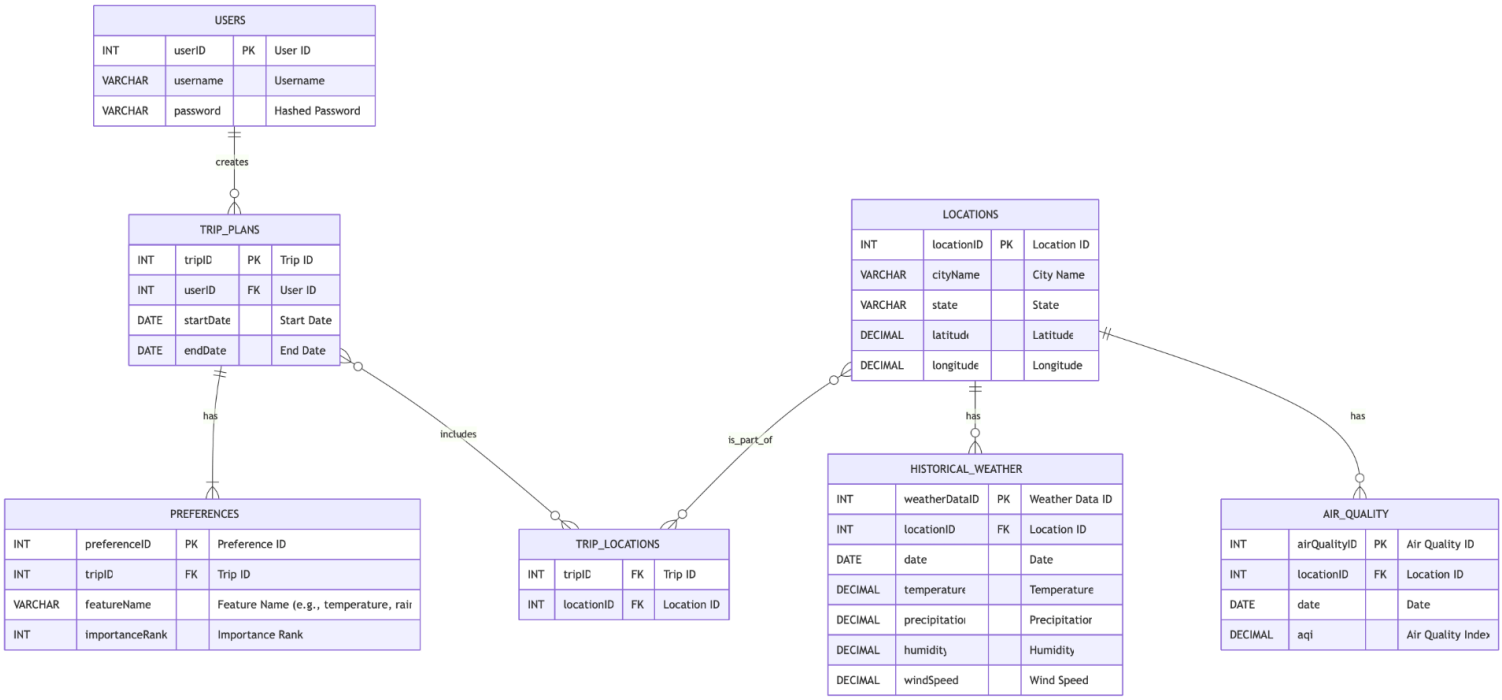
Team 095 PT1 Stage 2

1. **Draw your ER/UML diagram and save it as a single markdown or PDF file. Then save this file in the doc folder of your project GitHub repo.**



2. **Explain your assumptions for each entity and relationship in your model. Discuss why you've modeled something as an entity rather than an attribute of another entity. Describe the cardinality of relationships, like why a student is linked to only one advisor. These assumptions might come from customer requirements or application constraints. Please clarify them.**

   The database model is built on the fact that concepts like users, locations, and trip plans are distinct, primary objects. USERS is an entity to manage a member's data in order to save their research. LOCATIONS stores all locations that can be ranked, and having a separate entity for it avoids data duplication. For example a city like "Chicago" could be referenced across numerous trip plans so we want it to be distinct. A TRIP_PLAN acts as the main user-created object, serving as a container or link that a specific user can relate to the set of locations and preferences for a single comparison. Modeling these as separate entities rather than attributes prevents data redundancy and establishes a clear, scalable structure which is important when we try to do normalization.

Supporting data is separated into its own entities. PREFERENCES is an entity because each trip plan involves a variable number of ranked criteria; storing this one-to-many relationship in its own table is more flexible than adding a fixed number of columns to the TRIP_PLANS table. Similarly, the vast amount of time-series data for HISTORICAL_WEATHER and AIR_QUALITY means we have to give them their own tables. Storing thousands of daily records for each city as attributes within the LOCATIONS entity would be impractical.

The relationships between entities are defined by the application's logic. A one-to-many relationship is between USERS and TRIP_PLANS, since one user can create many plans, but each plan belongs to a single user. The core comparison feature requires a many-to-many relationship between TRIP_PLANS and LOCATIONS, implemented via the TRIP_LOCATIONS junction table, because a plan compares multiple cities and any city can be part of many plans. Finally, one-to-many relationships logically connect each location to its extensive historical data and each trip plan to its specific set of user preferences.

3. **Your ER/UML diagram should satisfy the following requirements. Your project must involve at least 5 entities with at most one entity representing user (accounts) information. This is why we asked you to identify two data sources in stage 1. It is often very hard to satisfy this requirement with a single dataset.It should involve at least two types of relationships (i.e., 1-1, 1-many, and many-many).**

Datasets Used:
NOAA Global Historical Climatology Network Daily Dataset (link)
EPA Air Quality System (AQS) Data (link)
Simplemaps U.S. Cities Database (link)

4. **Normalize your database. Apply BCNF or 3NF to your schema or show that your schema adheres to one of these normal forms.**

Our database adheres to 3NF

First of all our database attributes are atomic and there are no repeating groups in each database. For example, each location has a locationID, cityName, State, etc. This means that the database adheres to 1NF.

Every non-key attribute from each table depends on the table’s primary key. Every table has an attribute such as userID, tripID, etc.

Finally there are also no transitive dependencies where non-key attributes in a table depend on a different non-key attribute. For example, in Users, username and password both depend only on userID.

Therefore the database adheres to 3NF

5. **Convert your conceptual database design (ER/UML) to the logical design (relational schema). Note that a relational schema is NOT an SQL DDL command. Your relational schema should be formatted as follows: Table-Name(Column1:Domain [PK], Column2:Domain [FK to table.column], Column3:Domain,...). PK: Indicates that the column is a primary key for the table. FK: Indicates that the column is a foreign key referencing the primary key of table.column. Domain: INT, Decimal, VARCHAR(X),..**

USERS(userID: INT [PK],
username: VARCHAR(255),
password: VARCHAR(255)**)**

TRIP_PLANS(tripID: INT [PK],
userID: INT [FK to USERS.userID],
startDate: DATE,
endDate: DATE**)**

PREFERENCES(preferenceID: INT [PK],
tripID: INT [FK to TRIP_PLANS.tripID],
featureName: VARCHAR(255),
importanceRank: INT**)**

LOCATIONS(locationID: INT [PK],
cityName: VARCHAR(255),
state: VARCHAR(255),
latitude: DECIMAL,
longitude: DECIMAL**)**

HISTORICAL_WEATHER(weatherDataID: INT [PK],
locationID: INT [FK to LOCATIONS.locationID],
date: DATE,
temperature: DECIMAL,
precipitation: DECIMAL,
humidity: DECIMAL,
windSpeed: DECIMAL**)**

AIR_QUALITY(airQuality: INT [PK],
locationID: INT [FK to LOCATIONS.locationID],
date: DATE,
aqi: DECIMAL**)**

TRIP_LOCATIONS(tripID: INT [FK to TRIP_plans.tripID],
locationID: INT [FK to LOCATIONS.locationID**)**