**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. Assuming one engine (what does ‘this’ refer to?) is important in the design of the DBMS as the engine along with date and time of a particulare voyage (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, a voyage is a train travelling on a specific route (this sentence may be read to mean that a single voyage has multiple routes) 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. (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. Although we understand the databse cannot actually represent an entire real life passenger train database as there are to many complexities to model in the time give. (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. Many relations will allow for new types of relations such as new types of cars that can be pulled by the engine. If we only has a single relation representing the database adding new types of cars could be impossible as they might not fit the attribute constraints. (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 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)Using many relations rather than one “giant” relations increases performance by reducing the number of repeated tuples. There are less repeated tuples as commonalities among tuples of specific attributes, are made into their own relations which are not bags, and contain only one copy of the tuples. Because the relations contain foreign keys, and common attributes they can be combined to contain all information that would be found in the single “giant” relation.

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

**Breaches of Reality and Simplifications:**

* Assumed that all trains can be coupled and decoupled instantly.
* It is not required that a train or car begin its next voyage in the station where its previous voyage ended.

Considering the end position of each train car in the creation of new train voyages(specifically their start position) would require a level of complexity that we assumed to be to complex for the scope of this project. In order to consider this constraints would have to be written in the interface such that no new voyage can be created unless all cars are available at the start location. Also attributed of current location would have to be added to each train car and engine relation.

* Collisions are not prevented beyond two trains beginning their voyage at the same station in the same direction.

We assumed that two there would be parallel tracks between each train station with each set of tracks running in opposite directions, and trains comforming to their proper set of tracks. Because of the parallel tracks there should not be head on collisions in any situation, trains may be staggered headed in the same direction on the same section of track, as there could be different train voyages running the same sections.

* Staff are not designated to the dining cars, loading the baggage cars, etc.

The staff/voyage relation lists what staff are present on each voyage, we assume that the staff understand their responsibilities (such as loading baggage) when assigned to a specific position on a voyage.

* Train sizes are limited by the number of cars the engine can pull, not the weight the engine can pull.

If we wanted to limit train size by weight it would require that all train car relations include a weight attribute. This would add another level of complexity we believed to be ouside the scope of the project.

Unconnected train stations are not forbidden. (no tracks coming or goingIt is not the train stations that are important in the modeling of a route, rather the sections of track available. Train stations not connected by rail sections are not considered as the do not affect the modeling of a route. A constraint will be implemented where the section/rout relation cannot have null present in any of its tuples.

* Carless voyages, or voyages with no tickets sold are not forbidden.

Voyages are created before tickets are sold, and voyages require an engine ID, so all voyages will have minimally an engine. It will be up to a system administrator to decide when trains should be cancelled, based on circumstances surrounding the voyage.

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

It is assumed all supplies and space required for the regular functioning of a train will be present on the passenger, dinning, luggage cars and the engine.

* Calculating route distance and time length are estimated, so two otherwise identical routes might claim to be different lengths or have different travelling times

Route distance is calculated by adding distance of train sections travelled, this could differ based off slightly different sections travelled. Time length of a voyage is an educated estimation, this is common practice in the travel industry.

* Related to above: track sections have no associated time/distance cost
* Different trains might move at different speeds depending on the model or the number of cars being pulled.

Trains can travel along the same sections a rail, however it is assumed the engineer will be aware of other trains travelling on the same section and the speed they are moving at relative to themselves. The engineers will take the responsibility of controlling this situation.

* Trains may complete one voyage and depart on another at the same instant, i.e. no minimum layover time.

Start and end times of a voyage can be padded with additional minutes on each end to include the loading/unloading/fueling/boarding of a train. Adding layover time would require adding layover time attributes to train car relations, and a completion attribute to the car/voyage relation that must be updated when a voyage is complete, this would add more complexity that we though the scope of the project covered.

* we don't permit multiple engines per voyage

The engine on a voyage serves as a foreign key in many of our relations; it is the single thing that is common among the train cars, and passengers on a specific voyage. If our database allowed for multiple engines to partake in a voyage the foreign key relating train cars, passengers and voyage would be ruined.

* we ignore the coupling order of cars

The order of cars does not matter.