CPSC 471 Final Project
University of Calgary

Group Member:
Yifei Lin 30043746
Xudong Miao 30050905
Aidan Eyre 30038900
Tutorial:T05

Table of Contents

Abstract:

Our database allows an easy way to keep track of Canada's widespread railway system, including every train (including cars), train station, and route (for both freight and passenger trains). It has many different use cases, including for consumers, product producers, and the train companies themselves.

With our database, users will be able to retrieve information about all the passenger routes running across Canada and every city connected to the railway system. They will also be able to search for different routes between cities, and can then retrieve prices from different companies in order to compare and find the best prices.

Product producers will have the ability to look at all the different shipping routes and see what goods different trains carry. That way they can get helpful information to assist them in determining which railway company they should use to transport their goods.

Also, the train companies can use the database to track and update information about all their train routes. They can assign a pilot or mechanic to a specific train, or a train to a specific route. They can also add and remove trains or stations and update ticket prices.

Overall, our database just aims to be versatile in its ability to provide many different helpful functions to consumers, producers, and companies alike, as well as filling in a gap and bringing an abundance of railway information together into one place.

Introduction:

Currently, Canada's railway system is a large, complex entity spanning from coast to coast. Though it's mostly used for freight transportation at the moment, customers still have the ability to book tickets and travel across Canada in passenger trains. However, at the moment there is no central database covering the entire railway system. Instead, individual companies each have their own databases containing their routes and trains. While Via Rail provides a way for passengers to easily find routes and book tickets, it contains no information about shipping goods. All this different information is split into different, spread-out components, so we figured it would be handy if it were all brought together into one location.

That is where our database comes in. It takes all these different scattered pieces of information and brings them together. There will now be a way to track all passenger and freight trains for every train company in one handy database.

Brief description of the system:

We are creating an API with ASP.NET that connects to the database management system where stores the information required and the client may use this API to interact with the database for controlling the values.
More specific information about the API and the DBMS will be discussed in the following design and implementation sections.

Design Section:

Different users:

1.Passenger

```
public DataTable FindTicket(int routeNumber)
{
    MySqlParameter[] Parameters = new MySqlParameter[1];
    Parameters[0] = new MySqlParameter("@routeNumber", routeNumber);
    return Execute_Data_Query_Store_Procedure("FindTicket", Parameters);
}
```

The passenger is able to find the corresponding ticket information when API receives the route number.

```
public DataTable getTicketInfo(int ticketNumber,int ID)
{
    MySqlParameter[] Parameters = new MySqlParameter[2];
    Parameters[0] = new MySqlParameter("@ticketNumber", ticketNumber);
    Parameters[1] = new MySqlParameter("@ID", ID);
    return Execute_Data_Query_Store_Procedure("getTicketInfo", Parameters);
}
```

The other functionality for the passenger to use is retrieving the specific information of the ticket he/she has bought by entering the ticket number and passenger's id.

2.Route manager

```
public DataTable getRouteInfo(int routeNumber)
{
    MySqlParameter[] Parameters = new MySqlParameter[1];
    Parameters[0] = new MySqlParameter("@routeNumber", routeNumber);
    return Execute_Data_Query_Store_Procedure("GetRouteInfo", Parameters);
}

1 reference | 0 exceptions
public int AssignTrain(int routeNumber,int trainID)
{
    MySqlParameter[] Parameters = new MySqlParameter[2];
    Parameters[0] = new MySqlParameter("@routeNumber", routeNumber);
    Parameters[1] = new MySqlParameter("@trainID", trainID);
    return Execute_Non_Query_Store_Procedure("AssignTrain", Parameters);
}
```

For the route managers, they are able to retrieve the route information and assign a train to the route.

```
1 reference | 0 exceptions
public DataTable AddStation(int routeNumber, string stationName)
{
    MySqlParameter[] Parameters = new MySqlParameter[2];
    Parameters[0] = new MySqlParameter("@routeNumber", routeNumber);
    Parameters[1] = new MySqlParameter("@stationName", stationName);
    return Execute_Data_Query_Store_Procedure("AddStation", Parameters);
}
```

And they can also add a station for the route to pass by.

3.Employees who manage the train for one railway company

```
public int AddTrain( int trainID, int nocars, string company_name, bool passenger_flag, int economy, int business,bool freight_flag, 
{
    MySqlParameter[] Parameters = new MySqlParameter[9];
    Parameters[0] = new MySqlParameter("@trainID", trainID);
    Parameters[1] = new MySqlParameter("@NoofCars", nocars);
    Parameters[2] = new MySqlParameter("@company_name", company_name);
    Parameters[3] = new MySqlParameter("@passenger_flag", passenger_flag);
    Parameters[4] = new MySqlParameter("@economy_class", economy);
    Parameters[5] = new MySqlParameter("@business_class", business);
    Parameters[6] = new MySqlParameter("@freight_flag", freight_flag);
    Parameters[7] = new MySqlParameter("@max_cargo", max_cargo);
    Parameters[8] = new MySqlParameter("@max_volume", max_volume);
    return Execute_Non_Query_Store_Procedure("addtrain", Parameters);

}

1 reference | 0 exceptions
public DataTable GetTrainsFromCompany(string company_name)
{
    MySqlParameter[] Parameters = new MySqlParameter[1];
    Parameters[0] = new MySqlParameter("@company_name", company_name);
    return Execute_Data_Query_Store_Procedure("getTrainsFromCompany", Parameters);
}
```

The company manager can add the information of a new train or retrieve the information of the trains the company is currently owning.

```
public DataTable AdjustPrice(int routeNumber,string classType,int price)
{
    MySqlParameter[] Parameters = new MySqlParameter[3];
    Parameters[0] = new MySqlParameter("@routeNumber", routeNumber);
    Parameters[1] = new MySqlParameter("@classType", classType);
    Parameters[2] = new MySqlParameter("@priceUpdate", price);
    return Execute_Data_Query_Store_Procedure("AdjustPrice", Parameters);
}
```

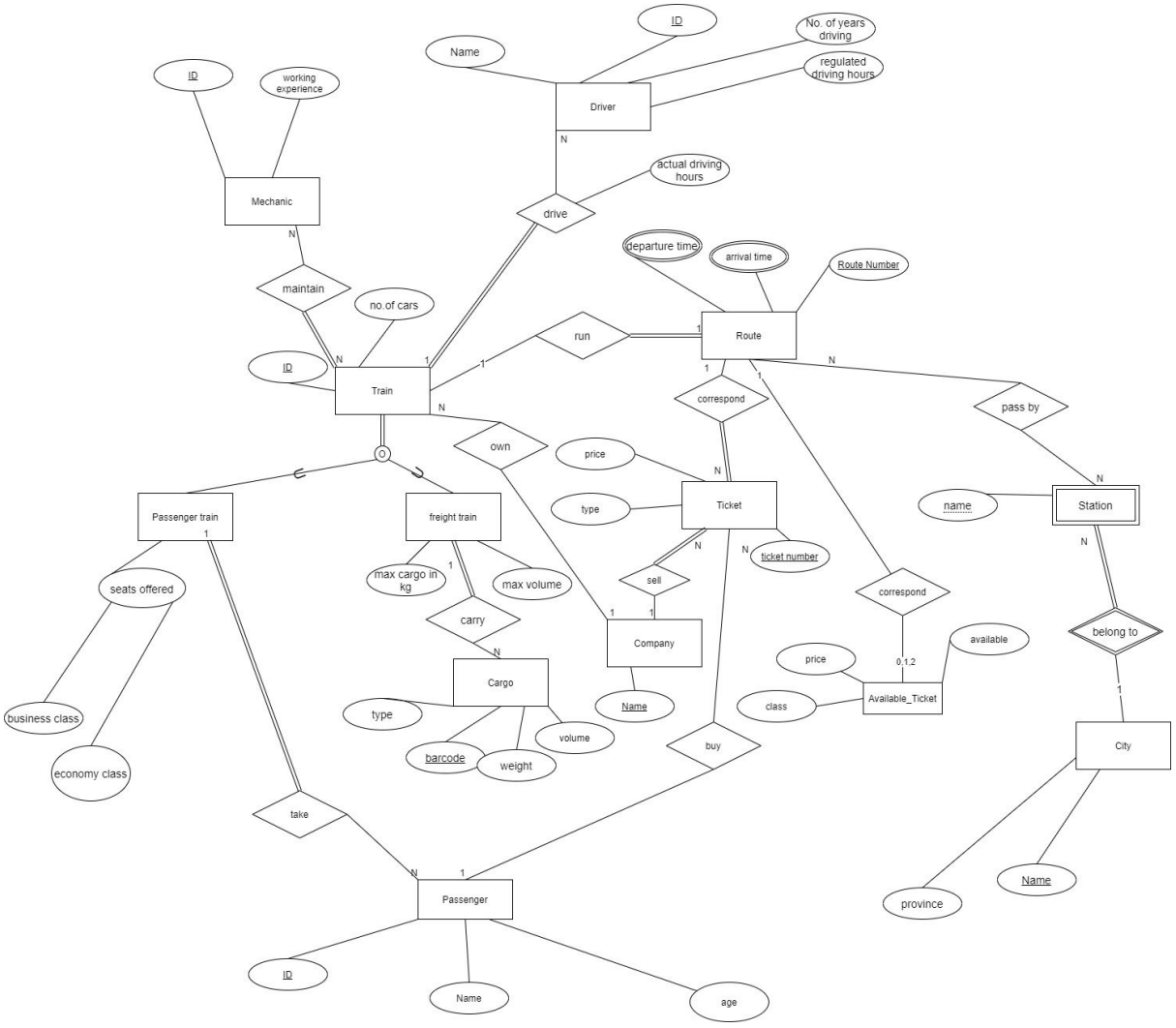
The company can also adjust the price of the ticket for the route.

```
public DataTable AssignDriver(int driverID,int trainID)
{
    MySqlParameter[] Parameters = new MySqlParameter[2];
    Parameters[0] = new MySqlParameter("@driverID", driverID);
    Parameters[1] = new MySqlParameter("@trainID", trainID);
    return Execute_Data_Query_Store_Procedure("AssignDriver", Parameters);
}

1 reference | 0 exceptions
public DataTable AssignMechanic(int mechanicID,int trainID)
{
    MySqlParameter[] Parameters = new MySqlParameter[2];
    Parameters[0] = new MySqlParameter("@mechanicID", mechanicID);
    Parameters[1] = new MySqlParameter("@trainID", trainID);
    return Execute_Data_Query_Store_Procedure("AssignMechanic", Parameters);
}
```

The assignment of a driver or a mechanic to a train is also a functionality provided for the company manager to use. This can also be done by the route manager if needed.

Revised ER diagram:



Remarks:

From the ER diagram, we have deleted the entity “track” for less complexity compared to the diagram we initially submitted.

An entity called “available_ticket” is added for distinguishing the “ticket” entity the passenger has actually bought with the unique ticket number while the entity “available_ticket” is for the result of searching for a ticket for a specific route and the passenger will get the actual “ticket” entity if he/she chooses to buy.

Implementation:

RM diagram:

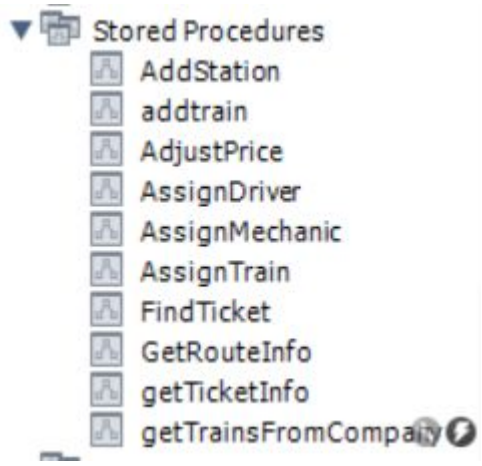
Changes made to the RM diagram is similar to that of the ER diagram by deleting the track entity and adding the available ticket entity for reducing complexity and adding more convenience for distinguishing the tickets bought or searched by the passenger.

Also, during the implementation stage, we found to remove some redundant primary keys for some entities, such as an ID for a city because the city name can be unique as long as the province of it matches.

DBMS:

We use MySQL as the database management platform and in order to incorporate it with the ASP.NET(default capable for MsSQL only), a nutget package library is added to the project for running all the objects of MySQL.

As we create 10 endpoints for the API, there are 10 corresponding stored procedures inside the MySQL database as well.



The procedures will be shown in this order.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddStation`(routeNumber int,stationName varchar(45))
BEGIN
Declare cityName varchar(45) Default '';
Select city_name
From station
Where name=stationName
INTO cityName;
Insert into pass_by
values(cityName,stationName,routeNumber);
Select distinct*
From pass_by
Where route_number=routeNumber;
END
```

Add a station for a route.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `addtrain`(trainID INT,
NOofCars INT,
company_name VARCHAR(45),
passenger_flag BOOLEAN,
economy_class INT,
business_class INT,
freight_flag BOOLEAN,
max_cargo DECIMAL(10,2),
max_volume DECIMAL(10,2)
)
Begin

INSERT INTO train values(trainID,NOofcars,company_name,passenger_flag,economy_class,business_class,freight_flag,ma

end
```

Add a new train for a company.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AdjustPrice`(routeNumber int, classType varchar(45), priceUpdate int)
BEGIN
Update available_ticket
SET price=priceUpdate
Where route_number=routeNumber AND class=classType;
Select *
from available_ticket
Where route_number=routeNumber AND class=classType;
END
```

Adjust the price of the ticket for a route.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AssignDriver`(driverID int,trainID int)
BEGIN
Insert into driver_train
values(driverID,trainID,0);
Select D.driverId,DR.Name,D.trainId
from driver_train AS D,driver AS DR
where D.trainId=trainID AND DR.driverID=D.driverID;
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AssignMechanic`(mechanicID int,trainID int)
BEGIN
Insert into maintain
values(mechanicID,trainID);
Select distinct M.mechanic_id,Me.name,M.train_id
from maintain AS M,mechanic AS ME
where M.train_id=trainID AND ME.mechanic_id=M.mechanic_id;
END
```

Assign a driver/mechanic to a train.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AssignTrain`(routeNumber int,trainID int)
BEGIN
UPDATE route
SET
train_id=trainID
WHERE route_number=routeNumber;

END
```

Assign a train to a route.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `FindTicket`(routeNumber int)
BEGIN
select route_number,price,class
from available_ticket
where route_number=routeNumber AND available=true;
END
```

Find available ticket information of a route.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetRouteInfo`(routeNumber int)
BEGIN
Select R.*,P.city_name,P.station_name
FROM route AS R, pass_by AS P
WHERE R.route_number=routeNumber AND R.route_number=P.route_number;
END
```

Retrieve specific route information of one route.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `getTicketInfo`(ticketNumber int,ID int )
BEGIN
select T.company_name,D.departure_time,A.arrival_time,T.seating_position
FROM ticket AS T, departure_time AS D,arrival_time AS A
WHERE T.ticket_number=ticketNumber AND T.passenger_id=ID AND D.route_number=T.route_number AND A.route_number=T.ro
END
```

Retrieve the ticket information bought by one passenger.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `getTrainsFromCompany`(company_name varchar(45))
BEGIN
SELECT *
FROM Train AS T
Where T.company_name=company_name;

END
```

Retrieve the listed information of all trains run by one company.

API documentation: https://documenter.getpostman.com/view/10825978/SzezaWaB?version=latest#9d6f79f6-a939-4364-9c33-7f1c12589f57

The description and a sample request are included for each postman request(10 in total).

Video demonstration: https://youtu.be/q2l63fVnC2g

References:

Alam, Sarfaraj. “Top 18 Database Projects Ideas for Students” LovelyCoding, https://cs.wmich.edu/gupta/teaching/cs4430/cs4430SummII19web/lectureNotesCS4430/Top%2018%20Database%20Projects%20Ideas%20for%20Students%20Lovelycoding_org.pdf.

Gupta, Vishal. “RAILWAY RESERVATION DATABASE SYSTEM.” *Academia.edu - Share Research*, www.academia.edu/10048716/RAILWAY_RESERVATION_DATABASE_SYSTEM.

Reddy, Shashank. “Railway Management System, Database Mini Project.” *LinkedIn SlideShare*, 2 Jan. 2018, www.slideshare.net/shashankkarnati/railway-management-system-database-mini-project.

Sarhan, Abed. “ProjectTemp” University of Calgary, https://d2l.ucalgary.ca/d2l/le/content/295209/viewContent/3949132/View

“CPSC 471” Messenger Group, https://www.messenger.com

“Train Management.” *Train Management*, 31 Jan. 2020, www.psitrans.de/en/solutions/train-management/.

“VIA Rail Canada.” *VIA Rail*, www.viarail.ca/en.