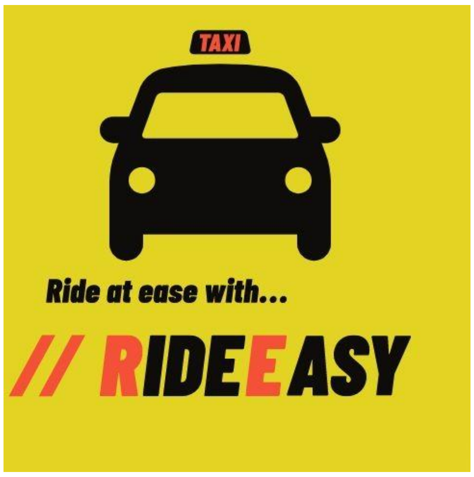
Group-5

Akshay Khandelwal, Patti Venkata Avinash Gupta, Saurabh Ambardekar, Sumit Malbari, Varada Kulkarni, Zarana Bhadricha

DATABASE DESIGN DOCUMENT – RIDE EASY

DATABASE PURPOSE:



The purpose of the RideEasy Database System is to bring its services to different cities and to allow drivers and riders to connect quickly and efficiently in a relatively smooth manner. RideEasy Database System allows transportation that's cheap and readily available. This system will help companies providing online cab booking services optimize the way they function by providing a robust database that will store all the required data systematically. This transportation system will also provide more opportunities for drivers and a source of income for the drivers as well as the car owners who are willing to rent their cars. RideEasy will bribe both riders as well as drivers by offering them interesting discounts and referral points to passengers and high incentives and bonuses to drivers.

BUSINESS PROBLEMS ADDRESSED:

- Provides information on rider (demand) requirements and driver (supply) inventory so that they can be matched efficiently reducing wait time and ensuring quick cab service.
- Enables car inventory administration which provides lucidity on vehicle availability. Provides riders with a wide range of cars to choose from thus enhancing rider experience.
- Allow dynamic management of user-profiles (drivers, riders, and car owners). E.g. storing crucial identification details of the driver will help the company verify driver details.
- Monitor drivers' activities and track their behavioral patterns that will help pick out skilled and professional drivers to enhance the service.
- Provides overall rides information that helps track the number of successful and canceled rides and factors at play. This data can be used to understand the reasons for canceled rides and steps can be taken to reduce them.
- Provide details on the outstanding payment for every rider. The customer will be charged for this due amount on their next invoice.
- Help analyze historic data about the services and develop new features to improve them through user feedback, such as making it easier for riders to find drivers during peak/rush hours.
- Increase new customer acquisition and ensure customer retention through targeted promo offers. Methodical storing of data makes the data more accessible for internal users who make business-critical decisions.
- Allows the companies to generate timely revenue reports that include essential user and rides KPIs.

BUSINESS RULES:

- Admin can give approvals to zero or more Users.
- Each User may hold one of the three profiles, i.e, Rider, Car Owner, or Driver.
- Each User can give zero or much feedback.
- Each Car Owner may have one or more Cars.
- Each Driver will be assigned to a car.
- Each Car can be associated with one or more locations.
- Each Car can have zero or many requests.

- Each Rider may have zero or more ride requests.
- Each Rider may have zero or more payment gateway.
- Each Request will have a source location and a Destination Location.
- Each Request may have zero or one promo code.
- A trip will be derived from a successful request.
- An invoice will be generated from a trip.
- An invoice must have at least one payment method.

DESIGN REQUIREMENTS (CREDIT TO PROFESSOR SIMON WANG):

- Use Crow's Foot Notation.
- The primary key field and Foreign key field are represented by "PK" and "FK" in each entity, respectively.
- Draw a line between two entities to show the relationship between them.
- Using vertical bars and Crow' Foot symbols at the start of the line we have represented the relationship between two entities whether it is one to one (or) one to many, respectively.
- For the representation of Superclass and Subclass relationships, we have used Enhanced Entity-Relationship techniques.
- For an understanding of participation whether it is optional or mandatory, we have used Crow's Foot Symbols.

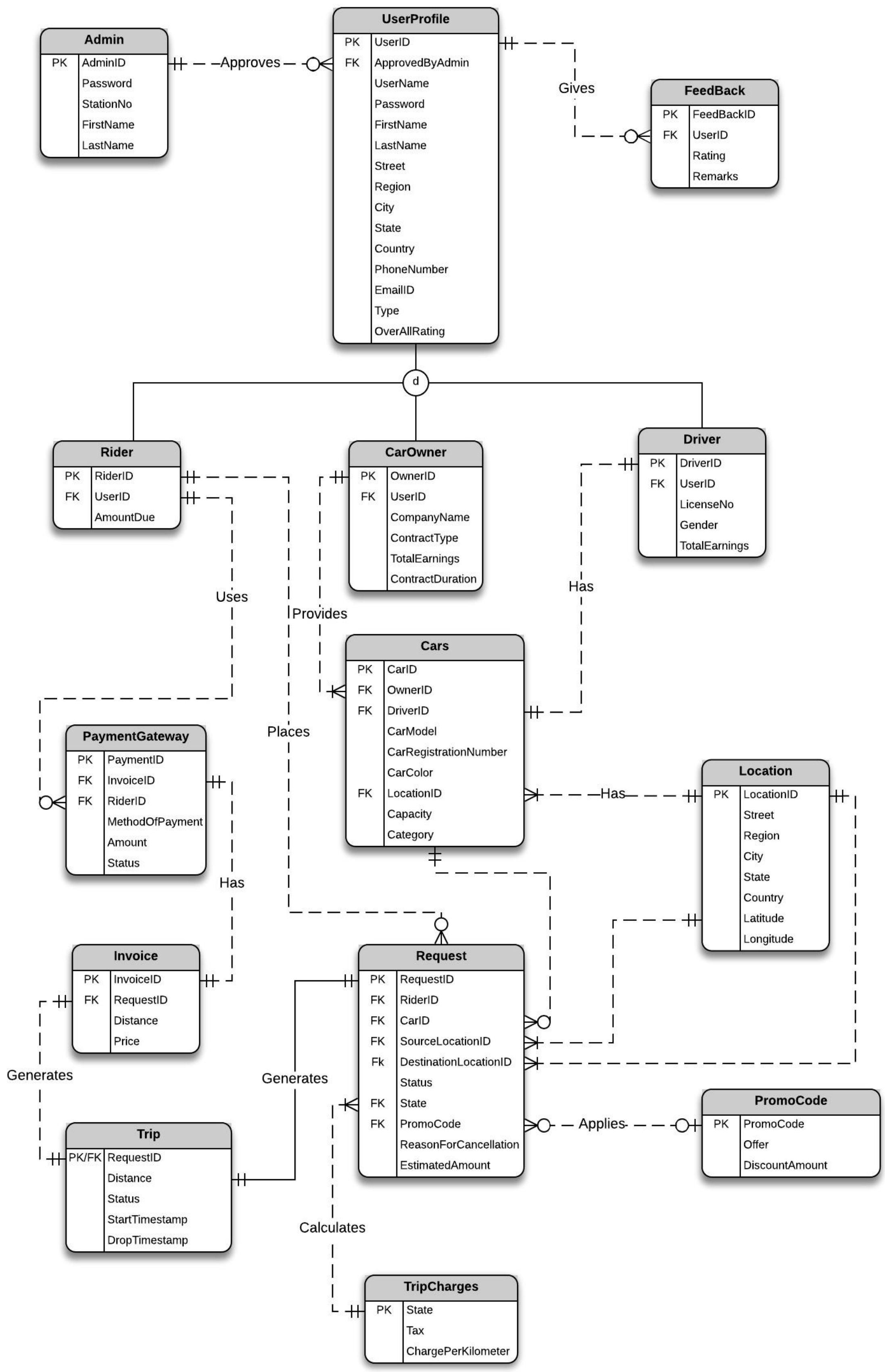
DESIGN DECISIONS:

ENTITY NAME	**WHY ENTITY INCLUDED**	**HOW ENTITY IS LINKED TO OTHER ENTITIES**
Admin	Admin's primary job is to update the database on a timely basis. Admin approvals are mandatory for activities like adding CarOwner or Driver Profiles or adding any promo Codes.	As one of the core entities in the database, its primary key is AdminID. It relates to the UserProfile. It holds a one-to-many relation with this entity as one Admin can give approvals to multiple user profiles. For this purpose, the ID of the Admin who reformed the user-profile last gets updated in the UserProfile Entity.
UserProfile	The system tracks all the users using RideEasy Services. A user must hold one of the profiles amongst Rider, Driver or Car Owner. Initial registration is necessary for every user Profile.	Being another core Entity of the Database, its primary key is UserID. User is represented as a superclass and it has three subclasses,i.e., car owner, driver, and rider. A user can register itself as a Rider, Driver, or a Car Owner and so it holds the one-to-one relationship with these entities.
CarOwner	The purpose of the CarOwner entity is to track the people/companies who want to offer their vehicles to be used as cabs. The company can check car contracts using this entity.	CarOwner entity is directly tied to Cars entity which stores the details about the cars offered by the car owners for cab service through a one-to-many relationship. CarOwner is also linked with UserProfile since every car owner is registered first as a user.
Driver	The company needs to track all the drivers registered through the app available for the cab service on any given day. The Driver entity helps achieve this. It helps the company track other important details such as license ID for every driver.	Since each driver gets assigned a car/vehicle, the Driver entity is directly associated with the Car entity through a one-to-one relationship. Driver entity becomes a subclass of UserProfile as every Driver registers as a User on the app.

Cars	The company maintains a record of all car details such as model, registration no, capacity, and type of the vehicle that we have included in the Cars Entity. This entity keeps a track of its Owner and Driver assigned to the respective car.	The Cars entity is linked with CarOwner and Driver entities with the help of OwnerID and DriverID respectively. This entity gives information about the Owner of the car and the allocated driver. Cars entity makes one-to-many relationships with the Request entity.
Rider	The system keeps track of all the users who have registered themselves as riders and are using our online cab services. Thus all such users come under Rider Entity. When a user signs-up as a customer a unique RiderID is generated for the user that distinguishes the User from CarOwner and Driver.	The Rider Entity is derived from UserProfile hence can be considered as Sub-Class to UserProfile. Rider Entity is linked with Payment gateway and Request Entities. When a customer requests for a cab then a unique RequestID is generated and both entities are linked by RiderID. To maintain track of payment associated with the rider we are linking Rider entity and Payment gateway with RiderID.
Request	This entity helps in tracking the ride requests placed by the riders. Multiple requests can be made by a rider. We also monitor the status of booking if the request was accepted by the driver or not and reason for cancellation if the rider canceled his/her ride.	As a rider requests a ride, the primary key of rider, i.e, RiderID is linked to request entity(rider entity making a one-to-many relationship with request entity). This request entity is linked to other entities like car and Location. A many-to-one relationship is made with these Location entities. Thus CarID, SourceLocationID,DestinationLocationID , State, PromoCode are linked to a foreign key in the request entity. Once the request is accepted by the driver, a trip record is created in the trip entity with a foreign key as requestid. The trip entity also has a one-to-one relationship with the request entity.
Location	This entity is required as it identifies a unique location id for a set of latitude and longitude. The location entity has details of the location like street, region, city, state, and country.	Location is having Many-to-One relationships with Request. The source id and destination id is derived from the location id which is required to generate requests. Thus, it is linked to the request entity.
Trip	Once the ride is accepted- the Trip begins containing all the details such as Start Time, End Time, Distance from source to destination.	The Trip entity is directly associated with the Request entity (one-to-one relationship), and Invoice entity (one-to-one relationship, as one single invoice, gets generated for a Trip).
TripCharges	The TripCharges entity provides the estimated and final fares of the trip/ride. The fare is calculated on the basis of per-mile charges applicable in a state. The total distance (between source and destination) is multiplied with the per-unit charge rate. The base fare	It has a one-to-one relation with the Request entity, to fetch the name of the state and to pass on the fare amount details.

	and tax are added to the final amount.	
Promocode	Promo Codes/special offers are introduced to cater to our potential customers. We are maintaining the data on a list of promo codes in the Promocode entity.	Promo Code entity is linked to the Request entity so that we can calculate the estimated cost for the trip. A promocode makes a one-to-many relationship with a Request entity.
Invoice	An invoice is generated on the successful completion of a trip to notify the rider with the trip charges. It also keeps track of the bill amount for all the trips.	Invoice generation is mandatory for every trip and so it holds a one-to-one relationship with the trip entity. If any promo code is applied to deduct some portion of the trip charges, then invoice entity has a zero-or-many
PaymentGateway	PaymentGateway is necessary to provide the rider with multiple payment options to pay the balance amount soon after the invoice is generated. This entity helps us to track whether the customer has completed his payment or not after the generation of the invoice. It also keeps a track of riders' methods of payment chosen by riders like Cash, Cards, or UPI.	PaymentGateway is connected to Rider and Invoice entities using RiderID and InvoiceID, so that we can maintain the payment status of each customer with the trip. It holds a one-to-one relationship with the invoice and many-to-one relationships with riders as one rider can choose among multiple modes of payment.
FeedBack	Feedbacks are necessary for the optimization of the system and making it more user friendly. On successful completion of a trip, the rider and the driver will provide their respective feedback which will then be reflected on their respective profiles.	A user can give multiple feedback on the completion of each ride. So there will be a one-to-many relationship between the user and the feedback. The userID gets updated in every feedback to track the users' suggestions.

ENTITY RELATIONSHIP DIAGRAM OF RIDE EASY DATABASE SYSTEM USING CROW'S FOOT NOTATION


Admin
PK AdminID
Password
StationNo
FirstName
LastName
Approves
UserProfile
PK UserID
FK ApprovedByAdmin
UserName
Password
FirstName
LastName
Street
Region
City
State
Country
PhoneNumber
EmailID
Type
OverAllRating
Gives
FeedBack
PK FeedBackID
FK UserID
Rating
Remarks
d
Rider
PK RiderID
FK UserID
AmountDue
CarOwner
PK OwnerID
FK UserID
CompanyName
ContractType
TotalEarnings
ContractDuration
Driver
PK DriverID
FK UserID
LicenseNo
Gender
TotalEarnings
Uses
Provides
Has
Cars
PK CarID
FK OwnerID
FK DriverID
CarModel
CarRegistrationNumber
CarColor
FK LocationID
Capacity
Category
Places
PaymentGateway
PK PaymentID
FK InvoiceID
FK RiderID
MethodOfPayment
Amount
Status
Has
Location
PK LocationID
Street
Region
City
State
Country
Latitude
Longitude
Has
Invoice
PK InvoiceID
FK RequestID
Distance
Price
Request
PK RequestID
FK RiderID
FK CarID
FK SourceLocationID
Fk DestinationLocationID
Status
FK State
FK PromoCode
ReasonForCancellation
EstimatedAmount
Generates
Generates
Applies
PromoCode
PK PromoCode
Offer
DiscountAmount
Trip
PK/FK RequestID
Distance
Status
StartTimestamp
DropTimestamp
Calculates
TripCharges
PK State
Tax
ChargePerKilometer