CS 387

COURSE PROJECT

RAILWAY MANAGEMENT SYSTEM

Members

- Andraju Manoj Varma - 190050016
- Valeti Bharat Chandu - 190050126
- Rajulapudi Kale Mastansha Goud - 190050098
- Dasari Gnana Heemanshuu - 190050029

Project Charter

➢ The *Indian Railways* is one of the largest organizations in the world. It employs an enormous number of people, runs over 21 thousand trains which are used by millions of people all over the country everyday.

Our project aims to build a ***Database Application*** to assist the administrative side of a railway organization. A railway administration using our application should be able to keep track of, update and manipulate relevant data (mentioned in the next part) for the organization.

➢ Owing to the gigantic size of the organization, a simple file system can lead to several problems with severe results, like

- **Data Redundancy and Inconsistency :** In our case, the data about railway stations has to be stored. If a station name or code is changed, or the station is no longer in function, having a rudimentary storage system would mean that we have to edit the station data for *each* train that arrives at that station. This can lead to heavy inconsistency across the data.
- **Difficulty in accessing data:** For each new task, such as to list out the trains that arrive at a station, a separate program has to be written or the data has to be manually checked. A program to count the number of tickets booked in a train would be very different from the mentioned above. There is a lot of grunt and repetitive work involved.
- **Integrity problems**: Integrity constraints such as two trains not being on the same platform at the same time become buried in the program code. Adding new constraints will be difficult.
- **Concurrent access and Security problems:** The application should give a unique selection result when several concurrent requests are made for the same seat at the same time. A Ticket Collector should be able to access only information about his own , but not of others, and thus that information has to be secure.

User classes and Interactions

We have a single user class : ***The Railway Administrator,*** referred to as ***Admin.***

The Admin uses the system by keeping track of, manipulating and updating the following:

- **Basic details of all trains:** Input can be a query and for which it should display a corresponding list of trains. For a given input, the details of each train can be updated. Also the trains can be added or deleted along with their details.
- **Coaches and Engines assigned to each train:** The Admin can assign what coaches and engines a train should run with in a particular route.
- **Stations and Routes:** Input can be a query for which it should display a list of corresponding stations, and the route taken along with the intermediate stations through it. For a given input, the details of each station can be updated. Also the stations can be added or deleted along with their details.
- **Booked and Available seats in the train (ACID transactions for booking):** When a request to book a particular seat is given as the input, the system should be able to update the booked and availability status of the seats, abiding all security measures.
- **Trains running between two stations:** Given an input of two stations for a particular date, and trains running from one station to the other one as a query, it should display all the trains running from that station to the other one, including all those trains which have those both stations as intermediate stops.
- **Maintenance details of coaches and trains:** This includes the query of what the fuel type or the power source is for the train
- **Traffic in a route or a station:** The Administrator can query the traffic, i.e., the number of trains passing per unit time through a particular route or checkpoint, and even stations.
- **Major employee details:** The system must maintain the details of some important people associated with the running of the train, like the train's locomotive pilot, ticket collector, etc.

  For every detail mentioned above, the system should be able to add or delete entries along with the details on an authenticated request for a valid change.

Logical Schema

The entities that our system uses will be

1. **Train**(Name, Train_ID, Start, Destination , Start_time, Destinaton_time, Train_type)
2. **Station**(Name, Station_ID, City, Platforms)
3. **Coach**(Coach_ID, seat_count, coach_type, coach_category)
4. **Engine**(Engine_ID, engine_type)
5. **Date(** Date, Day)
6. **Employee**(Employee_ID, Employee_name, age, profession, salary)
7. **Berth**(Berth_no, Berth_type)
8. **PF_status**(Station_ID, Platform_no, Status)
9. **Train_Composition**(TrComp_ID, Train_ID, Date, Engine_ID)
10. **Train_Coach**(TrComp_ID, Coach_ID)
11. **Train_Route**(TrRoute_ID, Train_ID, Date, Station_Count)
12. **Route_Stations**(TrRoute_ID, Station_ID, Time)
13. **Booking**(PNR, Train_ID, Date Berth, Coach_ID, Ticket_Category, Price)
14. **Train_Employee**(Train_ID, Date, Employee_ID)
15. **Station_Employee**(Station_ID, Date, Employee_ID)

Constraints

The constraints on entities are

1. Train (

   Name *type* char (null implies that the train may not have name eg:local trains)
   Train_ID *primary key type* number
   Start, Destination *foreign key* to station both not equal, if start or destination is deleted then the train should be deleted.
   Start_time, Destination_time *type* timestamp not null
   Start_time < Destination_time,
   Train_type *type* char (null implies that train type may not be known or runs on random days)
)

2. Station (
   Name *type* char (null implies that the station may not have a name),
   Station_ID *primary key type* char *capital letters*,
   City *type* char not null,
   Platforms *type* int not null
)

3. Coach(
   Coach_ID *primary key type* int,
   Seat_count *type int* not null,
   Coach_type *type* char (null implies that the coach type is unknown),
   Coach_category *type* char (null implies that the coach category is unknown)
)

4. Engine(
   Engine_ID *primary key type* int,
   Engine_type *type* char not null
)

5. Date(
   Date *primary key type* date,
   Day *type* char not null
)

6. Employee(
   Employee_ID *primary key type* int,
   Employee_name *type* char not null,
   Age *type* not null,
   Profession *type* char not null,
   Salary *type* int not null
)

7. Berth(
   Berth_no *type* int *primary key,*
   Berth_type *type* char not null
)

8. PF_Status(
   Station_ID *primary key foreign key to* station
   Platform_no *primary key to* platform
   Status *type* int 0 *or* 1 *or* null (null implies platform can't be used)
)

9. Train_Composition(
   TrComp_ID *primary key,*
   Train_ID *foreign key to* Train,
   Date *foreign key to* Date,
   Engine_ID *foreign key to* Engine
)

10. Train_Coach(
    TrComp_ID *primary key foreign key to* Train_Composition,
    Coach_ID *primary key foreign key to* Coach
)

11. Train_Route(
    TrRoute_ID *primary key,*
    Train_ID *foreign key to* Train,
    Date *foreign key to* Date
    Station_Count *type* int not null
)

12. Route_Stations(

TrRoute_ID *primary key foreign key to* Train_Route

Station_ID *primary key foreign key to* Station

Time *type* timestamp not null

)

13. Booking(

PNR,

Train_ID *foreign key to* Train,

Date *foreign key to* Date,

Coach_ID *foreign key to* Coach,

Berth_no *foreign key to* Berth,

Ticket_Category *type* char not null,

Price *type* int not null

)

14. Train_Employee(

Train_ID *primary key foreign key to* Train,

Date *primary key foreign key to* Date,

Employee_ID *foreign key to* Employee

)

15. Station_Employee(

Station_ID *primary key foreign key to* Station,

Date *primary key foreign key to* Date,

Employee_ID *foreign key to* Employee

)

Materialized Views

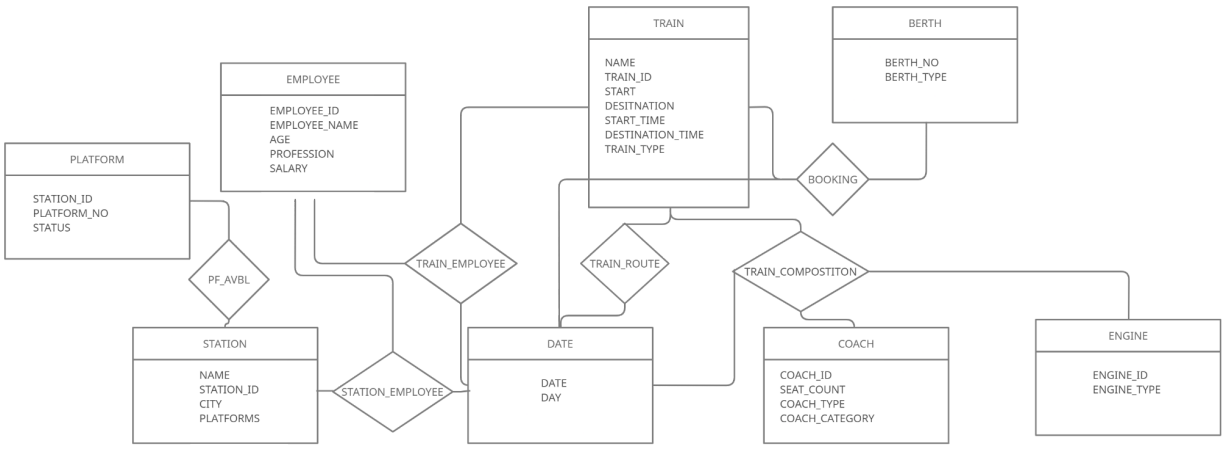
NONE

Relations

The relations used will be

1. **Train_route** on Train and Date with attributes no_of_stations,{intermediate_station}
2. **Train_composition** on Train, Date, Coach, Engine
3. **Train_Employee** on Train, Date and Employee
4. **Station_Employee** on Station, Date and Employee
5. **Booking** on Train, Date, Coach, Berth with attribute Price and Ticket_category
6. **Pf_Avbl** on Platform, station

ER Diagram



DB Operations

1. <u>The Basic Details of Train</u> :

   SELECT, UPDATE, INSERT, DELETE on the table *Train* modifies/fetches an entry in the table, and when a train is deleted, every entry in the other tables that references that entry of *Train* will get deleted.

2. <u>Coaches and Engines assigned to each train</u> :

   SELECT, UPDATE, INSERT, DELETE on *Train_Composition* relation between *Train*, *Coach* and *Engine* tables associates the given engine and coach to a particular

train on a particular date.

3. Stations and Routes :

   SELECT, UPDATE, INSERT, DELETE on the table *Station* modifies/fetches an entry in the table, and when deleted, every entry in the other tables that references that entry of *Station* will also get deleted.

4. Seat Availability Status :

   SELECT, UPDATE, INSERT, DELETE, WHERE on the table *Booking* to check if a seat is available and book an available seat and update the seat status if a request is issued.

5. Trains running between two stations :

   SELECT, JOIN(SELF JOIN), WHERE on *Train_Route* and *Route_Stations* tables when queried to find trains between 2 given stations as Start and Destination, tries to find the trains with *Start* and *Destination* as intermediate stations in the train's route.

6. Maintenance details of coaches and trains :

   SELECT, UPDATE, INSERT, DELETE on *Coach* and *Engine* to check and update the condition of coaches and engines.

7. Traffic in a route or a station :

   SELECT, TIMESTAMP, JOIN, WHERE on *Train*, *Station* and *Train_Route* to find the number of trains at a particular point in context to find the traffic.

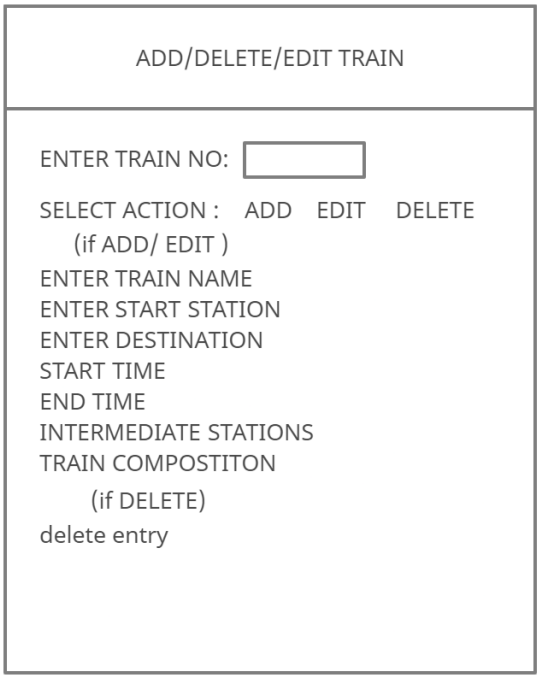
8. Major Employee Details :

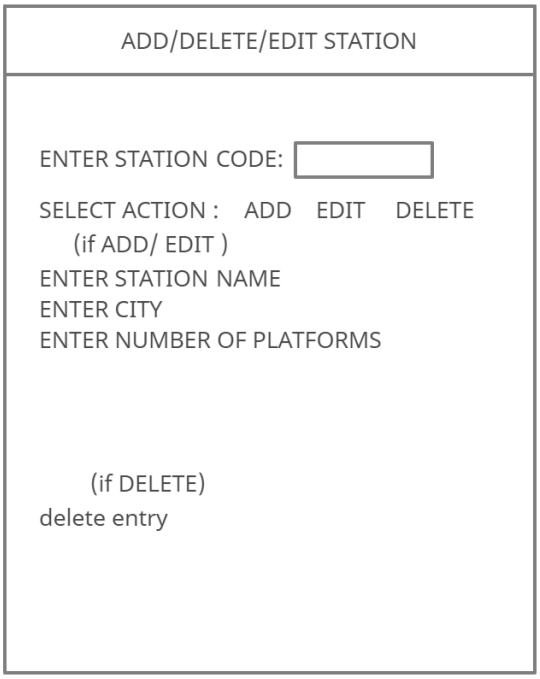
   SELECT, UPDATE, INSERT, DELETE on *Employee*, *Station_Employee*, *Train_Employee* tables to maintain the essential information about employees working within our system. Deletion of an entry in *Employee* will delete all other entries associated with that employee in all other tables as well.

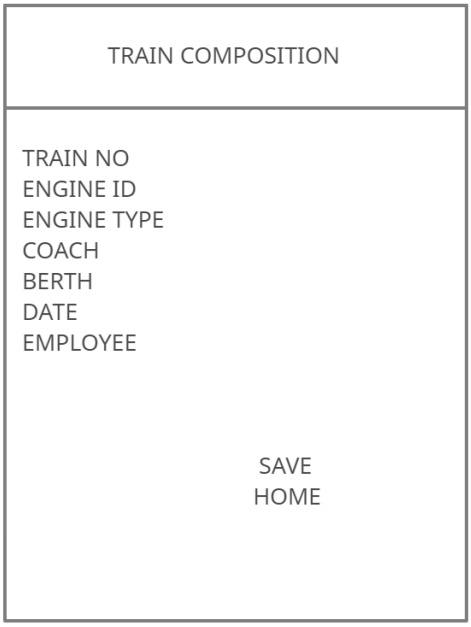
Softwares Used

We have employed angular for frontend, node.JS for backend, SQL as our query language, postgres and python scripts to generate large test data.

Forms







Results (What we could and could not do)

- **What we could do**
  - Created interfaces
    - To view, add, update basic train details
    - To view, add, update basic details of engines, coaches and stations
    - To view, add, update employee details
    - To find the schedule of a train given train number and date
    - To view all trains running between 2 stations
  - Implemented SQL queries in the backend and integrated with the frontend to avail the above described functionalities
- **Further work we had aimed to do**
  - Provide further interfaces for the railway administrator
    - To be able to update platfor details like its occupancy status when a train arrives there
    - See the statistics of a train - i.e., number of bookings, how many seats are of each type (such as lower, upper etc.)
  - Provide facility to auto-update the database based on instructions contained in a file that can be uploaded to the website
  - Provide authentication based on a login/logout system to ensure security of the database - (using the django user authentication model, for instance)
  - Improved styling, i.e., better organisation of available options on the website to make it easier for the user to navigate through
  - Ensure correctness of large test data - i.e., make sure that foreign keys in a relation only refer to values that already exist in the database.
  - Set platform number to auto-increment when a new platform is added to a station - to reduce the work the user has to do
  - Similarly, set coach number in the tr_coach relation to automatically increment when a new coach is added to a train
- **Further work**
  - The completed project can be extended to enable a simple user (unauthorised) to ONLY view trains scheduled (scheduler app)
  - Add user authentication to convert into a full fledged booking system