## UML Diagram

UML Diagram: https://miro.com/app/board/uXjVJ9Zrpq4=/

## Entities

User: The User entity stores information about each registered user of the Car Tinder platform, including their login credentials, display name, and home zip code for localized listings. It exists as its own entity because each user can have multiple related records in other tables, such as preferences and swipes, and this data would be redundant if stored elsewhere. Each user can have zero or many preferences and can perform many swipes on different car listings. A user must exist before any associated data (preferences or swipes) can be created.

Preference: The Preference entity captures a user's car search filters, including price range, mileage limits, body style, and fuel type. It is modeled as a separate entity rather than attributes within User because preferences can change over time and may need to be versioned or compared. Each user can have multiple saved preferences, but only one active at a time. This design allows the system to track user behavior and evolving search interests without losing history.

Swipe: The Swipe entity records every like or dislike action a user takes on a specific used car listing, along with the direction (right or left) and timestamp. It is an entity rather than a simple relationship attribute because each swipe represents an individual event that can be analyzed for recommendations and user engagement metrics. Each swipe connects one user and one used car listing, and a user can swipe on many listings while each listing can receive swipes from many users.

New Car Price: The New Car Price entity stores MSRP and cost-of-ownership data from the Cars 2025 dataset, including attributes like price, annual fuel cost, and effective date. It is a separate entity because pricing varies over time and comes from external datasets, while the base car specifications remain static. Each car can have multiple associated new price records representing different time periods or data sources. This prevents overwriting historical data and supports time-based analysis.

Car: The Car entity serves as the master reference table for all vehicles, containing make, model, year, engine details, body style, and emissions information. It consolidates data from the Vehicle Fuel Economy dataset and acts as the central hub linking all other datasets. It is a standalone entity because its data is shared across multiple other tables (New Car Price, Used Car Listing, Car Image) and would otherwise be duplicated. One car can have multiple price records, multiple used listings, and multiple images.

Used Car Listing: The Used Car Listing entity represents individual used car sale records from the Used Car Price Prediction dataset, including attributes like price, mileage, title status, and accident count. It is distinct from Car because these details describe specific instances of a car model rather than the model itself. Each used listing belongs to one car but can be viewed and swiped on by many users. This separation ensures that condition and listing specific data remain independent of general car specifications.

Car Image: The Car Image entity stores URLs and metadata for images from the Cars196 dataset, such as the image source and license information. It is a distinct entity because image information has its own attributes and sources, separate from the car’s specifications or pricing. However, the relationship between Car and Car Image is one-to-one. Each car has exactly one representative image, and each image corresponds to a single car to ensure uniform presentation of vehicles within the app.

## Normalization

| Schema | Functional Dependencies | Explanation |
|---|---|---|
| User | user_id->email, password, created_at | Email is a candidate key, so the schema is still BCNF |
| Preference | preference_id->user_id, created_at, min_price, max_price, min_year, fuel_type, transmission, is_active | The business rule that only one preference is active is enforced with a uniqueness constraint, which is not a functional dependency |
| NewCarPrice | new_price_id -> car_id, msrp | The attributes depend only on the key |
| Car | car_id ->make, model, year, mpg | All attributes depend on the primary key. We have not verified if make, model, and year form a candidate key, but we are fine in either case. |
| UsedCarListing | listing_id -> car_id, price, mileage | All attributes depend on the primary key |
| Swipe | swipe_id -> user_id, listing_id, action, created_at | All attributes depend on the primary key |
| CarImage | image_id -> car_id, image_url | All attributes depend on the key |

## Relational Schema

```
User(
  user_id: INT [PK],
  email: VARCHAR(120),
  password: VARCHAR(120),
  created_at: TIMESTAMP
)

Preference(
  preference_id: INT [PK],
  user_id: INT [FK to User.user_id],
  created_at: TIMESTAMP,
  min_price: DECIMAL(10,2),
  max_price: DECIMAL(10,2),
  min_year: INT,
  fuel_type: VARCHAR(20),
  transmission: VARCHAR(20),
  is_active: BOOLEAN
)

NewCarPrice(
  new_price_id: INT [PK],
  car_id: INT [FK to Car.car_id],
  msrp: DECIMAL(10,2)
)

Car(
  car_id: INT [PK],
  make: VARCHAR(40),
  model: VARCHAR(40),
  year: INT,
  mpg: INT
)

UsedCarListing(
  listing_id: INT [PK],
  car_id: INT [FK to Car.car_id],
  price: DECIMAL(10,2),
  mileage: INT
)

Swipe(
  swipe_id: INT [PK],
  user_id: INT [FK to User.user_id],
  listing_id: INT [FK to UsedCarListing.listing_id],
  action: VARCHAR(5),
  created_at: TIMESTAMP
)

CarImage(
  image_id: INT[PK],
  car_id: INT [FK to Car.car_id],
  image_url: VARCHAR(400)
)
```