**Assumptions / Business Rules**

Line Table:

Miles before will be populated when the row is created, but miles after will be null until the vehicle is returned

Vehicle can only be returned to the location in which it was rented from.

A line contains one and only one vehicle.

A vehicle can be in zero or many lines.

A line includes zero or many damages.

A damage can be in zero or many lines.

Vehicle Table:

A car that is rented must be returned to the location from which it was rented.

A vehicle belongs to one and only one rental class.

A rental class belongs to zero or many vehicles.

A vehicle can be bought from one and only one manufacturer.

A manufacturer can sell one or many vehicles to us.

A vehicle has one and only one location.

A location can have zero or many vehicles.

When a vehicles is returned, the number that is in MILES\_AFTER will become the new mileage for that vehicle.

Invoice Table:

An invoice is made by one and only one location.

A location has zero or many invoices.

An invoice contains one or many lines.

A line belongs to one and only one invoice.

An invoice is assigned to one and only one customer.

A customer can have zero or many invoices.

Employee

A employee works at zero or one locations.

A location has zero or many employees.

There is a certain amount that you pay upfront when you first rent the vehicle. This is the BASE\_PRICE from the RENTAL\_CLASS table. When the vehicle is returned, you pay for the amount of days rented, miles driven and for any damages that might have been done.

If a customer does not return the vehicle within a certain day, they are charged for the next day no matter how far into the next day it is.

The days are based on when the customer checks out the vehicle.

When a vehicle is returned, all money that is due must be payed right then and there.

**Normalization Steps**

Our group did not really have too many normalization steps as we attempted to design the database already in third normal form. However, as we worked on what we thought was the finished ERD more and more, we realized that there were many discrepancies with our thought process.

One such example is when somebody maybe wanted to rent more than one vehicle at a time. Initially they would have had to make separate purchases for each vehicle. We found a way around this by creating a line table that has a relationship with invoice. The invoice table acts as a sort of receipt, while the line table contains the line of that receipt. This allows for customers to rent more than one vehicle at a time.

Another problem we ran in to was when a vehicle happened to come back with multiple things wrong with it. In this case we created a bridge table between line and invoice to account for when a vehicle is returned with multiple different damages.

**SQL Statement to get Total Revenue for each Location**

-- Make sure we have the correct database selected

USE Timmy\_Car\_Rental;

GO

-- Create CTEs so that we do not grab info we will not use

;WITH BasePrices AS (

SELECT CLASS\_ID, BASE\_PRICE, COST\_PER\_DAY, COST\_PER\_MILE

FROM RENTAL\_CLASS),

Vehicles AS (

SELECT VEHICLE\_VIN, CLASS\_ID

FROM VEHICLE),

Lines AS (

SELECT LINE\_ID, INVOICE\_ID, RENT\_DATE, RETURN\_DATE, MILES\_BEFORE, MILES\_AFTER,

VEHICLE\_VIN

FROM LINE),

Invoices AS (

SELECT INVOICE\_ID, LOCATION\_ID

FROM INVOICE),

Locations AS (

SELECT LOCATION\_ID, LOCATION\_CITY, LOCATION\_STATE

FROM LOCATION),

Damages AS (

-- Grab the location ID and the total revenue it has reveived from damages

SELECT I.LOCATION\_ID, SUM(DMG\_COST) AS REVENUE\_FROM\_DAMAGES

FROM INVOICE AS I

LEFT OUTER JOIN LINE AS L

ON I.INVOICE\_ID = L.INVOICE\_ID

LEFT OUTER JOIN DAMAGE\_LIST AS DL

ON DL.LINE\_ID = L.LINE\_ID

LEFT OUTER JOIN DAMAGE AS D

ON D.DMG\_ID = DL.DMG\_ID

GROUP BY I.LOCATION\_ID)

SELECT I.LOCATION\_ID,

LO.LOCATION\_CITY,

LO.LOCATION\_STATE,

-- Sum up all the money that each location has got from all sources

SUM((DATEDIFF(DAY, RENT\_DATE, RETURN\_DATE) \* COST\_PER\_DAY) +

((MILES\_AFTER - MILES\_BEFORE) \* COST\_PER\_MILE) +

REVENUE\_FROM\_DAMAGES +

BASE\_PRICE) AS TOTAL\_REVENUE

FROM Invoices AS I

INNER JOIN Lines AS L ON L.INVOICE\_ID = I.INVOICE\_ID

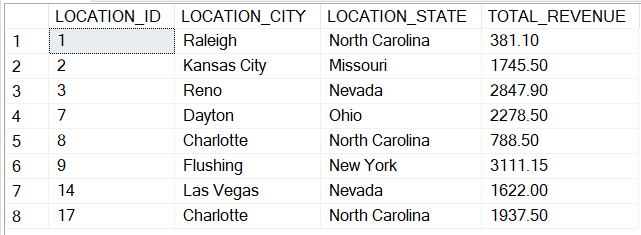
INNER JOIN Vehicles AS V ON L.VEHICLE\_VIN LIKE V.VEHICLE\_VIN

INNER JOIN BasePrices AS B ON B.CLASS\_ID = V.CLASS\_ID

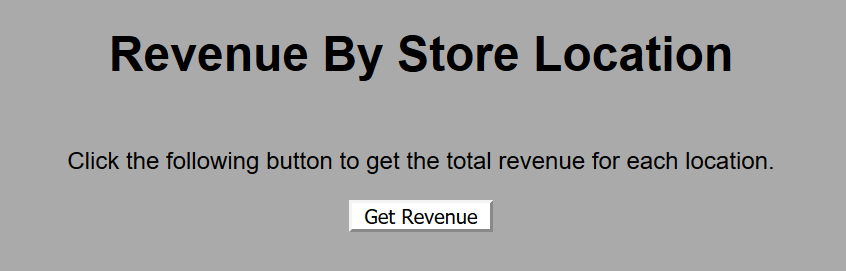
INNER JOIN Locations AS LO ON LO.LOCATION\_ID = I.LOCATION\_ID

INNER JOIN Damages AS D ON D.LOCATION\_ID = I.LOCATION\_ID

GROUP BY I.LOCATION\_ID, LO.LOCATION\_CITY, LO.LOCATION\_STATE;



**MySQL / PHP Web Form**



Clicking the button “Get Revenue” will output the below php form

