Vroom Rentals Inc.

Julia Kafato	260671487
Bozhong Lu	260683363
Hao Li	260711526
Muhang Li	260736135

1 Definitions

1.1 Entities

The following entities will be stored in tables of our relational database.

Customer

A **Customer** represents a person who has made one or more **Rentals** from Vroom.

- The customer's *email* is the key used to uniquely identify a Customer.
- The name and phone number are stored in our database as *name* and *phone*, respectively.

Note. The Customer set is distinct from the Driver set to account for Rentals for which the Customer is not the same person as the Driver. For example, someone at a company or organization may wish to make a Rental for its employee, in which case that person would be the Customer and the employee would be the Driver.

Member

A **Member** is a subclass of the **Customer** entity. The distinction between a **Member** and a **Customer** is that a **Member** has an account on our company's website. In addition to **Customer** attributes, the **Member** entity includes the following attributes:

- *password*, where a **Member**'s password is stored;
- *registration_date*, where their date of registration for the membership program is stored.

Note. *Each **Member** is a part of our company's points program. The number of points they have at any given time is calculated from the difference between the total amount of points they've earned from **Rentals** since their registration_date and the total amount of points they've redeemed. Every **Member** entity is also a **Customer** entity; thus, a **Member** can be uniquely identified by their email.*

Discount

This set stores all **Discounts** that can be applied to any **Rental**. The entity has the following artificial key and attributes:

- *discount_code*, (artificial key).
- *start_date* and
- *expiry_date* store the first day and last day, respectively, for which the **Discount** can be applied.
- *discount_value* describes how much will be discounted from the price.
- *type* describes whether a **Discount** is a flat value or percentage. For example, a **Discount** entity may have *type* "percentage" indicating that the *discount_value* is to be interpreted as a percentage to be taken off the total price.

Discounts associated with the company's points program have *discount_codes* and *discount_values* associated with them. All attributes of a Discount are determined by the company.

Employee

The **Employee** entity set represents the set of people who work for our car rental company. The entity holds the following attributes and key:

- *employee_id* (an artifical key)
- *phone*
- *name*
- *email*
- *dob* (the date of birth)
- *job_title*
- *start_date*
- *salary*

CustomerService

A **CustomerService** entity represents a customer service worker who works behind the desk at one or more of our Locations. They make sales and generally assist customers as needed. The **CustomerService** entity set is a subclass of the **Employee** entity set. Thus, every **CustomerService** entity is also an Employee. There are no attributes other than the ones inherited from **Employee**.

Maintenance

Maintenance entities represent maintenance personnel who transport and perform maintenance on cars. The **Maintenance** entity set is a subclass of the parent class **Employee**, so every **Maintenance** entity is also an Employee. Since **Maintenance** workers may drive the company's cars, our database stores the driving license details in the following form:

- *license_number*
- *license_expiry*

for all Maintenance personnel.

MaintenanceOperation

A **MaintenanceOperation** is any individual job done by a **Maintenance** worker to maintain a car. This includes oil changes, tire changes, and cleaning. All MaintenanceOperation entities have the attribute *type* which describes the type of MaintenanceOperation being done. We record the *start_time* and *end_time* for each operation. An artificial key called *opid* uniquely identifies each MaintenanceOperation.

Driver

A **Driver** entity represents any person who, during a **Rental**, drives the rented **Car**. The following attributes are recorded:

- *license_number*, the primary key for this Entity
- *license_expiry*
- *name*
- *dob* (the driver's date of birth),

Rental

A **Rental** entity represents the service provided to Vroom's customers wherein they pay for and use a rental car for some period of time.
Each **Rental** is uniquely identified by its artificial key, called *rental_id*.

Making Rentals a uniquely identifiable entity set enables the database to connect the **PaymentMethod**, **Customer**, **CarClass**, and **Location**(s) associated with a unique **Rental**, among other entity sets. This makes **Rental** the most essential entity in our database. *This is detailed in the Relationships section below.*

Payment

A **Payment** entity represents payment method, in the form of the credit or debit card used to pay for one or more **Rentals**. In order to accept a payment successfully, Vroom requires the information associated with the debit or credit card being used. This includes the following attributes and key:

- *card_number (*the primary key for this entity*)*
- *cardholder*,
- *billing_address*,
- *expiry_date*,
- the card's verification code (*cvc*)

Notes.

i. The *card_number* is the primary key used to identify a unique credit or debit card.

ii. We chose not to store the method of payment (*i.e.* credit or debit) as this can be derived from the *card_number*. Should a **Customer** wish to use a debit card, the *cvc* attribute for that **Payment** would be null.

AdditionalPurchase

AdditionalPurchase describes anything a customer would purchase from Vroom on top of their **Rental**. This could be car seats, insurance, GPS, or satellite radio, as examples. An **AdditionalPurchase** is a weak entity since the unique key of comes from its associated **Rental**, which is the owner entity.
Vroom stores

- *price*
- *product,* representing the item's name
- the *quantity* of the product being purchased.

Note. *The **AdditionalPurchase** is identified by the product name and its foreign key, rental_id from a **Rental**. This will always be unique as if you have two of any product associated with one **Rental**, this will be stored by its quantity attribute.*

Location

A **Location** represents Vroom's storefronts. This is where our **CustomerService** employees work and our cars are picked up and dropped off from. The following attributes and key are part of this entity set:

- *address*, the primary key for this entity set,
- *hours*,
- *name*,
- *phone,* the location's phone number.

CarClass

A **CarClass** refers to the categories each vehicle is classified within.
This classification is based on several criteria. Customers select a **CarClass** for their **Rental**, rather than a specific **Car**. Each **CarClass** has the following key and attributes:

- name is the primary key for this entity
- a number of seats (*num_seats*),
- *daily_price* representing the price charged for each day of renting this a car in this class,
- *features* which is a plaintext list of features shared between cars within that **CarClass**.
- *availability* describes dates available, represented in plain text.

Note. Names may include generic terms such as Economy, Compact, Intermediate, Standard, Full Size, Intermediate SUV, Standard SUV, Full Size SUV, or Minivan.

Car

A **Car** is a vehicle that Vroom rents to its customers. Each **Car** has the following attributes and key:

- *registration_num*, which is the registration number for the **Car** and the primary key for this entity,
- *plate_number*,
- *color*,
- *purchase_date*,
- *model*,
- *mileage,*
- *make*,
- *description* is used to store a plaintext description of any damage visible on the Car so that we can determine which Customers are accountable for which damage.

1.2 Relationships

Member ISA Customer

This ISA relationship indicates that all **Members** are also **Customers**. There is no covering constraint so it is not mandatory that a **Customer** be a **Member**.

{Maintenance, CustomerService} ISA Employee

This ISA relationship indicates that all **CustomerService** and **Maintenance** entities are also **Employee** entities. Since overlap constraint is allowed, It is possible that an **Employee** can be both a **CustomerService** person and **Maintenance** person.

redeemsPoints

A **Member** redeems his or her points for a **Discount**. One **Member** can have several **Discounts**. The redeemsPoints relationship has an attribute, *amt_of_points*, which indicates the amount of points redeemed. This is a many-to-many relationship because a **Member** can redeem points multiple times and the same **Discount** wherein points are redeemed can be used by multiple **Members**.

earnsPoints

A **Member** can earn points for each **Rental**. This is a one-to-many relationship. A **Member** can earn points from more than one **Rental**. However, a **Rental** can only distribute points to at most one **Member** given the key constraint between **Rental** and **earnsPoints**.

appliedTo

A **Discount** is applied to a **Rental**. This is a many-to-many relationship as a **Rental** can have zero or more **Discounts**. A **Discount** can be applied to more than one **Rental**.

works_at

An **Employee** works at a **Location**. There is a total participation constraint between **Employee** and **Location**. An **Employee** must work in at least one **Location** and a **Location** must have at least one **Employee** working there. Its attribute *hours* indicates the working hours of each **Employee**.

performs

A **Maintenance** worker performs **MaintenanceOperations**. This is a many-to-one relationship because one **MaintenanceOperation** must be performed by one **Maintenance** worker, but a **Maintenance** worker does not need to have performed any **MaintenanceOperations**, or they may have performed one or more **MaintenanceOperations**.

performedOn

A **MaintenanceOperation** is performed on a **Car**. There is a many-to-one relationship because a **MaintenanceOperation** must have a single **Car** it is performed on, however a **Car** may or may not need a **MaintenanceOperation** as it depends on the condition of the **Car**.

belongsTo

A **Car** belongs to a **CarClass**. In our company, a **Car** must belong to a **CarClass** because of the combination of key and participation constraint between **Car** and **CarClass**. Also, a **CarClass** must have at least one **Car** in it, because it would be useless for a **CarClass** to exist which has no **Cars** in it and therefore will not be available for a **Rental**.

primaryDriver

A **Driver** can be a primaryDriver for a **Rental**. There should be exactly one primaryDriver for a **Rental** since there is a combination of key and participation constraint between a **Rental** and a primaryDriver. This is a one-to-many relationship because one **Rental** must have a primaryDriver, but a **Driver** does not have to be a primaryDriver. They could exist only as an additionalDriver.

additionalDriver

A **Driver** can be an additionalDriver for a **Rental**. This relationship is many-to-many because it is not mandatory for a **Driver** to be an additionalDriver, though it is possible for them to be an additionalDriver for more than one **Rental**. Similarly, a **Rental** does not need to have additionalDrivers.

transportation

This is a ternary relationship between **Maintenance**, **Location**, and **Car**. A **Maintenance** worker transports **Cars** to different **Locations**. This relationship has two attributes, *travel_time* for us to use to calculate how many hours a **Maintenance** worker has spent working, and *date* which stores the *date* of transportation. This a many-many-many relationship. It is not mandatory for the **Maintenance** worker to have transported a **Car** to a **Location**, but they can transport more than one **Car** to more than one **Location** if need be. More than one **Car** can be transported to one **Location**. It is possible that a **Location** may have no **Cars** that have been brought there as part of a transportation, or many.

is_of

An **AdditionalPurchase** is of a **Rental**. **Rental** and **AdditionalPurchase** participate in a supporting one-to-many relationship where one **Rental** can have multiple **AdditionalPurchases**. Each **AdditionalPurchase** must exist as it relates to exactly one **Rental**.

isOrderFor

Rental is an order for a particular **CarClass**. **Customers** will choose the **CarClass** that they want when they make a **Rental**. This is a one-to-many relationship where a **Rental** can only be an order for exactly one **CarClass**. As for **CarClass**, a **CarClass** could be ordered by a **Rental** or perhaps it has never been rented by any **Customer**.

bookedBy

A **Rental** is booked by a **Customer**. This is a relationship with a participation constraint and key constraint. All entities in **Customer** have exactly one involvement in this relationship, which means each **Rental** can only be booked by exactly one **Customer**. Meanwhile, the participation constraint on **Customer** indicates each **Customer** must have booked at least one **Rental**.

paysWith

Customer pays with **Payment**. This is a many-to-many relationship because a **Customer** could use multiple forms of **Payment** and a form of **Payment** could be used by multiple **Customers**.

paidForBy

Rental is paid for by **Payment**. This is a relationship with a participation constraint and key constraint. All entities in **Payment** have exactly one involvement in this relationship, which means each **Rental** is paid for by exactly one **Payment**.

pickedUpAt

A **Rental** is picked up at a **Location**. This is a one-to-many relationship. **Rental** has a key constraint and participation constraint, which indicates that any given **Car** booked for a certain **Rental** is picked up at exactly one **Location**. One **Location**, however, may or may not have been used as a **Rental** pick-up **Location**. The *time* each **Rental** is picked up at a **Location** is recorded as an attribute of this relationship.

droppedOffAt

A **Rental Car** is dropped off at a **Location**. This is a one-to-many relationship. **Rental** has a key constraint and participation constraint, which indicates that any given **Car** booked for a certain **Rental** is dropped off

at exactly one **Location**. One **Location**, however, may or may not have been used as a **Rental** drop-off **Location**. The *time* each **Rental** is dropped off at a **Location** is recorded as an attribute of this relationship.

2 Relations

Car	Car(registration_num, make, plate_number, model, color, purchase_date, milage, name, employee_id, description)(name ref CarClass, employee_id ref Maintenance)
CarClass	CarClass(Name, daily_price, availability, num_seats, features, address)(address ref Location)
Customer	Customer(email, name, phone)
Discount	Discount(discount_code, discount_value, type, start_date, expiry_date)
Driver	Driver(Licence_number, name, dob, licence_expiry)
Employee	Employee(employee_id, phone, name, email, dob, start_date, job_title, salary)
CustomerService	CustomerService(employee_id)
Location	Location(address, hours, name, phone)(Relationship to CustomerService cannot be reflected because participation constraint)
Maintenance	Maintenance(employee_id, licence_number, licence_expiry)
MaintenanceOperation	MaintenanceOperation(opid, type, begin_time, end_time, registration_num)(Relationship to Maintenance is a participation constraint which cannot be reflected, registration_num ref Car)
Member	Member(email, password, registration_date)(email ref Customer)
Payment	Payment(card_number, cardholder, cvc, billing_address, expiry_date)
Rental	Rental(Rental_id, name, licence_number, email, card_number, address)(name ref CarClass, licence_number ref Driver, email ref Customer, card_number ref PaymentMethod, address ref Location)
AdditionalPurchase	AdditionalPurchase(rental_id, product, quantity, price)(rental_id ref Rental)
additionalDrivers	additionalDrivers(licence_number, rental_id)(licence_number ref Driver, rental_id ref Rental)
appliedTo	appliedTo(Discount_code, rental_id)(Discount_code ref Discount, rental_id ref Rental)
earnsPoints	earnsPoints(rental_id, email)(rental_id ref Rental, email ref Member)
droppedOffAt	droppedOffAt(time)(Key participation constraint from Rental to location)
pickedUpAt	pickedupAt(time)(Key participation constraint from Rental to location)
redeemsPoints	redeemsPoints(discount_code, email, amt_of_points)(discount_code ref Discount, email ref Member)
sells	sells(rental_id, employee_id)(rental_id ref Rental, employee_id ref CustomerService)

transportation	transportation(employee_id, registration_num, address, date, travel_time)(employee_id ref Maintenance, address ref Location, registration_num ref Car)
worksAt	worksAt(hours)(Between CustomerService and Location. Can not be reflected because it's a participation constraint)

3 Creativity and Thought Process

In the ER model for our car company, we offer our Customer a lot of freedom and options in their car rental. They may pick up and drop off their car from two different locations, they may make additional purchases of insurance, they can accrue points as part of our membership program which can be redeemed for a discount, and they may add additional drivers if they wish.

Given that our customers are able to pick up and drop off cars at any given location, we foresaw a potential issue that could arise. If too many customers take a certain type of car from a particular location to any other location, we could end up with a shortage of that type of car in that location. To handle this problem, we have determined that our employees must handle the transfer of cars from wherever there is an excess, to where there is a shortage. In our ER model, you will see a ternary one-to-one-to-one, if you will, relationship called "transportation" between the entities Employee, Location, and Car. This is how we represent the transfer of cars between locations so that we can ensure customers are free to pick up and drop off cars wherever they would like, while ensuring the type of car they wish to rent will almost always be available to them.

The additional purchases offered by our car company would include car seats for kids, insurance, GPS, Sirius XM radio, and ski racks. The entity AdditionalPurchase is a weak entity with Rental as its owner entity, as any additionally purchased product, as well as its quantity and cost, may be repeated over various rentals, but the customer cannot repeat a given additionally purchased product within one rental.

Most of the relationships between the Rental entity and one other entity have a combined constraint which dictates that there must be exactly one of that entity related to a given Rental.

The relationship primaryDriver is a one-to-many relationship. There must exist one Driver who is the primaryDriver for a given rental, but a Driver does not have to exist in our system as a primaryDriver. They could, for example, be related to a Rental or multiple Rentals only as an additionalDriver.

The relationship bookedBy is between Rental and Customer and it is one-to-many. Our ER model specifies that a rental must have exactly one customer. The relationship paidForBy is between Rental and PaymentMethod and specifies that a Rental must have exactly one PaymentMethod. The relationships pickedUpAt and droppedOffAt are between Rental and Location, and specify that a given Rental must have exactly one pick-up and one drop-off location. As such, they are one-to-many relationships. Lastly, isOrderFor is a relationship between Rental and Cardholder whose constraint dictates that a rental must be for exactly one CarClass.

We have multiple participation constraints. The relationship worksAt requires at least one Employee at each Location, and at least one Location for each Employee. The relationship belongsTo requires that each CarClass has at least one car associated to it.

We also specify the only two types of employees that would exist within our companies. These are CustomerService employees and Maintenance Employees. The relationship between these two entities and the Employee entity is an ISA relationship. This specification is necessary as only a Maintenance worker can perform a MaintenanceOperation, which we keep a record of in our system. A MaintenanceOperation requires exactly one Maintenance worker take care of it. These operations could include tire changes, oil changes, and other routine services. There is also an ISA relationship which specifies that the Guest entity

and Member entity are subclasses of the Customer entity. A Member needs to be an entity separate from a Customer because a Member is participating in our points program, and as such they have a pointsRedeemed relationship with Discount, as well as an attribute points wherein all points they have accumulated (1 per dollar spent) are stored.

4 Inspirations

Hertz, Avis