Introduction to Databases — Tsinghua University, R.O.C Taiwan
Prof. 陳宜欣 (Yi-Shin Chen) — Spring 2017
CS4710 — 2017.06.19

Final Project Report:
Traffic Accident Public Warning System
Group 2
Team 2-1 (MOI)

I. GROUP MEMBERS:

張展榕 102062307	朱凰華 102062314	王祥至 102062131
Erik Söderberg X1050003	蕭丞佑 104061133	Eva Arévalo(艾怡華)
林佳億 103062306	胡品捷 9962224	103062181

II. ER MODEL:

We separated all locations from the places (accident location, police station location, health center location) to facilitate distance computation. A Higher entity relationship Happened_at Links Locations, Accident Events and information on the units assigned to it (Assigned_to) together. We store Police/Health center id and the id of the location tuple (Longitude, Latitude) inside the Health_loc, Police_loc higher entity relations.

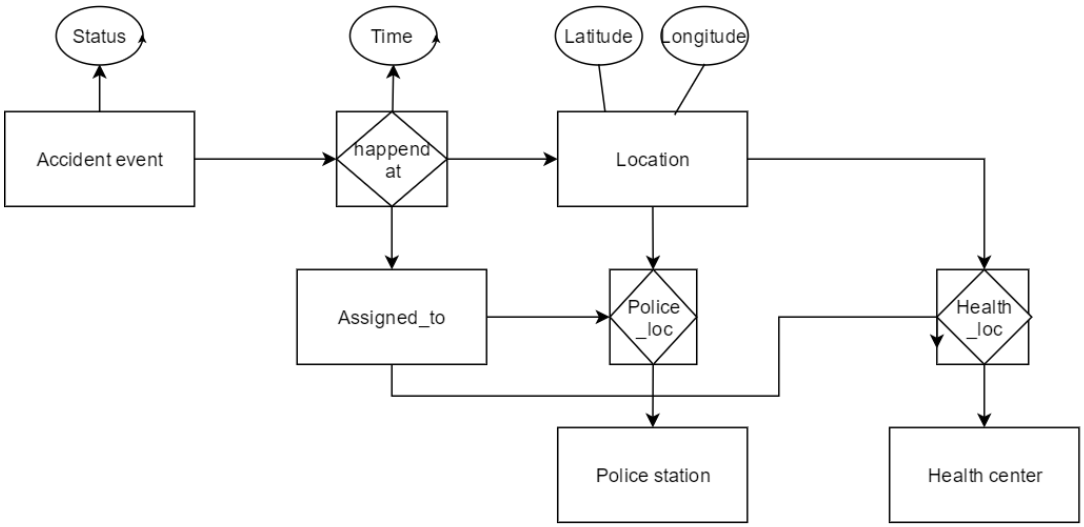

Figure 1. ER model

III. TABLE SCHEMAS:

We developed our schemas based on our ER Models. We used a table for each entity (including higher entity relationships). Primary Keys are underlined in the schemas below, they enforce Entity Integrity Constraints.

Relations are represented using Foreign Key attributes. They enforce the Referential Integrity Constraints.

Some extra attributes were added to facilitate queries, notably id attributes in certain tables.

Table: Health_Center

Name	Health_center_id	Health_center_name	Region	Zip_code	Address	Phone_number	URL
Dom	Int	Varchar(40)	Varchar(40)	Int	Varchar(40)	Int	Varchar(40)

Table: Health_Center_Located_At

Name	Locate_at_id	Health_center_id	Location_id
Domain	Int	Int	Int
F. key		Foreign key: Health_center_id	Foreign key: Location_id

Table: Police_Station

Name	Police_station_id	Chinese_name_of_the_center	English_name_of_the_center	Zip_code	Address	Phone_number
Dom	Int	Varchar(40)	Varchar(40)	Int	Varchar(40)	Int

Table: Police_Station_Located_At

Name	Located_at_id	Police_station_id	Location_id
Domain	Int	Int	Int
F. key		Foreign key: Police_station_id	Foreign key: Location_id

Table: Assigned_to

Name	Respone_id	Happened_at_id	Response_police_stations	Reponse_health_stations
Dom	Int	Int	Int	Int
F. key		Foreign key: Happened_At	Foreign key: Police_Station_Locate_At	Foreign key: Health_Center_Locate_At

Table: Happened_at

Name	Happened_at_id	Accident_event_id	Location_id
Domain	Int	Int	Int
F. key		Foreign Key: Accident_event_id	Foreign key: Location_id

Table: Detect_at

Name	Detect_at_id	Accident_event_id	Location_id
Domain	Int	Int	Int
		Foreign Key: Accident_event_id	Foreign key: Location_id

Table: Accident_Event

Name	Accident_id	Accident_status	Item_NO	Road_id	Road_type	Road_section_name	Road_direction	Milage
Dom	Int	Varchar(40)	Int	Int	Varchar(40)	Varchar(40)	Varchar(40)	Int

Table: Location

Name	Location_id	Latitude	Longitude
Domain	Int	Float	Float

View: Accident_Process_Report

Name	Accident_id	Accident_status
Domain	Int	Varchar(40)
	From table Accident Event	From table Accident Event

IV. TABLES SQL DEFINITION:

We used “serial” as the type of id attributes in some tables to set auto-incrementing of the id values. We used the “references” to set up foreign keys. Below is a list of the SQL definitions for the tables.

```
CREATE TABLE Accident_Event(Accident_id int Primary Key, Accident_status varchar(40),
    Item_NO text, Road_id int, Road_type varchar(40), Road_section_name varchar(40),
    Road_direction varchar(40), Milage int)
CREATE TABLE Location (Location_id serial Primary Key, Latitude Float, Longitude Float)
CREATE TABLE Happened_at (Happened_at_id serial Primary Key,
    Accident_event_id int REFERENCES Accident_Event(Accident_id),
    Location_id int REFERENCES Location(Location_id))
CREATE TABLE Detected_at (Detected_at_id serial Primary Key,
    Accident_event_id int REFERENCES Accident_Event(Accident_id),
    Location_id int REFERENCES Location(Location_id))
CREATE TABLE Health_Center (Health_center_id serial Primary Key,
    Health_center_name varchar(40), Region varchar(40), Zip_code int, Address varchar(40),
    Phone_number varchar(40), URL varchar(40))
CREATE TABLE Health_Center_Located_at (Located_at_id serial,
    Health_center_id int Primary Key REFERENCES Health_Center(Health_center_id),
    Location_id int REFERENCES Location(Location_id))
CREATE TABLE Police_station (Police_station_id serial Primary Key,
    Chinese_name_of_the_center varchar(40), English_name_of_the_center varchar(40),
    Zip_code int, Address varchar(40), Phone_number varchar(40))
CREATE TABLE Police_Station_Located_at (Located_at_id serial,
    Police_station_id serial Primary Key REFERENCES Police_Station(Police_Station_id),
    Location_id int REFERENCES Location(Location_id))
CREATE TABLE Assigned_to (Response_id serial Primary Key,
    Happened_at_id int REFERENCES Happened_at(Happened_at_id),
    Response_police_stations int REFERENCES Police_Station_Located_At(Police_station_id),
    Response_health_stations int REFERENCES Health_Center_Located_At(Health_center_id))
CREATE VIEW Accident_Process_Report as SELECT Accident_id, Accident_status FROM Accident_Event
```

Figure 2. Table Creation in SQL

V. DATA IMPORT:

We needed to populate the Health Center and Police Station Tables with info from the webpages of the Ministry Of Interior (MOI) webpage. After extracting the info from their webpages, we wrote a script in Python using the psycopg2 package, a PostgreSQL adapter for Python. We read the csv files in utf-8 encoding and perform the following steps:

1. **Establish connection:** With our database, db3 in the PostgreSQL database
2. **Table Location:** Insert new location
3. **Table Police Station/Health Center:** Insert new Police Station/Health Center
4. **Tables Health_center_located_at and Police_station_located_at:** Insert new Police Station/Health Center entry
5. **Commit the transaction:** To permanently save all changes

In order to do the 3rd step, we needed to save the *Location_id* and *Police_station_id* or *Health_center_id* of the new entries created in steps 1 and 2. This was accomplished by adding a "RETURNING " statement to the insertions and using psycospg2 fetchone() method on the connection element, which retrieves the result to the most recent query made.

VI. CONNECTION WITH TC DATABASE AND NEW WARNING DETECTION:

For connecting with TC's database, we used a script in Python using the psycopg2 package, a PostgreSQL adapter for Python. This scripts sets a connection object with our database db3 and another one for the db4 database to execute the view.

Whenever the script is running, it will detect new items inserted into the view *Accident Status Information* from TC's database.

After setting the connections, it continually extracts the view from the remote (TC's) database and compares the number of rows to the previous iteration.

If there is indeed a change, it will trigger the following steps:

1. **Accident Status Information View**: Execute a query that retrieves all items whose *accident_status* attribute is '*not clear'*. For each of the new uncleared accidents, we execute the following steps:
    i. **Table Location:** Insert new location for the accident
    ii. **Table Accident_Event:** Insert the info we got from the not clear tuple in the view to our own database
    iii. **Table Happened_at:** Insert a new Happened_at element that links a location to the Accident_id
2. **Commit:** Make all changes permanent

To get the result of an executed query we use psycopg2's fetchall() or fetchone() method on the connection objects. To get data from an insertion (like for example, the id of an automatic numbered new tuple), we use once again "RETURNING…" to return a value after an insertion.

VII. TRIGGERS:

We set 3 triggers originally. However, our current database doesn't use the 1st Trigger. We believe it is worth mentioning the work done.

Below are the definitions for each trigger in SQL:

```
CREATE TRIGGER NewWarning AFTER INSERT
    ON Accident Status Information FOR EACH ROW
    EXECUTE PROCEDURE CreateResponseEVENT();

CREATE OR REPLACE FUNCTION CreateResponseEVENT()
    RETURN TRIGGER AS
$$
BEGIN

INSERT INTO Location(Latitude, Longitude)
VALUES (NEW.sensor_latitude, NEW.sensor_longitude)
RETURNING Location_id INTO NewLocationId;

INSERT INTO Accident_Event(Accident_Status, Item_NO,
    Road_id, Road_type, Road_section_name, Road_direction, Milage)
VALUES ('not clear', NEW.Item_NO,NEW.Road_id,NEW.Road Direction,
    NEW.Road_Type, NEW.Road_Section_Name, NEW.Road Direction, NEW.Milage)
RETURNING Accident_id INTO NewAccidentId;

INSERT INTO Happened_at(Accident_event_id, Location_id)
VALUES(NewAccidentId, NewLocationId);

RETURN NEW;

END;
$$
LANGUAGE 'plpgsql';
```

Figure 3. SQL definition of 1st Trigger

```
CREATE  TRIGGER  detecting AFTER INSERT
    ON happened_at
FOR EACH ROW
    EXECUTE PROCEDURE InsertAccidentEVENT();

CREATE OR REPLACE FUNCTION InsertAccidentEVENT()
  RETURNS trigger AS
  $$
BEGIN
    INSERT INTO assigned_to(happened_at_id, response_health_stations)
    SELECT h.happened_at_id, hcl.located_at_id
    FROM happened_at as h, health_center_located_at as hcl, location as l1, location as l2
    WHERE NEW.happened_at_id = h.happened_at_id
    AND hcl.location_id = l1.location_id
    AND NEW.location_id = l2.location_id
    ORDER BY pow(l1.longitude-l2.longitude, 2) + pow(l1.latitude-l2.latitude, 2) ASC
    LIMIT 1;

    UPDATE assigned_to
    SET response_police_stations = (
            SELECT psl.located_at_id
            FROM happened_at as h, police_station_located_at as psl, location as l1, location as l2
            WHERE NEW.happened_at_id = h.happened_at_id
            AND psl.location_id = l1.location_id
            AND NEW.location_id = l2.location_id
            ORDER BY pow(l1.longitude-l2.longitude,2) + pow(l1.latitude-l2.latitude,2) ASC
            LIMIT 1
        )
    FROM happened_at as h1
    WHERE NEW.happened_at_id = h1.happened_at_id;

    RETURN NEW;

END;
  $$
LANGUAGE 'plpgsql';
```

Figure 4. SQL definition of 2nd Trigger

```
CREATE TRIGGER AccidentHandled AFTER INSERT
    ON Assigned_to FOR EACH ROW
    EXECUTE PROCEDURE ClearAccidentEVENT();

CREATE OR REPLACE FUNCTION ClearAccidentEVENT()
    RETURN TRIGGER AS
$$
BEGIN

UPDATE Accident_Event
SET Accident_Status = 'clear'
WHERE Accident_id = (SELECT Accident_event_id
  FROM Happened_at
  WHERE Happened_at_id = NEW.Happened_at_id);

RETURN NEW;

END;
$$
LANGUAGE 'plpgsql';
```

Figure 5. SQL definition of 3rd Trigger

The first trigger handled the creation of a new accident (and the corresponding *Location* and *Happened_at* items). However, it didn't take into consideration that triggers can't be set in remote views; and that for the Demo, we couldn't set Triggers on the other group's table. This is the reason why we don't use it, but it could theoretically be used if implemented on TC's database. A connection to our database should be added in the procedure though.

The second trigger is set on the table *Happened_at*. When a new event has been detected and created, we assign a Police Station and a Health Center to handle the accident based on the distance to the accident location. We get the distance using the Pythagorean theorem and the math function pow(xx,2) in PostgreSQL. The Police Stations and Health Centers are ordered by Ascending Distance and we select the top entry to handle the incident.

The third trigger automatically clears an accident when a Police Station and a Health Center have successfully been assigned to it. The trigger is set on the table *Assigned_to*. It triggers a query to *Happened_at_id* table using the assigned tuple's *Happened_at_id* attribute to get its *Accident_event_id* attribute. The accident Id is then used to update that entry and mark it as 'clear'.

VIII. MANUAL STEPS FROM DETECTION OF NEW WARNING:

Below is a list of the steps we take whenever we detect a new entry in the TC's view (this part is handled by a script, as explained above):

1. **Table Happened_at:** Query *Happened_at_id*
2. **Table Accident_Event:** Query *Accident_event_id*
3. **Table Accident_Event:** Update entry with mathching *Accident_event_id* : set *Accident_status* attribute as 'clear'.

IX. CHALLENGES AND PROBLEMS:

The biggest challenge were probably the triggers, especially the logic for the 2nd trigger, the one that assigns a Police Station and a Health Center to handle an accident based on distance.

We spent quite some time on the ER model and schema definition at the beginning but the rest of the project was smoother because we had already defined everything.

X. WORK ORGANIZATION:

1. Work Distribution:

ER model Design: 張展榕, Eva Arévalo, 朱凰華, Erik Söderberg
Table schema design: Erik Söderberg, 張展榕, 朱凰華, Eva Arévalo
Creating Tables: Erik Söderberg, Eva Arévalo
Automatic Accident warning detection script: 張展榕
Accident event creation: 張展榕,Eva Arévalo
Data import (Police Station, Health Center): 朱凰華, Eva Arévalo
Triggers: 張展榕,Eva Arévalo
Testing: 張展榕,Eva Arévalo
Report: Eva Arévalo

2. Meetings:

- *1st Meeting*:
  Attendees: 張展榕, Erik Söderberg, 蕭丞佑, 朱凰華, 王祥至, Eva Arévalo
  Topic: Project Specification reading, understanding specifications, planning
  Date: June 1st, 2017
- *2nd Meeting:*

Attendees: 張展榕, 朱凰華, Eva Arévalo

Topic: Discuss ER Model with Prof. Chen

Date: June 2nd, 2017

- *3rd Meeting:*

  Attendees: 張展榕, 朱凰華, Eva Arévalo

  Topic: Discuss ER Model with Prof. Chen

  Date: June 7th, 2017

- *4th Meeting:*

  Attendees: 張展榕, Erik Söderberg, 蕭丞佑, 朱凰華, Eva Arévalo

  Topic: Translating ER model into Tables, detecting modifications in other group's view

  Date: June 14th, 2017

- *5th Meeting:*

  Attendees: 張展榕, Eva Arévalo

  Topic: Data import, warning detection, Triggers

  Date: June 18th, 2017

- *6th Meeting:*

  Attendees: 張展榕, Eva Arévalo

  Topic: Testing, Linking database with TC group

  Date: June 18th, 2017

Note: Erik Söderberg went back to his home country on June 15th and wasn't able to attend the 5th and 6th meetings