EGE UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

322 INTRODUCTION TO DATABASES (3+2)

2023–2024 FALL SEMESTER

TERM PROJECT

Country Transportation Database

DELIVERY DATE

11/01/2024

PREPARED BY

05200000061 - Atakan Gesmeli

05230000194 - Anıl Akuygur

05210000905 - Ahmet Memiş

05210000249 - Deniz Egemen Keneş

ANALYSIS AND DESIGN

1- TURKISH AIRLINES

Data Requirements

Turkish Airlines is the national flag carrier airline of Turkey, headquartered in Istanbul. As one of the largest and most prominent airlines in the world, it operates flights to numerous domestic and international destinations.

for each passenger stored, the database will keep:
a passenger id, which is unique for all passengers
their name,
their age,
their passenger type (whether they are an elderly passenger, a student or a veteran etc.)
and their gender.

for each preference stored in database, the database will keep:
the preference id, which is unique for all preferences,
the class type,
the seat position,
the day time
and the meal menu.

for each airport stored, the database will keep:
the airport id, which is unique for all airports,
the name of the airport,
and the location of the airport.

for each ticket stored, the database will keep:
the ticket id, which is unique for all tickets
the ticket price, which is computed using the base price and the class type.
and the class type for the ticket (business, economy etc.)

for each flight stored, the database will keep
the flight id, which is unique for all train services,
the scheduled time for the flight,

for each payment card stored, the database will keep:
the card id for the payment card, which is unique for all payment cards
and the total amount inside the payment card.

for each airplane stored, the database will keep:
    it’s airplane id, which is unique for all airplanes,
    and the total number of seats of the airplanes.

for each airplane type stored, the database will keep:
    the airplane type id, which is unique for all train types
    the airplane type name,
    and the brand for the airplane type.

for each leg instance stored, the database will keep:
    the leg id, which is unique for all leg instances,
    the departure and arrival datetimes,
    the base price for the flight,
    and the number of available seats, which can be computed by subtracting every ticket bought for the flight leg instance from the total number of seats.

stewards help the people during the train service.
for each steward the database will keep:
    the id of the steward, which is unique for each of them,
    their gender,
    their name,
    and their age.

for each seat stored, the database will keep:
    the seat no, which is unique for all seats.

Relationships

passenger-ticket
each user can buy multiple tickets. but each ticket can be bought by only one person. a user can have no tickets to their name (no tickets bought by the user). a ticket must be bought in order for it to be arranged.

passenger-payment card
each passenger can at most have one passenger card. each passenger card can be attached to at most one passenger. all passengers must have at least one payment cards. each payment card must be attached to at least one payment card.

passenger-train service (search history)
each passenger can search multiple flights, or they can search no flight.
each flight can be searched by multiple passengers, or they can be searched by no passenger.

the relationship also holds the time that the flight was searched by the passenger.

flight - airport
each flight can depart from at most and at least one airport .
multiple flights can depart from one station, or there could be no departing flight from one airport.

each flight can arrive at at most and at least one airport.
multiple flights can arrive at one station, or there could be no departing airport from one airport.

flight-leg instance (weak)
each flight can have multiple leg instances, or it can have no leg instances.
the existence of a leg instance depends solely on the flight. it can’t exist without a train service.
each leg instance must be tied to at least and at most one flight.

leg instance-seat (weak)
each leg instance can have multiple seats, or it can have no seats.
the existence of a seat depends solely on the existence of a leg instance. it can’t exist without a leg instance
each seat must be tied to at least and at most one leg instance.

ticket-seat
each ticket must book at least and at most one seat.
a seat must be booked to at most one ticket. or it can be booked to no tickets.

steward-leg instance
every leg instance must have one steward for it.
each steward can be a guide in multiple leg instances
each leg instance must at least have one steward

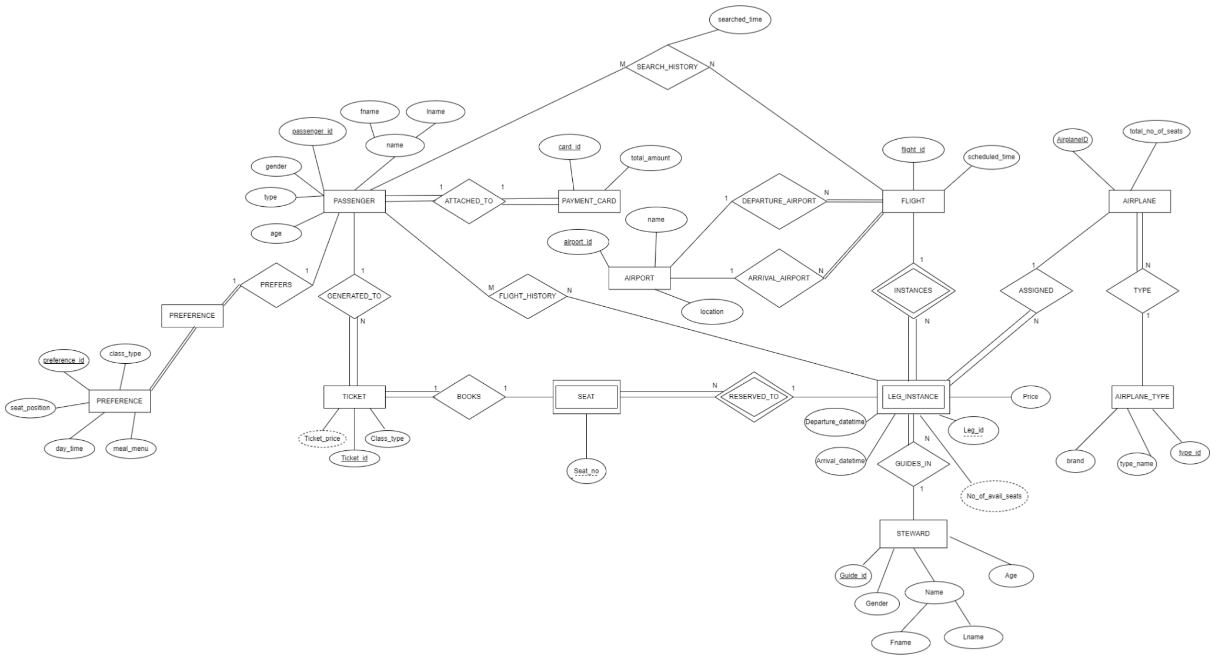
airplane -airplane type
each airplane must have only one airplane type.
each airplane type can be assigned to multiple airplanes.
every airplane must participate in this relationship.

airplane -leg instance
each leg instance must have at least and at most one airplane assigned to it.
an airplane can be assigned to multiple leg instances. an airplane can be assigned to no leg instance.

passenger-leg instance (flight history)
each passenger can travel in multiple leg instances, or they can be tied to no leg instances.

in each leg instance there can be multiple passengers, or there can be no passenger. (an empty leg instance)

- PASSENGER - PREFERENCE (1:1)
A PASSENGER can only have one PREFERENCE and A PREFERENCE can only be attached to one PASSENGER.



2- TCDD

Data Requirements

TCDD is a state owned train ride service provider company. The trains travel between stations. People buy tickets, get inside the train on one station and get out in another station. The tickets hold the info about these train rides. When buying tickets, users should consider factors such as arrival & departure station and the times to arrive and depart.

for each passenger stored, the database will keep:
a passenger id, which is unique for all passengers
their name,
their age,
their passenger type (whether they are an elderly passenger, a student or a veteran etc.)
and their gender.
for each preference stored in database, the database will keep:
the preference id, which is unique for all preferences,
the class type,
the seat position,

the day time
and the meal menu.

for each station stored, the database will keep:
the station id, which is unique for all stations,
the name of the station,
and the location of the station.

for each ticket stored, the database will keep:
the ticket id, which is unique for all tickets
the ticket price, which is computed using the base price and the class type.
and the class type for the ticket (business, economy etc.)

for each train service stored, the database will keep
the train service id, which is unique for all train services,
the scheduled time for the train,

for each payment card stored, the database will keep:
the card id for the payment card, which is unique for all payment cards
and the total amount inside the payment card.

for each train stored, the database will keep:
it's train id, which is unique for all trains,
and the total number of seats of the train.

for each train type stored, the database will keep:
the train type id, which is unique for all train types
the train type name,
and the brand for each train type.

for each leg instance stored, the database will keep:
the leg id, which is unique for all leg instances,
the departure and arrival datetimes,
the base price for the train service,
and the number of available seats, which can be computed by subtracting every ticket bought for the train service from the total number of seats.

train attendants help the people during the train service.
for each train attendant the database will keep:
the id of the train attendant, which is unique for each of them,
their gender,
their name,
and their age.

for each seat stored, the database will keep:
    the seat no, which is unique for all seats.

Relationships

passenger-ticket
each user can buy multiple tickets. but each ticket can be bought by only one person. a user can have no tickets to their name (no tickets bought by the user). a ticket must be bought in order for it to be arranged.

passenger-payment card
each passenger can at most have one passenger card. each passenger card can be attached to at most one passenger. all passengers must have at least one payment cards. each payment card must be attached to at least one payment card.

passenger-train service (search history)
each passenger can search multiple train services, or they can search no train service.
each train service can be searched by multiple passengers, or they can be searched by no passenger.
the relationship also holds the time that the train service was searched by the passenger.

train service - station
each train service can depart from at most and at least one station.
multiple train services can depart from one station, or there could be no departing train service from one station.

each train service can arrive at at most and at least one station.
multiple train services can arrive at one station, or there could be no departing train service from one station.

train service-leg instance (weak)
each train service can have multiple leg instances, or it can have no leg instances.
the existence of a leg instance depends solely on the train service. it can’t exist without a train service.
each leg instance must be tied to at least and at most one train service.

leg instance-seat (weak)
each leg instance can have multiple seats, or it can have no seats.
the existence of a seat depends solely on the existence of a leg instance. it can’t exist without a leg instance
each seat must be tied to at least and at most one leg instance.

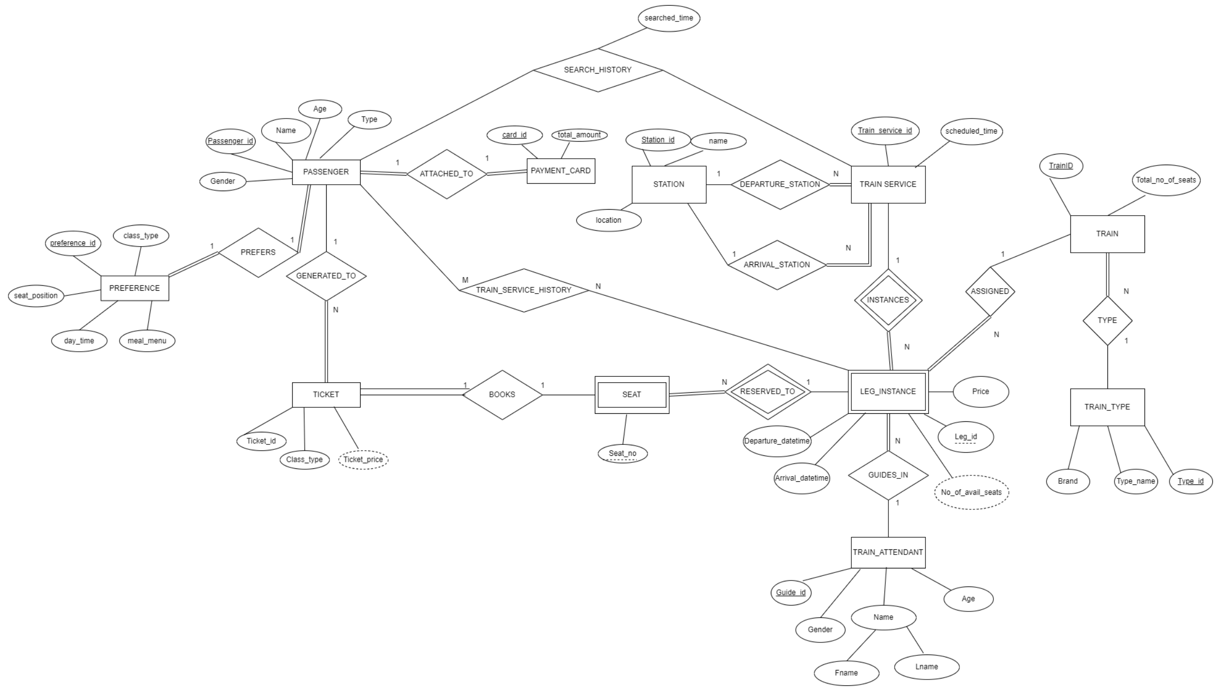
ticket-seat
each ticket must book at least and at most one seat.
a seat must be booked to at most one ticket. or it can be booked to no tickets.

train attendant-leg instance
every leg instance must have one train attendant for it.
each train attendant can be a guide in multiple leg instances
each leg instance must at least have one train attendant

train-train type
each train must have only one train type.
each train type can be assigned to multiple trains.
every train must participate in this relationship.

PASSENGER - PREFERENCE (1:1)
A PASSENGER can only have one PREFERENCE and A PREFERENCE can only be attached to one PASSENGER.
train-leg instance
each leg instance must have at least and at most one train assigned to it.
a train can be assigned to multiple leg instances. a train can be assigned to no leg instance.

passenger-leg instance (train service history)
each passenger can travel in multiple leg instances, or they can be tied to no leg instances.
in each leg instance there can be multiple passengers, or there can be no passenger. (an empty leg instance)



3- MARTI

Data Requirements

Martı is a website/mobile app where you can rent electric vehicles such as scooters and bikes to travel small distances. We designed our data requirements in a way such that when you are renting a vehicle, you must rent it (start driving it) from a renting station where all vehicles are charged and taken care of. After renting the vehicles, the vehicles should be dropped off in another renting station. When you rent, you must declare the two stations: one that you rent from and one that you leave your vehicle to.

for each user, the database will keep:

    their user id,
    their name,
    their date of birth,
    their phone number,
    their email account.

each unique user can be identified by their user id

for each preference stored in database, the database will keep:

    the preference id, which is unique for all preferences,
    the class type,
    the seat position,
    the day time
    and the meal menu.

for each city, the database will keep:

    the city's id,
    the city's name

each unique city can be identified by its id.

for each renting station, the database will keep:

    the stations id,
    the stations name,
    the number of vehicles that are actively waiting to be rented in the station.

each station can be identified by its station id.

for each vehicle, the database will keep:

    the vehicle's id,
    the vehicle's status (whether it is currently being used, or waiting to be used etc.)
    and the type of the vehicle (for example: bike, scooter etc.)

each vehicle can be identified by its unique vehicle id.

for each renting, the database will keep:
the rent id,
the rent time (estimated time that it is going to be rented for that's going to be calculated by the system.)
the rent price (estimated price that it is going to be rented for) (going to be calculated by the system)
each renting can be identified by its unique id.

Relationships

1. RIDE_HISTORY - PASSENGER Relationship: (1:N)

- Each passenger (PASSENGER) can have multiple entries in the ride history (RIDE_HISTORY), connecting a passenger to their historical ride records. This relationship allows for tracking the ride history associated with each passenger.

2. PAYMENT_CARD - PASSENGER Relationship: (1:1)

- Each passenger (PASSENGER) can have one payment card (PAYMENT_CARD), linking a passenger to their payment information. This relationship is essential for managing payment transactions and ensuring seamless transactions for each passenger.

3. RIDE - CHARGE_STATION Relationship: (1:N)

- Each ride (RIDE) involves both an arrival station and a departure station, both of which are charge stations (CHARGE_STATION). This relationship is crucial for identifying the charging stations associated with each ride.

4. LEG_INSTANCE - CHARGE_STATION Relationship: (M:N)

- Each leg instance (LEG_INSTANCE) is associated with specific charge stations (CHARGE_STATION) where vehicles might be charged. This relationship aids in tracking the charging locations related to each leg instance.

5. RIDE - VEHICLE Relationship: (1:1)

- Each ride (RIDE) involves a vehicle (VEHICLE), specifying the type of vehicle used for the ride. This relationship helps in identifying the vehicle associated with each ride instance.

6. LEG_INSTANCE - VEHICLE Relationship: (M:N)

- Each leg instance (LEG_INSTANCE) involves a vehicle (VEHICLE), determining the type of vehicle used for that specific leg instance. This relationship is vital for monitoring the vehicles associated with each leg instance.

7. PASSENGER prefers PREFERENCE (1:1)

A PASSENGER can only have one PREFERENCE and A PREFERENCE can only be attached to one PASSENGER.

8. PAYMENT_CARD - LEG_INSTANCE Relationship: (M:N)

- Each payment card (PAYMENT_CARD) used for transactions related to vehicle charging is linked to a leg instance (LEG_INSTANCE). This relationship is crucial for monitoring payment details associated with charging services.

9. RIDE - LEG_INSTANCE Relationship: (1:N)

- Each ride (RIDE) involves one or more leg instances (LEG_INSTANCE). This relationship helps in tracking the legs associated with each ride, including their arrival and departure details.

10. PASSENGER - RIDE Relationship: (1:N)

- Each ride (RIDE) is associated with a passenger (PASSENGER), linking the ride to the passenger who took the journey. This relationship is crucial for managing ride information for each passenger.

11. LEG_INSTANCE - RIDE Relationship: (M:N)

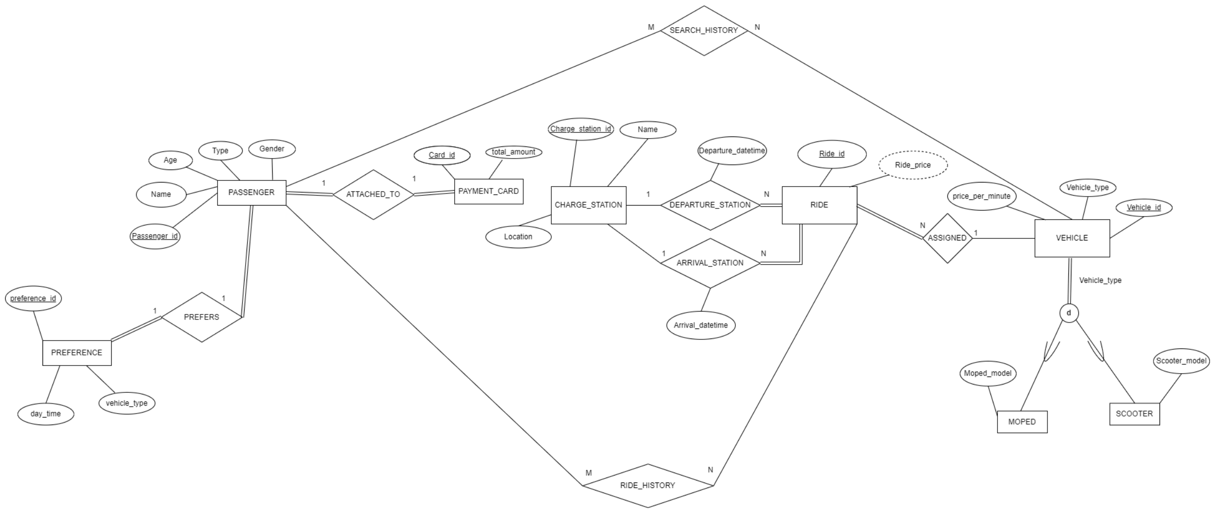
- Each leg instance (LEG_INSTANCE) is part of a ride (RIDE), establishing the connection between a specific leg of a journey and the overall ride instance. This relationship aids in organizing and tracking ride legs.

12. RIDE_HISTORY - LEG_INSTANCE Relationship: (M:N)

- Each leg instance (LEG_INSTANCE) is part of a ride history (RIDE_HISTORY), connecting specific legs to historical ride records. This relationship is essential for maintaining a comprehensive record of past ride legs.

13. RIDE_HISTORY - PASSENGER Relationship: (M:N)

- Each ride history entry (RIDE_HISTORY) is associated with a passenger (PASSENGER), linking historical ride records to the passengers who took those rides. This relationship is vital for tracking the ride history of each passenger.



4- IDO

Data Requirements

IDO (İstanbul Deniz Otobüsleri) is a public transport company that offers ferry rides. The ferries travel between ports. People buy tickets, get inside the ferry in one port and get out in another port. When buying tickets, passengers should consider factors such as arrival & departure ports and the times to arrive and depart.

for each ferry stored, the database will keep:
the id, which is unique for all ferries
and its total seats for humans.

for each ferry_name stored, the database will keep:
the id, which is unique for all ferry_types,
the name of type
and the maximum capacity of seats.

for each voyage stored, the database will keep,
the id, which is unique for all voyages
and the scheduled time.

for each leg_instance stored, the database will keep,
the voyage id,

the leg instance id, which is unique for all leg instances,
the departure date and time,
the arrival date and time
and the price.

for each steward stored in database, the database will keep,
the guide id, which is unique for all stewards,
the name (Fname and Lname),
the age
and the gender.

for each port stored in the database, the database will keep:
the port id, which is unique for all ports,
the name
and the location.

for each seat stored in the database, the database will keep:
the voyage id,
the leg instance id,
and the seat no, which is unique for all seats.

for each ticket stored in the database, the database will keep:
the ticket id, which is unique for all tickets
the price,
and the class type.

for each passenger stored in the database, the database will keep:
the passenger id, which is unique for all passengers,
the name (Fname and Lname),
the gender,
the age
and the passenger type.

for each payment card stored in database, the database will keep:
the card id, which is unique for all payment cards
and the total amount.

for each preference stored in database, the database will keep:
the preference id, which is unique for all preferences,
the class type,
the seat position,
the day time
and the meal menu.

Relationships

- FERRY_TYPE type FERRY (1:N)
A FERRY can only have one FERRY_TYPE but there may be FERRYs with zero or multiple FERRY_TYPEs.

- FERRY assigned to LEG_INSTANCE (1:N)
A FERRY can be assigned to zero or multiple LEG_INSTANCEs but a LEG_INSTANCE can only have one FERRY.

- VOYAGE instances LEG_INSTANCE (1:N) (weak relationship)
A VOYAGE can instance zero or multiple LEG_INSTANCEs but a LEG_INSTANCE can be instanced to only one VOYAGE.

- STEWARD guides in LEG_INSTANCE (1:N)
A STEWARD can guide in zero or multiple LEG_INSTANCEs but a LEG_INSTANCE can assign only one STEWARD.

- VOYAGE has departure PORT (1:N)
A VOYAGE can have only one departure PORT but a PORT can be assigned to zero or multiple VOYAGEs as departure.

- VOYAGE has arrival PORT (1:N)
A VOYAGE can have only one arrival PORT but a PORT can be assigned to zero or multiple VOYAGEs as arrival.

- SEAT reserved to LEG_INSTANCE (1:N) (weak relationship)
A SEAT can be reserved to only one LEG_INSTANCE but a LEG_INSTANCE can reserve zero or multiple SEATs.

- TICKET books SEAT (1:1)
A TICKET can book only one SEAT and a SEAT can be booked by only one TICKET.

- TICKET generated to PASSENGER (1:N)
A TICKET can be generated to only one PASSENGER but a PASSENGER can have zero or multiple TICKETs.

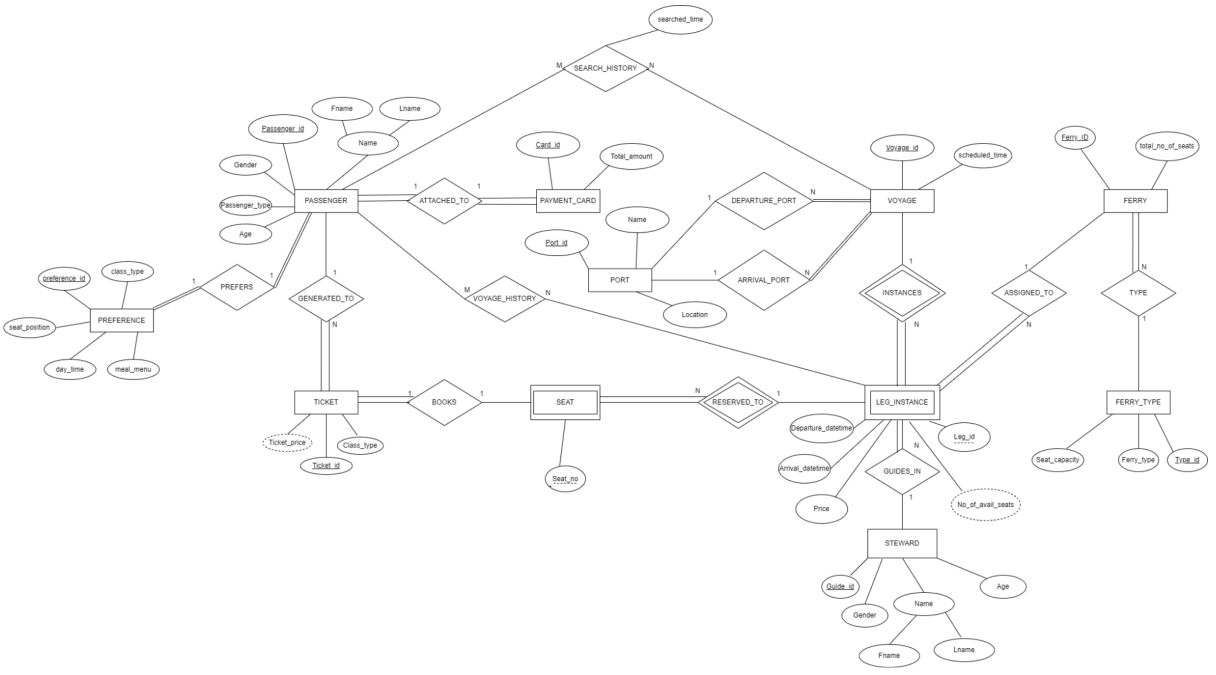
- PAYMENT_CARD attached to PASSENGER (1:1)
A PAYMENT_CARD can be attached to only one PASSENGER and a PASSENGER can only have one PAYMENT_CARD.

- PASSENGER search history VOYAGE (M:N)

Search history can have zero or multiple PASSENGERs or VOYAGEs. And search history relationship also have a searched time attribute.

- PASSENGER voyage history LEG_INSTANCE (M:N)
Flight history can have zero or multiple PASSENGERs or LEG_INSTANCEs.

- PASSENGER prefers PREFERENCE (1:1)
A PASSENGER can only have one PREFERENCE and A PREFERENCE can only be attached to one PASSENGER.



5- IZBAN

Data Requirements

İZBAN (İzmir Banliyö Sistemi) is a suburban commuter rail system serving the city of İzmir, Turkey. Operated in collaboration between İzmir Metropolitan Municipality and the Turkish State Railways, İZBAN provides efficient and rapid train services connecting the city center with its surrounding districts.

for each PASSENGER, the database will keep:

passenger id,
Passenger id,
name as a composite attribute(first name, last name),
age,
type (retired, healthcare worker, veteran etc.),
their gender value.

each Passenger can be uniquely identified by their passenger id.

Passenger has PREFERENCES, the database also keep PREFERENCES of passengers.
For each PASSENGER, the database will keep:
Preference id,
day_time(which part of the day passenger prefers to travel)
each PREFERENCES is uniquely identified by preference_id

for each PAYMENT CARD, the database will keep:
card id,
the owners passenger id,
the total amount of money stored inside the card.
each payment card can be uniquely identified by the card id.

For each STATION, the database will keep:
It's station id,
It's name,
it's location.
each station can be uniquely identified by it's location id.

For each METRO, the database will keep:
metro id
total capacity,
number of vagons
each specific metro can be uniquely identified by its metro id.

Ride entity is equal to METRO_SERVICE entity in izban's er diagram.
For each METRO_SERVICE the database will keep:
Metro_service_id
Each metro service id is uniquely identified by metro_service_id

The database will also keep LEG_INSTANCE as a weak entity.
For each LEG_INSTANCE the database will keep:
Leg_id
Price
Arrival date and time
Deparature date and time
Also there is a computed attrbiute that store number of available seats.
Each leg instane is unqiuely identified by the combination of Metro_Service_id of it's strong entity and Leg_id of the leg instance.

for each preference stored in database, the database will keep:

the preference id, which is unique for all preferences,
the class type,
the seat position,
the day time
and the meal menu.

Relationships

- PAYMENT_CARD – PASSENGER Relationship (1:1) ATTACHED_TO:

- Each driver (PASSENGER) possesses one payment card (PAYMENT_CARD). However, each payment card is uniquely associated with a single passenger, emphasizing the individuality of payment cards for each passenger. Each passenger must possess a payment card and for each card there must be a passenger who owns a card.

- STATION – METRO_SERVICE Relationship (1:N) DEPARTURE STATION AND ARRIVAL_STATION:

- Each METRO_SERVICE is linked to a specific arrival and departure location, but there may be a STATION which is not related to any METRO_SERVICE. The STATION can be used in multiple METRO_SERVICE, but each METRO_SERVICE is uniquely connected to one set of arrival and departure STATIONS.

- METRO_SERVICE – LEG_INSTANCE Relationship (1:N) INSTANCES:

- METRO_SERVICE is linked to specific LEG_INSTANCE, but there may be a METRO_SERVICE which has no LEG_INSTANCE. Each LEG_INSTANCE must be related to METRO_SERVICE. METRO_SERVICE may have more than one LEG_ISNTANCE, however, LEG_INSTANCE can only be related to one METRO_SERVICE.

- METRO – LEG_INSTANCE Relationship (1:N) ASSIGNED:

- METRO_SERVICE is assigned to LEG_INSTANCE, but there may be a METRO which has not assigned any LEG_INSTANCE yet. Each LEG_INSTANCE must have a METRO assigned to it. METRO may be assigned more than one LEG_ISNTANCE, however, LEG_INSTANCE can only have one METRO.

- PASSENGER- PREFERENCE Relationship (1:1) PREFERS:

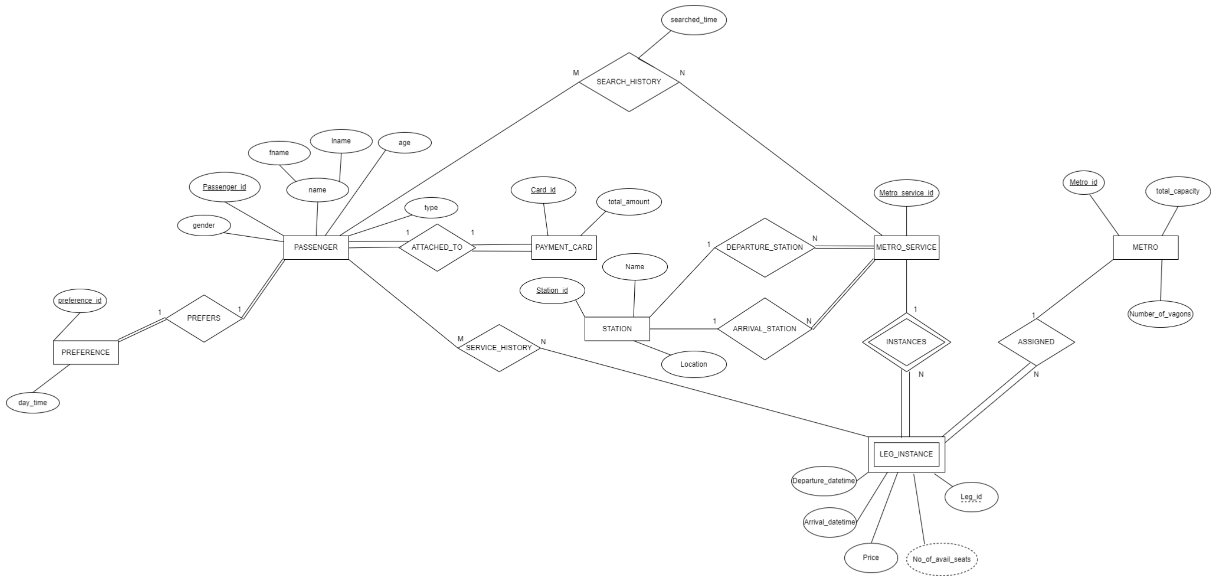
-Each PASSENGER has a PREFERENCES, also each PREFERENCES is related to PASSENGER. PASSENGER can only have one PREFERENCES, because we store

PREFERENCES as a combination of all features. Also, each PREFERENCE is related to only one PASSENGER.

- PASSENGER – LEG_INSTANCE Relationship (M:N) SERVICE_HISTORY:
    - PASSENGER may have more than one LEG_INSTANCE Also, each LEG_INSTANCE may be related to more than one PASSENGER (we do it like that because we want RIDE_HISTORY as a table. We manage the problem at SQL level.). There may be a PASSENGER, who has no LEG_INSTANCE yet, and there may be a LEG_INSTANCE, which is not related to any PASSENGER.

-PASSENGER – METRO_SERVICE Relationship (M:N) SEARCH_HISTORY:

- The database also stores SEARCH_HISTORY of PASSENGERs. PASSENGER may search more than one METRO_SERVICE, and METRO_SERVICE may search more than one PASSENGERs. There may be a PASSENGER who has no SEARCH_HISTORY, also there may be a METRO_SERVICE that has not been searched by any PASSENGER.



6- YOLCU360

Yolcu360.com is a car rental platform that brings together the world's leading car rental companies. By gathering rental cars under a single system, Yolcu360.com enables its users to find the most suitable car for them at the point and day they want to rent a car. At the same time, it offers the user the opportunity to filter according to different company and vehicle features.

Data Requirements

for each DRIVER, the database will keep:

their driver id,
their driver id,
their name as a composite attribute(first name, last name),
their age,
their type (retired, healthcare worker, veteran etc.),
and their gender value.
each driver can be uniquely identified by their driver id.

for each PAYMENT CARD, the database will keep:

its card id,
and the total amount of money stored inside the card.
each payment card can be uniquely identified by the card id.

For each RENTING_LOCATION, the database will keep:

It’s location id,
It’s name,
and it’s location values.
each station can be uniquely identified by it’s location id.

For each CAR, the database will keep:

car id
it’s license plate no,
each specific car can be uniquely identified by its car id.

Each car has a CAR_MODEL
Database will also keep the car’s CAR_MODEL.
For each CAR MODEL the database will keep:

Type of the car
Name of the model
Model id
Also the number of resources of car is limited and the database must keep available car numbers for each CAR_MODEL.

Each car model has a different price per minute and the database also keeps this information for each CAR_MODEL.
Car_MODEL is uniquely identified by Model_id

RIDE provides the information for a generally available, common ride
For each RIDE, the database will keep:
the ride id,
the arrival location for the ride,
the departure station for the ride.
a ride can be uniquely identified by its ride id.
Also ride has a total price and it's computed according to type of the car.
Database also keep Driver's preference.
For each PREFERENCE, the database will keeP:
Preference_id
Vehicle_model
Day_time (preference of ride time)
Each preference is uniquely identified by preferences_id

Relationships

- PAYMENT_CARD - DRIVER Relationship (1:1) ATTACHED_TO:

Each driver (DRIVER) possesses one payment card (PAYMENT_CARD). However, each payment card is uniquely associated with a single driver, emphasizing the individuality of payment cards for each driver. Each driver must possess a payment card and for each card there must be a driver who owns card.

- RENTING_LOCATION - RIDE Relationship (1:N) RENTING_FROM:

Each RIDE is linked to a specific arrival and departure location, but there may be a RENTING_LOCATION which is not related to any ride. The location can be used in multiple rides, but each ride is uniquely connected to one set of arrival and departure locations. Driver must bring the car back to the same renting location. Also we must store arrival and the departure time of the ride.

- CAR - RIDE Relationship (1:N) ASSIGNED:

Each RIDE is associated with a vehicle (CAR), This relationship indicates that a vehicle may be utilized in multiple rides, but each ride is exclusively connected to one specific vehicle. For each ride, there must be a vehicle, however there may be a vehicle which is not belongs to any ride.

- CAR – CAR_MODEL Relationship (1:N) MODEL:
Each car (CAR) has a model (CAR_MODEL), however, there may be a CAR_MODEL which does not belong to any CAR. Each CAR has only one CAR_MODEL, but more than one CAR may have the same CAR_MODEL.

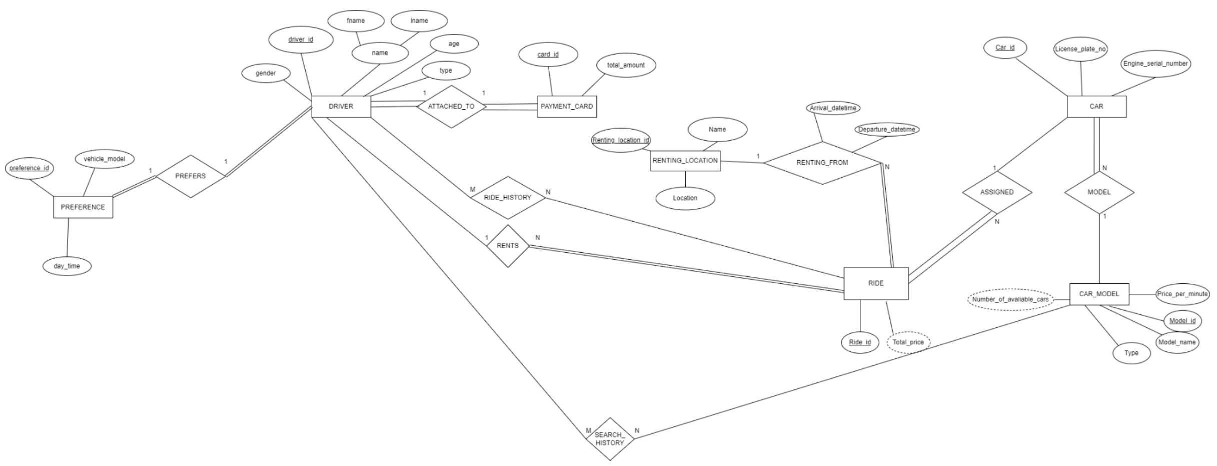
- DRIVER – RIDE Relationship (1:N) RENTS:
For each RIDE there must be a DRIVER. However, there may be a DRIVER who isn't related to any RIDE. DRIVER may have more than one RIDE, but RIDE is associated with only one DRIVER.

- DRIVER - PREFERENCE Relationship (1:1) PREFERS:
Each DRIVER has a PREFERENCES, also each PREFERENCES is related to DRIVER. DRIVER can only have one PREFERENCES, because we store PREFERENCES as a combination of all features. Also, each PREFERENCE is related to only one DRIVER.

- DRIVER-RIDE Relationship (M:N) RIDE_HISTORY:
DRIVER may have more than one RIDE. Also, RIDE may be related to more than one driver (we do it like that because we want RIDE_HISTORY as a table. We manage the problem at SQL level.). There may be a Drıver, who has no RIDE yet, and there may be a RIDE, which is not related to any DRIVER.

- PASSENGER – METRO_SERVICE Relationship (M:N) SEARCH_HISTORY:
The database also stores SEARCH_HISTORY of PASSENGERs. PASSENGER may search more than one METRO_SERVICE, and METRO_SERVICE may be searched more than one PASSENGERS. There may be a PASSENGER who has no SEARCH_HISTORY, also there may be a METRO_SERVICE that has not be searched by any PASSENGER.

- DRIVER – CAR_MODEL Relationship (M:N) SEARCH_HISTORY:
The database also stores SEARCH_HISTORY of DRIVERs. DRIVER may search for more than one CAR_MODEL, and CAR MODEL may be searched by more than one PASSENGERS. There may be a DRIVER who has no SEARCH_HISTORY, also there may be CAR_MODEL that has not be searched by any DRIVER.



7- OBİLET

Obilet is a Turkey-based online travel platform. Founded in 2010, the company provides users with the opportunity to purchase transportation options such as bus, train, and plane tickets online. Known as a platform that facilitates the easy organization of users' travel plans, Obilet collaborates with various transportation operators to offer access to a broad travel network and provides users with various pricing options.

Data Requirements

for each passenger, the database will keep:
their passenger id,
their name,
their age,
their type (retired, healthcare worker, veteran etc.),
and their gender value.
each passenger can be uniquely identified by their passenger id.

for each preference stored in database, the database will keep:
the preference id, which is unique for all preferences,
the class type,
the seat position,
the day time
and the meal menu.

for each location, the database will keep:
its location id,
its name,
and its location values.
each location can be uniquely identified by its port id.

for each vehicle, the database will keep:
its vehicle id,
its company,
its transportation type..
a vehicle can uniquely be identified by its vehicle id.

for each payment card, the database will keep:
its card id,
the owners passenger id,
and the total amount of money stored inside the card.
each payment card can be uniquely identified by the card id.

the expedition entity provides the information for a generally available expedition.
for each expedition, the database will keep:
the expedition id,
the arrival location for the expedition,
the departure location for the expedition,
an expedition can be uniquely identified by its expedition id.

leg instance specifies more info for the expedition. While the expedition provides the general structure, leg instance provides more info.
for each leg instance, the database will keep,
the leg id,
the expedition id,
the arrival & departure datetimes,
a leg instance can be uniquely identified by its leg id and voyage id.

expedition history stores every user's past ride info.
for each entry of the expedition history, the database will keep:
the passenger id,
the expedition id,
the leg id
the entity can be uniquely identified by using all three of these entitites.

for each ticket, the database will keep:
the ticket id,
the passenger id,
the expedition id,
the leg id,
the class type,
the ticket price.
the ticket price will be computed by using the user type and the class type.
each ticket can be uniquely identified by its ticket id.

for each seat of a leg instance, the database will keep:
the seat no,
the expedition id,
and the leg id.
each seat can be uniquely identified by using all three of these entities.

Relationships

1. ATTACHED_TO (between PASSENGER and PAYMENT_CARD) 1:1

- Each passenger (PASSENGER) is attached to exactly one payment card (PAYMENT_CARD), establishing a one-to-one relationship. This ensures that each passenger has a unique and individual payment card associated with them.

2. GENERATED_TO (between PASSENGER and TICKET) 1:N

   - For each passenger (PASSENGER), multiple tickets (TICKET) can be generated. This one-to-many relationship signifies that a passenger can have multiple tickets, but each ticket is uniquely generated for a specific passenger.

3. BOOKS (between TICKET and SEAT) 1:1

   - Each ticket (TICKET) books exactly one seat (SEAT), forming a one-to-one relationship. This indicates that a ticket corresponds to a specific seat, ensuring a direct association between the two entities.

4. RESERVED_TO (between LEG_INSTANCE and SEAT) 1:N

   - Each leg instance (LEG_INSTANCE) can have multiple seats (SEAT) reserved. This one-to-many relationship signifies that a leg instance can accommodate reservations for multiple seats, but each seat is uniquely reserved within a specific leg instance.

5. EXPEDITION_HISTORY (between PASSENGER and LEG_INSTANCE) M:N

   - The relationship between passengers (PASSENGER) and leg instances (LEG_INSTANCE) is many-to-many, denoted by M:N. This implies that a passenger can be associated with multiple leg instances, and conversely, a leg instance can be connected to multiple passengers, forming a complex and flexible relationship.

6. RENTS (between PASSENGER and RIDE) 1:N

   - Each passenger (PASSENGER) can rent multiple rides (RIDE), establishing a one-to-many relationship. This indicates that a passenger may have multiple ride transactions, but each ride is uniquely associated with a specific passenger.

7. RENTAL_ARR_LOC (between LOCATION and RIDE) 1:N

   - Each location (LOCATION) can be associated with multiple rental arrivals (RIDE). This one-to-many relationship signifies that a location can serve as the arrival point for multiple ride rentals, ensuring flexibility in the assignment of arrival locations.

8. RENTAL_DEP_LOC (between LOCATION and RIDE) 1:N

   - Similar to RENTAL_ARR_LOC, each location (LOCATION) can be associated with multiple rental departures (RIDE). This one-to-many relationship indicates that a location can serve as the departure point for multiple ride rentals.

9. TICKET_ARRIVAL_LOCATION (between LOCATION and EXPEDITION) 1:N

- Each location (LOCATION) can be associated with multiple expedition arrivals (EXPEDITION). This one-to-many relationship signifies that a location can be utilized as the arrival point for multiple expeditions.

10. TICKET_DEPARTURE_LOCATION (between LOCATION and EXPEDITION) 1:N

- Similar to TICKET_ARRIVAL_LOCATION, each location (LOCATION) can be associated with multiple expedition departures (EXPEDITION). This one-to-many relationship indicates that a location can be utilized as the departure point for multiple expeditions.

11. RENTED (between RENTAL_VEHICLES and RIDE) 1:N

- Each rental vehicle (RENTAL_VEHICLES) can be rented for multiple rides (RIDE), forming a one-to-many relationship. This indicates that a rental vehicle can be used for multiple ride transactions, but each ride is uniquely associated with a specific rental vehicle.

12. INSTANCES (between EXPEDITION and LEG_INSTANCE) 1:N

- Each expedition (EXPEDITION) can have multiple instances (LEG_INSTANCE), forming a one-to-many relationship. This signifies that an expedition can be comprised of multiple legs, each represented as a distinct leg instance.

13. GUIDES_IN (between GUIDE and LEG_INSTANCE) 1:N

- Each guide (GUIDE) can be associated with multiple leg instances (LEG_INSTANCE), forming a one-to-many relationship. This indicates that a guide can be assigned to guide multiple legs, but each leg instance is uniquely associated with a specific guide.

14. ASSIGNED (between TICKET_VEHICLE and LEG_INSTANCE) 1:N

- Each ticket vehicle (TICKET_VEHICLE) can be assigned to multiple leg instances (LEG_INSTANCE), forming a one-to-many relationship. This indicates that a ticket vehicle can be utilized in multiple legs, but each leg instance is uniquely associated with a specific ticket vehicle.

- PASSENGER prefers PREFERENCE (1:1)
A PASSENGER can only have one PREFERENCE and A PREFERENCE can only be attached to one PASSENGER.

Superclass/Subclass Specialization/Generalizations:

1. VEHICLE has these two entities: RENTAL_VEHICLES and TICKET_VEHICLE

- The superclass VEHICLE has a specialization relationship with two subclasses: RENTAL_VEHICLES and TICKET_VEHICLE. This means that VEHICLE serves as a generic or overarching category, while RENTAL_VEHICLES and TICKET_VEHICLE are more specific types or classes under the VEHICLE umbrella.

- RENTAL_VEHICLES represent vehicles that are available for rental purposes, such as cars, bikes, or scooters. This subclass inherits attributes and behaviors from the VEHICLE superclass but may also have additional properties specific to rental transactions.

- TICKET_VEHICLE represents vehicles associated with ticketing systems, possibly used for transportation services. Similar to RENTAL_VEHICLES, TICKET_VEHICLE inherits attributes from the VEHICLE superclass but may have unique features relevant to ticketing.

2. VEHICLE has these three entities: AIRPLANE, BUS, and FERRY

- The superclass VEHICLE further specializes into three subclasses: AIRPLANE, BUS, and FERRY. Each of these subclasses represents a specific type of vehicle within the broader category of VEHICLE.

- AIRPLANE is a subclass representing vehicles that are designed for air transportation. This may include attributes and behaviors specific to airplanes, such as registration numbers or aviation-related features.

- BUS is a subclass representing vehicles designed for land transportation, typically accommodating multiple passengers. Attributes specific to buses, such as plate numbers, are part of this subclass.

- FERRY is a subclass representing watercraft designed for transporting passengers or vehicles across bodies of water. Attributes like the ferry name may be specific to this subclass.

SEARCH_HISTORY (for the history of EXPEDITION, PASSENGER, and RENTAL_VEHICLES entities)

- The SEARCH_HISTORY relationship is established to maintain a historical record of three entities: EXPEDITION, PASSENGER, and RENTAL_VEHICLES. This relationship serves as a repository for tracking changes, updates, or historical data related to these entities over time.

- Attributes:

  - search_id (PK): A primary key attribute uniquely identifying each entry in the search history.

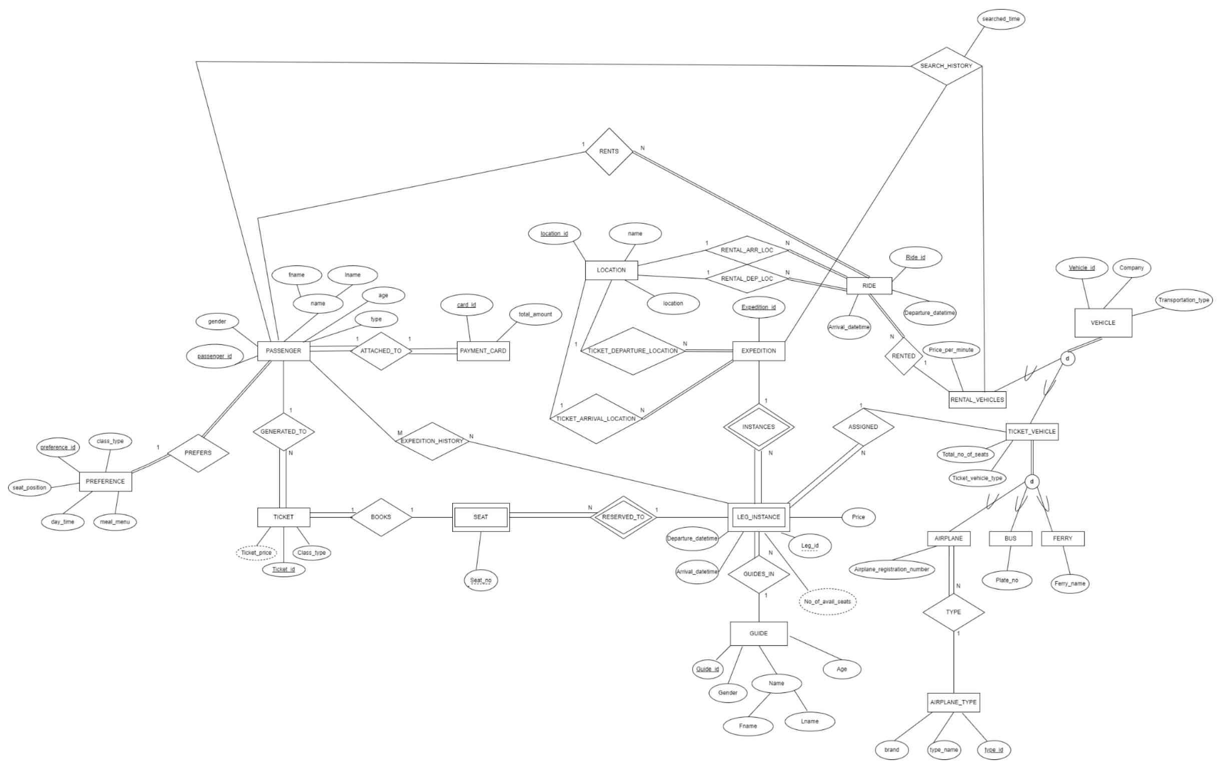
  - entity_type: Indicates the type of entity (EXPEDITION, PASSENGER, or RENTAL_VEHICLES) associated with the specific historical record.

- entity_id (FK): A foreign key attribute linking to the unique identifier of the corresponding entity (e.g., expedition_id, passenger_id, rental_vehicle_id).

- search_timestamp: Represents the timestamp when the historical record was created or modified.

- The inclusion of `search_id` as the primary key ensures a unique identifier for each historical entry, while `entity_type` distinguishes the type of entity involved. The foreign key `entity_id` establishes a link to the specific entity that the historical record refers to, and `search_timestamp` provides the chronological context for the historical data.

- This SEARCH_HISTORY relationship is valuable for auditing, analyzing changes over time, and maintaining a comprehensive historical perspective on the EXPEDITION, PASSENGER, and RENTAL_VEHICLES entities within the system. It enables tracking modifications, updates, or other relevant events for these entities.



ASSEMBLY PROCESS

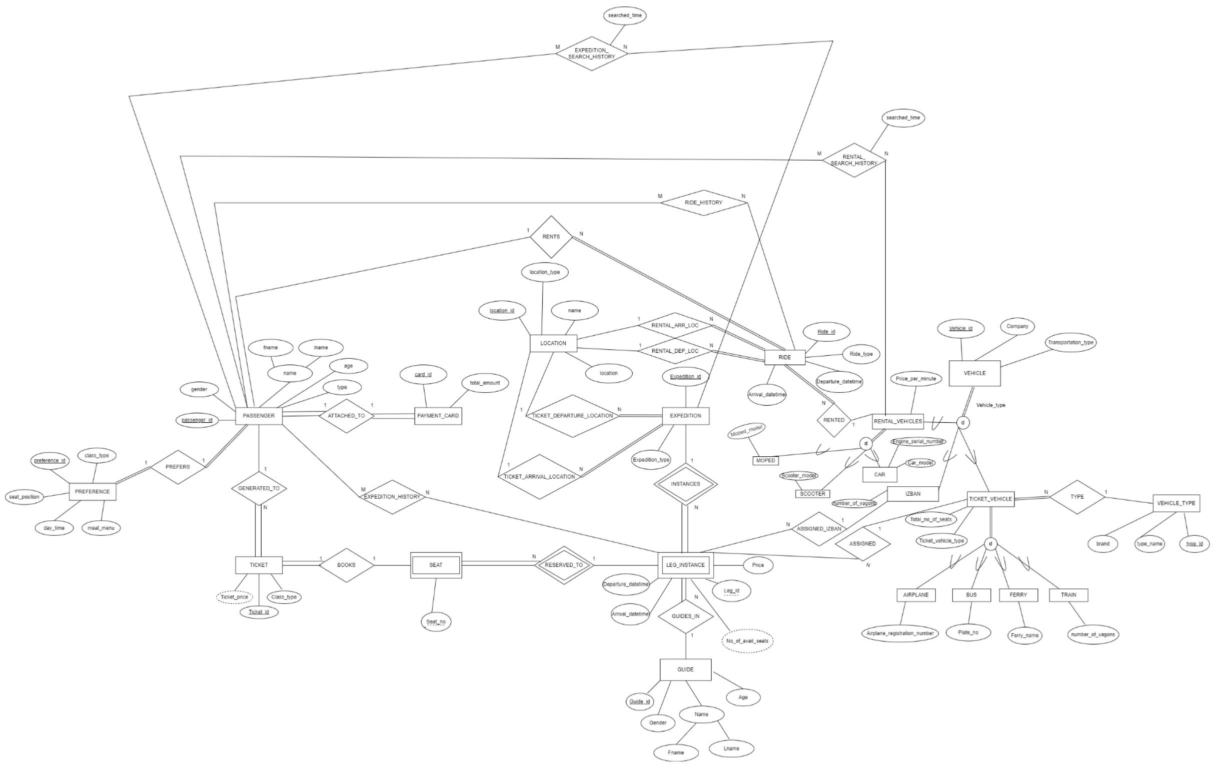
Combining all the EERs / Merging

After we have created the ER/EER diagrams for all seven websites, we discussed our method of combining all the diagrams together. As it is an important and indefinite process, (there is no exact algorithm for combining diagrams) we all had different opinions. But to put it broadly, we saw two roads ahead of us:
One of these roads were to combine all the diagrams loosely, meaning that we keep all entities of separate diagrams separate and we don't merge them, but we use relationships to "combine" the diagrams together, so we can say that this is a more specialized version as every diagram will keep most of their entities separate from other diagrams. But this would potentially slow the database down and make our SQL (both DDL AND DML) statements much harder to write.

The other road was to combine all the diagrams in a more generalized fashion, meaning that we would use one entity to represent the entities that had the same/alike relationships and purposes in their respective diagrams. This way also has its downfalls. Generalizing the diagrams in a more compact and one-size-fits-all way would result in some of the properties of the diagrams fading away. But we thought that it wouldn't be that big of a problem because we can still (in a way) save those properties and meanings of the diagrams using SQL.

In the end, we went with the second road and decided to combine all the diagrams in a more generalized way.
We combined all the stations, airports, ports etc. into a new entity called
Location
We combined all the vehicles and used EER specializations so that the important properties of the diagrams could still be represented (ticketed_vehicle, izban and rental_vehicle as they serve different purposes and have different characteristics in their respective diagrams)
we combined all expedition types for some vehicles (flight, train ride, bus ride, ferry ride) in a single expedition entity
We created a separate entity for rental vehicles' rides and combined them in that entity (for cars, mopeds and scooters) because it has different characteristics that cannot be represented in any other way
We merged the expedition_search_history, expedition_history, ride_search_history and ride_search history in their own respective relationships because all the relationships corresponding to the relationships that have been mentioned serve the same exact purpose.

All in all we can say that we didn't just use the 2nd way, we definitely used the 1st way for some situations but the process as a whole was more on the side of the 2nd way.



EER TO RELATIONAL MAPPING

1ST ITERATION:

STEP 1: REGULAR ENTITIES

PREFERENCE(<u>Preference_id</u>, class_type, seat_position, day_time, meal_menu)
PASSENGER(<u>Passenger_id</u>, Gender, Fname, Lname, Age, Type)
PAYMENT_CARD(<u>Card_id</u>, Total_amount)
TICKET(<u>Ticket_id</u>, Class_type)
LOCATION(<u>Location_id</u>, Name, Location, Location_type)
RIDE(<u>Ride_id</u>, Departure_datetime, Arrival_datetime, Ride_type)
GUIDE(<u>Guide_id</u>, Gender, Fname, Lname, Age)
EXPEDITION(<u>Expedition_id,</u> Expedition_type)
VEHICLE(<u>Vehicle_id</u>, Company, Transportation_type)
VEHICLE_TYPE(<u>Type_id</u>, Type_name, Brand)

STEP 2: MAPPING OF WEAK ENTITY TYPES

LEG_INSTANCE(Expedition_id, Leg_id, Price, Departure_datetime, Arrival_datetime)
//INSTANCES
EXPEDITION.Expedition.id ← LEG_INSTANCE.Expedition.id

STEP 3: MAPPING OF BINARY 1:1 RELATIONSHIP TYPES

PASSENGER(Passenger_id, gender, Fname, Lname, Age, Type, Card_id, Total_amount)
//ATTACHED_TO
//WE HAVE MERGED THE TWO ENTITIES PASSENGER AND PAYMENT_CARD THAT ARE RELATED TO EACH OTHER WITH THE ATTACHED_TO RELATIONSHIP AS THERE ARE TOTAL PARTICIPATION OF BOTH ENTITIES TO THE RELATIONSHIP
//The Card_id attribute should be set UNIQUE at the SQL level.

PREFERENCE(Preference_id, Passenger_id, Class_type, Seat_position, Day_time, Meal_menu)
//PREFERS
//**PASSENGER.Passenger_id ← PREFERENCE.Passenger_id**
//We could have used the approach above where we merged the entities PASSENGER and PAYMENT_CARD. But we chose to just include Passenger_id as a foreign key in the PREFERENCE relation because if we were to merge PREFERENCE and PASSENGER entities, the resulting table would be too horizontally big and it would be hard to manage it (too many attributes).

STEP 4: MAPPING OF BINARY 1:N RELATIONSHIP TYPES

TICKET(Ticket_id, Class_type, Passenger_id) //GENERATED_TO

EXPEDITION(Expedition_id, Expedition_type, *Arrival_location_id*)
//TICKET_ARRIVAL_LOCATION

EXPEDITION(Expedition_id, Expedition_type, *Arrival_location_id, Departure_location_id*)
//TICKET_DEPARTURE_LOCATION

RIDE(Ride_id, Departure_datetime, Arrival_datetime, *Arrival_location_id*)
//RENTAL_ARR_LOC

RIDE(Ride_id, Departure_datetime, Arrival_datetime, *Arrival_location_id, Departure_location_id*)
//RENTAL_DEP_LOC

RIDE(Ride_id, Departure_datetime, Arrival_datetime, *Arrival_location_id, Departure_location_id, Passenger_id*) //RENTS

LEG_INSTANCE(*Expedition_id*, Leg_id, Price, Departure_datetime, Arrival_datetime, *Guide_id*) //GUIDES_IN

STEP 5: MAPPING OF BINARY M:N RELATIONSHIP TYPES

EXPEDITION_HISTORY(*passenger_id, Expedition_id, Leg_id*) //EXPEDITION_HISTORY

EXPEDITION_SEARCH_HISTORY(*Passenger_id, Expedition_id,* searched_time)
//EXPEDITION_SEARCH_HISTORY

RIDE_HISTORY(*Passenger_id, Ride_id*) //RIDE_HISTORY

STEP 6: MAPPING OF MULTIVALUED ATTRIBUTES

No multivalued attributes.

STEP 7: MAPPING OF N-ARY RELATIONSHIPS

No n-ary relationships.

STEP 8: OPTIONS FOR MAPPING SPECIALIZATION OR GENERALIZATION

OPTION 8A: MULTIPLE RELATIONS-SUPERCLASS AND SUBCLASSES

RENTAL_VEHICLE(*Vehicle_id*, Price_per_minute) //Vehicle_type

IZBAN*(Vehicle_id*, number_of_vagons) //Vehicle_type

TICKET_VEHICLE(*Vehicle_id*, Total_no_of_seats, Ticket_vehicle_type) //Vehicle_type

STEP 9: MAPPING OF UNION TYPES (CATEGORIES)

No union types.

2ND ITERATION:

STEP 1: REGULAR ENTITIES

There are no regular entities left at this point.

STEP 2: MAPPING OF WEAK ENTITY TYPES

SEAT(*Expedition_id, Leg_id*, Seat_no) //RESERVED_TO

STEP 3: MAPPING OF BINARY 1:1 RELATIONSHIP TYPES

TICKET(Ticket_id, Class_type, Ticket_price, *Passenger_id, Expedition_id, Leg_id, Seat_no*) //BOOKS

STEP 4: MAPPING OF BINARY 1:N RELATIONSHIP TYPES

TICKET_VEHICLE(*Vehicle_id*, Total_no_of_seats, Ticket_vehicle_type, *type_id*) //TYPE

LEG_INSTANCE(*Expedition_id*, Leg_id, Price, *Departure_datetime, Arrival_datetime, Guide_id*, *Vehicle_id*) //ASSIGNED_IZBAN and ASSIGNED

RIDE(Ride_id, Departure_datetime, Arrival_datetime, *Arrival_location_id, Departure_location_id, Passenger_id, Vehicle_id*) //RENTED

STEP 5: MAPPING OF BINARY M:N RELATIONSHIP TYPES

RENTAL_SEARCH_HISTORY(*Passenger_id,Vehicle_id*, Searched_time) //RENTAL_SEARCH_HISTORY

STEP 6: MAPPING OF MULTIVALUED ATTRIBUTES

No multivalued attributes.

STEP 7: MAPPING OF N-ARY RELATIONSHIPS

No n-ary relationships.

STEP 8: OPTIONS FOR MAPPING SPECIALIZATION OR GENERALIZATION

OPTION 8C: SINGLE RELATION WITH ONE TYPE ATTRIBUTE

RENTAL_VEHICLE(*Vehicle_id*, Price_per_minute, Rental_vehicle_type, Moped_model, Scooter_model, Engine_serial_number, Car_model) //Rental_vehicles specialization

TICKET_VEHICLE(*Vehicle_id*, Total_no_of_seats, Ticket_vehicle_type, *type_id,* Airplane_registration_number, Plate_no, Ferry_name, number_of_vagons) //Ticket_vehicles specialization

STEP 9: MAPPING OF UNION TYPES (CATEGORIES)

No union types.

MAPPING IS COMPLETED.

FINAL STATE OF THE TABLES:

TICKET_VEHICLE(*Ticket_vehicle_id*, Total_no_of_seats, Ticket_vehicle_type, *Type_id,* Airplane_registration_number, Plate_no, Ferry_name, number_of_vagons) //Ticket_vehicles specialization (previously vehicle_id)

RENTAL_VEHICLE(*Rental_vehicle_id*, Price_per_minute, Rental_vehicle_type, Moped_model, Scooter_model, Engine_serial_number, Car_model) //Rental_vehicles specialization (previously vehicle_id)

RENTAL_SEARCH_HISTORY(*Passenger_id,Vehicle_id*, Searched_time) //RENTAL_SEARCH_HISTORY

LEG_INSTANCE(*Expedition_id*, Leg_id, Price, Departure_datetime, Arrival_datetime*, Guide_id*, *Ticket_vehicle_id, izban_id,* number_of_available_seats) //ASSIGNED_IZBAN and ASSIGNED (previously vehicle_id)

RIDE(Ride_id, Departure_date_time, Arrival_date_time, *Arrival_location_id, Departure_location_id, Passenger_id, Vehicle_id*) //RENTED

TICKET(Ticket_id, Class_type, Ticket_price, *Passenger_id, Expedition_id, Leg_id, Seat_no*) //BOOKS

SEAT(*Expedition_id, Leg_id,* Seat_no) //RESERVED_TO

IZBAN*(izban_id*, number_of_vagons) //Vehicle_type (previously vehicle_id)

RIDE_HISTORY(*Passenger_id, Ride_id*) //RIDE_HISTORY

EXPEDITION_HISTORY(*passenger_id, Expedition_id, Leg_id*) //EXPEDITION_HISTORY

EXPEDITION_SEARCH_HISTORY(*Passenger_id, Expedition_id,* Searched_time) //EXPEDITION_SEARCH_HISTORY

EXPEDITION(Expedition_id, Expedition_type, *Arrival_location_id, Departure_location_id*) //TICKET_DEPARTURE_LOCATION

PREFERENCE(Preference_id, *Passenger_id*, Class_type, Seat_position, Day_time, Meal_menu)

PASSENGER(Passenger_id, Gender, Fname, Lname, Age, Type, Card_id, Total_amount)

LOCATION(Location_id, Name, Location, Location_type)

GUIDE(Guide_id, Gender, Fname, Lname, Age)

VEHICLE(Vehicle_id, Company, Transportation_type)

VEHICLE_TYPE(Type_id, Type_name, Brand)

PRIMARY KEYS FOR EACH TABLE:

VEHICLE: VEHICLE.Vehicle_id
VEHICLE_TYPE: VEHICLE_TYPE.Type_id
GUIDE: GUIDE.Guide_id
PASSENGER: PASSENGER.Passenger_id
PREFERENCE: PREFERENCE.Preference_id
EXPEDITION: EXPEDITION.Expedition_id
TICKET_VEHICLE: TICKET_VEHICLE.Ticket_vehicle_id
RENTAL_VEHICLE: RENTAL_VEHICLE.Rental_vehicle_id
RENTAL_SEARCH_HISTORY:
RENTAL_SEARCH_HISTORY.{*Passenger_id,Vehicle_id*}
EXPEDITION_SEARCH_HISTORY:
EXPEDITION_SEARCH_HISTORY.{*Passenger_id, Expedition_id*}
LEG_INSTANCE: LEG_INSTANCE.{*Expedition_id*, Leg_id}
RIDE_HISTORY: RIDE_HISTORY.{*Passenger_id , Ride_id*}
RIDE: RIDE.Ride_id
TICKET: TICKET.Ticket_id
SEAT: SEAT{*Expedition_id, Leg_id*, Seat_no}
EXPEDITION_HISTORY: EXPEDITION_HISTORY.{*Passenger_id, Expedition_id, Leg_id*}
IZBAN: IZBAN.*izban_id*

List all the references to each primary key:

VEHICLE.Vehicle_id:

VEHICLE.Vehicle_id ← TICKET_VEHICLE.Ticket_vehicle_id
VEHICLE.Vehicle_id ← RENTAL_VEHICLE.Rental_vehicle_id
VEHICLE.Vehicle_id ← RENTAL_SEARCH_HISTORY.Vehicle_id
VEHICLE.Vehicle_id ← LEG_INSTANCE.izban_id
VEHICLE.Vehicle_id ← LEG_INSTANCE.ticket_vehicle_id
VEHICLE.Vehicle_id ← RIDE.Vehicle_id
VEHICLE.Vehicle_id ← IZBAN.izban_id

VEHICLE_TYPE.Type_id:

VEHICLE_TYPE.Type_id ← TICKET_VEHICLE.Type_id

GUIDE.Guide_id:

GUIDE.Guide_id ← LEG_INSTANCE.Guide_id

PASSENGER.Passenger_id:

PASSENGER.Passenger_id ← RENTAL_SEARCH_HISTORY.Passenger_id

PASSENGER.Passenger_id ← RIDE.Passenger_id

PASSENGER.Passenger_id ← TICKET.Passenger_id

PASSENGER.Passenger_id ← RIDE_HISTORY.Passenger_id

PASSENGER.Passenger_id ← EXPEDITION_HISTORY.Passenger_id

PASSENGER.Passenger_id ← PREFERENCE.Passenger_id

PREFERENCE.Preference_id:
-

EXPEDITION.Expedition_id

EXPEDITION.Expedition_id ← LEG_INSTANCE.Expedition_id

EXPEDITION.Expedition_id ← TICKET.Expedition_id

EXPEDITION.Expedition_id ← SEAT.Expedition_id

EXPEDITION.Expedition_id ← EXPEDITION_HISTORY.Expedition_id

EXPEDITION.Expedition_id ← EXPEDITION_SEARCH_HISTORY.Expedition_id

TICKET_VEHICLE.Ticket_vehicle_id:
-

RENTAL_VEHICLE.Rental_vehicle_id:
-

RENTAL_SEARCH_HISTORY.{*Passenger_id,Vehicle_id}*:

-

EXPEDITION_SEARCH_HISTORY.*{Passenger_id, Expedition_id}*:

-

LEG_INSTANCE.{*Expedition_id*, Leg_id}:

LEG_INSTANCE.{*Expedition_id*, Leg_id} ← SEAT.{*Expedition_id, Leg_id*}

LEG_INSTANCE.{*Expedition_id*, Leg_id} ← EXPEDITION_HISTORY.{*Expedition_id*, Leg_id}

RIDE_HISTORY.*{Passenger_id , Ride_id}*:

-

RIDE.Ride_id:

RIDE.Ride_id ← RIDE_HISTORY.Ride_id

TICKET.Ticket_id:

-

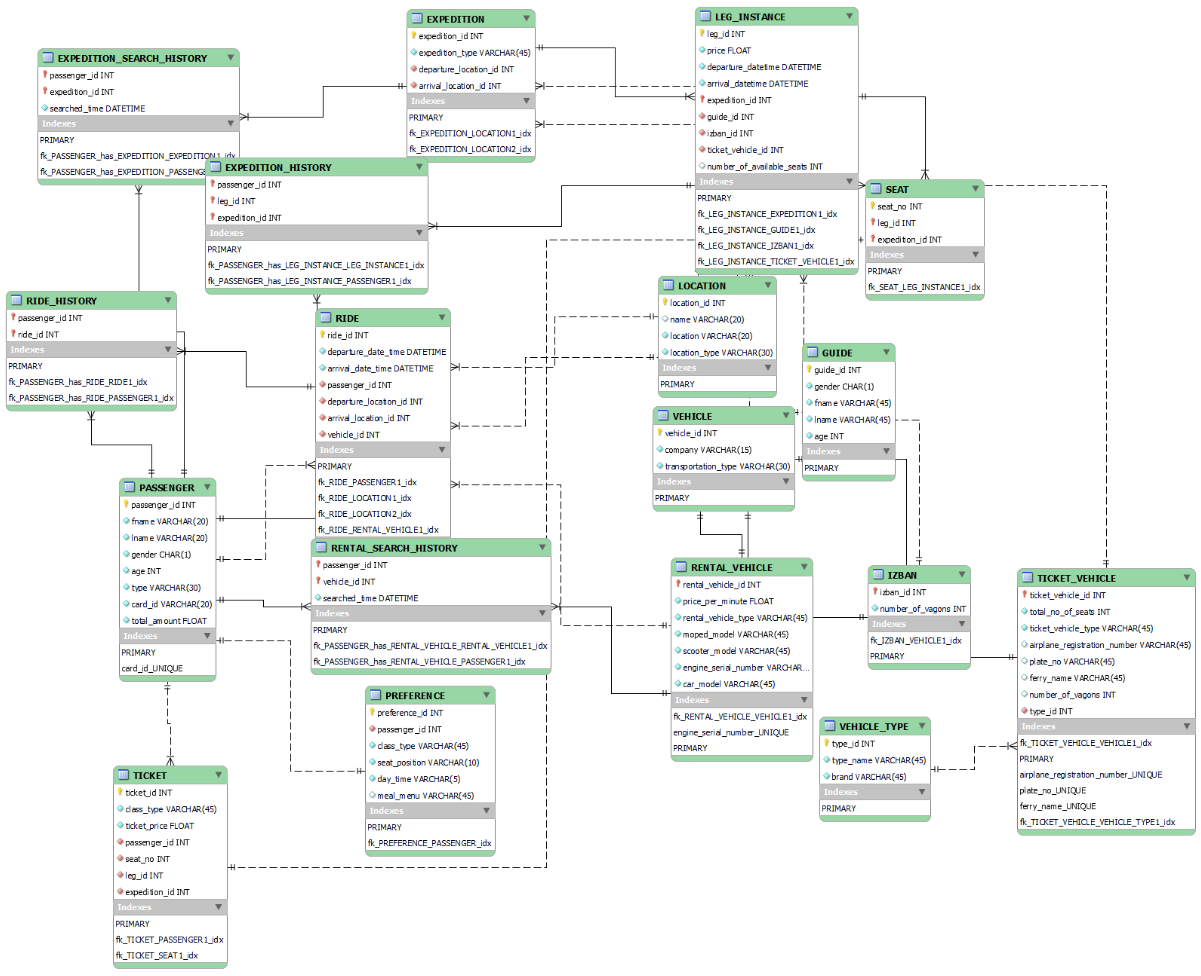
SEAT.{*Expedition_id, Leg_id*, Seat_no}:

SEAT.{Expedition_id, Leg_id, Seat_no} ← TICKET.{Expedition_id, Leg_id, Seat_no}

EXPEDITION_HISTORY.*{Passenger_id, Expedition_id, Leg_id}*:

-

IMPLEMENTATION-PHYSICAL MODEL



CREATING TABLES

```
DROP DATABASE transportation_db;
CREATE DATABASE transportation_db;
USE transportation_db;

CREATE TABLE PASSENGER (
  passenger_id INT AUTO_INCREMENT,
  fname VARCHAR(20) NOT NULL,
  lname VARCHAR(20) NOT NULL,
  gender CHAR(1) NOT NULL,
  age INT NOT NULL DEFAULT 18,
  type VARCHAR(30) NOT NULL,
  card_id VARCHAR(20) NOT NULL UNIQUE,
  total_amount FLOAT NOT NULL DEFAULT 0,
  PRIMARY KEY (passenger_id),
  CONSTRAINT check_age CHECK (age >= 0),
  CONSTRAINT check_amount CHECK (total_amount >= 0)
);

CREATE TABLE PREFERENCE (
  preference_id INT AUTO_INCREMENT,
  passenger_id INT NOT NULL,
  class_type VARCHAR(45),
  seat_position VARCHAR(10) DEFAULT 'corridor',
  day_time VARCHAR(5) NOT NULL DEFAULT 'day',
  meal_menu VARCHAR(45) DEFAULT NULL,
  PRIMARY KEY (preference_id),
  FOREIGN KEY (passenger_id) REFERENCES
PASSENGER(passenger_id) ON DELETE CASCADE
);

CREATE TABLE LOCATION (
  location_id INT AUTO_INCREMENT,
  name VARCHAR(50) NULL DEFAULT NULL,
  location VARCHAR(50) NOT NULL,
  location_type VARCHAR(50) NOT NULL,
  PRIMARY KEY (location_id)
);

CREATE TABLE EXPEDITION (
  expedition_id INT AUTO_INCREMENT,
  expedition_type VARCHAR(45) NOT NULL,
  departure_location_id INT NOT NULL,
  arrival_location_id INT NOT NULL,
  PRIMARY KEY (expedition_id),
  FOREIGN KEY (departure_location_id) REFERENCES
LOCATION(location_id) ON DELETE CASCADE,
  FOREIGN KEY (arrival_location_id) REFERENCES
LOCATION(location_id) ON DELETE CASCADE
);

CREATE TABLE GUIDE (
  guide_id INT AUTO_INCREMENT,
  gender CHAR(1) NOT NULL,
  fname VARCHAR(45) NOT NULL,
  lname VARCHAR(45) NOT NULL,
  age INT NOT NULL DEFAULT 18,
  PRIMARY KEY (guide_id)
);

CREATE TABLE VEHICLE (
  vehicle_id INT AUTO_INCREMENT,
  company VARCHAR(30) NOT NULL,
  transportation_type VARCHAR(50) NOT NULL,
  PRIMARY KEY (vehicle_id)
);

CREATE TABLE IZBAN (
  izban_id INT AUTO_INCREMENT,
  number_of_vagons INT NOT NULL DEFAULT 6,
  PRIMARY KEY (izban_id),
  FOREIGN KEY (izban_id) REFERENCES VEHICLE(vehicle_id)
ON DELETE CASCADE,
  CONSTRAINT check_number_of_vagons CHECK
(number_of_vagons >= 1)
);

CREATE TABLE VEHICLE_TYPE (
  type_id INT AUTO_INCREMENT,
  type_name VARCHAR(45) NOT NULL,
  brand VARCHAR(45) NOT NULL,
  PRIMARY KEY (type_id)
);

CREATE TABLE TICKET_VEHICLE (
  ticket_vehicle_id INT AUTO_INCREMENT,
  total_no_of_seats INT NOT NULL DEFAULT 100,
  ticket_vehicle_type VARCHAR(45) NOT NULL,
  airplane_registration_number VARCHAR(45) UNIQUE DEFAULT
NULL,
  plate_no VARCHAR(45) UNIQUE DEFAULT NULL,
  ferry_name VARCHAR(45) UNIQUE DEFAULT NULL,
  number_of_vagons INT DEFAULT NULL,
  type_id INT,
  PRIMARY KEY (ticket_vehicle_id),
  FOREIGN KEY (ticket_vehicle_id) REFERENCES VEHICLE
(vehicle_id) ON UPDATE CASCADE ON DELETE CASCADE,
  FOREIGN KEY (type_id) REFERENCES VEHICLE_TYPE (type_id)
ON DELETE SET NULL
);

CREATE TABLE LEG_INSTANCE (
  leg_id INT AUTO_INCREMENT,
  price FLOAT NOT NULL DEFAULT 0,
  departure_datetime DATETIME NOT NULL DEFAULT
'1999-01-01',
  arrival_datetime DATETIME NOT NULL DEFAULT
'1999-01-01',
  expedition_id INT NOT NULL,
  guide_id INT,
  izban_id INT,
  ticket_vehicle_id INT,
  number_of_available_seats INT,
  PRIMARY KEY (leg_id, expedition_id),
  FOREIGN KEY (expedition_id) REFERENCES EXPEDITION
(expedition_id) ON DELETE CASCADE,
  FOREIGN KEY (guide_id) REFERENCES GUIDE (guide_id),
  FOREIGN KEY (izban_id) REFERENCES IZBAN (izban_id),
  FOREIGN KEY (ticket_vehicle_id) REFERENCES
TICKET_VEHICLE (ticket_vehicle_id),
  CONSTRAINT check_price CHECK (price >= 0),
  CONSTRAINT check_seats CHECK (number_of_available_seats
>= 0)
);

CREATE TABLE SEAT (
  seat_no INT AUTO_INCREMENT,
  leg_id INT NOT NULL,
  expedition_id INT NOT NULL,
  PRIMARY KEY (seat_no, leg_id, expedition_id),
  FOREIGN KEY (leg_id , expedition_id) REFERENCES
LEG_INSTANCE (leg_id , expedition_id) ON UPDATE CASCADE
ON DELETE CASCADE
);

CREATE TABLE TICKET (
  ticket_id INT AUTO_INCREMENT,
  class_type VARCHAR(45) NOT NULL,
  ticket_price FLOAT NOT NULL DEFAULT 0,
  passenger_id INT NOT NULL,
  seat_no INT NOT NULL,
  leg_id INT NOT NULL,
  expedition_id INT NOT NULL,
  PRIMARY KEY (ticket_id),
  FOREIGN KEY (passenger_id) REFERENCES PASSENGER
(passenger_id) ON DELETE CASCADE,
  FOREIGN KEY (seat_no , leg_id , expedition_id)
REFERENCES SEAT (seat_no , leg_id , expedition_id)
);
```

```
CREATE TABLE RENTAL_VEHICLE (
  rental_vehicle_id INT AUTO_INCREMENT,
  price_per_minute FLOAT NOT NULL,
  rental_vehicle_type VARCHAR(45) NOT NULL,
  moped_model VARCHAR(45),
  scooter_model VARCHAR(45),
  engine_serial_number VARCHAR(45) UNIQUE,
  car_model VARCHAR(45),
  PRIMARY KEY (rental_vehicle_id),
  FOREIGN KEY (rental_vehicle_id) REFERENCES VEHICLE
(vehicle_id) ON DELETE CASCADE
);

CREATE TABLE RIDE (
  ride_id INT AUTO_INCREMENT,
  departure_date_time DATETIME NOT NULL DEFAULT
'1999-01-01',
  arrival_date_time DATETIME NOT NULL DEFAULT
'1999-01-01',
  passenger_id INT NOT NULL,
  departure_location_id INT NOT NULL,
  arrival_location_id INT NOT NULL,
  vehicle_id INT NOT NULL,
  PRIMARY KEY (ride_id),
  FOREIGN KEY (passenger_id) REFERENCES PASSENGER
(passenger_id) ON DELETE CASCADE,
  FOREIGN KEY (departure_location_id) REFERENCES LOCATION
(location_id) ON DELETE CASCADE,
  FOREIGN KEY (arrival_location_id) REFERENCES LOCATION
(location_id) ON DELETE CASCADE,
  FOREIGN KEY (vehicle_id) REFERENCES RENTAL_VEHICLE
(rental_vehicle_id)
);
```

```
CREATE TABLE RIDE_HISTORY (
  passenger_id INT NOT NULL,
  ride_id INT NOT NULL,
  PRIMARY KEY (passenger_id, ride_id),
  FOREIGN KEY (passenger_id) REFERENCES PASSENGER
(passenger_id) ON DELETE CASCADE,
  FOREIGN KEY (ride_id) REFERENCES RIDE (ride_id) ON
DELETE CASCADE
);

CREATE TABLE RENTAL_SEARCH_HISTORY (
  passenger_id INT NOT NULL,
  vehicle_id INT NOT NULL,
  searched_time DATETIME NOT NULL DEFAULT '1999-01-01',
  PRIMARY KEY (passenger_id, vehicle_id),
  FOREIGN KEY (passenger_id) REFERENCES PASSENGER
(passenger_id) ON DELETE CASCADE,
  FOREIGN KEY (vehicle_id) REFERENCES RENTAL_VEHICLE
(rental_vehicle_id)
);

CREATE TABLE EXPEDITION_SEARCH_HISTORY (
  passenger_id INT NOT NULL,
  expedition_id INT NOT NULL,
  searched_time DATETIME NOT NULL DEFAULT '1999-01-01',
  PRIMARY KEY (passenger_id, expedition_id),
  FOREIGN KEY (passenger_id) REFERENCES PASSENGER
(passenger_id) ON DELETE CASCADE,
  FOREIGN KEY (expedition_id) REFERENCES EXPEDITION
(expedition_id)
);

CREATE TABLE EXPEDITION_HISTORY (
  passenger_id INT NOT NULL,
  leg_id INT NOT NULL,
  expedition_id INT NOT NULL,
  PRIMARY KEY (passenger_id, leg_id, expedition_id),
  FOREIGN KEY (passenger_id) REFERENCES PASSENGER
(passenger_id) ON DELETE CASCADE,
  FOREIGN KEY (leg_id , expedition_id) REFERENCES
LEG_INSTANCE (leg_id , expedition_id)
);
```

TRIGGERS

```
USE transportation_db;

-- DROP TRIGGER check_location_for_expedition_type;
-- DROP TRIGGER check_vehicle_type_for_ride;
-- DROP TRIGGER
check_vehicle_is_ticket_vehicle_before_ticket_insert;
-- DROP TRIGGER check_vehicle_type_before_seat_insert;
-- DROP TRIGGER
expedition_departure_arrival_location_check;
-- DROP TRIGGER
leg_instance_departure_arrival_datetime_check;
-- DROP TRIGGER ride_departure_arrival_datetime_check;
-- DROP TRIGGER leg_instance_add_no_of_seats;
-- DROP TRIGGER ticket_insert_update_and_check_seats;

DELIMITER //
CREATE TRIGGER check_location_for_expedition_type
BEFORE INSERT ON EXPEDITION
FOR EACH ROW
BEGIN
    DECLARE departure_location_type VARCHAR(30);
    DECLARE arrival_location_type VARCHAR(30);

    -- Fetch location types for departure and arrival
locations
```

```
    SELECT location_type INTO departure_location_type
    FROM LOCATION
    WHERE location_id = NEW.departure_location_id;

    SELECT location_type INTO arrival_location_type
    FROM LOCATION
    WHERE location_id = NEW.arrival_location_id;

    -- Check expedition type
    IF NEW.expedition_type NOT IN ('bus', 'train',
'izban', 'ferry', 'flight') THEN
         SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'Invalid expedition type!';
     END IF;

    -- Check for flight type
    IF NEW.expedition_type = 'flight' THEN
        IF departure_location_type != 'airport' OR
arrival_location_type != 'airport' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'For flights, both
departure and arrival locations must be airports!';
        END IF;
    END IF;

    -- Check for ferry type
    IF NEW.expedition_type = 'ferry' THEN
        IF departure_location_type != 'port' OR
arrival_location_type != 'port' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'For ferries, both
departure and arrival locations must be ports!';
        END IF;
    END IF;
    -- Check for bus type
    IF NEW.expedition_type = 'bus' THEN
        IF departure_location_type != 'bus_station' OR
arrival_location_type != 'bus_station' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'For buses, both departure
and arrival locations must be bus stations!';
        END IF;
    END IF;

    -- Check for train type
    IF NEW.expedition_type = 'train' THEN
        IF departure_location_type != 'train_station' OR
arrival_location_type != 'train_station' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'For trains, both
departure and arrival locations must be train stations!';
        END IF;
    END IF;

    -- Check for IZBAN type
    IF NEW.expedition_type = 'izban' THEN
        IF departure_location_type != 'izban_station' OR
arrival_location_type != 'izban_station' THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'For izban, both departure
and arrival locations must be izban stations!';
        END IF;
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER check_vehicle_type_for_ride
```

```
BEFORE INSERT ON RIDE
FOR EACH ROW
BEGIN
    DECLARE vehicle_type VARCHAR(45);
    DECLARE departure_location_type VARCHAR(30);

    -- Fetch the vehicle type
    SELECT rental_vehicle_type INTO vehicle_type
    FROM RENTAL_VEHICLE
    WHERE rental_vehicle_id = NEW.vehicle_id;

    -- Fetch the departure location type
    SELECT location_type INTO departure_location_type
    FROM LOCATION
    WHERE location_id = NEW.departure_location_id;

    -- Check for Car type and departure location
    IF vehicle_type = 'Car' THEN
        IF departure_location_type != 'renting_location'
THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'For Car rentals, the
departure location must be a car rental station';
        END IF;
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER
check_vehicle_is_ticket_vehicle_before_ticket_insert
BEFORE INSERT ON TICKET
FOR EACH ROW
BEGIN
```

```
    DECLARE fetched_ticket_vehicle_id INT;
    DECLARE is_ticket_vehicle BOOLEAN;

    -- Fetch the ticket_vehicle_id from LEG_INSTANCE
    SELECT ticket_vehicle_id INTO
fetched_ticket_vehicle_id
    FROM LEG_INSTANCE
    WHERE leg_id = NEW.leg_id AND expedition_id =
NEW.expedition_id;

    -- Check if the vehicle is a TICKET_VEHICLE
    SELECT EXISTS (
        SELECT 1
        FROM TICKET_VEHICLE
        WHERE ticket_vehicle_id =
fetched_ticket_vehicle_id
    ) INTO is_ticket_vehicle;

    -- If the vehicle is not a TICKET_VEHICLE, prevent
the insertion
    IF NOT is_ticket_vehicle THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Tickets can only be added for
vehicles that are TICKET_VEHICLES';
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER check_vehicle_type_before_seat_insert
BEFORE INSERT ON SEAT
FOR EACH ROW
BEGIN
    DECLARE leg_instance_ticket_vehicle_id INT;
    DECLARE is_ticket_vehicle BOOLEAN;
    DECLARE is_izban BOOLEAN;

    -- Fetch the ticket_vehicle_id from LEG_INSTANCE
    SELECT ticket_vehicle_id INTO
leg_instance_ticket_vehicle_id
    FROM LEG_INSTANCE
    WHERE leg_id = NEW.leg_id AND expedition_id =
NEW.expedition_id;

    -- Check if the vehicle is a TICKET_VEHICLE
    SELECT EXISTS (
        SELECT 1
        FROM TICKET_VEHICLE
        WHERE ticket_vehicle_id =
leg_instance_ticket_vehicle_id
    ) INTO is_ticket_vehicle;

    -- Check if the vehicle is an IZBAN
    SELECT EXISTS (
        SELECT 1
        FROM IZBAN
        WHERE izban_id = leg_instance_ticket_vehicle_id
    ) INTO is_izban;

    -- If the vehicle is not a TICKET_VEHICLE or if it's
an IZBAN, prevent the insertion
    IF NOT is_ticket_vehicle OR is_izban THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Seats can only be added for
vehicles that are TICKET_VEHICLES and not IZBAN';
    END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER
expedition_departure_arrival_location_check
BEFORE INSERT ON EXPEDITION
FOR EACH ROW
BEGIN
    IF NEW.departure_location_id =
NEW.arrival_location_id THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Departure and arrival
locations cannot be same!';
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER
leg_instance_departure_arrival_datetime_check
BEFORE INSERT ON LEG_INSTANCE
FOR EACH ROW
BEGIN
    IF NEW.departure_datetime >= NEW.arrival_datetime
THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid departure and arrival
datetimes!';
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER ride_departure_arrival_datetime_check
```

```
BEFORE INSERT ON RIDE
FOR EACH ROW
BEGIN
    IF NEW.departure_date_time >= NEW.arrival_date_time
THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid departure and arrival
datetimes!';
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER leg_instance_add_no_of_seats
BEFORE INSERT ON LEG_INSTANCE
FOR EACH ROW
BEGIN
    DECLARE number_of_seats INT DEFAULT NULL;

    SELECT TIV.total_no_of_seats INTO number_of_seats
    FROM TICKET_VEHICLE AS TIV
    WHERE TIV.ticket_vehicle_id = NEW.ticket_vehicle_id;

    SET NEW.number_of_available_seats = number_of_seats;
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER ticket_insert_update_and_check_seats
BEFORE INSERT ON TICKET
FOR EACH ROW
BEGIN
    DECLARE available_seats INT;
    SELECT number_of_available_seats INTO available_seats
    FROM LEG_INSTANCE
    WHERE leg_id = NEW.leg_id AND expedition_id =
NEW.expedition_id;

    IF available_seats <= 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot add ticket because no
available seats!';
    ELSE
        UPDATE LEG_INSTANCE
        SET number_of_available_seats =
number_of_available_seats - 1
        WHERE leg_id = NEW.leg_id AND expedition_id =
NEW.expedition_id;
    END IF;
END;
//
DELIMITER ;
```

INSERTING VALUES

```
USE transportation_db;

INSERT INTO VEHICLE_TYPE (type_name, brand) VALUES
('Airplane', 'Airbus A380'),
('Bus', 'Mercedes-Benz Sprinter'),
('Ferry', 'Metal Shark'),
('Train', 'Siemens Velaro'),
('Airplane', 'Boeing 777'),
('Bus', 'Volvo 9700'),
('Ferry', 'Cruise Ferry 2000'),
('Train', 'Alstom Euroduplex'),
('Airplane', 'Embraer E190'),
('Bus', 'Neoplan Tourliner');

INSERT INTO LOCATION (name, location, location_type)
VALUES
('Istanbul Ataturk Airport', 'Istanbul', 'airport'),
('Taksim Square', 'Istanbul', 'charging_station'),
('Kusadasi Port', 'Kusadasi', 'port'),
('Esenler Bus Station', 'Istanbul', 'bus_station'),
('Alsancak Izban Station', 'Izmir', 'izban_station'),
('Antalya Airport', 'Antalya', 'airport'),
('Cappadocia Charging Station', 'Nevsehir',
'charging_station'),
('Bodrum Port', 'Bodrum', 'port'),
('Ankara Bus Station', 'Ankara', 'bus_station'),
('Gaziemir Izban Station', 'Izmir', 'izban_station'),
('Sabiha Gokcen Airport', 'Istanbul', 'airport'),
('Izmir Konak Square Charging Station', 'Izmir',
'charging_station'),
('Marmaris Port', 'Marmaris', 'port'),
('Busan Bus Station', 'Busan', 'bus_station'),
('Karsiyaka Izban Station', 'Izmir', 'izban_station'),
('Dalaman Airport', 'Dalaman', 'airport'),
('Konya Charging Station', 'Konya', 'charging_station'),
('Canakkale Port', 'Canakkale', 'port'),
('Trabzon Bus Station', 'Trabzon', 'bus_station'),
('Bornova Izban Station', 'Izmir', 'izban_station'),
('Izmit Charging Station', 'Izmit', 'charging_station'),
('Cesme Port', 'Cesme', 'port'),
('Gaziantep Bus Station', 'Gaziantep', 'bus_station'),
('Alsancak Renting Location 1', 'Izmir',
'renting_location'),
('Istanbul Renting Location 1', 'Istanbul',
'renting_location'),
('Ephesus Charging Station', 'Selcuk',
'charging_station'),
('Fethiye Port', 'Fethiye', 'port'),
('Samsun Bus Station', 'Samsun', 'bus_station'),
('Kars Renting Location 2', 'Kars', 'renting_location'),
('Adana Renting Location 2', 'Adana',
'renting_location'),
('Antakya Charging Station', 'Antakya',
'charging_station'),
('Bursa Bus Station', 'Bursa', 'bus_station'),
('Izmir Konak Renting Location 3', 'Izmir',
'renting_location'),
('Istanbul Renting Location 2', 'Istanbul',
'renting_location'),
('Mugla Port', 'Mugla', 'port'),
('Erzurum Bus Station', 'Erzurum', 'bus_station'),
('Konya Renting Location 1', 'Konya',
'renting_location'),
('Trabzon Port', 'Trabzon', 'port'),
('Denizli Bus Station', 'Denizli', 'bus_station'),
('Eskisehir Renting Location 4', 'Eskisehir',
'renting_location'),
('Eskisehir Train Location', 'Eskisehir',
'train_station'),
('Erzurum Train Location', 'Erzurum', 'train_station');

INSERT INTO VEHICLE (company, transportation_type) VALUES
('TCDD', 'Ticketed_vehicle'),
('THY', 'Ticketed_vehicle'),
('IDO', 'Ticketed_vehicle'),
('Martı', 'Rental_vehicle'),
('OBilet', 'Ticketed_vehicle'),
('Yolcu360', 'Rental_vehicle'),
('IZBAN', 'IZBAN'),
('TCDD', 'Ticketed_vehicle'),
('THY', 'Ticketed_vehicle'),
('IDO', 'Ticketed_vehicle'),
('Martı', 'Rental_vehicle'),
('OBilet', 'Rental_vehicle'),
('Yolcu360', 'Rental_vehicle'),
('IZBAN', 'IZBAN'),
('TCDD', 'Ticketed_vehicle'),
('THY', 'Ticketed_vehicle'),
('IDO', 'Ticketed_vehicle'),
('Martı', 'Rental_vehicle'),
('OBilet', 'Ticketed_vehicle'),
('TCDD', 'Ticketed_vehicle');

INSERT INTO PASSENGER (fname, lname, gender, age, type,
card_id, total_amount) VALUES
('Ali', 'Yılmaz', 'M', 23, 'Student', 'CARD12345', 500),
('Ayşe', 'Demir', 'F', 35, 'Regular', 'CARD23456', 700),
('Mustafa', 'Arslan', 'M', 40, 'Regular', 'CARD314567',
600),
('Fatma', 'Kaya', 'F', 28, 'Student', 'CARD456718', 550),
('Hüseyin', 'Öztürk', 'M', 19, 'Student', 'CARD156789',
480),
('Zeynep', 'Çelik', 'F', 75, 'Elderly', 'CARD678190',
800),
('Mehmet', 'Koç', 'M', 33, 'Regular', 'CARD781901', 720),
('Aslı', 'Yıldız', 'F', 80, 'Elderly', 'CARD82', 900),
('Can', 'Şahin', 'M', 21, 'Student', 'CARD190123', 520),
('Seda', 'Aydın', 'F', 70, 'Elderly', 'CARD012134', 850),
('Emre', 'Ergin', 'M', 29, 'Healthcare_worker',
'CARD1231456', 750),
('Nur', 'Avcı', 'F', 25, 'Student', 'CARD21345678', 480),
('Ercan', 'Uzun', 'M', 32, 'Regular', 'CARD34567189',
690),
('Gizem', 'Yavuz', 'F', 22, 'Student', 'CARD1456789',
510),
('Tolga', 'Güler', 'M', 68, 'Elderly', 'CARD5678190',
670),
('Elif', 'Aydın', 'F', 65, 'Healthcare_worker',
'CARD67189012', 880),
('Burak', 'Kurt', 'M', 26, 'Healthcare_worker',
'CARD178901', 780),
('Esra', 'Erdem', 'F', 70, 'Elderly', 'CARD890112', 710),
('Cem', 'Kara', 'M', 24, 'Student', 'CARD910123', 530),
('Duygu', 'Güneş', 'F', 42, 'Regular', 'CARD012345617',
720);

INSERT INTO GUIDE (gender, fname, lname, age) VALUES
('M', 'İsmail', 'Yıldırım', 36),
('F', 'Aylin', 'Can', 42),
('M', 'Burhan', 'Arslan', 38),
('F', 'Ceren', 'Aksoy', 29),
('M', 'Kadir', 'Öztürk', 27),
('F', 'Nihan', 'Taş', 48),
('M', 'Mert', 'Koç', 35),
('F', 'Gamze', 'Kaya', 41),
('M', 'Engin', 'Şahin', 31),
('F', 'Gülçin', 'Aydın', 43),
('M', 'Emir', 'Erdem', 33),
('F', 'Selma', 'Avcı', 30),
('M', 'Tarkan', 'Uzun', 37),
('F', 'Ece', 'Yavuz', 28),
('M', 'Okan', 'Güler', 42),
('F', 'Şule', 'Aydın', 50),
('M', 'Umut', 'Kurt', 29),
('F', 'Elif', 'Erdem', 26),
('M', 'Kaan', 'Kara', 34),
('F', 'Cansu', 'Güneş', 39);

INSERT INTO TICKET_VEHICLE (ticket_vehicle_id,
total_no_of_seats, ticket_vehicle_type,
airplane_registration_number, plate_no, ferry_name,
number_of_vagons, type_id) VALUES
(1, 150, 'Train', NULL, NULL, NULL, 5, 4),
(2, 200, 'Airplane', 'ABC123', NULL, NULL, NULL, 1),
(3, 100, 'Ferry', NULL, NULL, 'FerryA', NULL, 3),
(5, 50, 'Bus', NULL, 'XY123', NULL, NULL, 2),
(8, 150, 'Train', NULL, NULL, NULL, 4, 4),
(9, 200, 'Airplane', 'XYZ789', NULL, NULL, NULL , 1),
(10, 100, 'Ferry', NULL, NULL, 'FerryB', NULL, 3),
(15, 150, 'Train', NULL, NULL, NULL, 3, 4),
(16, 200, 'Airplane', 'DEF456', NULL, NULL, NULL, 1),
(17, 100, 'Ferry', NULL, NULL, 'FerryC', NULL, 3),
(19, 50, 'Bus', NULL, 'WX789', NULL, NULL, 2);

INSERT INTO RENTAL_VEHICLE (rental_vehicle_id,
price_per_minute, rental_vehicle_type, moped_model,
scooter_model, engine_serial_number, car_model) VALUES
(4, 0.2, 'Moped', 'MopedModelA', NULL, NULL, NULL),
(6, 0.5, 'Car', NULL, NULL, 'EngineSerialA',
'CarModelA'),
(11, 0.3, 'Scooter', NULL, 'ScooterModelB', NULL, NULL),
(12, 0.6, 'Car', NULL, NULL, 'EngineSerialB',
'CarModelB'),
(13, 0.7, 'Car', NULL, NULL, 'EngineSerialC',
'CarModelC'),
(18, 0.25, 'Moped', 'MopedModelB', NULL, NULL, NULL);


INSERT INTO RENTAL_SEARCH_HISTORY (passenger_id,
vehicle_id, searched_time) VALUES
(1, 4, '2023-03-15 10:30:00'),
(2, 6, '2023-04-20 15:45:00'),
(3, 11, '2023-05-12 08:20:00'),
(4, 12, '2023-06-08 12:10:00'),
(5, 13, '2023-07-05 17:00:00'),
(6, 18, '2023-08-18 09:55:00'),
(7, 4, '2023-09-23 14:30:00'),
(8, 6, '2023-10-14 11:15:00'),
(9, 11, '2023-11-27 19:40:00'),
(10, 12, '2023-12-09 22:05:00'),
(11, 13, '2024-01-30 16:50:00'),
(12, 18, '2024-02-14 07:25:00'),
(13, 4, '2024-03-08 18:15:00'),
(14, 6, '2024-04-04 13:05:00'),
(15, 11, '2024-05-19 20:45:00'),
(16, 12, '2024-06-22 10:00:00'),
(17, 13, '2024-07-17 23:30:00'),
(18, 18, '2024-08-03 05:40:00'),
(19, 4, '2024-09-11 08:55:00'),
(20, 6, '2024-10-27 14:20:00');


INSERT INTO RIDE (departure_date_time, arrival_date_time,
passenger_id, departure_location_id, arrival_location_id,
vehicle_id) VALUES
('2024-01-01 08:00:00', '2024-01-01 12:00:00', 1, 25, 30,
6),
('2024-01-02 10:30:00', '2024-01-02 14:45:00', 2, 9, 10,
4),
('2024-01-03 14:00:00', '2024-01-03 17:30:00', 3, 5, 6,
11),
('2024-01-04 08:45:00', '2024-01-04 12:15:00', 4, 33, 34,
6),
('2024-01-05 09:30:00', '2024-01-05 12:45:00', 5, 9, 10,
4),
('2024-01-06 11:15:00', '2024-01-06 15:30:00', 6, 34, 37,
13),
('2024-01-07 14:30:00', '2024-01-07 17:45:00', 1, 15, 16,
11),
('2024-01-08 07:45:00', '2024-01-08 11:15:00', 2, 37, 38,
11),
('2024-01-09 12:00:00', '2024-01-09 15:30:00', 3, 5, 6,
11),
('2024-01-10 10:15:00', '2024-01-10 14:30:00', 4, 37, 33,
6);

INSERT INTO IZBAN (izban_id, number_of_vagons) VALUES
(7, 6),
(14, 9);

INSERT INTO EXPEDITION (expedition_type,
departure_location_id, arrival_location_id) VALUES
('flight', 1, 6),
('ferry', 3, 8),
('izban', 5, 10),
('train', 41, 42),
('bus', 4, 19),
('flight', 6, 1),
('ferry', 8, 3),
('izban', 10, 5),
('train', 42, 41),
('bus', 19, 4);
```

```
INSERT INTO PREFERENCE (passenger_id, class_type,
seat_position, day_time, meal_menu) VALUES
(1, 'First Class', 'window', 'day', 'Chicken'),
(2, 'Economy', 'corridor', 'night', 'Fish'),
(3, 'Business', 'window', 'day', 'Vegetarian'),
(4, 'Economy', 'corridor', 'night', 'Pasta'),
(5, 'First Class', 'window', 'day', 'Beef'),
(6, 'Business', 'corridor', 'night', 'Steak'),
(7, 'Economy', 'corridor', 'day', 'Fish'),
(8, 'First Class', 'window', 'night', 'Pasta'),
(9, 'Business', 'corridor', 'day', 'Chicken'),
(10, 'Economy', 'corridor', 'night', 'Vegetarian'),
(11, 'First Class', 'window', 'day', 'Beef'),
(12, 'Business', 'window', 'night', 'Steak'),
(13, 'Economy', 'corridor', 'day', 'Pasta'),
(14, 'First Class', 'window', 'night', 'Chicken'),
(15, 'Business', 'corridor', 'day', 'Fish'),
(16, 'Economy', 'corridor', 'night', 'Steak'),
(17, 'First Class', 'window', 'day', 'Beef'),
(18, 'Business', 'window', 'night', 'Pasta'),
(19, 'Economy', 'corridor', 'day', 'Vegetarian'),
(20, 'First Class', 'window', 'night', 'Chicken');

INSERT INTO LEG_INSTANCE (leg_id, price,
departure_datetime, arrival_datetime, expedition_id,
guide_id, izban_id, ticket_vehicle_id) VALUES
(1, 75.5, '2024-01-01 08:00:00', '2024-01-01 10:30:00',
1, 3, NULL, 2),
(2, 45.0, '2024-01-03 14:30:00', '2024-01-03 18:00:00',
2, 8, NULL, 3),
(3, 30.0, '2024-01-04 09:45:00', '2024-01-04 11:30:00',
3, 12, 7, NULL),
(4, 55.75, '2024-01-25 17:30:00', '2024-01-25 20:15:00',
4, 6, NULL, 1),
(6, 65.0, '2024-02-08 07:30:00', '2024-02-08 09:45:00',
6, 18, NULL, 9),
(8, 27.75, '2024-02-22 10:00:00', '2024-02-22 11:45:00',
8, 10, 14, NULL),
(10, 35.0, '2024-03-07 12:30:00', '2024-03-07 16:15:00',
10, 7, NULL, 19);

INSERT INTO RIDE_HISTORY (passenger_id, ride_id)
VALUES
(3, 1),
(5, 2),
(8, 3),
(10, 4),
(15, 5),
(2, 6),
(7, 7),
(12, 8),
(18, 9),
(20, 10);

INSERT INTO EXPEDITION_SEARCH_HISTORY (passenger_id,
expedition_id, searched_time) VALUES
(1, 1, '2024-01-06 10:00:00'),
(3, 2, '2024-01-06 11:30:00'),
(6, 3, '2024-01-06 12:45:00'),
(9, 4, '2024-01-06 14:15:00'),
(12, 5, '2024-01-06 15:30:00'),
(15, 6, '2024-01-06 16:45:00'),
(18, 7, '2024-01-06 18:00:00'),
(2, 8, '2024-01-06 19:15:00'),
(5, 9, '2024-01-06 20:30:00'),
(20, 10, '2024-01-06 21:45:00');

INSERT INTO EXPEDITION_HISTORY(passenger_id,
expedition_id, leg_id) VALUES
```

```
(1,1,1),
(2,1,1),
(3,2,2),
(4,2,2),
(5,4,4),
(6,4,4),
(7,6,6),
(8,6,6),
(9,10,10),
(10,10,10);

INSERT INTO SEAT (seat_no, leg_id, expedition_id) VALUES
(1, 1, 1),
(2, 1, 1),
(3, 2, 2),
(4, 2, 2),
(5, 4, 4),
(6, 4, 4),
(7, 6, 6),
(8, 6, 6),
(9, 10, 10),
(10, 10, 10);

INSERT INTO TICKET (seat_no, leg_id, expedition_id,
passenger_id, class_type) VALUES
(1, 1, 1, 1, 'economy'),
(2, 1, 1, 2,'first class'),
(3, 2, 2,3,'business'),
(4, 2, 2,4, 'economy'),
(5, 4, 4,5, 'first class'),
(6, 4, 4,6,'business'),
(7, 6, 6,7, 'economy'),
(8, 6, 6,8,'first class'),
(9, 10, 10,9,'business'),
(10, 10, 10,10, 'economy');
```

INSERT - DELETE - UPDATE STATEMENTS

```
USE transportation_db;

INSERT INTO TICKET (seat_no, leg_id, expedition_id,
passenger_id, class_type) VALUES (5, 4, 4, 1,
'business');
DELETE FROM PASSENGER WHERE fname LIKE '%a%';
UPDATE VEHICLE SET company = 'Turkish Airlines' WHERE
company = 'THY';
UPDATE TICKET_VEHICLE SET ticket_vehicle_id = 20 WHERE
ticket_vehicle_id = 15;
```

SELECT QUERIES

```
USE transportation_db;

-- Using minimum 2 tables
SELECT fname, lname, gender, age, type, class_type,
seat_position, day_time, meal_menu
FROM PASSENGER, PREFERENCE
WHERE PREFERENCE.passenger_id = PASSENGER.passenger_id;

SELECT COUNT(*) AS vehicle_count, ticket_vehicle_type
FROM TICKET_VEHICLE, LEG_INSTANCE
WHERE TICKET_VEHICLE.type_id =
LEG_INSTANCE.ticket_vehicle_id
GROUP BY ticket_vehicle_type;


-- Using minimum 3 tables
SELECT fname, lname, searched_time, expedition_type,
departure_location_id, arrival_location_id
FROM PASSENGER AS P, EXPEDITION_SEARCH_HISTORY AS ESH,
EXPEDITION AS EXP
```

```
WHERE P.passenger_id = ESH.passenger_id AND
ESH.expedition_id = EXP.expedition_id;

SELECT fname, lname, RH.ride_id, departure_date_time,
arrival_date_time, departure_location_id,
arrival_location_id, vehicle_id
FROM PASSENGER AS P, RIDE_HISTORY AS RH, RIDE AS R
WHERE P.passenger_id = RH.passenger_id AND RH.ride_id =
R.ride_id;

SELECT P.Fname, P.Lname, RV.Car_model
FROM PASSENGER AS P, RENTAL_VEHICLE AS RV, RIDE_HISTORY
AS RH, RIDE AS R
WHERE P.Passenger_id = RH.Passenger_id
AND RH.Ride_id = R.Ride_id
AND R.Vehicle_id = RV.rental_vehicle_id
AND Car_model IS NOT NULL;

-- Critical SELECT statements

-- Flights from Antalya to Istanbul
SELECT LGI.leg_id, LGI.expedition_id, DL.name, AL.name
FROM LEG_INSTANCE AS LGI, EXPEDITION AS E, LOCATION AS
DL, LOCATION AS AL
WHERE LGI.Expedition_id = E.Expedition_id
AND E.Expedition_type = 'flight'
AND E.Departure_location_id = DL.location_id
AND E.Arrival_location_id = AL.location_id
AND DL.Location = 'Antalya'
AND AL.Location = 'İstanbul';
-- All upcoming expeditions
SELECT EXP.expedition_type, DL.name AS
departure_location, AL.name AS arrival_location,
LGI.departure_datetime, LGI.arrival_datetime
```

```sql
FROM LEG_INSTANCE AS LGI, EXPEDITION AS EXP, LOCATION AS
DL, LOCATION AS AL
WHERE LGI.expedition_id = EXP.expedition_id
AND EXP.departure_location_id = DL.location_id
AND EXP.arrival_location_id = AL.location_id
AND LGI.departure_datetime > NOW();

-- Most preferred car model
SELECT COUNT(*) AS number_of_ride, RV.car_model
FROM RIDE_HISTORY AS RH, RIDE AS R, RENTAL_VEHICLE AS RV
WHERE RH.ride_id = R.ride_id
AND R.vehicle_id = RV.rental_vehicle_id
AND RV.car_model IS NOT NULL
GROUP BY RV.car_model
ORDER BY number_of_ride DESC
LIMIT 1;

-- Total price for upcoming leg instances for each
passenger
SELECT P.fname, P.lname, SUM(price) AS total_price
FROM PASSENGER AS P, (SELECT passenger_id, price
FROM TICKET AS T, SEAT AS S, LEG_INSTANCE AS LGI
WHERE T.seat_no = S.seat_no
AND S.leg_id = LGI.leg_id
AND LGI.departure_datetime > NOW()) AS LGI_DATA
WHERE P.passenger_id = LGI_DATA.passenger_id
GROUP BY P.fname, P.lname
ORDER BY total_price;
```

PROJECT JOURNAL

2023-11-06 We met up for the first time and talked about what we are going to do.
2023-11-08 We started by checking out the THY website. We discussed what is relevant and what is not for our project. We started writing data requirements for the THY website.
2023-12-04 We felt OK with the THY data requirements and moved to next website: TCDD. We started writing data requirements for the THY website.
2023-12-07 We checked out multiple websites. We wrote some rough data requirements for IDO, Obilet, Yolcu360, Izban and Martı.
2023-12-09 We designed the ER diagram for THY
2023-12-10 We designed the ER diagrams for IDO
2023-12-11 We designed the ER diagrams for TCDD and Yolcu360
2023-12-12 We designed the ER diagrams for Martı
2023-12-12 We designed the ER diagrams for IZBAN
2023-12-13 We designed the ER diagrams for Obilet. During all the diagram designing, we came up with a strategy for combining all ERs. We started to combine all the ERs around Obilet.
2023-12-14
2023-12-15
2023-12-16 We thought of how to combine the diagrams. We decided to combine the rentals, then combine ticketed vehicles and expeditions, and finally we combined all the diagrams on the Obilet diagram.
2023-12-20 We worked on combining the diagrams and finalized the process.
2023-12-23 We applied the 9 step algorithm to the final diagram
2023-12-25 We updated the diagram and the 9 step algorithm
2023-12-27 We started to work on SQL (creating tables, referential integrity constraints etc.)
2023-12-28 We started to work on SQL (creating tables, referential integrity constraints etc.)
2023-12-31 Worked on inserting tuples into the database.
2024-01-01 Worked on inserting tuples into the database.
2024-01-02 Worked on inserting tuples into the database.
2024-01-03 Worked on triggers, inserts, updates, deletes and queries.
2024-01-04 Worked on triggers, inserts, updates, deletes and queries. Finished the project. Worked on the document.
2024-01-05 Worked on the project document.