**Overview/Uses:**

We wish to design a database to store information about a passenger railway (‘passenger railway information’ may be ambiguously read as ‘railway information about a passenger’, instead of ‘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 different (not necessarily different, what a user can see and modify may be a subset of what an administrator can see and modify) components of the database. The database will be accessed through a web interface using PHP.

* A train database will be advantageous in the smooth operation of passenger trains for a multitude of reasons, including:
* Eliminating human error in tasks such as scheduling (ex: scheduling a train without assigning an engine car). (you should mention that this is because of constraints the DB imposes)
* Allowing passengers to more easily access up-to-date train schedules, (not quite how this word is used) resulting in higher customer satisfaction.
* Tracking passengers on specific train cars efficiently. (but we don’t actually ensure that the passenger gets on the train, all we know is that they bought a ticket)
* Track passengers’ luggage on specific baggage cars, helping to avoid lost belongings. (our database doesn’t do this)
* Aiding employees in mapping their work schedules.

**Specifications/Assumptions:**

The railway system may be imagined as a graph, in which the stations are vertices and the tracks are edges. A train route describes a path in this graph. A specific route is made up of sections of rail with a station at either end. A train route also has a start station and an 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. This (what does ‘this’ refer to?) is important in the design of the DBMS as the engine along with date and time (date and time of what?) are a train’s unique identifier in the grand scheme of all train voyages. (be careful to distinguish between a ‘train’ and ‘train voyage’, you use them interchangeably here)

Trains may embark on voyages, which are train routes (this sentence may be read to mean that a single voyage has multiple routes) being travelled at a specific date and time. Each train is manned by a crew, including the conductor, engineer, stewards, etc (our only constraint is that there is at least one qualified engineer, no guarantee of auxiliary staff). A specific crew will be assigned to each train voyage; it is assumed they will stay with the train for the entirety of the train route.

There are passenger, dinning and baggage cars. There are 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 at meal times. (our database doesn’t deal with specific meals)

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. Ticket pricing is calculated by multiplying the base voyage price by the cost of selected traveling class.

**Goals:**

The passenger train database should be helpful to all its users. It will cover all of the above uses, and also create a convenient, simple, and useful tool to access, store, and manipulate a passenger train network. Based on our specifications, we hope our DBMS will accurately model a passenger railway system. (but we know we can’t accurately model the railway system—that’s what the necessary evils are about) We will create many relations in the hopes of making the database easily extendible with the creation of new relations. (you might want to specify that we’re creating these relations carefully and cleverly, not just adding a lot of them) Creating many relations will also hopefully eliminate repetition making our database able to perform fast lookups, and require minimal memory usage. (you need to justify how more relations equals less memory usage)

(Benji: explain why we chose to do these possibly bad things with a couple of sentences each)

**Breaches of Reality and Simplifications:**

* we assume all trains can be coupled and decoupled instantly
* we do not require that a train or car begin its next voyage in the station where its previous voyage ended
* we do not prevent collision past except to prevent two trains from beginning their voyage at the same station in the same direction
* we do not require staff for the dining cars, loading the baggage cars, etc.
* train sizes are limited by the number of cars the engine can pull, not the weight the engine can pull
* we don't forbid unconnected train stations (no tracks coming or going)
* we don't forbid carless voyages, or voyages with no tickets sold
* we don't have staff cars, fuel cars, supply cars, etc.
* we don't have a rigorous way of calculating route distance or time length, so two otherwise identical routes might claim to be different lengths or have different travelling times
* related to above: track sections have no associated time/distance cost
* we don't consider that different trains might move at different speeds depending on the model or the number of cars being pulled
* trains may complete one voyage and depart on another at the same instant, i.e. no minimum layover time
* we don't permit multiple engines per voyage
* we ignore the coupling order of cars