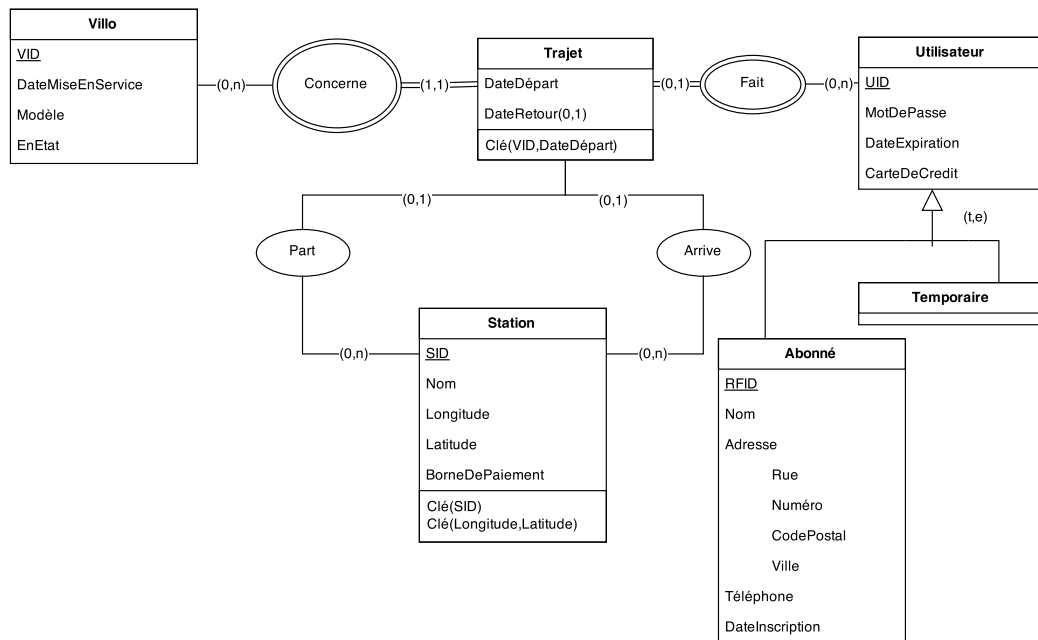


# Projet INFO-H-303 : Villo!

Hereman Nicolas, Van Brande Rodrigue

24 avril 2015

# Diagramme entité-association



## Contraintes et hypothèses

- La *DateExpiration* d'un *Abonné* doit être strictement supérieure à sa *DateInscription*
- La *DateDépart* d'un *Trajet* doit être strictement supérieure à la *DateInscription* de l'*Abonné* qui le fait.
- La *DateDépart* d'un *Trajet* pot être strictement inférieure à la *DateExpiration* de l'*Utilisateur*.
- La *DateDépart* d'un *Trajet* doit être strictement inférieure à la *DateExpiration* de l'*Utilisateur* qui le fait.
- La *DateRetour* d'un *Trajet* doit être strictement supérieure à la *DateDépart* de ce même *Trajet*.
- La *DateDépart* d'un *Trajet* doit être strictement supérieure à la *DateMiseEnService* du *Villo* concerné.
- A l'instant *DateRetour*, la *Station* de retour d'un *Trajet* doit contenir moins de *Villos* que sa *Capacité*. Le calcul du nombre de *Villos* présents dans une *Station* à l'aide de l'entité *Trajet*.
- Un même *Villo* ne peut pas concerner deux *Trajets* en même temps.
- Un même *Utilisateur* ne peut pas faire deux *Trajets* en même temps.
- Un *Trajet* *i* qui suit directement un autre *Trajet* *j* pour un même *Villo* doit avoir la même *Station* de départ que la *Station* d'arrivée du *Trajet* *j*.
- Si un *Trajet* n'a pas d'*Utilisateur*, il n'a pas de *Station* de départ et inversement.
- Si un *Trajet* n'a ni *Utilisateur* ni *Station* de départ, sa *DateDépart* est 0000/00/00 - 00:00:00
- Pour chaque *Villo*, il n'y a qu'un *Trajet* sans *Utilisateur* et *Station* de départ. Ni plus ni moins.
- Si un *Trajet* n'a pas de *Station* de retour, il n'a pas de *DateRetour* et inversement.
- Pour chaque *Villo*, il y a au maximum un *Trajet* sans *DateRetour* et *Station* de retour.

- Si un *Trajet* n'a pas de *DateDépart*, il a une *DateRetour*.
- Si un *Trajet* n'a pas de *DateRetour*, il a une *DateDépart*.

## Traduction relationnelle

- Utilisateur(UID, MotDePasse, DateExpiration, CarteDeCredit)
- Abonne( UID, RFID, Nom, Rue, Numéro, CodePostal, Ville, Téléphone, DateInscription)
  - UID référence Utilisateur.UID
- Station(SID, Nom, Longitude, Latitude, Capacité, BorneDePaiement)
- Villo(VID, DateMiseEnService, Modèle, EnEtat)
- Trajet(VID, DateDépart, *UID*, *StationDépart*, *DateRetour*, *StationRetour*)
  - UID référence Utilisateur.UID
  - VID référence Villo.VID
  - StationDépart référence Station.SID
  - StationRetour référence Station.SID

## Remarque

Un Utilisateur.UID n'existant pas dans Abonné.UID est un utilisateur temporaire

## Justification et hypothèses de modélisation

On utilise une table *Utilisateur* pour stocker leurs données. On a créé une table *Abonné* afin des les différencier des utilisateurs *Temporaire*. Comme l'héritage des table est totale et exclusive et que toutes les informations dont on a besoin pour ces derniers sont déjà dans la table *Utilisateur*, ils n'ont pas besoin d'une table pour eux. Les utilisateurs temporaires seront ceux qui n'ont pas leurs *UID* dans la table *Abonné*.

Les *Stations* et *Villos* ont aussi droit à leur table pour qu'on y sauvegarde leurs données.

On sauvegarde aussi la liste des *Trajets* dans une table. Le fait que l'*Utilisateur* et la *StationDépart* soit optionnel peut paraître illogique par rapport à la réalité. Mais cela se justifie par le fait que les *Villo* doivent être placés une première fois. Comme on ne sauvegarde pas la *Station* dans laquelle ils sont stockés, on se repère au *Trajet* pour le savoir. On les place donc à l'aide de *Trajet* sans *Station* de départ et sans *Utilisateur*. Les *Trajets* sans *Station* de retour et *DateRetour* sont les *Trajets* encore en cours.

## Script SQL DDL de création de base de données

```

1 CREATE TABLE 'Utilisateur' (
2     'UID' int unsigned NOT NULL,
3     'MotDePasse' smallint(4) unsigned zerofill NOT NULL,
4     'CarteDeCredit' bigint(16) unsigned zerofill NOT NULL,
5     'DateExpiration' datetime NOT NULL,
6     PRIMARY KEY ('UID')
7 );
8
9 CREATE TABLE 'Villo' (
10     'VID' smallint unsigned NOT NULL,
11     'DateMiseEnService' datetime NOT NULL,
12     'Modèle' varchar(12) NOT NULL,
13     'EnEtat' tinyint(1) NOT NULL,
14     PRIMARY KEY ('VID')
15 );
16
17 CREATE TABLE 'Abonné' (
18     'UID' int unsigned NOT NULL,
19     'RFID' char(20) NOT NULL,
20     'Nom' varchar(50) NOT NULL,
21     'Rue' varchar(100) NOT NULL,
22     'Numéro' smallint unsigned NOT NULL,
23     'CodePostal' smallint(4) unsigned NOT NULL,
24     'Ville' varchar(50) NOT NULL,
25     'Téléphone' char(10) NOT NULL,
26     'DateInscription' datetime NOT NULL,
27     PRIMARY KEY ('RFID'),
28     FOREIGN KEY ('UID') REFERENCES Utilisateur('UID')
29 );
30

```

```

31 CREATE TABLE 'Station' (
32     'SID' smallint unsigned NOT NULL,
33     'Nom' varchar(50) NOT NULL,
34     'Longitude' float NOT NULL,
35     'Latitude' float NOT NULL,
36     'Capacité' tinyint unsigned NOT NULL,
37     'BorneDePaiement' tinyint(1) NOT NULL,
38     PRIMARY KEY ('SID', 'Longitude', 'Latitude')
39 );
40
41 CREATE TABLE 'Trajet' (
42     'VID' smallint unsigned NOT NULL,
43     'DateDépart' datetime NOT NULL DEFAULT '0000-00-00_00:00:00',
44     'UID' int unsigned DEFAULT NULL,
45     'StationDépart' smallint unsigned DEFAULT NULL,
46     'DateRetour' datetime DEFAULT NULL,
47     'StationRetour' smallint unsigned DEFAULT NULL,
48     PRIMARY KEY ('VID', 'DateDépart'),
49     FOREIGN KEY ('VID') REFERENCES Villo('VID'),
50     FOREIGN KEY ('UID') REFERENCES Utilisateur('UID'),
51     FOREIGN KEY ('StationDépart') REFERENCES Station('SID'),
52     FOREIGN KEY ('StationRetour') REFERENCES Station('SID')
53 );

```

## Les requêtes en algèbre relationnel et calcul relationnel tuples

### R1

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

#### Algèbre relationnelle

$$\begin{aligned}
 UTS &\leftarrow (Abonne * Trajet) \bowtie_{StationDepart=SID} Station \\
 Sel &\leftarrow \sigma_{Nom=Flagey \wedge Ville=Ixelles}(UTS) \\
 \pi_{UID}(Sel)
 \end{aligned}$$

## Calcul relationnel tuple

$$\{a.UID | Abonne(a) \wedge a.Ville = 'Ixelles' \wedge \exists t \exists s (Trajet(t) \wedge Station(s) \wedge t.UID = a.UID \wedge s.Nom = 'Flagey')\}$$

## SQL

```
1 SELECT a.UID FROM Trajet t, Abonné a, Station s
2 WHERE a.UID=t.UID AND s.SID=t.StationDépart
3 AND a.Ville="Ixelles" AND s.Nom="Flagey"
```

## Imprécisions

L'énoncé ne dit pas si il faut récupérer l'UID où le nom de l'utilisateur. Nous avons donc supposé qu'il s'agissait de l'UID.

## R2

Les utilisateurs ayant utilisé Villo au moins 2 fois

## Algèbre relationnelle

$$\begin{aligned} T(u, v, d) &\leftarrow \pi_{UID, VID, DateDepart}(Trajet) \\ Temp &\leftarrow Trajet \bowtie_{UID=u} T \\ &\pi_{UID}(\sigma_{VID \neq v \vee DateDepart \neq d}(Temp)) \end{aligned}$$

## Calcul relationnel tuple

$$\{u.UID | Utilisateur(u) \wedge \exists t1 \exists t2 (Trajet(t1) \wedge Trajet(t2) \wedge t1.UID = u.UID \wedge t2.UID = u.UID \wedge [t1.VID \neq t2.VID \vee t1.DateDepart \neq t2.DateDepart])\}$$

## SQL

```
1 SELECT t1.UID FROM Trajet t1, Trajet t2 WHERE t1.UID=t2.UID
2 AND ( t1.DateDépart!=t2.DateDépart OR t1.VID!=t2.VID )
3 GROUP BY t1.UID
```

## Imprécisions

Voir R1.

## R3

Les paires d'utilisateurs ayant fait un trajet identique

### Algèbre relationnelle

$$\begin{aligned} t2(UID2, dep2, ret2) &\leftarrow \pi_{UID, StationDepart, StationRetour}(Trajet) \\ Temp &\leftarrow Trajet \bowtie_{StationDepart=dep2 \wedge StationRetour=ret2} t2 \\ \pi_{UID, UID2}(\sigma_{UID \neq UID2} Temp) \end{aligned}$$

### Calcul relationnel tuple

$$\{u1.UID, u2.UID | Utilisateur(u1) \wedge Utilisateur(u2) \wedge u1.UID \neq u2.UID \wedge \exists t1 \exists t2 (Trajet(t1) \wedge Trajet(t2) \wedge t1.UID = u1.UID \wedge t2.UID = u2.UID \wedge t1.StationDepart = t2.StationDepart \wedge t1.StationRetour = t2.StationRetour)\}$$

## SQL

```
1 SELECT t1.UID, t2.UID FROM Trajet t1, Trajet t2
2 WHERE t1.StationDépart=t2.StationDépart
3 AND t1.StationRetour=t2.StationRetour
4 AND t1.UID!=t2.UID
5 GROUP BY t1.UID, t2.UID
```

### Imprécisions

Voir *R1*.

## R4

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

### Algèbre relationnelle

$$\begin{aligned} T1(v1, d1, s1) &\leftarrow \pi_{VID, DateDepart, StationRetour}(Trajet) \\ T2(v2, d2, s2) &\leftarrow \pi_{VID, DateDepart, StationDepart}(Trajet) \\ T3(v3, d3) &\leftarrow \pi_{VID, DateDepart}(Trajet) \\ T12 &\leftarrow T1 \bowtie_{v1=v2 \wedge d1 < d2} T2 \\ Temp &\leftarrow T12 \bowtie_{d1 < d3 \wedge d3 < d2 \wedge v3=v1} T3 \\ \pi_{v2}(\sigma_{s1 \neq s2 \wedge d3=NULL}(Temp)) \end{aligned}$$



## Calcul relationnel tuple

$$\{v.VID | V(v) \wedge \exists t1 \exists t2 (Trajet(t1) \wedge Trajet(t2) \wedge t1.VID = v.VID \wedge t2.VID = v.VID \wedge t1.DateDepart < t2.DateDepart \wedge t1.StationRetour \neq t2.StationDepart \wedge \nexists t3 (t3.VID = v.VID \wedge t1.DateDepart < t3.DateDepart \wedge t3.DateDepart < t2.DateDepart))\}$$

## SQL

```
1 SELECT t1.VID FROM Trajet t1, Trajet t2
2 WHERE t1.VID = t2.VID
3 AND t1.DateDepart < t2.DateDepart
4 AND t1.StationRetour != t2.StationDepart
5 AND NOT EXISTS
6     (SELECT * FROM Trajet t3
7      WHERE t3.VID = t1.VID
8      AND t1.DateDepart < t3.DateDepart
9      AND t3.DateDepart < t2.DateDepart)
10 GROUP BY t1.VID
```

## R5

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

```
1 SELECT t1.uid, a.DateInscription, t1.nb, di.dist, di.adist
2 FROM
3     (SELECT Trajet.UID, t.nb FROM
4      Trajet,
5      (SELECT tr.UID as uid, COUNT(*) as nb
6       FROM Trajet as tr GROUP BY tr.UID ) as t
7      WHERE Trajet.UID = t.uid
8      GROUP BY t.uid ) as t1,
9     Abonné a,
10    (SELECT tra.UID as uid,
11     SUM( SQRT(POW(s2.Longitude-s1.Longitude,2) +
12           POW(s2.Latitude-s1.Latitude,2)) ) as dist,
13     AVG( SQRT(POW(s2.Longitude-s1.Longitude,2) +
```

```

14          POW(s2.Latitude-s1.Latitude,2)) ) as adist
15 FROM Trajet tra, Station s1, Station s2
16 WHERE tra.StationDépart = s1.SID
17 AND tra.StationRetour = s2.SID
18 GROUP BY tra.UID) as di
19 WHERE t1.uid = a.UID
20 AND di.uid = a.UID
21 ORDER BY di.dist

```

### Imprécisions

Comme pour *R1*, on a décidé d'utiliser l'*UID* plutôt que le *Nom* de l'*Utilisateur*. Pour les distances, il n'est pas précisé d'unité. Nous avons donc pris la décision d'utiliser directement les coordonnées géographiques ( Longitude et Latitude ) sans aucune conversion pour les calculer.

## R6

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 différent ayant utilisé la station et ce pour toutes les stations ayant été utilisées au moins 10 fois.

### SQL

```

1 SELECT v.s, v.nbvillo, u.nbuser
2 FROM
3 (SELECT tr.StationRetour as s, COUNT(*) as nbvillo
4 FROM Trajet tr GROUP BY s) as v,
5 (SELECT tr.StationRetour as s, COUNT(DISTINCT tr.UID) as nbuser
6 FROM Trajet tr GROUP BY s) as u
7 WHERE v.s = u.s
8 HAVING v.nbvillo >= 10

```