SELECT COUNT(*) AS TrailCount FROM CW1.Trail;

-- Check for NULL values
```

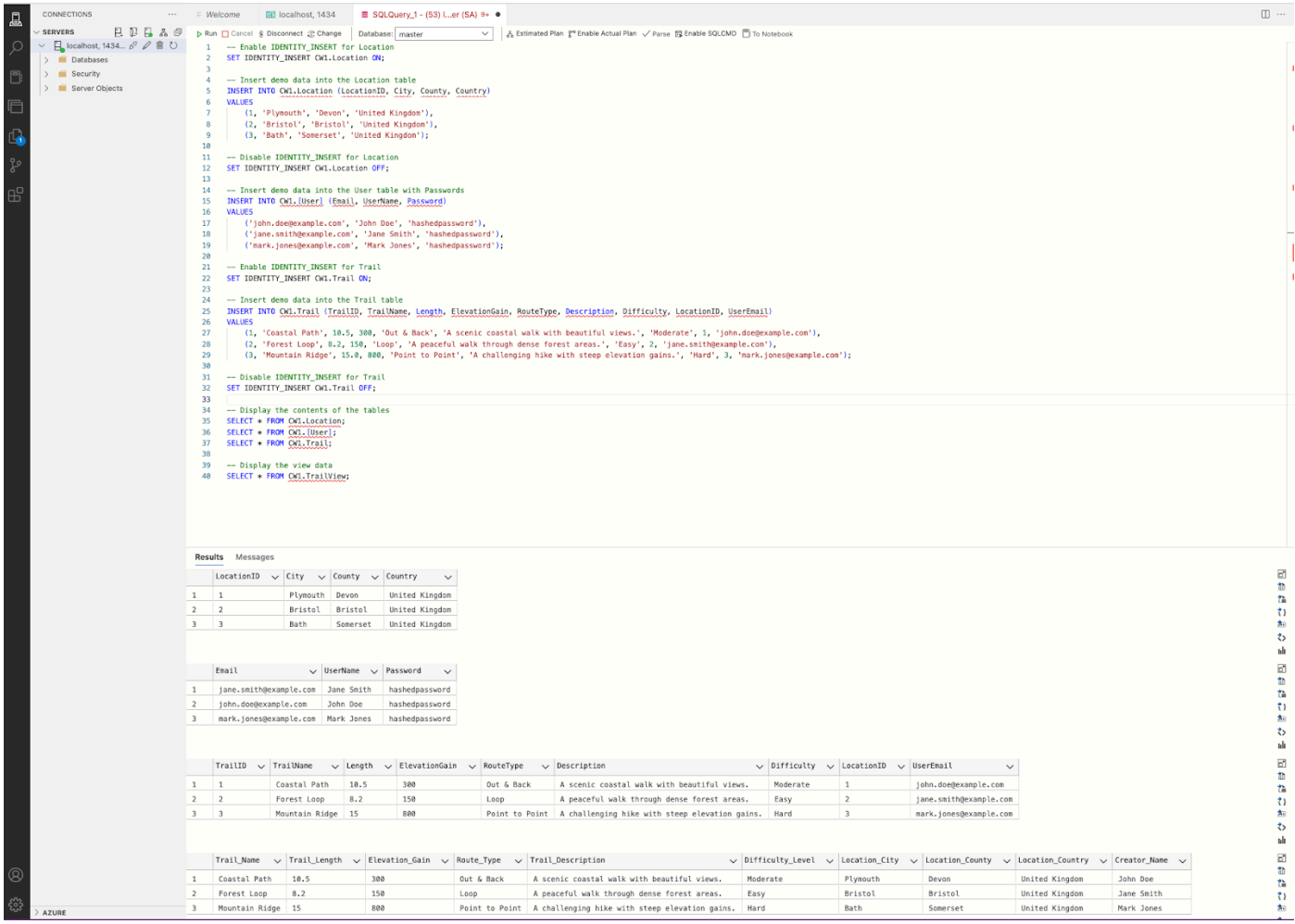
```sql
SELECT * FROM CW1.Trail WHERE TrailName IS NULL OR Length IS NULL OR Difficulty IS NULL;

-- This should fail
INSERT INTO CW1.Trail (TrailName, Length, ElevationGain, RouteType, Difficulty, LocationID, UserEmail)
VALUES ('Test Trail', 10, 100, 'Loop', 'Easy', 999, 'nonexistent@example.com');
```

}

Set-exercise 5



2001 Set Exercises



I needed to use the SET IDENTITY_INSERT commands to manually insert data into identity columns like LocationID and TrailID, data that is normally auto-generated

SQL code {

1st query

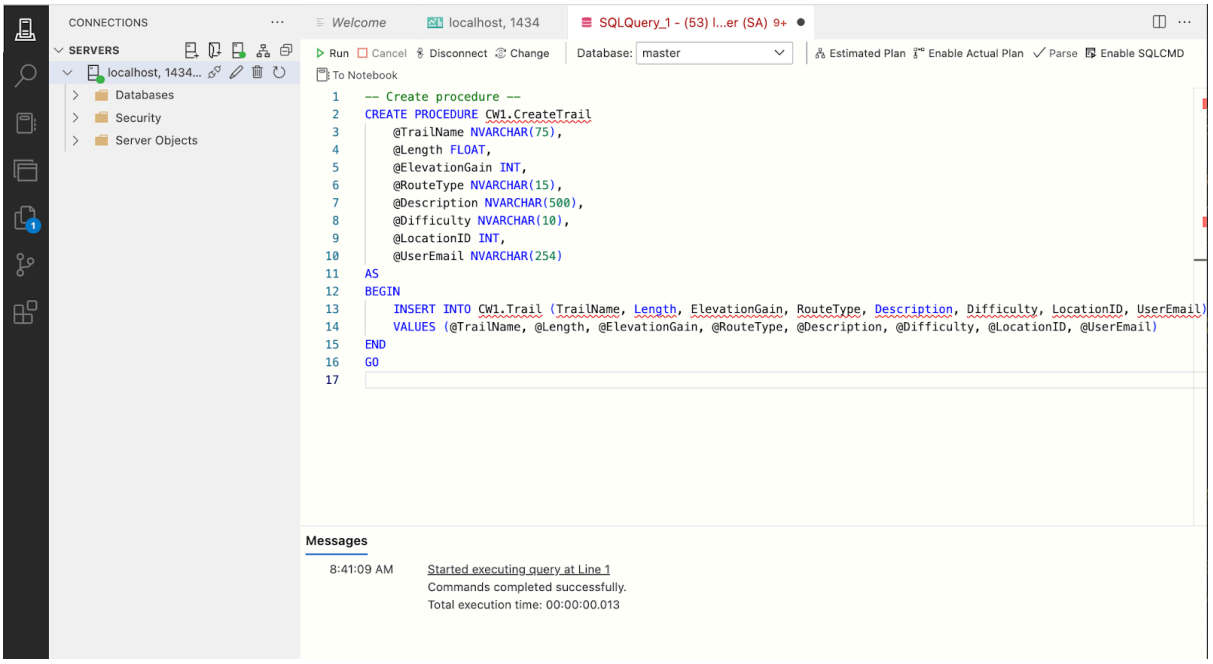
```
-- Create view to display trail information
CREATE VIEW CW1.TrailView AS
SELECT
   t.TrailName AS Trail_Name,
   t.Length AS Trail_Length,
   t.ElevationGain AS Elevation_Gain,
   t.RouteType AS Route_Type,
   t.Description AS Trail_Description,
   t.Difficulty AS Difficulty_Level,
   l.City AS Location_City,
   l.County AS Location_County,
   l.Country AS Location_Country,
   u.UserName AS Creator_Name
FROM
   CW1.Trail t
   JOIN CW1.Location l ON t.LocationID = l.LocationID
   JOIN CW1.[User] u ON t.UserEmail = u.Email;
```

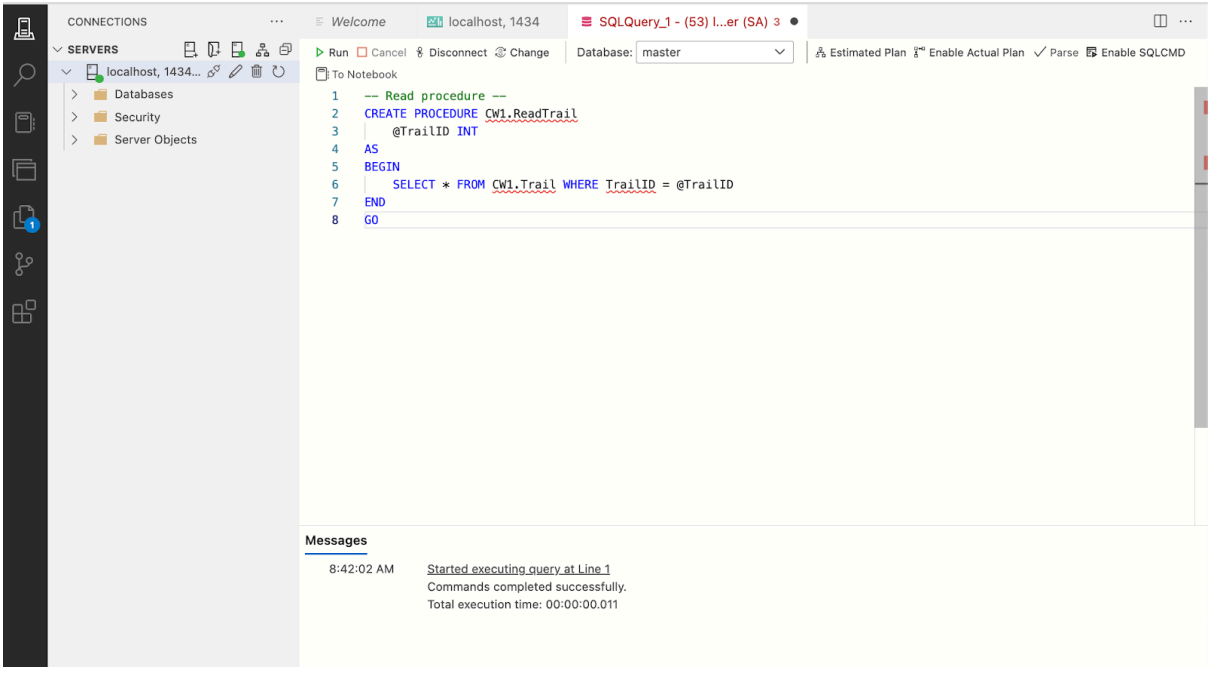
2nd query

```sql
-- Enable IDENTITY_INSERT for Location
SET IDENTITY_INSERT CW1.Location ON;

-- Insert demo data into the Location table
INSERT INTO CW1.Location (LocationID, City, County, Country)
VALUES
   (111, 'Plymouth', 'Devon', 'United Kingdom'),
   (222, 'Bristol', 'Bristol', 'United Kingdom'),
   (333, 'Bath', 'Somerset', 'United Kingdom');

-- Disable IDENTITY_INSERT for Location
SET IDENTITY_INSERT CW1.Location OFF;

-- Insert demo data into the User table with Passwords
INSERT INTO CW1.[User] (Email, UserName, Password)
VALUES
   ('john.doe@example.com', 'John Doe', 'hashedpassword'),
   ('jane.smith@example.com', 'Jane Smith', 'hashedpassword'),
   ('mark.jones@example.com', 'Mark Jones', 'hashedpassword');

-- Enable IDENTITY_INSERT for Trail
SET IDENTITY_INSERT CW1.Trail ON;

-- Insert demo data into the Trail table
INSERT INTO CW1.Trail (TrailID, TrailName, Length, ElevationGain, RouteType,
Description, Difficulty, LocationID, UserEmail)
VALUES
   (6, 'Coastal Path', 10.5, 300, 'Out & Back', 'A scenic coastal walk with
beautiful views.', 'Moderate', 1, 'john.doe@example.com'),
   (7, 'Forest Loop', 8.2, 150, 'Loop', 'A peaceful walk through dense forest
areas.', 'Easy', 2, 'jane.smith@example.com'),
   (8, 'Mountain Ridge', 15.0, 800, 'Point to Point', 'A challenging hike with
steep elevation gains.', 'Hard', 3, 'mark.jones@example.com');

-- Disable IDENTITY_INSERT for Trail
SET IDENTITY_INSERT CW1.Trail OFF;

-- Display the contents of the tables
SELECT * FROM CW1.Location;
SELECT * FROM CW1.[User];
SELECT * FROM CW1.Trail;

-- Display the view data
SELECT * FROM CW1.TrailView;
```

}

Set-exercise 6





.sql code:

{

```
-- Set-exercise 6: Procedures --


-- Create procedure --
CREATE PROCEDURE CW1.CreateTrail
   @TrailName VARCHAR(75),
```

```
    @Length FLOAT,
    @ElevationGain INT,
    @RouteType VARCHAR(15),
    @Description VARCHAR(500),
    @Difficulty VARCHAR(10),
    @LocationID INT,
    @UserEmail NVARCHAR(254)
AS
BEGIN
    INSERT INTO CW1.Trail (TrailName, Length, ElevationGain, RouteType, Description,
Difficulty, LocationID, UserEmail)
    VALUES (@TrailName, @Length, @ElevationGain, @RouteType, @Description,
@Difficulty, @LocationID, @UserEmail)
END
GO

-- Read procedure --
CREATE PROCEDURE CW1.ReadTrail
    @TrailID INT
AS
BEGIN
    SELECT * FROM CW1.Trail WHERE TrailID = @TrailID
END
GO

-- Update procedure --
CREATE PROCEDURE CW1.UpdateTrail
    @TrailID INT,
    @TrailName VARCHAR(75),
    @Length FLOAT,
    @ElevationGain INT,
    @RouteType VARCHAR(15),
    @Description VARCHAR(500),
    @Difficulty VARCHAR(10),
    @LocationID INT,
    @UserEmail NVARCHAR(254)
AS
BEGIN
    UPDATE CW1.Trail
    SET TrailName = @TrailName,
        Length = @Length,
        ElevationGain = @ElevationGain,
        RouteType = @RouteType,
        Description = @Description,
        Difficulty = @Difficulty,
```

```
        LocationID = @LocationID,
        UserEmail = @UserEmail
    WHERE TrailID = @TrailID
END
GO

-- Delete procedure --
CREATE PROCEDURE CW1.DeleteTrail
    @TrailID INT
AS
BEGIN
    DELETE FROM CW1.Trail WHERE TrailID = @TrailID
END
GO
```

}

Set-exercise 7

```
-- Set-exercise 7: Triggers

-- Create log table to store trail addition details
CREATE TABLE CW1.TrailLog (
    LogID INT PRIMARY KEY IDENTITY(1,1),
    TrailID INT,
    UserEmail NVARCHAR(254),
    AddedOn DATETIME,
    FOREIGN KEY (TrailID) REFERENCES CW1.Trail(TrailID),
    FOREIGN KEY (UserEmail) REFERENCES CW1.[User](Email)
);

-- Create trigger to log trail additions
CREATE TRIGGER CW1.LogNewTrail
ON CW1.Trail
AFTER INSERT
AS
BEGIN
    INSERT INTO CW1.TrailLog (TrailID, UserEmail, AddedOn)
    SELECT inserted.TrailID, inserted.UserEmail, GETDATE()
    FROM inserted;
END
GO
```

I created the log table to store the trail addition details already in the university database, and forgot screenshot this; however as we can see, the trigger query successfully ran

Welcome | DIST-6-505.uopnet.plymouth.ac.uk:COMP2001_ANurse | SQLQuery_1 - (80) D...ANurse)

Run | Cancel | Disconnect | Change | Database: COMP2001_ANurse | Estimated Plan | Enable Actual Plan

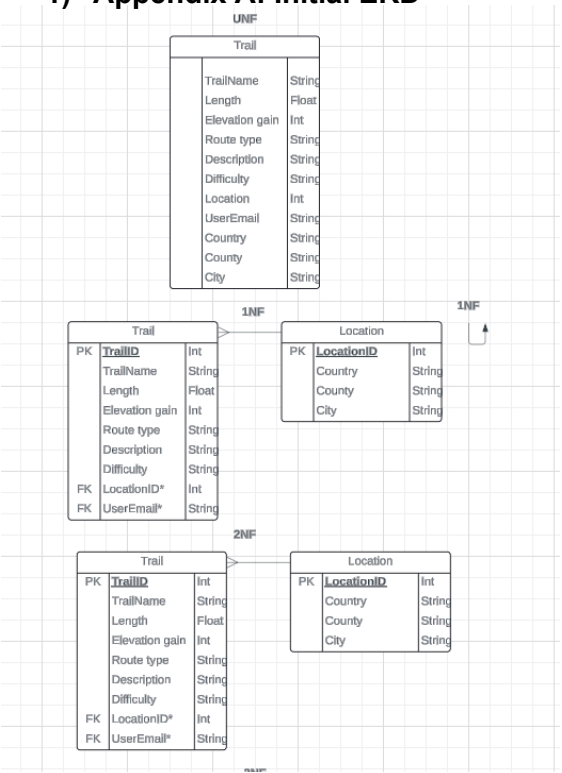
```sql
-- Create trigger to log trail additions
CREATE TRIGGER CW1.LogNewTrail
ON CW1.Trail
AFTER INSERT
AS
BEGIN
    INSERT INTO CW1.TrailLog (TrailID, UserEmail, AddedOn)
    SELECT inserted.TrailID, inserted.UserEmail, GETDATE()
    FROM inserted;
END
GO
```

Messages

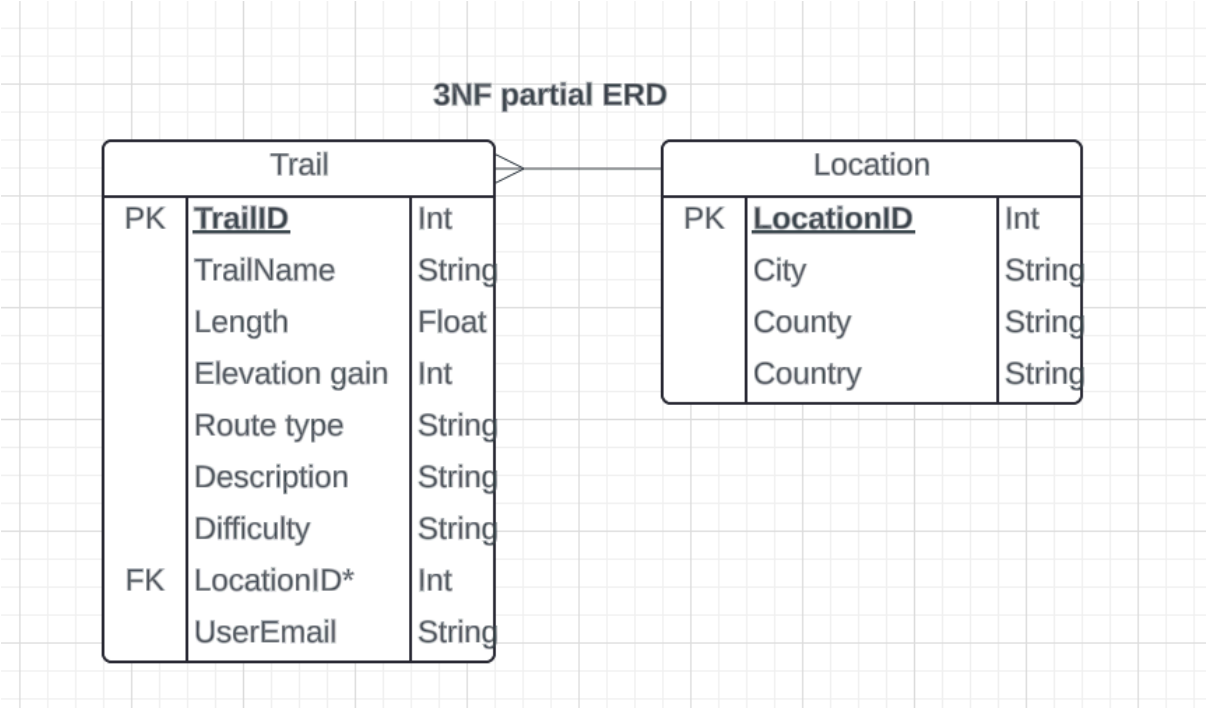
3:28:01 PM Started executing query at Line 1
Commands completed successfully.
Total execution time: 00:00:00.097

Appendix:

1) Appendix A: Initial ERD

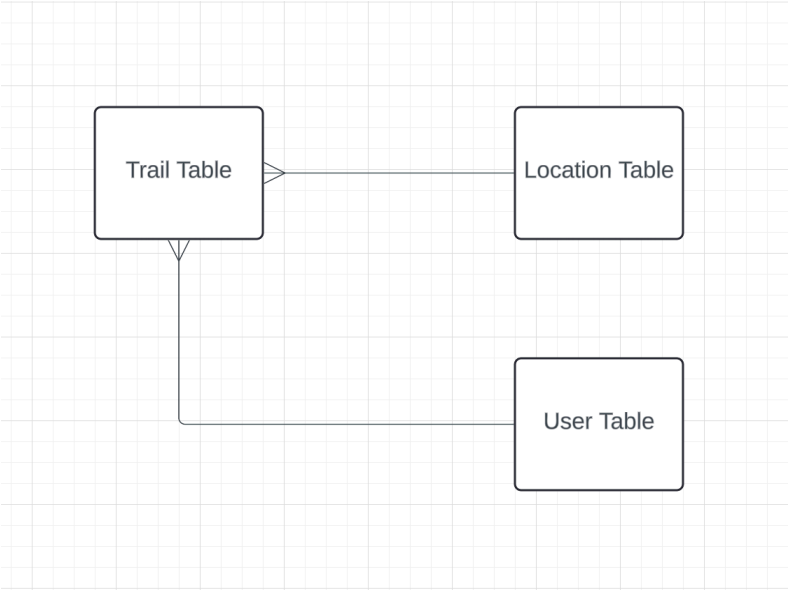


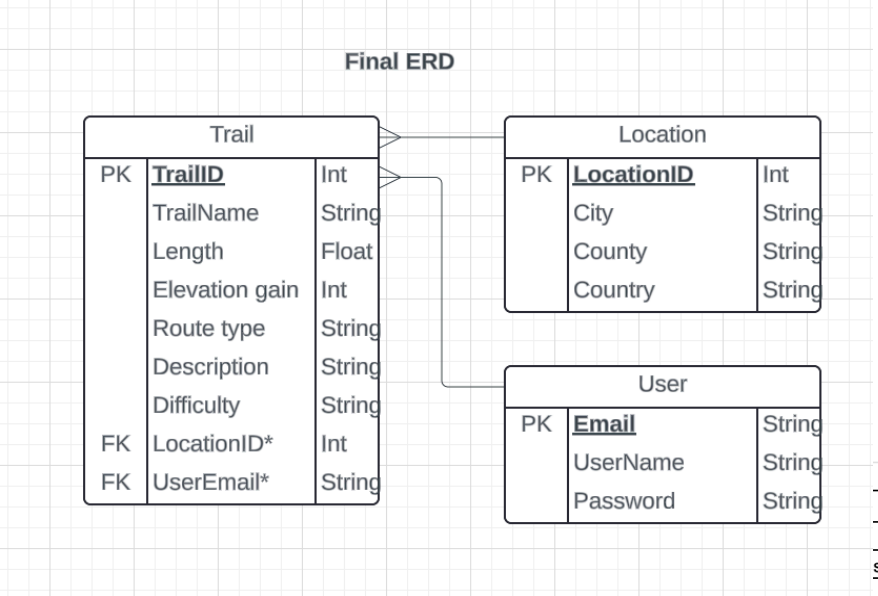
2) **Appendix B: Partial ERD**



3) **Appendix C: Preliminary ERD & Final ERD**

2001 Set Exercises





					s for each trail
Length	Float	N/A	Not Null	N/A	Length of trail (km)
Elevation Gain	Int	N/A	Not Null	N/A	Elevation gain (metres)
Route Type	String	Varies	Not Null	N/A	Type of loop (e.g., loop)
Description	String	Varies	N/A	N/A	Description of trail
Complexity	String	Varies	Not Null	N/A	Complexity of the trail
LocationID	Int	N/A	Foreign Key (Location)	N/A	Location of the trail
UserEmail	String	Varies	Foreign Key (User)	N/A	Email of the user who owns the trail

Location table

Field name	Data Type	Size	Constraints	Default Value	Description
LocationID	Int	N/A	Primary Key	N/A	Unique Identifier for location
City	String	Varies	Not Null	N/A	City where the trail is located
County	String	Varies	N/A	N/A	County where the trail is located
Country	String	Varies	N/A	N/A	Country where the trail is located

User Table

Field name	Data Type	Size	Constraints	Default Value	Description
Email	String	Varies	Primary Key	N/A	User email (unique identifier)
UserName	String	Varies	Not Null	N/A	Name of the user
Password	String	Varies	Not Null	N/A	Password or user login

Appendix D - Database ran on university db