Team Cyclone – Group 104

Cyclone Train Ticketing System

Sebastian Gajardo

Bryan Gronberg

1. **Project URL**

[**http://flip1.engr.oregonstate.edu:3234/**](http://flip1.engr.oregonstate.edu:3234/index.html)

1. **Project Outline Database Outline ERD and Schema Updated Version**

### **a)** Overview

A small city is reinvesting in its low cost railway in an effort to appeal to tourists, commuters, and young people looking to relocate downtown. Cyclone trains are projected to generate $10 million in trip sales revenue every year. There are 15 unique trips which connect 5 stations and it is expected that 2,000 to 3,000 tickets will be sold each day across all stations in the city. Cyclone trains consist of one train type with a capacity for 200 passengers and because the system is low cost, a trip has one cost regardless of day or time.

In the past, the ticketing system was criticized for being outdated. There was no way for ticket collectors to validate or issue tickets electronically. Passengers needed to purchase their tickets at the station. It was also difficult for the city to analyze purchase activity because minimal data was collected at the point of sale. To support the improvements to the infrastructure of the railway, additional investment will be made to design a database backend for purchasing tickets. By creating this database, mobile and web applications can later be designed as front end portals for customers. Also, electronic ticket validation can happen consistently and in real time while keeping track of sales and passenger numbers.

Passengers can buy zero or many Tickets (1:M); and many Trips can be taken by many Passengers (M:M). A single ticket must belong to a single Passenger (1:1), and there’s one Trip per individual Ticket (1:1). Zero or more Tickets can be bought for one Trip (M:1) A Ticket must have a single Trip assigned to it (1:1) and can have zero or one Train assigned to it. Each Trip must have one Departure and one Arrival (1:1). A Train can be on zero or many Tickets (train can be NULL in case the train number is unknown at point of sale (regular train needs maintenance, was decommissioned etc.)). Arrival and departure stations can be part of many Trips or none (1:M) (NULL, can add arrival and departure stations without connecting to Trip).

### 

### 

### **b)** Database Outline, in Words

Concept: Train Ticket System

Entities:

1. Passengers (Entity): Record details of passengers that ride our trains.

* passengerId: INT, auto\_increment, unique, not NULL, PK.
* fistName: TEXT, NULLable.
* lastName: TEXT, NULLable.
* dob: DATE, NULLable.
* **relationships: 1:M with Tickets with passengerID as an FK inside Tickets.**

1. Trips (Entity): Available trips on our train system and prices associated with them. Each departure and arrival pairing is given a tripId number to act as the PK for simplicity.

* tripId: INT, auto\_increment, unique, not NULL, PK.
* departureId: INT, not Null, FK.
* arrivalId: INT, not Null, FK.
* price: DECIMAL, not NULL.
* **relationships: 1:M with Tickets with tripId as an FK inside Tickets.**

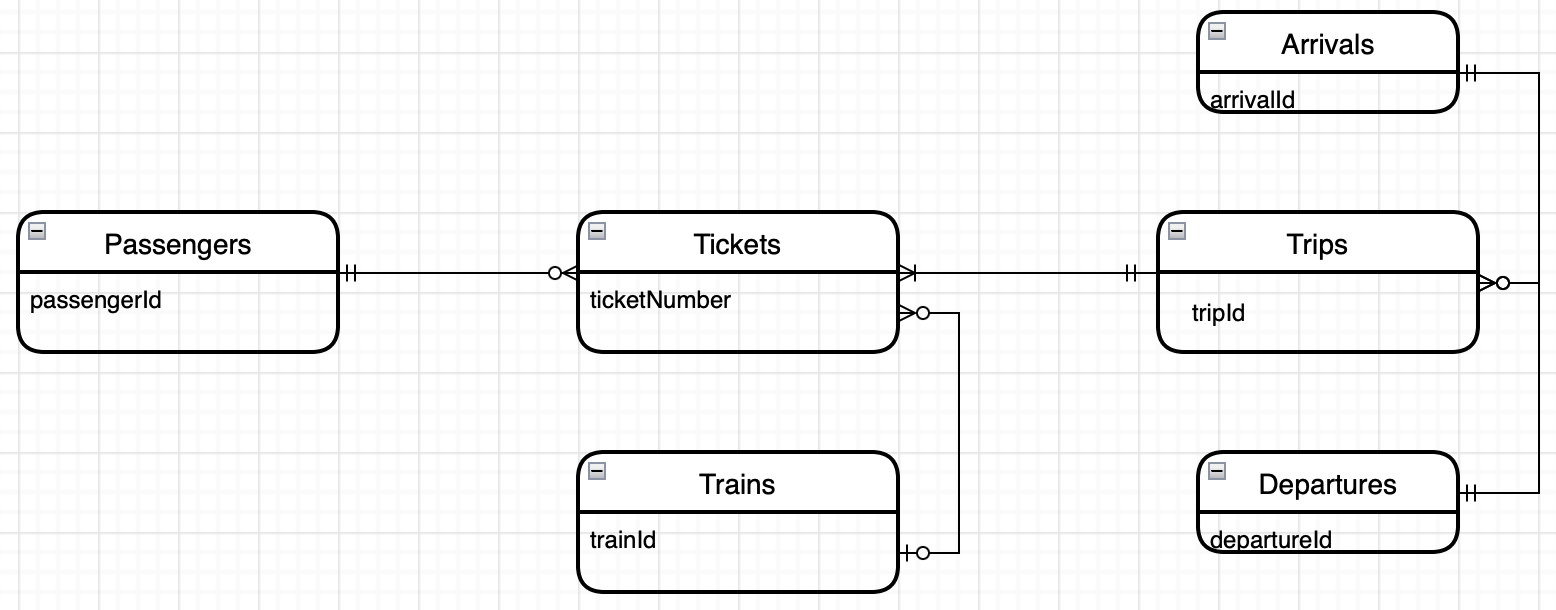
1. Tickets (Entity & Intersection Table): Keep track of tickets sold per date. Also acts as an intersection table between Passengers and Trips. TrainId is optional and can be NULLED.
   * ticketNumber: INT, auto\_increment, unique, not NULL, PK.
   * passengerId: INT, not NULL, FK.
   * tripId: INT, not NULL, FK.
   * trainId: INT, NULLABLE, FK.
   * tripDate: DATE, not NULL.
   * **relationships: Intersection table for Passengers and Trips M:M relationship, M:1 for both with passengerID and tripID as FK’s. Also M:1 with Trains with trainId as FK.**
2. Departures (Entity): Stations trains depart at.

* departureId: CHAR, unique, not NULL, PK.
* departureLocation: VARCHAR, not NULL.
* **relationships: 1:M with Trips, with departureId as an FK inside Trips.**

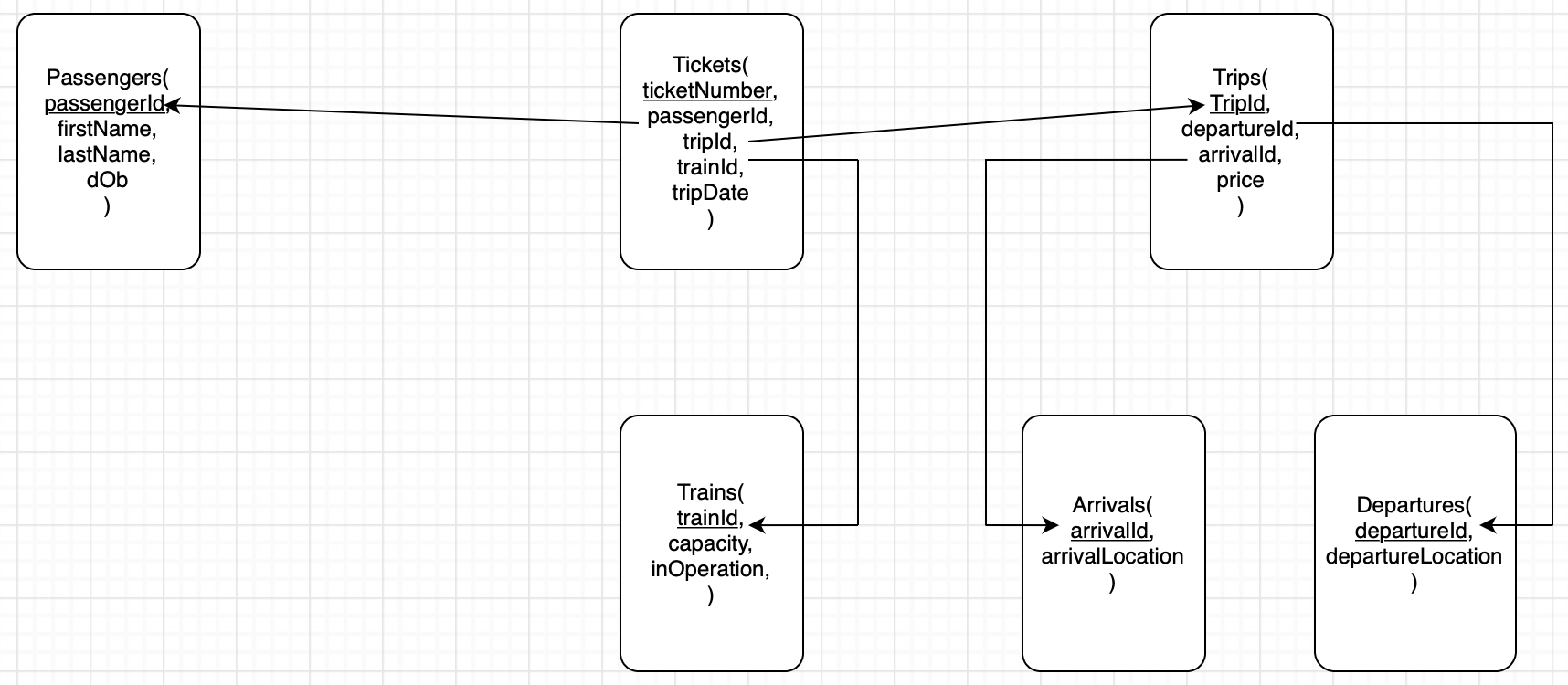
1. Arrivals (Entity): Stations trains arrive at.
   * arrivalId: CHAR, unique, not NULL, PK.
   * arrivalLocation: VARCHAR, not NULL.
   * **relationships: 1:M with Trips, with arrivalId as an FK inside Trips.**
2. Trains (Entity): Details of trains owned by the company, including if in operation or maintenance. All trains have a capacity of 200.

* trainId: INT, auto\_increment, unique, not NULL, PK.
* capacity: INT, not NULL.
* inOperation: BOOL, not NULL.
* **relationships: 1:M with Tickets with trainId as a NULLABLE FK in Tickets**

**c) Entity-Relationship Diagram**

****

**d) Schema**

****

1. **Fixes Based on Feedback from Previous Steps (Steps 4 & 5)**

From step 3 the grader noticed we had the arrows pointing the wrong direction in our Schema from PK to FK instead of from FK to PK so we flipped the arrows around. He also pointed out we needed to update our ERD to reflect that trains is nullable for tickets.

Our peer feedback mentioned we had a name mishap between our schema and sql files for Ticket where date was tripDate. We decided to go with tripDate for both. Besides that they mentioned we should be careful with our use of ‘cascade if delete’. We kept our cascade if delete the same for now as we don’t feel we’re far enough along to test that properly and will revisit before next week.

On our own we decided to rename the location attribute in Arrivals to arrivalLocation and did the same in Departures to departureLocation. This was done for simplicity's sake to easily be able to differentiate between the two. We also decided to change the display of the Tickets table to include the passenger first and last name and trip in arrival- departure format, instead of the passengerId and tripId for ease of use. We did the same thing to the Trips by adding the departureLocation and arrivalLocation to the table instead of departureId and arrivalId, this again makes it much more user friendly and easier to interact with.