UNIVERSITÉ LIBRE DE BRUXELLES

Rapport : Villo !

Pierre Gérard, Titouan Christophe

INFO-H-303 Base de données

Esteban Zimányi, Michaël Waumans

Année académique 2014 - 2015

Table des matières

1 Diagramme entité association

1.1 Diagramme

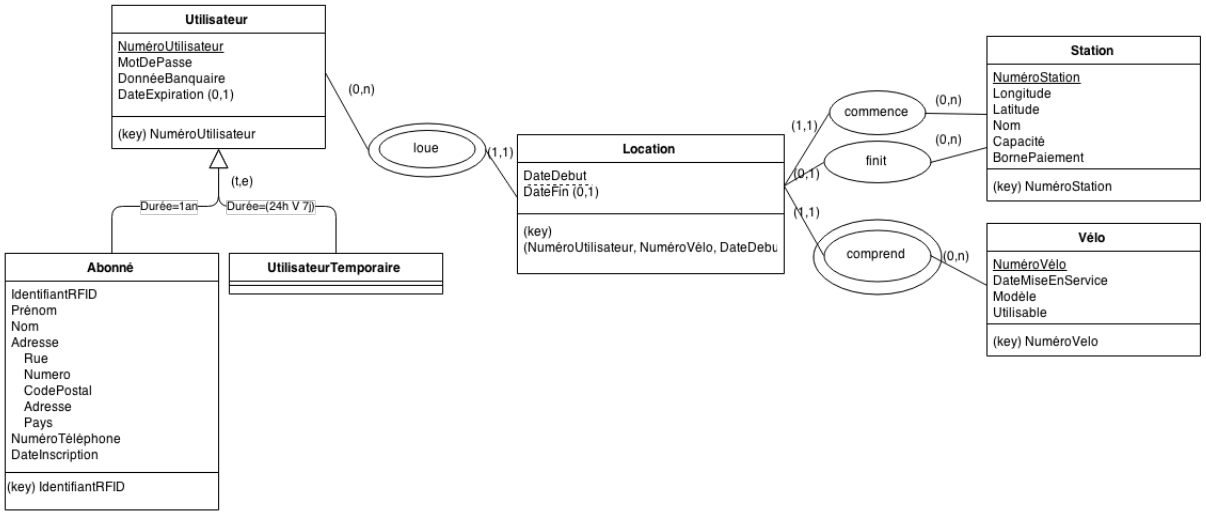

FIGURE 1 – Diagramme entité association

1.2 Contraintes

Les contraintes sont les suivantes :

— La DateDebut d'une Location doit précéder la DateFin,
— La DateMiseEnService d'un Vélo doit précéder la DateDebut de chacune de ses Locations,
— La DateExpiration d'un Utilisateur doit être postérieure à la DateDebut de toutes ses Locations,
— Le couple (Longitude, Latitude) est unique,
— La DateExpiration d'un Utilisateur doit être postérieure à la DateFin de toutes ses Locations,
— Un utilisateur qui a une date d'expiration non nul ne peut pas avoir plus d'une Location ayant une DateFin nul,
— Pour une Station, le nombre de vélo dont la dernière Location finit dans cette station ne doit pas dépasser sa capacité,
— Un Vélo ne peut pas avoir de déplacement disjoints. C'est a dire que la Station de départ du trajet n doit être similaire a la Station d'arrivé du trajet (n-1) pour n¿1,
— Un Utilisateur ayant une DateExpiration non nul ne peut pas prendre un Vélo si Usable est faux.

2 Modèle relationnel

2.1 Modèle

Utilisateur(NuméroUtilisateur, MotDePasse, DonnéeBanquaire, *DateExpiration*)

Abonné(NuméroUtilisateur, IdentifiantRFID, Nom, Rue, Numero, CodePostal, Adresse, Pays DateInscription, NuméroTélephone)

Abonné.NuméroUtilisateur référence Utilisateur.NuméroUtilisateur

Location(NuméroUtilisateur, NuméroVélo, DateDebut,*DateFin*, NuméroStationDépart , NuméroStationFin)

Location.NuméroUtilisateur référence Utilisateur.NuméroUtilisateur

Location.NuméroVélo référence Vélo.NuméroVélo

Location.NuméroStationDépart référence Station.NuméroStation

Location.NuméroStationFin référence Station.NuméroStation

(NuméroUtilisateur, DateDebut) est unique

(NuméroVélo, DateDebut) est unique

Station(NuméroStation, Longitude, Latitude, Nom, Capacité, BornePaiement)

Vélo(NuméroVélo, DateMiseEnService, Modèle, Utilisable)

2.2 Contraintes

Les contraintes sont les suivantes :

— Une Location a au plus une station d'arrivé,
— Une Location a une et une seule Station de départ,
— Une Location a un et un seul Utilisateur,
— Une Location a un et un seul Vélo,
— La DateDebut d'une Location doit précéder la DateFin,
— La DateMiseEnService d'un Vélo doit précéder la DateDebut de chacune de ses Locations,
— Le couple (Longitude, Latitude) est unique,
— La DateExpiration d'un Utilisateur doit être postérieure à la DateDebut de toutes ses Locations,
— Un utilisateur qui a une date d'expiration non nul ne peut pas avoir plus d'une Location ayant une DateFin nul,
— Pour une Station, le nombre de vélo dont la dernière Location finit dans cette station ne doit pas dépasser sa capacité,
— Un Vélo ne peut pas avoir de déplacement disjoints. C'est a dire que la Station de départ du trajet n doit être similaire a la Station d'arrivé du trajet (n-1) pour n¿1,
— Un Utilisateur ayant une DateExpiration non nul ne peut pas prendre un Vélo si Usable est faux.

3 Hypothèses sur le modèle

Il existe des utilisateur "admin", ce sont ces derniers et uniquement eux qui n'ont pas de date d'expiration.

Si un Abonné a son abonnement qui expire, il peut re-utiliser le NuméroUtilisateur et MotDePasse dans le futur, l'entité n'est pas supprimée.

Dans le cas ou des employés villo déplacent un vélo la nuit, alors ce déplacement doit être enregistré dans la base de donnée par un utilisateur admin.

Dans le cas ou un vélo serait cassé et devrait sortir du circuit de location, un Utilisateur admin vient le chercher et la Location ne finit jamais, c'est à dire pas de DateFin.

Dans le cas ou la société villo achète des nouveaux vélo et les met en circulation, un utilisateur admin fait une location de ce nouveau vélo qui a une date de départ égale à la date d'arrivée et une station de départ égale à la station d'arrivée.

Le champs MotDePasse contient un hash cryptographique du mot de passe et non le mot de passe lui-même.

4 Justification

Afin d'éviter la redondance et de garantir la cohérence du modèle, nous avons choisis de :

— Ne pas mettre d'attribut PlaceUtilisé dans Station,
— Ne pas mettre d'attribut Endroit dans Vélo,
— ect ...

En effet, ces informations peuvent être déduite de la suite de Location.

Pour obtenir une clé primaire à l'entité Location, nous avons rendu obligatoire le champs DateDebut et c'est pour cela que la mise en circulation et la mise a la retraite des vélos sont différentes.

Pour la généralisation nous avons choisis, une solution permettant d'avoir une relation a une seule table depuis la location et une solution permettant de mettre une contrainte d'existence sur tous les champs excepté la DateExpiration.

5 Requêtes

5.1 Les utilisateurs habitant Ixelles ayant utilisé un Villo de la station Flagey

5.1.1 SQL

```sql
-- Les utilisateurs habitant Ixelles ayant utilise un Villo de la station Flagey

SELECT DISTINCT subscriber.user_id,
                subscriber.firstname,
                subscriber.lastname
FROM   subscriber
       INNER JOIN trip
               ON subscriber.user_id = trip.user_id
       INNER JOIN station
               ON trip.departure_station_id = station.id
WHERE  station.NAME = "FLAGEY"
       AND subscriber.address_zipcode = 1050;
```

5.1.2 Algèbre relationnelle

$$\Pi_{subscriber.user\_id,subscriber.firstname,subscriber.lastname}(\sigma_{station.name="FLAGEY"ANDsubscriber.address\_zipcode=1050}(\tag{1}$$

$$subscriber \bowtie_{subscriber.user\_id=trip.user\_id} (trip \bowtie_{trip.departure\_station_id=station.id} station)) \tag{2}$$

5.1.3 Calcul relationnel

$$\{sb.user_id, sb.firstname, sb.lastname | subscriber(sb) \wedge trip(tp) \wedge station(st) \wedge tp.departure\_station\_id = st.id \tag{3}$$

$$\wedge sb.user\_id = tp.user\_id \wedge st.NAME = "FLAGEY" \wedge tp.departure\_station_id = st.id\} \tag{4}$$

5.2 Les utilisateurs ayant utilisé Villo au moins 2 fois

5.2.1 SQL

```sql
-- Les utilisateurs ayant utilise Villo au moins 2 fois

SELECT DISTINCT trip1.user_id
FROM   trip trip1
       INNER JOIN trip trip2
               ON trip1.user_id = trip2.user_id
                  AND trip1.departure_date != trip2.departure_date;
```

5.2.2 Algèbre relationnelle

$$\Pi_{trip1.user\_id}(trip1 \bowtie_{trip1.user\_id=trip.user\_idANDtrip1.departure\_date!=trip2.departure\_date} trip2)$$

5.2.3 Calcul relationnel

$$\{t1.user_id | trip(t2) \wedge trip(t1) \wedge trip1.user\_id = trip2.user\_id \wedge trip1.departure\_date != trip2.departure\_date\}$$

5.3 Les paires d'utilisateurs ayant fait un trajet identique

5.3.1 SQL

```sql
-- Les paires d'utilisateurs ayant fait un trajet identique

SELECT DISTINCT t1.user_id,
                t2.user_id
FROM   trip t1
       INNER JOIN trip t2
               ON t1.departure_station_id = t2.departure_station_id
                  AND t1.arrival_station_id = t2.arrival_station_id
                  AND t1.user_id < t2.user_id;
```

5.3.2 Algèbre relationnelle

$$\Pi_{t1.user\_id, t2.user\_id}(\tag{5}$$

$$t1 \bowtie_{t1.departure\_station_id = t2.departure\_station_id AND t1.arrival\_station_id = t2.arrival\_station\_id AND t1.user\_id < t2.user\_id;} t2) \tag{6}$$

5.3.3 Calcul relationnel

$$\{t1.user\_id, t2.user\_id | trip(t2) \wedge trip(t1) \wedge t1.departure\_station_id = t2.departure\_station_id \tag{7}$$

$$\wedge t1.arrival\_station_id = t2.arrival\_station_id \wedge t1.user\_id < t2.user\_id \tag{8}$$

$$\wedge trip1.departure\_date != trip2.departure\_date\} \tag{9}$$

5.4 Les vélos ayant deux trajets consécutifs disjoints (station de retour du premier trajet différente de la station de départ du suivant)

5.4.1 SQL

Voici, deux requêtes qui semblent prendre un temps équivalent à s'exécuter.

```sql
-- Les velos ayant deux trajets consecutifs disjoints (station de retour du
-- premier trajet differente de la station de depart du suivant)

SELECT DISTINCT bike_id FROM (
    SELECT t1.bike_id AS bike_id,
           t1.arrival_station_id AS s1,
           t2.departure_station_id AS s2,
           t1.arrival_date AS arrival,
           t2.departure_date AS departure
    FROM trip t1
    INNER JOIN trip t2 ON t1.bike_id=t2.bike_id AND t1.arrival_date<t2.departure_date
    WHERE t1.arrival_date NOT NULL --AND t1.bike_id=5
    GROUP BY t2.departure_date
    ORDER BY t1.departure_date,t2.departure_date)
WHERE s1 != s2;
```

```sql
-- Les velos ayant deux trajets consecutifs disjoints (station de retour du
-- premier trajet differente de la station de depart du suivant)

SELECT DISTINCT t1.bike_id
FROM   trip t1
       INNER JOIN trip t2
               ON t1.bike_id = t2.bike_id
       LEFT OUTER JOIN trip AS t3
              ON t1.bike_id = t3.bike_id
                 AND t1.arrival_date < t3.arrival_date
                 AND t3.arrival_date < t2.departure_date
WHERE  t1.arrival_station_id != t2.departure_station_id
       AND t1.arrival_date <= t2.departure_date
       AND t3.bike_id IS NULL;
```

5.4.2 Algèbre relationnelle

$$\Pi_{t1.bike\_id}(\sigma_{t1.arrival\_station_id!=t2.departure\_station_idANDt1.arrival\_date<=t2.departure\_dateANDt3.bike_idISNULL}(\tag{10}$$

$$(t1 \bowtie_{t1.bike_id=t2.bike_id} t2) \rtimes_{t1.bike\_id=t3.bike\_idANDt1.arrival\_date<t3.arrival\_dateANDt3.arrival_date<t2.departure\_date} t3)) \tag{11}$$

5.4.3 Calcul relationnel

5.5 Les utilisateurs, la date d'inscription, le nombre total de trajet effectués, la distance totale parcourue et la distance moyenne parcourue par trajet, classés en fonction de la distance totale parcourue

```sql
-- Les utilisateurs, la date d'inscription, le nombre total de trajet effectues,
-- la distance totale parcourue et la distance moyenne parcourue par trajet,
-- classes en fonction de la distance totale parcourue

-- SQLite ne dispose pas de fonction de la fonction sqrt(), ou de fonctions
-- trigonometriques, necessaires au calcul de la distance sur terre. Nous avons
-- donc implemente la formule de Haversine en tant qu'extension SQLite a
-- charger au demarrage, et qui integre la fonction
-- geodistance(lat1,long1,lat2,long2) -> km a l'environnement SQL.

SELECT   subscriber.firstname,
         subscriber.lastname,
         subscriber.entry_date,
         COUNT(trip.arrival_station_id) AS trip_count,
         SUM(geodistance(from_.latitude, from_.longitude, to_.latitude, to_.longitude))
    AS total_km,
         AVG(geodistance(from_.latitude, from_.longitude, to_.latitude, to_.longitude))
    AS avg_km
FROM     trip
INNER JOIN station AS from_ ON trip.departure_station_id = from_.id,
           station AS to_ ON trip.arrival_station_id = to_.id,
           subscriber ON trip.user_id = subscriber.user_id
WHERE    trip.arrival_station_id NOT null
GROUP BY subscriber.firstname,
         subscriber.lastname
ORDER BY count(trip.arrival_station_id);
```

5.6 Les stations avec le nombre total de vélos déposés dans cette station (un même vélo peut-être comptabilisé plusieurs fois) et le nombre d'utilisateurs différents ayant utilisé la station et ce pour toutes les stations ayant été utilisées au moins 10 fois.

```
-- Les stations avec le nombre total de velos deposes dans cette station
-- (un meme velo peut-etre comptabilise plusieurs fois) et le nombre
-- d'utilisateurs differents ayant utilise la station et ce pour toutes les
-- stations ayant ete utilisees au moins 10 fois.


SELECT station.NAME,
       Count(trip.arrival_station_id),
       Count(DISTINCT trip.user_id)
FROM   station
       INNER JOIN trip
               ON trip.arrival_station_id = station.id
GROUP  BY station.id
HAVING Count(trip.arrival_station_id) >= 10;
```

6 Script DDL de création de la base de données

```
PRAGMA foreign_keys = ON;

CREATE TABLE IF NOT EXISTS station (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    payment BOOLEAN NOT NULL,
    capacity INTEGER NOT NULL
        CHECK(capacity >= 0),
    latitude REAL NOT NULL
        CHECK(-90<=latitude AND latitude<=90),
    longitude REAL NOT NULL
        CHECK(-180<=longitude AND longitude<=180),
    name VARCHAR(32) NOT NULL,

    UNIQUE(latitude, longitude)
);

CREATE TABLE IF NOT EXISTS bike (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    entry_date VARCHAR(20) NOT NULL
        CHECK(entry_date IS strftime(entry_date)),
    model VARCHAR(32) NOT NULL,
    usable BOOLEAN NOT NULL
);

CREATE TABLE IF NOT EXISTS user (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    password VARCHAR(64) NOT NULL,
    card VARCHAR(64) NOT NULL,
    expire_date VARCHAR(20)
        CHECK(expire_date IS strftime(expire_date))
);

CREATE TABLE IF NOT EXISTS subscriber (
    user_id INTEGER PRIMARY KEY,
    rfid TEXT UNIQUE NOT NULL,
    firstname TEXT NOT NULL,
    lastname TEXT NOT NULL,
    address_street TEXT NOT NULL,
    address_streenumber INTEGER NOT NULL,
    address_zipcode INTEGER NOT NULL,
```

```
    address_city TEXT NOT NULL,
    address_country TEXT NOT NULL,
    entry_date VARCHAR(20) NOT NULL
        CHECK(entry_date IS strftime(entry_date)),
    phone_number VARCHAR(20),

    FOREIGN KEY(user_id) REFERENCES user(id)
);

CREATE TABLE IF NOT EXISTS trip (
    departure_station_id INTEGER NOT NULL,
    departure_date VARCHAR(20) NOT NULL
        CHECK (departure_date IS strftime(departure_date)),
    arrival_station_id INTEGER,
    arrival_date VARCHAR(20)
        CHECK(arrival_date IS strftime(arrival_date))
        CHECK(arrival_date >= departure_date),
    user_id INTEGER NOT NULL,
    bike_id INTEGER NOT NULL,

    PRIMARY KEY(user_id, bike_id, departure_date),

    FOREIGN KEY(departure_station_id) REFERENCES station(id),
    FOREIGN KEY(arrival_station_id) REFERENCES station(id),
    FOREIGN KEY(user_id) REFERENCES user(id),
    FOREIGN KEY(bike_id) REFERENCES bike(id)
);
```