# Public Transport & Disaster Database

## Introduction

This reflection journey presents how I built the ER diagram and database and what I learned from this assignment.

A database is a data structure that stores organized information. Users use it to integrate, store, update and analyze information. These parts are about the application level. Although building an ER diagram and database structure are in the initial levels, it also plays an essential role as the foundation, which determines what information the database have and how to do further analysis. In this case, use their intrinsic relationship as Foreign Key to connect bus routes, bus stop, city area and disaster.

## Create an ER Diagram

An entity relationship diagram shows the relationships of entity sets stored in a database. An entity in this context is a component of data. In other words, ER diagrams illustrate the logical structure of databases. Before set up the database, I build the ER diagram that is a graphical representation of data requirements for a database.

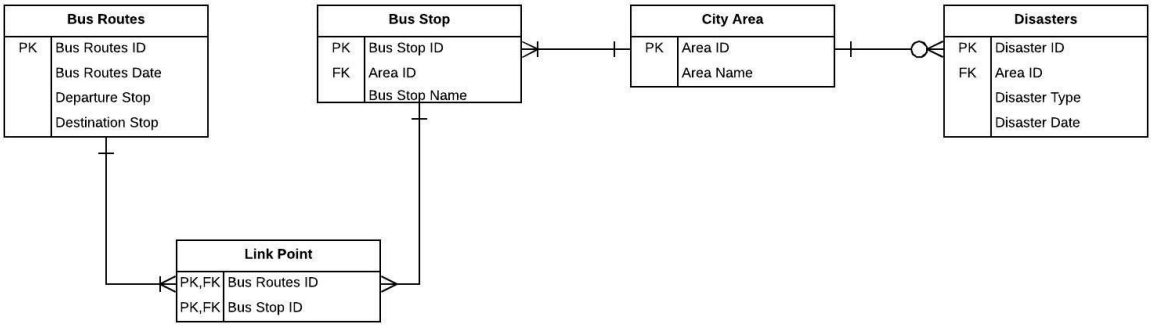
1. **Relationship**

   First, the basic entities are Bus Routes and Disaster. Disaster happens in some city area, so Area is an attribute as a foreign key in disaster table. An area could have a disaster or not, so it is the “zero to many” relationship.

   Meanwhile, every bus route has several bus stops, and some bus stops connect different bus routes, so they are “many to many”relationship.

   Every city area has several Bus Stops that consist of the specific Bus Routes.

   Finally, in my data modelling, it contains 5 entities - Bus Routes, Bus Stop, Link Point, City Area and Disaster.




### 2. Entities and Attributes

- **Bus Routes:** Bus Routes ID is the primary key to identify the unique bus routes. Bus routes run on the regular date, and the date here will connect to the disaster date to make sure whether the disaster impact on this specific route. Meanwhile, Bus Routes also have Departure Bus Stop and Destination Bus Stop as attributes.
- **Bus Stop:** Contain Bus Stop ID as Primary Key, Area ID as Foreign Key, and Bus Stop Name.
- **City Area**: Area ID is the primary key and Area name.
- **Link Point:** It is a linking table to connect Bus Routes and Bus Stop tables by their primary key as the PK and FK in Linking table
- **Disasters**: Disaster ID is the primary key, and disaster happens in the specific area, so it contains Area ID as FK. Then bus stop connects each area. So could  Additional, Disaster Type and Disaster Date are the attributes.

## Build Database

First I create tables in the DB browser, set the Primary Key and Foreign Key to connect each table. Then select data from the Metlink website, use INSERT command to input the data piece by piece. Noticed that need to put the data in the table that don't contain PK and FK first. Because of the FK restriction, can't insert the data into the linking table directly.
Also, used UPDATE and DELETE commands to revise the database.

## Write Queries

### 1. All routes that have ever been affected

Link the Bus Routes table with Disaster table by LinkPoint tables. So use JOIN... ON… to connect multi-tables. Meanwhile, add condition WHERE to make sure the disaster date is equal to bus routes date.

**Query:**

```
SELECT DISTINCT BusRoutes.BusRoutesID,Disaster.DisasterType
FROM BusRoutes
JOIN LinkPoint, BusStop,Disaster
ON BusRoutes.BusRoutesID=LinkPoint.BusRoutesID
AND BusStop.BusStopID=LinkPoint.BusStopID
WHERE BusRoutes.BusRoutesDate=Disaster.DisasterDate
```

**Result of Query:**

| | BusRoutesID | DisasterType |
|---|---|---|
| 1 | 1 | Earthquakes |
| 2 | 3 | Tsunami |
| 3 | 4 | Flooding |
| 4 | 5 | Earthquakes |
| 5 | 5 | Landslide |
| 6 | 6 | Landslide |
| 7 | 7 | Earthquakes |

## 2. All routes that have never been affected

Select all bus routes which don't show in the disaster-related table (Join Disaster table with Bus Stop table by linking table). The query will be used is the WHERE... NOT IN…
In order to make it more clear, added the DISTINCT command as well.

**Query:**

```
SELECT DISTINCT BusRoutes.BusRoutesID
FROM LinkPoint JOIN BusRoutes USING (BusRoutesID)
WHERE LinkPoint.BusRoutesID NOT IN
(SELECT DISTINCT LinkPoint.BusRoutesID
FROM LinkPoint JOIN Disaster,BusRoutes,BusStop
ON LinkPoint.BusRoutesID=BusRoutes.BusRoutesID
AND LinkPoint.BusStopID=BusStop.BusStopID
```

AND Disaster.AreaID=BusStop.AreaID)

**Result of Query:**

| | BusRoutesID |
|---|---|
| 1 | 2 |

3. **The 2 routes that have been affected the most (Just have 7 bus routes in the database)**
   Firstly, connect the Bus Routes table with the Disaster table by linking table, and group by the Bus Routes ID. Then count how many times of disaster for each affected bus routes. Finally, sort by the count in descending order and choose the top 2.

   The difficult part of this query is using GROUP BY function.

   **Query:**

```
SELECT COUNT(BusRoutes.BusRoutesID) AS 'Num of Disaster',BusRoutes.BusRoutesID
FROM BusRoutes
JOIN LinkPoint, Disaster,BusStop
ON BusRoutes.BusRoutesID=LinkPoint.BusRoutesID
AND BusStop.AreaID=Disaster.AreaID
AND LinkPoint.BusStopID=BusStop.BusStopID
WHERE BusRoutes.BusRoutesDate=Disaster.DisasterDate
GROUP BY BusRoutes.BusRoutesID
ORDER BY COUNT(*) DESC
LIMIT  2;
```

   **Result of Query:**

| | Num of Disaster | BusRoutesID |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 2 | 5 |

4. **The 3 routes that have been affected the least (Just have 7 bus routes in the database)**
   The same logic as the third query. But at end of the query, using the ascending order to sort the count of disaster in the affected bus routes. Then choose the top 3 in this order which are the routes have been affected the least

**Query:**

```
SELECT COUNT(BusRoutes.BusRoutesID) AS 'Num of Disaster',BusRoutes.BusRoutesID
FROM BusRoutes
JOIN LinkPoint, Disaster,BusStop
ON BusRoutes.BusRoutesID=LinkPoint.BusRoutesID
AND BusStop.AreaID=Disaster.AreaID
AND LinkPoint.BusStopID=BusStop.BusStopID
WHERE BusRoutes.BusRoutesDate=Disaster.DisasterDate
GROUP BY BusRoutes.BusRoutesID
ORDER BY COUNT(*) ASC
LIMIT  3;
```

**Result of Query:**

| | Num of Disaster | BusRoutesID |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1 | 4 |
| 3 | 1 | 6 |

5. **All routes have been affected in the last 5 years**

This query needs the DATE function when setting the condition.

The difficult part is figuring out how to set the DATE type and its format in SQLite.

First, SQLite Date function has a slight difference from other SQL. Secondly, the date format in the table should be “YYYY-MM-DD”.

Finally,  there are different DATE data types could be used in SQLite  - TEXT, INTEGER and REAL for storing the date and time values. Each value stored in an SQLite database has one of the following storage classes. I used INTEGER, so the following query is using “strftime”

**Query:**

```
SELECT DISTINCT BusRoutes.BusRoutesID
FROM BusRoutes
JOIN LinkPoint, BusStop,Disaster
ON BusRoutes.BusRoutesID=LinkPoint.BusRoutesID
AND BusStop.AreaID=Disaster.AreaID
AND LinkPoint.BusStopID=BusStop.BusStopID
WHERE strftime('%Y','now') - BusRoutes.BusRoutesDate<=5
```

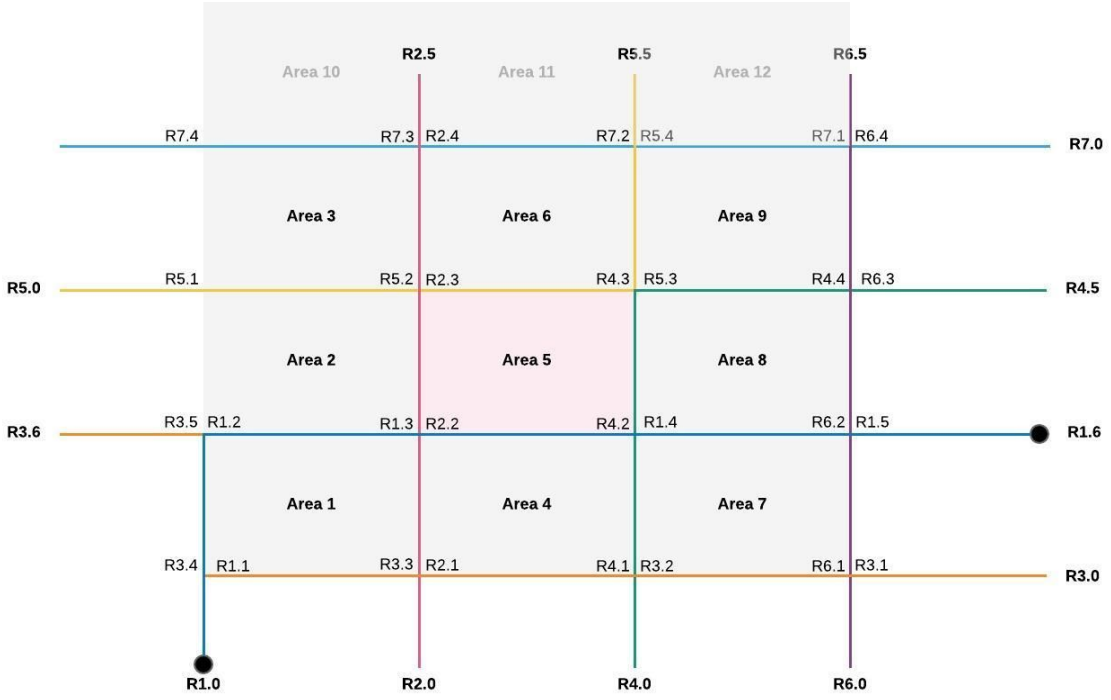
**Result of Query:**

| | BusRoutesID |
|---|---|
| 1 | 3 |
| 2 | 4 |
| 3 | 5 |
| 4 | 6 |
| 5 | 7 |

## Alternative Route

How to recover the public transportation after a disaster is about to design the alternative routes.

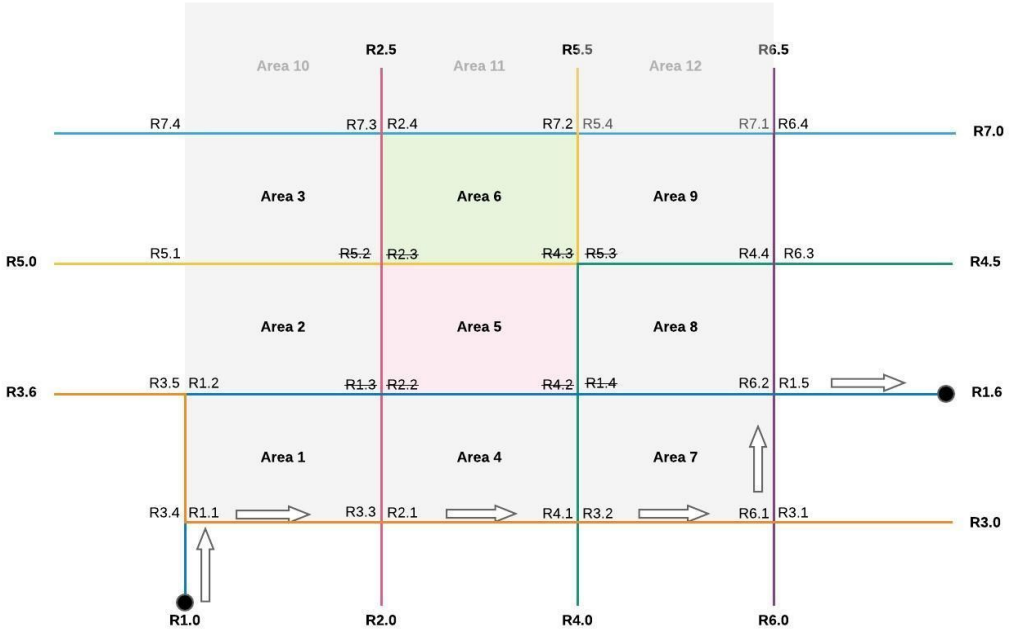
In my ER diagram, the relationship is Disaster → Area → Stop → Route. So when a disaster happens, select area and bus stop which have been affected. To seek the alternative routes, So I draw a 2D map to describe this effect.



Let's see if I want to travel from R1.0 to R1.6.

After the disaster happened in area 5, bus stop R1.4 in Route 1, R2.2 and R2.3 in Route 2, R4.2 and R4.3 in Route 4, R5.2 in Route 5 are been impacted, which means can't be used. The available routes are R3, R6 and R7.

Next step, we need to find route 1,3,6,7 whether can be connected by the transfer stops. The bus stops R3.4/R1.1, R6.1/R3.1 and R6.2/R1.5 could take the passenger from R1.0 to R1,6.




First, SELECT the bus routes and bus stops which have not been impacted.

Then need to find out whether have available transfer stops. So using SELECT, GROUP BY and COUNT to filter transfer bus stops which are connected at least 2 routes.

Finally, These available transfer bus stops have to exist in the available bus routes.Need to use WHERE...IN... command.

In summary, if want to find out whether has alternative routes. Need to match with following situation:

1. There are at least two available bus routes
2. There are available transfer bus stops connect these available routes
3. Last but not least, The available bus routes shared the same transfer stops with the affected routes which contain the departure and destination bus stops.

According to the current database and the relationship among tables, it could help to find out whether to find an alternative route when it can meet the requirements as above.

## Summary

Since start to build the ER diagram, need to keep asking myself that do this analysis/query need this data/information, then restructure the ER diagram. The process of normalization helps to reduce data redundancy.

From filling data into the database to write queries, I applied all the SQL commands I learned in the lectures. Additionally, I searched and learned some new commands in the online community, such as *Group By* and *Date data type*. Therefore, I think the key point of writing queries is not only about how many SQL commands you master (certainly as many as possible) but also it is about the logic and critical thinking. I tried many times to pull out the data from the database and failed many times as well. Then I figure it out that I need to fully understand my database structure first,  disassemble the queries into pieces, then assemble these pieces of information in a logic way.

Logic and critical thinking are involved in the whole process. Meanwhile, a sense of accomplishment achieved by writing queries makes me want to learn more SQL and data analysis.