**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.

**Teleporting trains?**How do we constrain these, if at all?

**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.

**How do we handle teleporting trains?**