Proposal Outline
Anthony Trinh, Daniel Safarov
March 18th, 2020
Project #28 | Train Company
URL: http://access.engr.oregonstate.edu:7950/home

Executive Summary

Steps 0 - 2:

The first few steps were focused on a project proposal and design. During this time we received quite a bit of feedback on the initial design of our project. We needed to focus less on project details and instead incorporate more justification for our proposal by using facts for a train website. One of our major issues was that we had too many unnecessary tables. Our initial design was problematic being that it had redundant entities. The tables we kept had unclear purposes (for example, our Trains table didn't make it clear if we were referring to a physical train or to a scheduled run of a train on a specific route at a date/time). We made sure to redesign our tables and remove those that were deemed unnecessary (like Mechanics and Conductors). The initial design also housed multiple M:M relationships, greatly complicating our tasks. Redesign in that department was essential as well. We reduced our M:M relationships to one core relationship (routes through stations).

Steps 3 - 4:

In steps 3 - 4 we did a lot of database refactoring and query building. In the overview, based on TA feedback, we made sure to add consistent capitalization for all our entities and lowercase the names for all their attributes. We needed to make sure that our foreign keys were not allowed to be NULL. Also, we had huge cardinality issues that overly complicated the flow of our databases, so with the assistance of a TA we were able to simplify relationships between Trains and Stations and Conductors and Trains (we deleted the Conductors entity along with its foreign key in Trains completely removing the redundant relationship. Our goal was to shorten the number of relationships). With this simplification we were able to add attributes to our entities that tied our database flow in a nice bow (we added "stationname" to Stations, "routename" to Routes, created a join table for Stations and Routes and added "conductorfirstname"/"conductorlastname" to Trains).

Steps 5 - 6:

Most of our feedback for these steps were UI related. We made sure to clean up how our cards looked, the color scheme or our webpage and we added necessary functionality for data creation, navigation, deletion and readability. By step 5 we had CREATE and READ working flawlessly. By step 6 our website could SEARCH, UPDATE and DELETE. With the implementation of SEARCH, we noticed that fundamentally we were implementing our backend incorrectly, so major refactoring was required. By the end of step 6, our backend code was cleaned up and our UI was refined.

Project Outline and Database Outline - Updated Version:

AnyTrak sells $100,000 worth of train tickets every year. Instead of paper tracking, they will use a database driven website in order to track train routes and stations. They are having trouble keeping track of which Routes a certain Trains is on and what Stations certain Routes travel through. With the use of a database, they will be able to more efficiently look up and update Routes. Each Trains can either have one Stations that they reside at or none at all due to out of service issues. When inserting an entry into Trains, its main station must be assigned. When the Station is deleted, this is the only time that the Trains' main station can be NULL. Each Routes must have one and only one Trains running it, but Trains can run multiple Routes or none at all (1:M). Routes will have multiple Stations that they go through and Stations can have multiple Routes that go through it (M:M).

Entities:

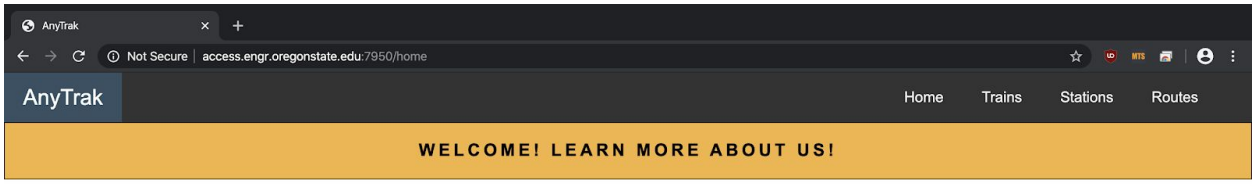
- **Trains: The physical train itself. Will be assigned route(s) to travel on.**
    - trainID: int, auto_increment, unique, not NULL, PK
    - stationID: FK,
    - conductorfirstname: varchar
    - conductorlastname: varchar
    - model: varchar, not NULL
    - cost: int, not NULL
    - capacity: int, not NULL
    - Relationship: 1: M relationship between Trains and Routes is implemented with trainID as a FK inside of Routes
    - Relationship: 1:1 relationship between Trains and Stations with stationID as FK in Trains

- **Stations: Houses trains and are the start and ending point of every train's journey.**
    - stationID: int, auto_increment, unique, not NULL, PK
    - stationname: varchar, not NULL, unique
    - address: varchar, not NULL
    - state: varchar, not NULL
    - city: varchar, not NULL
    - zipcode: int, not NULL
    - Relationship: 1:M relationship between Stations and RoutesThruStations with StationID as FK in RoutesThruStations (Part of M:M between Routes and Stations)

    - Relationship: 1:1 relationship between Trains and Stations with StationID as FK in Trains
- **Routes: The system by which trains navigate through to arrive at specific stations.**
    - routeID: int, auto_increment, unique, not NULL, PK
    - trainID: FK, not NULL
    - routename: varchar, not NULL, unique
    - ticketprice: int, not NULL
    - Relationship: 1: M relationship between Trains and Routes is implemented with trainID as a FK inside of Routes
    - Relationship: 1:M relationship between Routes and RoutesThruStations with routeID as FK in RoutesThruStations (Part of M:M between Routes and Stations)
- **RoutesThruStations: What route leads to what station.**
    - routesthrustationsID: PK, auto_increment, not NULL, unique
    - routeID: FK, not NULL
    - stationID: FK, not NULL
    - travelduration: int, NOT NULL
    - milestraveled: int, NOT NULL
    - Relationship: 1:M relationship between Routes and RoutesThruStations with routeID as FK in RoutesThruStations (Part of M:M between Routes and Stations)
    - Relationship: 1:M relationship between Stations and RoutesThruStations with stationID as FK in RoutesThruStations (Part of M:M between Routes and Stations)

Database Screenshots

- Home (Nothing, just explains entities).



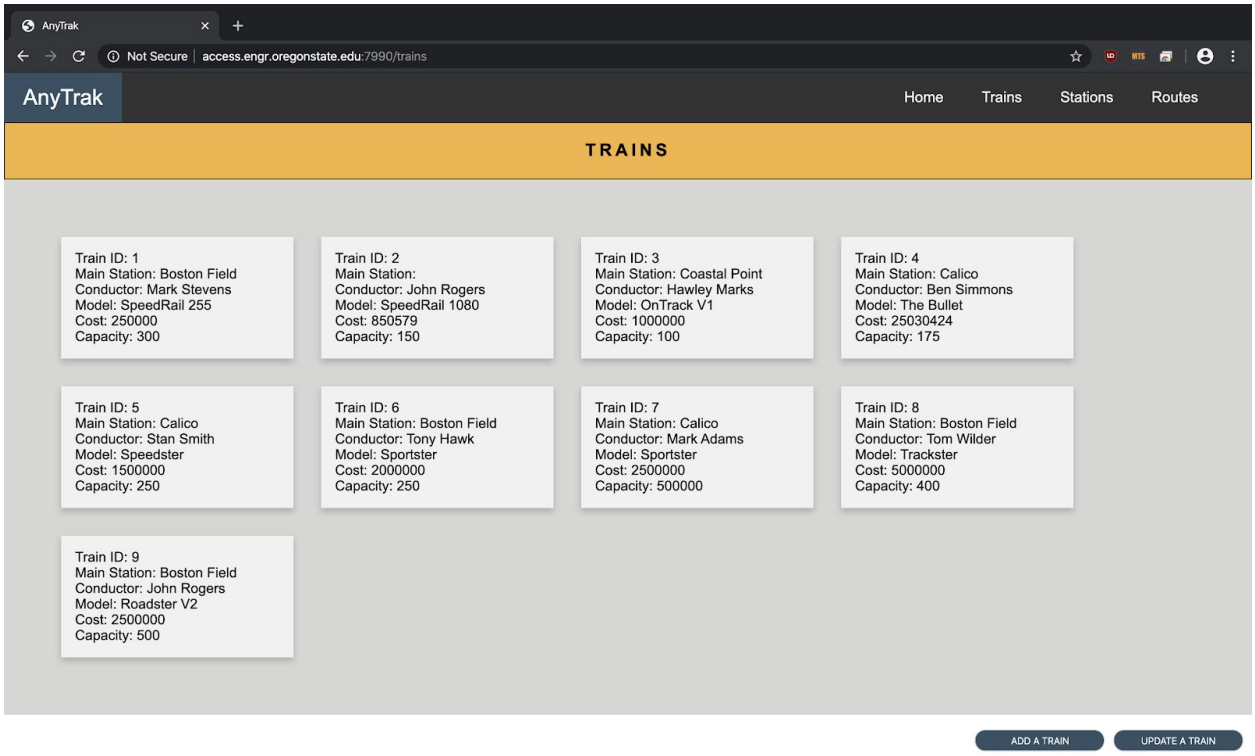
Welcome! This website will allow you to track routes, view stations, and see trains!

Trains: Lists all of the Trains. User can add or update trains.

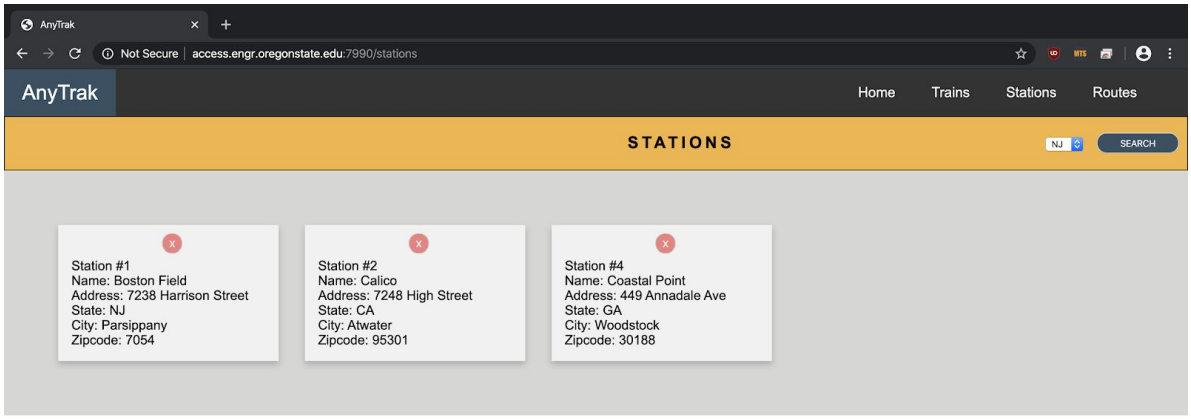
Stations: Lists all of the trains. User can add a station. They can also filter the station by their states.

Routes: Lists all of the Routes with the RoutesThruStations intersection table connected to each Routes. Shows each Routes and its corresponding Stations below in order.

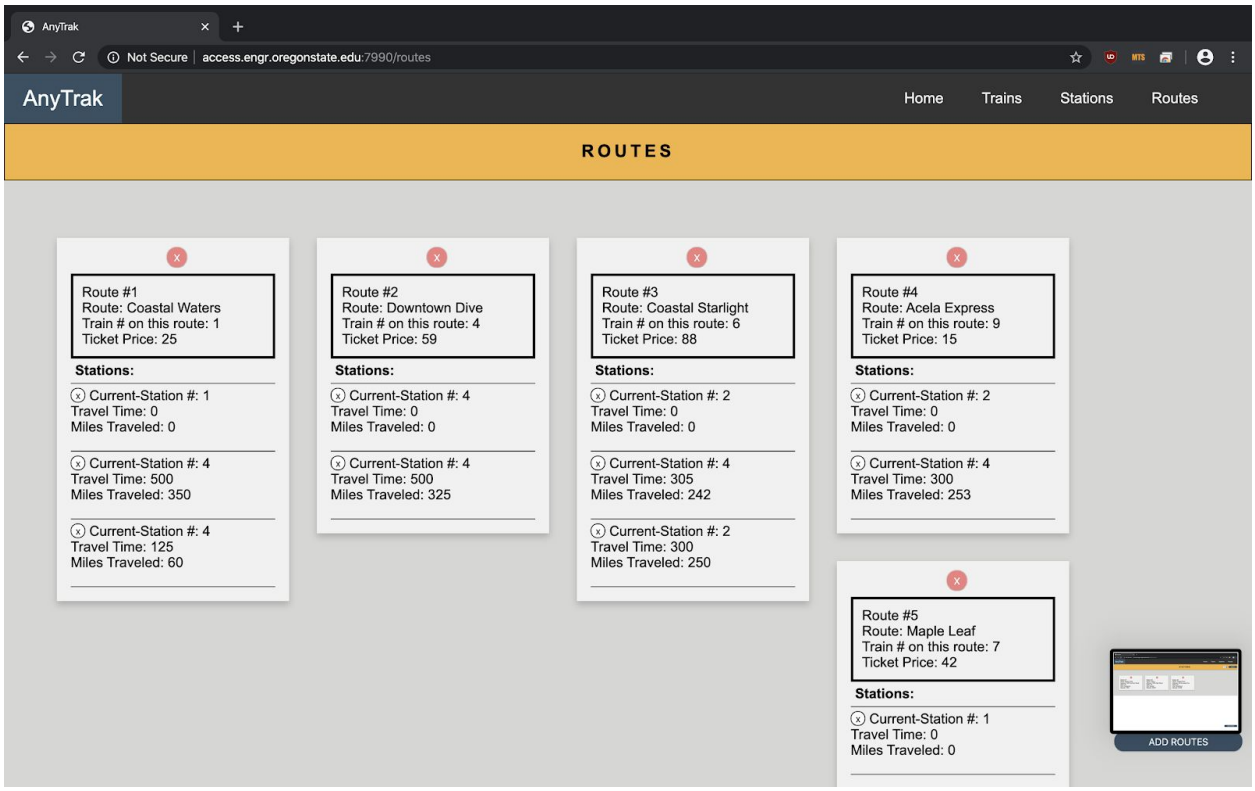
- Trains (CREATE, UPDATE)



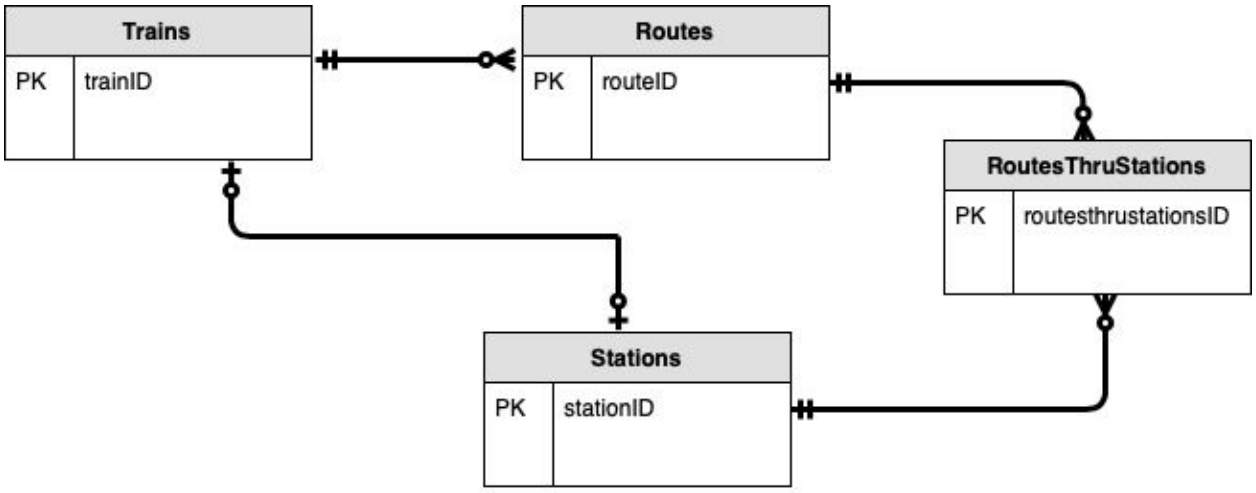
- Stations (CREATE, DELETE, FILTER)



- Routes (CREATE, DELETE) + RoutesThruStations (CREATE, DELETE)



Entity-Relationship Diagram



Schema