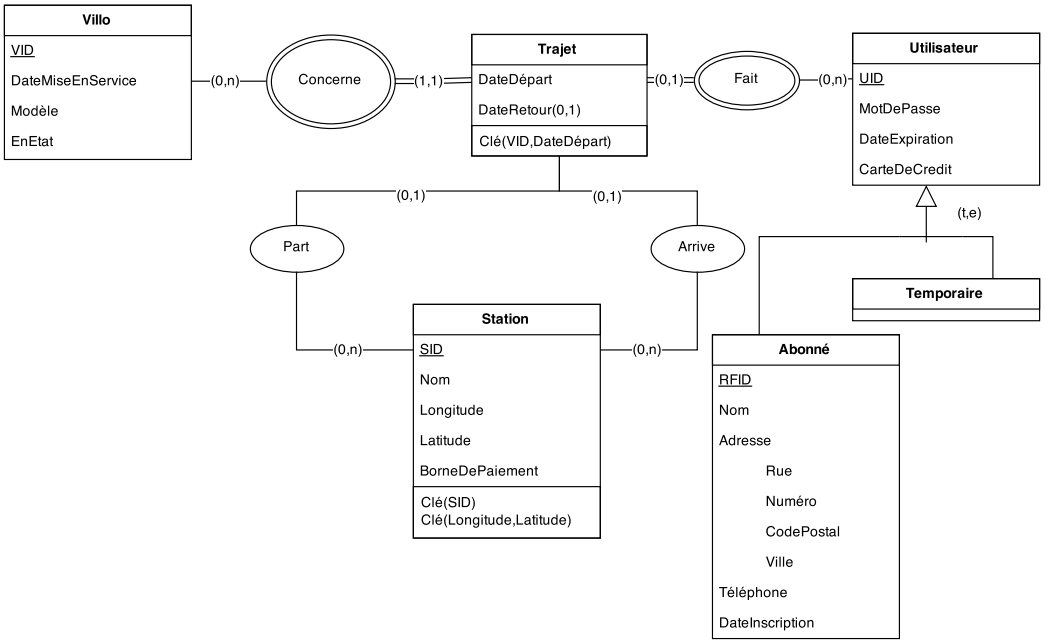
Projet INFO-H-303 : Villo!

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 placé 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` mediumint 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` mediumint 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` varchar(10) NOT NULL,
26         `DateInscription` datetime NOT NULL,
27         PRIMARY KEY (`RFID`),
28         FOREIGN KEY (`UID`) REFERENCES Utilisateur(`UID`)
29 );
30
```

```
CREATE TABLE `Station` (
        `SID` smallint unsigned NOT NULL,
        `Nom` varchar(50) NOT NULL,
        `Longitude` float NOT NULL,
        `Latitude` float NOT NULL,
        `Capacité` tinyint unsigned NOT NULL,
        `BorneDePaiement` tinyint(1) NOT NULL,
       PRIMARY KEY (`SID`,`Longitude`,`Latitude`)
);

CREATE TABLE `Trajet` (
        `VID` smallint unsigned NOT NULL,
        `DateDépart` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
        `UID` mediumint unsigned DEFAULT NULL,
        `StationDépart` smallint unsigned DEFAULT NULL,
        `DateRetour` datetime DEFAULT NULL,
        `StationRetour` smallint unsigned DEFAULT NULL,
       PRIMARY KEY (`VID`,`DateDépart`),
       FOREIGN KEY (`VID`) REFERENCES Villo(`VID`),
       FOREIGN KEY (`UID`) REFERENCES Utilisateur(`UID`),
       FOREIGN KEY (`StationDépart`) REFERENCES Station(`SID`),
       FOREIGN KEY (`StationRetour`) REFERENCES Station(`SID`)
);
```

Les requêtes en algèbre relationnel et calcul relationnel tuples

R1

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

Algèbre relationnelle

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

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

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

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

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

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

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

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

```
SELECT t1.VID FROM Trajet t1, Trajet t2
WHERE t1.VID = t2.VID
AND t1.DateDepart < t2.DateDepart
AND t1.StationRetour != t2.StationDepart
AND NOT EXISTS
        (SELECT * FROM Trajet t3
    WHERE t3.VID = t1.VID
    AND t1.DateDepart < t3.DateDepart
    AND t3.DateDepart < t2.DateDepart)
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

```
SELECT t1.uid, a.DateInscription, t1.nb, di.dist, di.adist
FROM
(SELECT Trajet.UID, t.nb FROM
        Trajet,
        (SELECT tr.UID as uid, COUNT(*) as nb
                FROM Trajet as tr GROUP BY tr.UID ) as t
                WHERE Trajet.UID = t.uid
                GROUP BY t.uid ) as t1,
Abonné a,
(SELECT tra.UID as uid,
        SUM( SQRT(POW(s2.Longitude-s1.Longitude,2) +
                POW(s2.Latitude-s1.Latitude,2)) ) as dist,
        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
```

Justification et hypothèses

Les seules hypothèses faites en plus de celles données pour la modélisation sont par rapport au types des colonnes dans la base de donnée. Voici les explications.

Tout d'abord pour la table *Utilisateur*. L'*UID* est stocké dans un mediumint unsigned car il s'agit d'un chiffre positif. Ce type allant jusqu'à 16 277 215, nous avons jugé cela suffisant par rapport au nombre d'utilisateur

potentiel. Les *MotDePasse* des fichiers data fournis pour le projet était tous des nombres de 4 chiffres, nous avons décider de les stocker dans des smallint unsigned de taille 4. La *CarteDeCredit* est une suite de 16 chiffres, on a donc pu la stocker dans un bigint unsigned de taille 16. Pour la *DateExpiration*, on a fait comme toutes les dates du projet, nous l'avons enregistré au format datetime.

Ensuite pour la table *Villo*. Le *VID* est stocké dans un smallint unsigned. Ce type permet de stocker des entier positif allant jusqu'à 65 535. Nous avons jugé ce nombre suffisant. Le modèle est un varchar de taille 12 qui est la taille du texte de tous les modèles des fichiers data. *EnEtat*, qui est un booléen étant vrai lorsque le villo n'est pas cassé, est stocké dans un tinyint qui vaut 0 lorsque le booléen est faux et 1 lorsqu'il est vrai.

La table suivante est la table *Abonné*. Le *Nom*, la *Ville* et la *Rue* sont stockées dans des varchar dont les tailles sont respectivement 50, 50 et 100. Ces tailles ont été choisies arbitrairement par rapport à ce que nous pensions suffisant pour stocker ces données. Le *Numéro* est un smallint unsigned car un tinyint unsigned ne va que jusque 255 et bien que ce nombre paraisse assez grand, le smallint unsigned nous permet une meilleure marge de sécurité. Les *CodePostal* belges sont des nombres de 4 chiffres, ils sont donc stockés dans un smallint unsigned de taille 4. Bien que le téléphone soit une suite de 9 (fixes) ou 10 (gsm) chiffres, nous avons décider de le stocker dans un varchar de taille 10 pour continuer à stocker le 0 à l'avant. Le *RFID* est un nombre de 20 chiffres qui est donc trop grand pour un bigint unsigned. Nous le stockons donc dans un varchar de taille 20. Les RFID ne se suivant pas nécessairement, il n'est pas incrémenté mais généré aléatoirement à l'inscription.

Ensuite pour la table *Station*. Le *SID* est stocké dans un smallint unsigned qui nous semble amplement suffisant alors qu'un tinyint risquait de limiter l'augmentation du nombre de station. La *Longitude* et la *Latitude* sont des float car la précision des double n'était pas nécéssaire. La *Capacité* est stocké dans un tinyint car 255 nous semble une limite bien assez importante. *BorneDePaiement* étant un booléen, il est stocké dans un tinyint.

Il n'y a rien de particulier à dire sur la table *Trajet* étant donné que celle ci est uniquement composé de date ainsi que de foreign key.