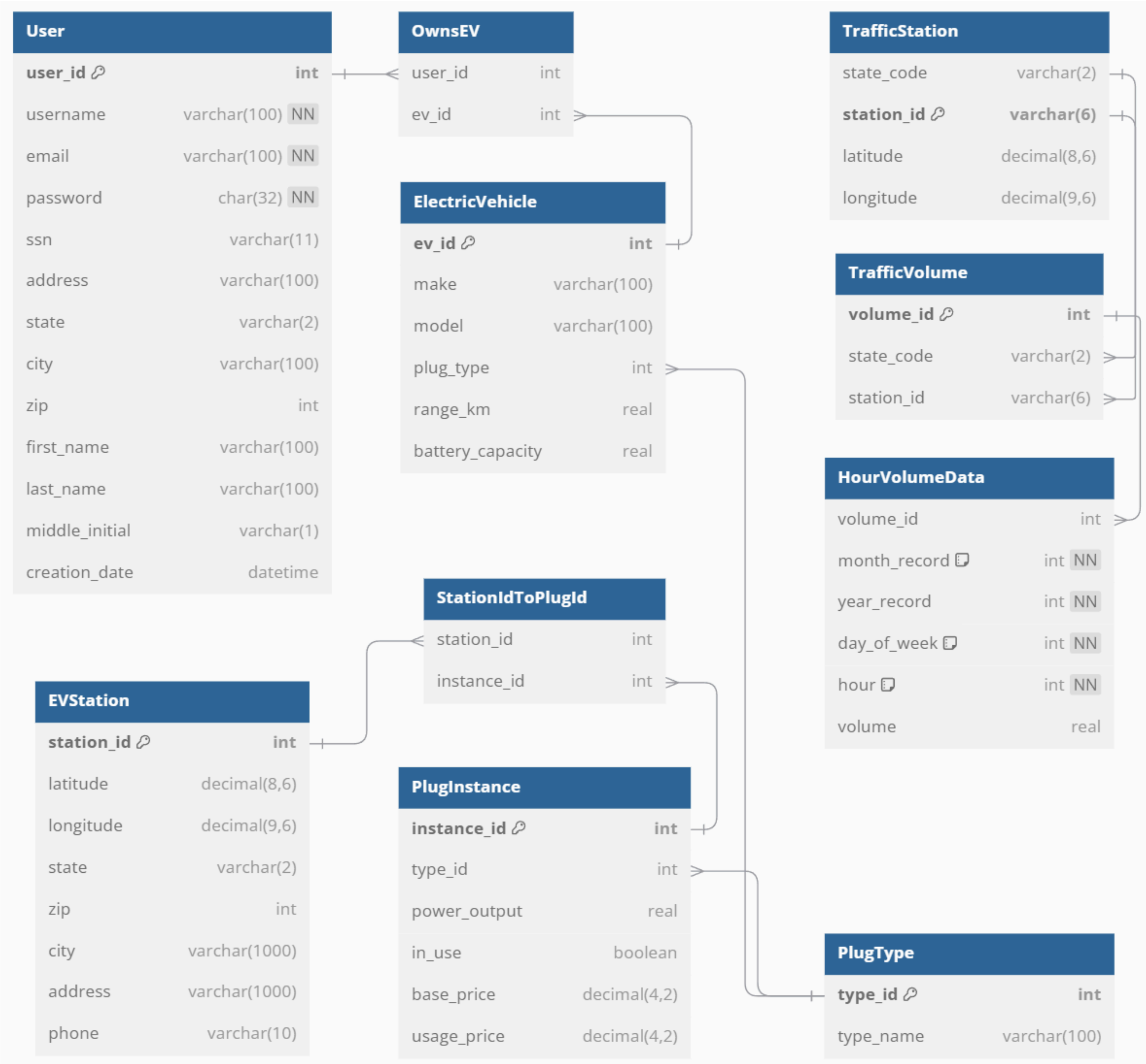
Table of Contents

1 & 3: ER Diagram

There are 10 > 5 different entities. There is only 1 entity related to Users. We utilize both 1-many and many-many relationships. (See phase 2/5 for examples).



2 & 5: Assumptions and Justifications

User

Assumption

The User table represents the system's users. We assume that ownership of a vehicle means that the user is of the legal age required to collect cookies without their consent. We also assume any other critical information like driver's license is not required and stored elsewhere.

Why Entity

Since a user can own multiple EVs and interact with the other entities, it needs to be a standalone entity instead of as an attribute of another entity.

Cardinality

One-to-many (User, OwnsEV): A user can own multiple electric vehicles

Data Type Justification

```
User(
   user_id: INT[PK], -- Primary key for uniqueness
   username: VARCHAR(100) UNIQUE, -- Username, unique to each user
   email: VARCHAR(100) UNIQUE, -- Email, unique for user and necessary for authentication
   password: CHAR(32), -- Password stored as a hash (MD5)
   ssn: VARCHAR(11), -- SSN is stored as VARCHAR to accommodate standard formatting
   address: VARCHAR(100), -- User's address
   state: VARCHAR(2), -- State stored as a 2-character abbreviation like CA
   city: VARCHAR(100), -- City name
   zip: INT, -- ZIP code stored as an integer (5-digit format)
   creation_date: DATETIME, -- Captures the date and time the user account was created
   first_name: VARCHAR(100), -- User's first name
   last_name: VARCHAR(100), -- User's last name
   middle_initial: VARCHAR(1) -- Middle initial, assuming one character suffices
)
```

OwnsEV

Assumption

This table is how we form relationships between User entities and their owned electric vehicle types since a user may own multiple types of EVs. We assume that many Users may own multiple ElectricVehicles and many different types of ElectricVehicle can be owned by multiple users. We also assume that we don't need any specific information about the instance of the ElectricVehicle owned by the User such as license plate or mileage since this project seeks to provide a broad overview of EV charging options.

Why Entity

We modeled this relationship as a separate entity since a single user can own many electric vehicles and to adhere to normalization constraints, we need to ensure that each record is only accessible via the `user_id` that owns a particular `ev_id`.

Cardinality

Many-to-many (OwnsEV, User): A user can own multiple electric vehicle types and multiple electric vehicle types can be owned by multiple different users.
Many-to-many (OwnsEV, ElectricVehicle): A user can own multiple electric vehicle types and multiple different electric vehicle types can be owned by multiple different users.

Note: The relations between User ←→ OwnsEV ←→ ElectricVehicle creates an indirect relation between the User and their owned ElectricVehicles in a normalized manner.

Data Type Justification

All types are foreign keys

```
OwnsEV(
  user_id: INT [PK][FK to User.user_id], -- Foreign key to the User table
  ev_id: INT [PK][FK to ElectricVehicle.ev_id] -- Foreign key to the ElectricVehicle table
)
```

EVStation

Assumption

This table stores details about charging stations, like location, address, and contact information. It is assumed that EVStations can have multiple PlugInstances and that a PlugInstance is unique to each EVStation.

Why Entity

The station's attributes are independent and frequently queried (location data, joined with number of plugs, etc.), so it should be a separate entity to avoid duplication of information.

Cardinality

One-to-many (EVStation, StationIdToPlugId): Each station can have multiple plug instances.

Data Type Justification

```
EVStation(
    station_id: INT [PK], -- Primary key for uniqueness
    latitude: DECIMAL(8,6), -- Latitude stored with six decimal places as standard
    longitude: DECIMAL(9,6), -- Longitude stored with six decimal places as standard
    state: VARCHAR(2), -- Two-character state abbreviation
    zip: INT, -- ZIP code for the station's location
    city: VARCHAR(1000), -- City name stored as VARCHAR to handle large names
    address: VARCHAR(1000), -- Address stored with flexibility for full street information
    phone: VARCHAR(10) -- Phone number stored in numeric-only format, preprocessed to
remove hyphens or parentheses
)
```

PlugType

Assumption

This table stores different types of plugs that EVs can use for charging (the different charging standards). We assume all the PlugTypes are valid and standardized across different records with the same PlugType.

Why Entity

Plug types are standardized across multiple stations and vehicles, so it would be more efficient to define them as a separate entity instead of repeating them in PlugInstance or ElectricVehicle. We also need to ensure that any plug type for an EV or station is valid meaning it exists in this table as a referenced relation. By definition, we have a one-to-many relationship due to the implicit understanding that a single wire running into the ground is a single wire running into the ground (a single wire running into the ground that can be used to charge a vehicle).

Cardinality

One-to-many (PlugType, PlugInstance): A plug type can be used in many instances

Data Type Justification

PlugType(
  type_id: INT [PK], -- Primary key for uniqueness
  type_name: VARCHAR(100) -- Plug type name like “Type 2”.
)

PlugInstance

Assumption

This table tracks the individual plug instances at EV stations, including details like power output, availability (in_use), and pricing. We assume that all tax is calculated by the user by hand through papyrus or bamboo parchment and mailed via carrier pigeon to our location in the bahamas.

Why Entity

This allows fine-grained management of each charging port's status and pricing. It is essential to make this a separate entity rather than just an attribute. A plug instance changing should not change a station entity. Additionally, individual EV stations may charge different rates depending on individual instances (e.g., if they had a premium charging station in the shade)

Cardinality

Many-to-one (PlugInstance, PlugType): Each instance has a single plug type since we assume the adapters aren't interchangeable.

Data Type Justification

```
PlugInstance(
   instance_id: INT AUTOINCREMENT [PK], -- Primary key for uniqueness
   type_id: INT [FK to PlugType.type_id], -- Foreign key to the PlugType table
   power_output: REAL, -- The power output of the plug in kW
   in_use: BOOLEAN, -- Tracks whether the plug is currently in use (true/false)
   base_price: DECIMAL(4,2), -- Base price for connecting to the plug in dollars
   usage_price: DECIMAL(4,2) -- Price charged per unit of usage in kWh
)
```

StationIdToPlugId

Assumption

This table allows us to map between an EVStation and a given PlugInstance. We need this as a separate table because a single EVStation may have multiple different types of PlugTypes, each with their own attributes like charging rate, price, and plug type. This table maps each PlugInstance to a specific EVStation. There aren't any assumptions here since this is a normalization-required mapping table.

Why Entity

This needs to be a separate entity since multiple different plug instances can exist within a single station. Separating this into a separate table allows us to adhere to normalization constraints and ensure that each plug instance is tied to an EVStation meaning there are no PlugInstance(s) without a parent station.

Cardinality

One-to-many (EVStation, PlugInstance): A single EV station can have multiple instances of different plugs, but each plug instance is dependent on an EVStation to exist.

Data Type Justification

```
StationIdToPlugId(
   station_id: INT [PK][FK to EVStation.station_id], -- Foreign key to the EVStation table
   instance_id: INT [PK][FK to PlugInstance.instance_id] -- Foreign key to the PlugInstance table
)
```

ElectricVehicle

Assumption

This table tracks information and specifications about various electric vehicles, including their make, model, range, and the type of plug they use for charging. One of our main assumptions is that an EV will only have a single PlugType which requires us to make new records if new ElectricVehicles are released, although this is a minor assumption and doesn't strictly need to be applied at the database level.

Why Entity

Since an electric vehicle has so many of its own characteristics, and a vehicle can be owned by multiple users or linked to various PlugTypes, it needs to be a separate entity.

Cardinality

Many-to-one (ElectricVehicle, PlugType): Each vehicle can use one plug type.

Data Type Justification

```
ElectricVehicle(
    ev_id: INT UNIQUE AUTOINCREMENT [PK], -- Primary key for uniqueness
    make: VARCHAR(100), -- Make of the electric vehicle
    model: VARCHAR(100), -- Model of the electric vehicle
    plug_type: INT [FK to PlugType.type_id], -- Foreign key to the PlugType table
    range_km: REAL, -- Vehicle range in kilometers
    battery_capacity: REAL -- Battery capacity in kWh
)
```

TrafficStation

Assumption

A TrafficStation is a location where any given traffic volume data was collected. These are collected directly from the DoT, so many of the assumptions they make will be applied here as well. Some assumptions include the fact that we are assuming 1 TrafficStation per location and that all TrafficStations are uniquely identifiable by a combination of their station_id and state_code. We apply this under the assumption that traffic stations have been observed as inanimate entities existing in a distinctual location traveling through the Milky Way galaxy hulled by the gravity of the Earth, Sun, and supermassive blackhole known as Sagittarius A*.

Why Entity

These need to be modeled as a separate entity because they have unique IDs which are tied to the traffic volume measurements and they also contain vital location information which is needed to localize any individual measurements. We decided to create this as a separate entity as otherwise, we would have to tie a latitude and longitude to each individual traffic observation which is highly redundant and wastes a lot of space.

Cardinality

Many-to-one (TrafficVolume, TrafficStation): A single traffic station is tied to multiple instances in the TrafficVolume table each of which represents a set of traffic observations.

Data Type Justification

TrafficStation(

state_code:VARCHAR(2) [PK], – 2 letter state code, e.g. IL for Illinois

station_id:VARCHAR(6) [PK], – 6 character state-unique station ID

latitude:DECIMAL(8, 6), – Latitude, -90 - 90 with 6 degrees of precision

longitude:DECIMAL(9,6) – Longitude, -180 - 180 with 6 degrees of precision

)

TrafficVolume

Assumption

This table captures the traffic volume recorded at a specific station. This table doesn't make many assumptions since it exists solely as a way to map between TrafficVolume and HourVolumeData.

Why Entity

It allows for the capture of traffic data over time, which is flexible in ways to help store and analyze traffic volume trends.

Cardinality

One-to-many (TrafficVolume, HourVolumeData): Each traffic volume can have multiple hour-by-hour breakdowns.
One-to-many (TrafficStation, TrafficVolume): Each TrafficVolume record is tied to the single TrafficStation that generated it, however each TrafficStation generates multiple TrafficVolume records.

Data Type Justification

TrafficVolume(
    volume_id:INT [PK], – Unique INT ID to identify this record
    state_code:VARCHAR(2) [FK to TrafficStation.state_code], – Defined by foreign key
    station_id:VARCHAR(6) [FK to TrafficStation.station_id] – Defined by foreign key
)

HourVolumeData

Assumption

This table stores hourly traffic data for a specific traffic volume entry, including the exact month, year, day of the week, etc. A major assumption for this table is made implicity by the connection to VolumeData. Since VolumeData is tied to a single TrafficStation, by transitivity, we create an implicit relationship between HourVolumeData and TrafficStation. We also assume that there is only one valid traffic volume measurement for any given time. Records may be updated if required, but no duplicate times at the same TrafficStation are permitted. This constraint would have to be enforced by a trigger.

Why Entity

Detailed traffic volume data is recorded hourly, so this granularity is required to manage complexity. Otherwise, we would repeat much data such as longitude and latitude or create a horrendous attribute in the TrafficStation table.

Cardinality

Many-to-one (HourVolumeData, TrafficVolume).

Data Type Justification

HourVolumeData(
    volume_id:INT [PK][FK to TrafficVolume.volume_id], – Unique integer ID
    month_record:INT, – INT from 1-12 to represent month
    year_record:INT, – INT to represent year
    day_of_week:INT, – INT from 1-7 to represent day of week
    hour:INT, – INT from 0-23 to represent hour of day
    volume:REAL – REAL traffic measurement in cars/hour
)

4. Normalization Justification

The tables are in 3NF because every non-primary key attribute is fully dependent on the primary key, and there are no transitive dependencies. So every primary key is its own superkey i.e. 3NF. We have included the functional dependencies to justify this fact if the reader so desires.

Functional Dependencies

User = {
    user_id, username, email -> password,
    user_id, username, email-> ssn,
    user_id, username, email-> address,
    user_id, username, email-> state,
    user_id, username, email-> city,
    user_id, username, email-> zip,
    user_id, username, email -> creation_date,
    user_id, username, email -> first_name,
    user_id, username, email-> last_name,
    user_id, username, email-> middle_initial
}

OwnsEV = {
    user_id, ev_id -> user_id, ev_id
}

EVStation = {
    station_id -> latitude,
    station_id -> longitude,
    station_id -> state,
    station_id -> zip,
    station_id -> city,
    station_id -> address,
    station_id -> phone
}

PlugType = {
    type_id -> type_name
}

```
PlugInstance = {
    instance_id -> instance_id
    instance_id -> type_id
    instance_id -> power_output
    instance_id -> in_use
    instance_id -> base_price
    instance_id -> usage_price
}

StationIdToPlugId = {
    station_id, instance_id-> station_id, instance_id
}

ElectricVehicle = {
    ev_id -> make,
    ev_id -> model,
    ev_id ->plug_type,
    ev_id ->range_km,
    ev_id ->battery_capacity
}

TrafficStation = {
    state_code, station_id -> latitude,
    state_code, station_id -> longitude
}

TrafficVolume = {
    volume_id -> state_code,
    volume_id -> station_id
}

HourVolumeData = {
    volume_id -> month_record,
    volume_id -> year_record,
    volume_id -> day_of_week,
    volume_id -> hour,
    volume_id -> volume
}
```

5. Relational Schema

User

```
User(
        user_id:INT [PK],
        username:VARCHAR(100) UNIQUE,
        email:VARCHAR(100) UNIQUE,
        password:CHAR(32),
        ssn:INT,
        address:VARCHAR(100),
        state:VARCHAR(2),
        city:VARCHAR(100),
        zip:INT,
        creation_date:DATETIME,
        first_name:VARCHAR(100),
        last_name:VARCHAR(100),
        middle_initial:VARCHAR(1)
)
```

OwnsEV

```
OwnsEV(
        user_id:INT [PK][FK to User.user_id],
        ev_id:INT [PK][FK to ElectricVehicle.ev_id]
)
```

EVStation

```
EVStation(
        station_id:INT [PK],
        latitude:DECIMAL(8,6),
        longitude:DECIMAL(9,6),
        state:VARCHAR(2),
        zip:INT,
        city:VARCHAR(1000),
        address:VARCHAR(1000),
        phone:VARCHAR(10)
)
```

PlugType

```
PlugType(
	type_id:INT [PK],
	type_name:VARCHAR(100)
)
```

PlugInstance

```
PlugInstance(
	instance_id:INT AUTOINCREMENT [PK],
	type_id:INT [FK to PlugType.type_id],
	power_output:REAL,
	in_use:BOOLEAN,
	base_price:DECIMAL(4,2),
	usage_price:DECIMAL(4,2)
)
```

StationIdToPlugId

```
StationIdToPlugId(
	station_id:INT [PK][FK to EVstation.station_id],
	instance_id:INT [PK][FK to PlugInstance.instance_id]
)
```

ElectricVehicle

```
ElectricVehicle(
	ev_id:INT UNIQUE AUTOINCREMENT [PK],
	make:VARCHAR(100),
	model:VARCHAR(100),
	plug_type:INT [FK to PlugType.plug_type]
	range_km:REAL,
	battery_capacity:REAL
)
```

TrafficStation

```
TrafficStation(
	state_code:VARCHAR(2) [PK],
	station_id:VARCHAR(6) [PK],
	latitude:DECIMAL(8, 6),
	longitude:DECIMAL(9,6)
)
```

TrafficVolume

```
TrafficVolume(
	volume_id:INT [PK],
	state_code:VARCHAR(2) [FK to TrafficStation.state_code],
	station_id:VARCHAR(6) [FK to TrafficStation.station_id]
)
```

HourVolumeData

```
HourVolumeData(
	volume_id:INT [PK][FK to TrafficVolume.volume_id],
	month_record:INT,
	year_record:INT,
	day_of_week:INT,
	hour:INT,
	volume:REAL
)
```