Alex Keeling, Benjamin Weichman and Caroline Quinn

November 5th, 2014

Databases Term Project Draft

Database Modeling System for a Passenger Train Network

Overview:

We wish to design a database to store information about a passenger railway. The model will be concerned with train cars, engines, passengers, and routes. Different views of the information will allow different types of users (passenger and system administrators) to view and manipulate specific components of the database. The database will be accessed through a web interface using PHP.

A train database will help passenger rail companies in a number of ways, including:

- Elimination of human error in tasks such as scheduling (for example, scheduling a train without assigning an engine). This is because of the constraints we impose on the database.
- Allowing passengers to easily access up-to-date train schedules, resulting in greater customer satisfaction.
- Tracking passengers' movements' across multiple voyages
- Helping employees to map their work schedule.

Goals:

The passenger train database should be helpful to all users. It will cover all of the above uses, and also create a simple, useful tool to access and manipulate information relating to a passenger train network. Based on our specifications, we hope our database management system will model a passenger railway system with reasonable accuracy.

We will design our relation schemas such that the database may gracefully accommodate extension. This way our database will be able to expand and adapt according to the client's needs.

We will also strive for memory and computational efficiency. By adhering to Boyce-Codd Normal Form, we will ensure that our database has as little redundancy as possible. Through careful selection of foreign keys between relations, we have attempted to simplify and optimize what we suspect will be common join operations.

Views:

Passenger

This view allows a passenger to see their past and future tickets. It also allows a passenger to see some information about upcoming voyages, including the date/time information (a schedule of upcoming trains).

Travel Agent

This view allows for a travel agent to book tickets for passengers on upcoming train voyages. Consequently, this lets the travel agent view the Train Voyages, the Passenger Cars associated with particular voyages, the route information for each voyage, as well as train station information. Additionally, the travel agent can create, update, and delete both Passengers and Tickets.

Employee

This view allows an employee to see their work schedule for the upcoming voyages.

Manager (Employee Scheduler)

This view allows a manager to schedule staff for particular voyages. This means that the manager can see the voyages, the Cars and Engine assigned to the voyage, as well as the Engine Type. Of course, the manager can also create, update and delete Employees and Employee/Voyage schedule information.

Manager (Train Scheduler)

This view allows a manager to schedule specific voyages, or "runs", of routes. This means that the manager can create, update and delete Train Voyages.

Head Office

This view allows a senior member of the company to add, update, or delete things that rarely change. This includes the Train Routes, Track Sections, and Train Stations. It also includes Engines, Engine Type, all types of Cars (Passenger, Dining, and Baggage).

Specifications and Assumptions:

The railway system may be imagined as a graph, in which the stations are vertices and the tracks joining pairs of stations are edges. A train route describes a path in this graph. A single route is composed of an ordered set of track sections with start and end station.

A train is an engine pulling zero or more train cars. It is assumed that all trains will have exactly one engine pulling them. One engine is important in the design of the DBMS, as the engine along with date and time of a particular voyage are a train voyage's unique identifier in the grand scheme of all train voyages. (Isn't a voyage's unique identifier the Voyage ID?)

A train may embark on a voyage. A voyage is a train travelling on a specific route departing at a specific date and time. Each train is manned by a staff that includes at minimum one qualified engineer. A specific staff will be assigned to each train voyage. It is assumed they will remain on the train for the duration of the voyage.

There are passenger, dining and baggage cars. There must be an appropriate number of baggage cars to house all the passengers' belongings, as well as an appropriate number of dinning cars to feed all passengers. These quantities will be calculated based on the assumption that all passengers have luggage and will dine on board the train.

A passenger may own many tickets for different voyages. Each ticket represents a seat reserved on a specified passenger car embarking on a specific voyage. The ticket price is calculated as the product of the base voyage price, the cost of the selected traveling class, and any applicable ticket discounts.

Design Choices

Many voyages may follow a single route

In early design stages, we had planned to create a unique route for each voyage. We reasoned that this would allow each of our voyages to follow a different route of need be. By speaking with our client, we came to realize that this was a poor design choice that did not reflect reality.

Rather than having each voyage follow a unique route, we then chose to have voyages travel along set routes instead. We rationalized that most passenger trains follow daily or weekly schedules, so the added flexibility of unique routes was unnecessary. This saves us from an enormous amount of duplication. Routes may be re-used, rather than created anew for each voyage.

Tickets link passengers to passenger cars

Initially we intended to have passenger cars and passengers relate directly to one another. This seemed to make sense, since there is a direct physical relationship between passengers and their train car seats. However, we decided that it would be simpler to make the ticket serve as the intermediate between the passenger and the voyage/car information.

This design choice allowed us to separate our concerns. A Person entity is concerned with information about that person, and a Ticket entity is concerned with the specifics of a voyage. This actually models better than our former design. A person doesn't need to commit their seat ticket, train number and departure time to memory—they just look at their ticket.

Train routes are ordered track segments

Our model treats network of stations as a graph. Each station is a vertex, and each segment of track joining two stations is a directed edge. We assume that that all connected stations are joined by a pair of track edges—one directed in either direction. Thus a train route describes a path in our graph.

We considered describing a train route as an ordered set of stations. This design would allow us to cleanly and easily define a path. However, we would be unable to treat tracks as entities. It would require extra complexity to record whether a track section is under repairs and differentiate parallel going-to and coming-from tracks.

We chose to instead create a track section entity defined by a start and endpoint station, and build our routes from these. By describing a route by edges instead of vertices, it became easier to constrain voyages to prevent trains from colliding or traversing out-of-service tracks. The downside is

that we require a Track Section entity, increasing the size of our database. Given how much this entity reduces the logical complexity of our constraints, we believe this is a worthwhile tradeoff.

Train cars are directly related to their voyage, not their engine

Initially we were very concerned with coupling and decoupling train cars in order to attach them to different trains. We were bewildered by the complexity of car ordering, swapping and abandoning cars mid-voyage, etc. We realized that it was far simpler to discard the relationship between an engine and its cars, and simply have both be related to the same voyage.

This solves a three problems. First, we are no longer concerned with train car coupling or ordering. We assume another system will deal with these issues. As far as our database is concerned, train reassembling may occur instantly between voyages. Second, it becomes simpler to query cars by a particular voyage, since the two are directly related. Finally, we preserve the many-to-many relationship between cars and engines through their common voyages.

Model Constraints

Each voyage has an operator

Every voyage must be attended by at least one employee qualified to operate that particular model of train. This means no remote-controlled or preprogrammed voyages.

Tickets are issued for future voyages

Tickets may only be distributed for a particular voyage before that voyage has begun. That is, ticket sales are locked at the moment a train voyage departs.

Values are positive

Every attribute that stores a price, quantity, number of years, distance, etc. must be positive.

Passenger cars are not oversold

The number of tickets available on a passenger car for a particular voyage must not exceed the number of seats actually on that passenger car. We assume that passengers are not permitted to cancel their seat reservations, therefore a rail company has no incentive to overbook.

One seat per individual per voyage

A person may not be issued multiple tickets in their name for the same voyage. That is, a single person may occupy one seat in one car, not an arbitrary number of seats in an arbitrary number of cars.

The baggage cars can fit all passengers' baggage

On a given voyage, there must be baggage cars with great enough capacity to store baggage for all ticketholders. The sum of all baggage cars' baggage capacities must be greater than the number of passengers on a given voyage.

The dining cars can feed all passengers

On a given voyage, there must be dining cars capable of satiating all ticketholders. The sum of all dining cars' capacities must be greater than the number of passengers on a given voyage.

Each voyage has one engine

There must be exactly one engine per voyage. This is a slight break from reality, as double-headed and push-pull trains do exist. However, this saves us the trouble of determining which engines might be compatible with which, etc.

The engine can pull all the cars

On any voyage, there cannot be more cars than the engine is capable of pulling. Each engine is limited by how many cars it can pull. Our model ignores that car weights may vary individually based on contents, model, etc.

Engines types are invented in the past

An engine type's date of invention may not be later than the current date. Our model does not account for time-travelling trains.

Voyages arrive after they depart

A voyage's time of arrival may not be before its time of departure. Again, our model does not account for time-travelling trains.

Trains do not collide at their outset

No two voyages starting at the same time may begin in the same direction on the same track segment. Note that our model does not handle train collisions after the initial segment—it is assumed that synchronization and mutual exclusion is handled by another system.

All active trains and cars are in service

No train or car currently on a voyage may be out of service, and no voyage may depart if any of its constituent cars or engine are out of service.

Routes have at least one stop

A train route needs to leave the station, meaning that the number of track segments on that route must be greater than zero. It is unlikely that patrons will pay for a voyage that remains at the station for several hours before the conductor ushers them back onto the platform.

Routes are continuous

Trains may only travel between stations if a track exists between them. This will require that each track segment endpoint be equal to the next sequential track segment's start point. In addition, the first and last segments' respective start and endpoints must match the voyage's start and endpoint.

Engines and cars may not be on multiple simultaneous voyages

A single engine or car may not be on multiple voyages at once, even if those voyages follow the same route. We assume that a car or engine may only be in one place at once.

Necessary Simplifications:

Trains can be coupled and decoupled instantly.

This is unrealistic, but it would require an incredibly complex amount of logic to determine how many couplings and de-couplings are required to switch a train from one configuration to another. This is not a central concern of the database.

We ignore the coupling order of cars.

The coupling order is only important if we actually intend to couple or decouple trains. We do not, therefore this detail may be ignored.

Engines and cars may teleport between stations in between voyages.

This is a large break from reality that we may mend at a later stage. We are considering two possible solutions. Either we establish a constraint stating that a car may only embark on a voyage if that voyage's starting station is identical to the terminal station of that car's most recent voyage. The upside of this solution is that this does not require that we modify our relation schema. The downside is that we need to perform a query whenever we want to find a car's current location. Also, it is unclear how we would establish a newly-created car's location.

Another option is to create a "current station" attribute on each car and engine. This solution is more explicit and sensible than the former, but it would require that we automatically update a car's location at the end of each voyage.

There are cases where collisions may occur.

We assume that all tracks are one-way, so head-on collisions are not feasible. However, there is nothing to prevent two trains from attempting to merge onto the same track at the same time. We cannot prevent these cases without predicting them, and our database does not store enough information to calculate train locations at a given point in time.

Staff are not required to operate dining cars or baggage cars.

Rather than attempting to calculate staff size based on the number of dining and baggage cars, we assume that in the worst case all dining cars are self-serve and all baggage cars are coin-operated.

Engine power is described by car limit, not weight limit.

If we wished to limit train size by weight, we would require that all train cars include a weight attribute that would functionally determine the amount of passengers and cargo on that car. This would add too much complexity.

"Island" train stations are not forbidden.

An island station has no connections to any other stations and would not operate in reality. Although pointless, these stations are harmless given the constraints outlined further on. Therefore we see no reason to constrain them.

Voyages without passengers or cars are permitted.

A passenger train without passengers seems pointless. Consider, however, if an engine broke down and a replacement was dispatched from another station. There may be cases where non-commercial voyages may be useful.

There are no staff cars, fuel cars, supply cars, etc.

It is assumed all supplies and space required for the staff will be present on the passenger, dinning, luggage cars and the engine. Fuel is too complex to consider, as the amount of space required for fuel would depend on the voyage length and the engine fuel type.

Route distance and travel times are assigned, not calculated.

This means that nothing prevents two otherwise identical routes from claiming to be of different lengths. This would be nonsensical. We may attempt to rectify this later by adding a distance attribute to a Track Segment that would be used to calculate route distances.

To calculate travel time, we would be required to know a train's average speed and the length of its stops and layovers. Our database does not track this information, and it would be complex and costly to do so. Therefore, travel time cannot be simply calculated.

Trains may embark on a new voyage the instant their current voyage finishes.

> We could solve this problem by requiring a buffer period between the arrival and departure of a car or engine to allow loading/unloading/refueling/boarding. A simpler solution might be to add a fixed amount of time to a voyage's arrival time so it can use that additional time to load and unload goods and people at the arrival platform.

We do not permit multiple engines per voyage.

> Some real-life trains do have two engines. We chose to disallow these to save us from the complexity of determining which pairs of engines are compatible.

Relations:

Ticket

- (primary key) ticket ID : integer
- (foreign key) passenger ID : integer
- (foreign key) train voyage ID : integer
- (foreign key) passenger car ID : integer
- price : float
- seat number: integer

Functional Dependencies:

- Ticket ID --> passenger ID
- Ticket ID --> train voyage ID
- Ticket ID --> passenger car ID
- Ticket ID --> seat number
- train voyage ID, passenger car ID, discount --> price

BCNF Decomposition: *(Ticket was the only relation containing a BCNF violation)*

Violating FD: train voyage ID, passenger car ID, discount --> price

Result:

Relation 1:

- train voyage ID
- passenger car ID
- discount,
- price

Relation 2:

- voyage ID
- passenger car ID
- discount
- seat number
- Ticket ID

The Ticket relation represents the information required to connect a passenger with a seat on a particular voyage. It includes its own Ticket ID, a foreign key to the Passenger relation, the Voyage relation, and the Passenger Car relation. These foreign keys allow the ticket to connect the specific passenger with the specific voyage along a route, in a particular car on that voyage. The Ticket relation also includes a price attribute which is calculated at the time of purchase, and a specific seat number.

Car/Voyage Pair

- (foreign key) voyage ID : integer
- (foreign key) car ID : integer

The Car/Voyage Pair relation simply stores foreign keys from both the Car and Voyage relations. It connects a particular train car and a particular voyage.

Employee/Voyage Pair

- (foreign key) voyage ID : integer
- (foreign key) engine ID : integer

The Employee/Voyage Pair relation simply stores foreign keys from both the Employee and Voyage relations. It connects a particular employee and a particular voyage.

Car

- primary key ID : integer
- in service state : boolean

The Car relation is the super relation for all types of train cars. Thus, it contains the things that all train car types have in common: a Car ID, and an in service indicator.

Passenger Car

- number of seats : integer

- class : float

Functional Dependencies:

- car ID --> number of seats
- car ID --> class

The Passenger Car relation is a sub-relation of the Car relation. It contains the number of seats available for passengers, as a class multiplier for calculating ticket prices (i.e. a business class train car might have a value of 1.2, while an economy class car might have a value 1.0).

Baggage Car

- baggage capacity : integer

Functional Dependency:

- car ID --> baggage capacity

The Baggage Car relation is a sub-relation of the Car relation. It contains the number of spots available for passengers' bags.

Dining Car

- passenger limit : integer

Functional Dependency

- car ID --> passenger limit

The Dining Car relation is a sub-relation of the Car relation. It contains the number of people it can feed over the course of one voyage.

Engine

- (primary key) engine ID : integer
- year of construction : integer
- (foreign key) engine model name : string
- in service state : Boolean

Functional Dependencies:

- engine ID --> year of construction
- engine ID --> engine model name

The Engine relation contains information about a single engine, including its ID, year of construction, a reference to its Engine Type relation, and an in service indicator.

Engine Type

- (primary key) name : string
- fuel type : string
- year of invention : integer
- inventor (optional) : string
- country of origin (optional) : string

- maximum cars pulled : integer

The Engine Type relation contains information about different models of engines. It contains an engine type ID, the type of fuel this model uses, the year the model was invented, the inventor's name, the country of origin, and the maximum name of cars this model can pull.

Voyage

- (primary key) voyage ID : integer
- departure date and time : date
- arrival date and time : date
- (foreign key) train route : integer

Functional Dependencies:

- voyage ID --> departure date and time
- voyage ID --> arrival date and time
- voyage ID --> train route

The Voyage relation refers to specific Train Voyages (or "runs") along existing routes. It contains a voyage ID, a starting date & time, a reference to the engine ID of the engine which will be on this voyage, and a reference to the route in the route relation the train will follow.

Train Station

- (primary key) name : string
- (primary key) station ID : integer (unused?)
- latitude (optional) : string
- longitude (optional) : string
- address : string

Functional Dependency:

- name --> station ID

The Train Station relation stores each train station. It contains a station ID, a station name, the address of the train station, and the latitude & longitude.

Track Section

- (primary key) section ID : integer
- (foreign key) station of origin name : string
- (foreign key) terminal station name : string
- in service state: Boolean

Functional Dependencies:

- section ID --> station of origin name
- section ID --> terminal station name

The Track Section relations represents a set of tracks running in a particular direction between two stations. The attributes are a section ID, a reference to the origin station, a reference to the destination station, and an in service indicator.

Train Route

- (primary key) route ID : integer
- (foreign key) start station : string

- (foreign key) endpoint : string
- distance : float
- base cost : float
- travel time : integer

Functional Dependencies:

- route ID --> start station
- route ID --> endpoint
- route ID --> distance
- route ID --> base cost

The Train Route relation represents an established route that a train might follow on a voyage. It contains a route ID, a reference to the starting station, a reference to the final destination station, the total distance, the travel time, and a base ticket cost.

Section/Route

- (foreign key) Track Section ID : integer
- (foreign key) Train Route ID : integer
- order of visitation : integer

The Section/Route relation represents all the discrete sections of track between stations that add together to create an entire route. It contains a reference to the route it's a part of, as well as a reference the section of track, and an integer indicating the order of visitation (n is the nth section of track for a given route).

Passenger

- (primary key) passport number : string
- (primary key) passenger ID : integer
- name : string
- nationality : string
- address : string
- phone : string
- payment type : string

Functional Dependencies:

- passenger ID --> name
- passenger ID --> nationality
- passenger ID --> address
- passenger ID --> phone

The Passenger relation represents an actual customer. It contains a passenger ID, name, address, phone number, and payment information.

Employee

- (primary key) employee ID : integer
- name : string
- title : string
- years of employment : integer

Functional Dependencies:

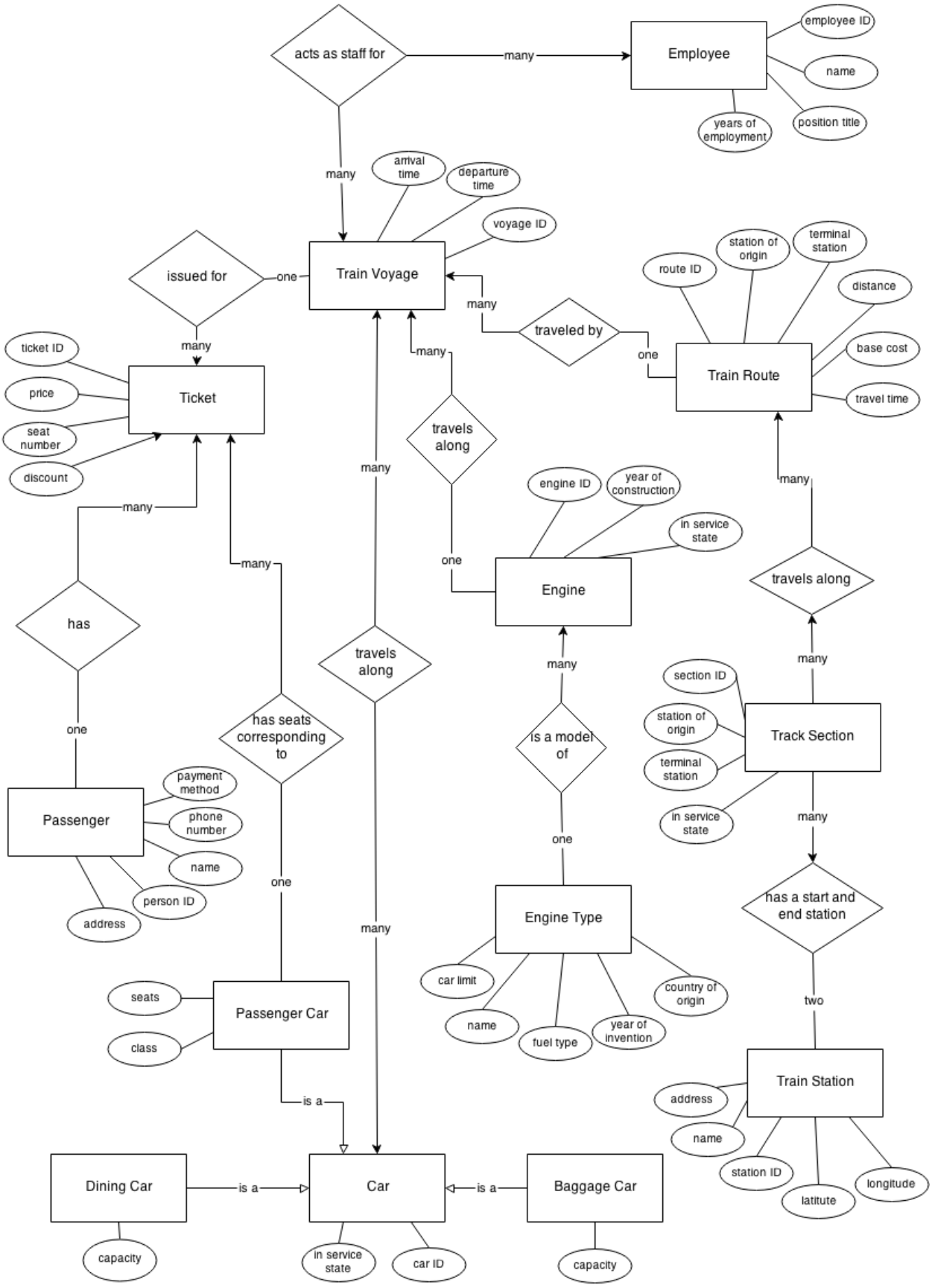
- employee ID --> name
- employee ID --> years of employment
- employee ID --> title

A tuple in the Employee relation represents an employee. The relation contains an employee ID, name, title, and employment start date. The title is a string describing the employee's position, e.g. "Senior Engineer" or "Assistant Lavatory-Scrubber in Training".

Engine Type/Employee Pairs

- (foreign key) employee ID : integer
- (foreign key) engine type name : string

The Engine Type/Employee Pairs relation simply stores foreign keys from both the Engine Type and Employee relations. It connects a particular employee and a particular engine type, to indicate that this employee is qualified to operate this type of engine.


acts as staff for
many
Employee
employee ID
name
position title
years of employment
many
arrival time
departure time
voyage ID
issued for
one
Train Voyage
many
traveled by
one
Train Route
route ID
station of origin
terminal station
distance
base cost
travel time
many
ticket ID
price
seat number
discount
Ticket
many
travels along
many
engine ID
year of construction
in service state
one
Engine
many
travels along
many
has
one
has seats corresponding to
many
travels along
many
is a model of
one
Engine Type
car limit
name
fuel type
year of invention
country of origin
many
section ID
station of origin
terminal station
in service state
Track Section
many
has a start and end station
two
Train Station
address
name
station ID
latitute
longitude
Passenger
payment method
phone number
name
person ID
address
one
Passenger Car
seats
class
is a
Dining Car
capacity
is a
Car
in service state
car ID
is a
Baggage Car
capacity