Systèmes de Gestion de Bases de Données

Rapport du projet SGBD : Flotte de Vélos

Auteurs :

Dkhissi Youness
Jamil Mohammed
Toumi Youssef

Encadrents :

M. Lombardy Sylvain
M. Mosbah Mohamed

Table des matières

1 Introduction

Ce projet consiste à réaliser une base de données pour stocker et gérer les données nécessaires au suivi d'une flotte de vélos électriques sur une métropole. Dans ce rapport, nous détaillons nos choix de conception et d'implémentation de la base de données en quatre parties principales : La première partie, consiste en une description et une analyse du contexte de l'application pour ensuite présenter notre modélisation du problématique sous forme d'un modèle conceptuel de données. Dans la deuxième partie, nous faisons la traduction du modèle entité-association en un modèle relationnel de données sous la 3^{eme} forme normale, tout en respectant les contraintes d'intégrité et dépendances fonctionnelles. La troisième partie quant à elle, présente l'implémentation de la base en `SQL` ainsi les services qu'elle offre. Et finalement, dans la quatrième partie, on s'intéresse à l'interface graphique de l'application en fournissant une notice d'utilisation.

2 Modélisation des données

La projection de la situation réelle d'une flotte de vélos électriques sur une base de données nécessite tout d'abord l'établissement d'un schéma conceptuel qui regroupe l'ensemble des informations utiles sous formes d'entités et qui représente les associations entre ces entités. Dans cette partie nous présentons les choix qui ont été fait sur la conception de le modèle entité-association.

2.1 Description du contexte de l'application

Nous nous intéressons à la gestion et le suivi d'une flotte de vélos sur la métropole de Bordeaux. Chacun de ces vélos a un numéro de référence, un modèle, date de mise en service, kilométrage, état, niveau de charge de la batterie et un prix horaire pour l'utilisation. Ces derniers sont distribués par des fournisseurs ayant un nom, une adresse et une ville. Les vélos vont être contenue dans plusieurs stations ayant une adresse et un nombre de bornes qui représentent le nombre maximal de vélos qui peuvent être accueilli par cette station. Les stations se situent dans des communes et elles sont éloignées les unes par rapport aux autres par une distance précise. Les vélos électriques vont être utilisés par des adhérents qui sont identifiés par des numéros adhérent qui leur ont été attribués au moment de leur adhésion. Puisque l'utilisation des vélos est tarifée, chaque adhérent dispose d'un solde qui lui permettra de payer le prix d'utilisation des vélos.

2.2 Modèle entité-association

Afin de concevoir le modèle entité-association de notre base de données, nous avons dû prendre plusieurs décisions quant au choix des entités et des associations qui les relient, ainsi que le choix des cardinalités maximales et minimales. Notre schéma conceptuel contient 6 entités :

- **Vélos** : Cette entité représente les vélos, elle a pour identifiant le `numéro de référence`, et contient d'autres attributs qui donnent des informations sur le `modèle`, la `date de mise en service`, le `kilométrage`, `l'état du vélo`, le `prix horaire` et le `niveau de batterie`.
- **Fournisseurs** : Représente les fournisseurs de vélo, chaque fournisseur est identifié par un unique `numéro de fournisseur`, et possède trois autres attributs : `nom de fournisseur`, `adresse`, et `ville`.
- **Stations** : Représente un ensemble de stations du métropole, elle a pour identifiant un numéro de station, et des attributs fournissant l'adresse du station et son nombre de bornes qui correspond au nombre maximale de vélos que peut contenir une station.
- **Communes** : Chaque commune possède un attribut `nom de commune`, et un identifiant `numéro de commune`.
- **Adhérents** : Représente la personne adhérente à ce service, elle est identifiée par son unique `numéro d'adhérent`, et a pour attributs, `nom`, `prénom`, `adresse`, `date d'adhésion` et `solde adhérent`.
- **Emprunts** : Cette entité représente les enregistrements des emprunts des vélos par les adhérents. Elle a comme attributs `date de début d'emprunt`, `date de fin d'emprunt`, la `distance parcourue`, et elle a pour identifiant l'attribut `numéro adhérent`. La distance parcourue par un adhérent ne peut pas être calculée à partir des stations de départ et d'arrivée, puisqu'elle est, en général, plus grande que la distance entre ces deux stations. d'où le choix d'ajouter un attribut `distance parcourue` dans l'entité **Emprunts**.

Ces entités sont liée entre elles à travers plusieurs associations, chacune sa signification et ses caractéristiques :

- **Fournir** : Cette association relie les entités **Fournisseurs** et **Vélos**. Un fournisseur peut fournir (ou pas) plusieurs vélos, mais un vélo ne peut être fourni que par un seul fournisseur, on aura donc comme cardinalités minimales et maximale $0, N$ du coté Fournisseurs, et $1, 1$ du coté Vélos.
- **Être contenue** : Cette association relie les entités **Vélos** et **Stations**. Un vélo ne peut être contenu que dans une seule station, ou bien mis en service par un adhérent ce qui justifie le choix des cardinalités $0, 1$. Une station peut contenir plusieurs vélos, mais il se peut qu'une station soit vide d'où le choix des cardinalités $0, N$.
- **Se trouver** : Associe l'entité `Stations` à `Commune`. Une station appartient à une seule commune ce qui justifie le choix des cardinalités $1, 1$. Une commune peut avoir plusieurs (ou aucune) stations, donc des cardinalités $0, N$.
- **Être éloigné** : Cette association possède un attribut distance, qui représente la distance entre deux stations. Nous n'avons pas considéré la distance entre une station et elle même d'où le choix de mettre des cardinalités $0, N$ des deux cotés de l'association.
- **Être station de départ** : Associe un emprunt à sa station de départ, un emprunt doit forcément avoir une et une seule station de départ, alors qu'une station peut être associée à plusieurs emprunts, d'où les cardinalités $1, 1$ du coté Emprunts, et $0, N$ du coté Stations.
- **Être station d'arrivée** : Associe un emprunt à sa station de d'arrivée, contrairement à l'association précédente l'emprunt d'un vélo en cours d'utilisation n'a pas de station d'arrivée d'où le choix des cardinalités $0, 1$ du coté Emprunts.
- **Utiliser dans** : Cette association relie les entités **Vélos** et **Emprunts**. Un vélo peut apparaître (ou non) dans plusieurs emprunts, ce qui se traduit par les cardinalités $0, N$. Un emprunt doit forcément être associé à un et un seul vélo. Pour éviter la suppression des emprunts après avoir enlevé un vélo de la base, nous avons choisi les cardinalités $0, 1$, afin de calculer des statistiques précises.
- **Effectuer** : Cette association relie les entités **Adhérents** et **Emprunts**. Un adhérent peut effectuer (ou non) plusieurs emprunts, d'où par les cardinalités $0, N$. Un emprunt doit forcément être associé à un et une seule adhérent, mais pour les même raisons que l'association précédente, nous avons choisi les cardinalités $0, 1$.

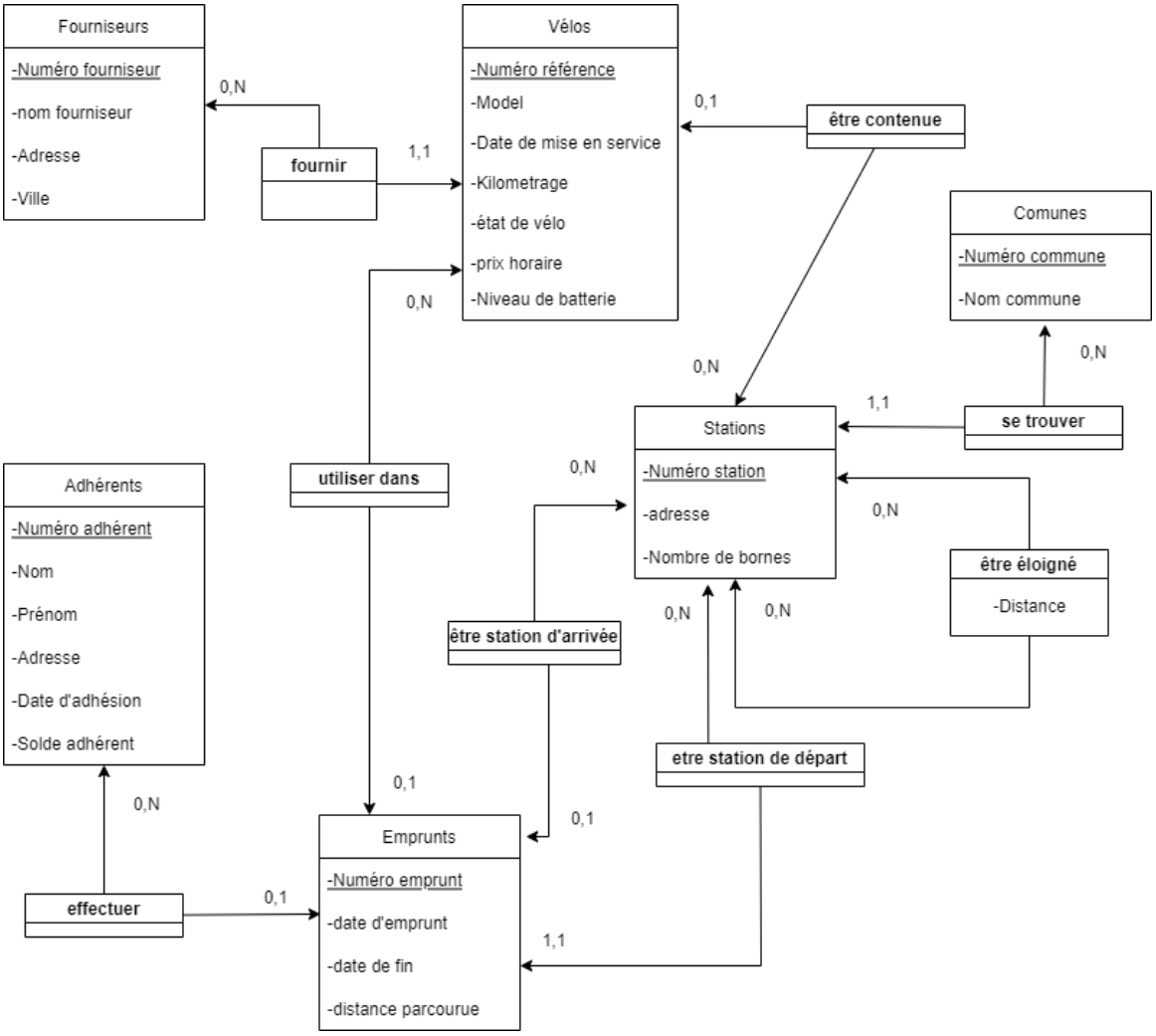

FIGURE 1 – Le schéma conceptuel de flotte de vélos

3 Modèle relationnel

Après avoir constituer le schéma conceptuel, nous sommes passés à la traduction en schéma relationnel qui sert de support à l'implémentation de la base de données en **SQL**. Ce schéma permet aussi de mettre en évidence les clés primaires et étrangères de chaque relation.

3.1 Traduction du modèle conceptuel

Les entités du schéma conceptuel ont été transformés dans le schéma relationnel sous forme de relations dont les clés primaires et les attributs sont ceux des entités du modèle conceptuel. L'association réflexive *être éloigné* qui relie l'entité *Stations* à elle-même. est devenue une relation qui a comme clé primaire le couple de deux clés étrangères de la relation *stations*, et elle a comme un attribut *distance* présent dans le schéma entité-association. Dans le cas des associations de type 1 : N. Nous avons ajouté les identifiants des entités qui ont une cardinalité maximale N dans les entités de cardinalité maximale 1 sous forme des clés étrangères :

- La relation *Emprunts* qui a comme clés étrangères, les clés primaires de *Adhrents*, *velos* et *stations*.
- La relation *Stations* a une clé étrangère de la relation *Communes*.
- La relation *Velos* qui a comme clés étrangères, les clés primaires des relations *Stations* et *Fournisseurs*.

3.2 Contraintes d'intégrité et dépendances fonctionnelles

Nous nous sommes basés sur les cardinalités des associations du schéma conceptuel afin d'extraire et spécifier les contraintes d'intégrité et les dépendances fonctionnelles. Dans l'association entre *Emprunts* et *Station*, nous avons de côté de l'entité *Emprunts* une cardinalité minimale égale à 1, ce qui impose à ce que la clé étrangère de *Station* qui est dans Emprunts soit non nulle. Pour la même raison, la clé étrangère de *Communes* présente dans *Stations* doit être elle aussi non nulle. De même pour la clé étrangère de *station* qui réfère à la station de départ de l'emprunt. En ce qui concerne les dépendances fonctionnelles, nous avons construit l'arbre de couverture minimal de la Figure 2 pour normaliser notre schéma relationnel.

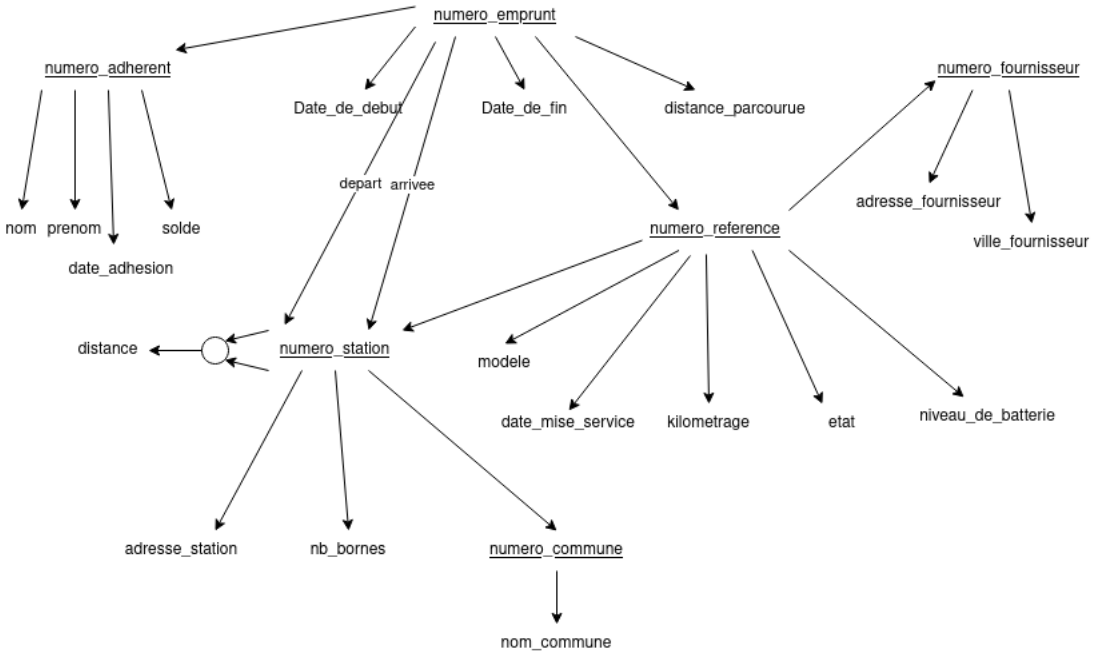

FIGURE 2 – L'arbre de couverture minimal du schéma conceptuel

3.3 Schéma relationnel en 3^{me} forme normale

D'après la section 3.1 et 3.2, nous avons construit le schéma relationnel représenté dans la figure 3. Ce modèle respecte bien la 3^{me} forme normale car les champs de notre schéma ne prennent pas une liste de valeurs (spécification de la 1^{re} forme normale). Tous les champs dépendent de l'entièreté de la clé primaire (spécification de la 2^{me} forme normale) et seulement de celle-ci (spécification de la 3^{me} forme normale).

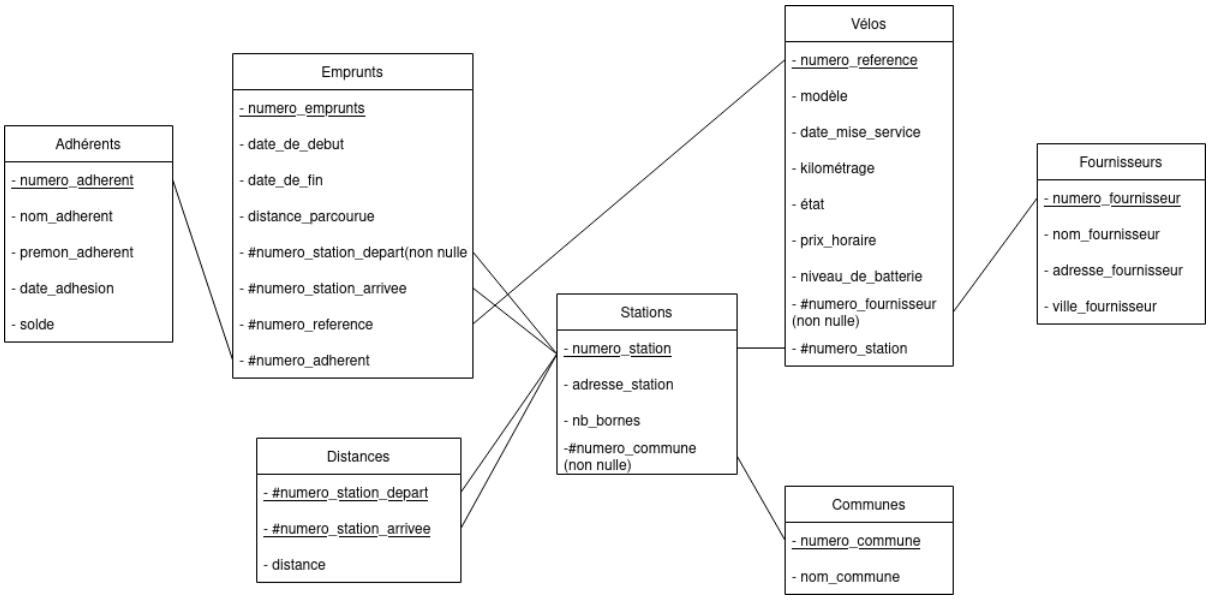

FIGURE 3 – Le schéma relationnel de notre base

4 Implémentation en SQL

Dans cette partie nous allons nous intéresser aux étapes d'implémentation de notre base de données. Nous avons choisit *Oracle* comme système de gestion de base de données. l'implémentation de la base passe par trois étapes principales : création et peuplement de la base, implémentation des requêtes de consultation et des statistiques, et finalement la gestion des mise à jour.

4.1 Création et peuplement de la base

Comme nous l'avons détaillé dans la partie précédente. Notre base de données contient 7 tables. Les requête de création de ses tables se trouvent dans un fichier `creation.sql`. Voici un exemple de création d'une table :

```
create table COMMUNES(
    NUMERO_COMMUNE number(4) primary key not null,
    NOM_COMMUNE char(20)
);
```

Pour le peuplement de la base de donnée nous avons écrit un script `Python` pour générer des données aléatoires. Le code SQL géneré par ce script est mis par la suite dans le fichier `peuplement.sql` qui permet d'insérer les données dans notre base. le code SQL suivant est un exemple d'ajout d'une ligne dans une table :

```
insert into COMMUNES values (1, 'Talence');
```

4.2 Consultation et statistiques

Pour consulter les données de notre base de données, nous avons écris plusieurs requêtes. Pour visualiser l'entièreté d'une table par exemple. Nous avons aussi écrit quelques requêtes pour visualiser des données plus intéressantes. Ces requêtes de consultations nous permettent également de consulter les données après chaque modification, suppression ou ajout de nouvelles données. Voici quelques requêtes que nous avons implémentées dans le fichier `consultation.sql` :

- La liste des vélos par stations.
- La liste des stations dans une commune donnée.
- La liste des adhérents qui ont emprunté plusieurs ou au moins deux vélos différents pour un jour donné.
- La liste des stations dans une commune donnée.
- La liste des vélos en cours d'utilisation.

Nous avons aussi implémenté des requêtes SQL qui permettent d'avoir des statistiques sur les données contenues dans la base. ces requêtes sont écrites dans le fichier `statistiques.sql` et renvoient les informations suivantes :

- La moyenne de nombre d'usagers par vélos par jour.
- La moyenne des distances parcourues par les vélos sur une semaine.
- La moyenne des distances parcourues par vélo sur une période donnée.
- Le classement des stations par nombre de places disponibles par commune.
- Le classement des vélos les plus chargés par station.

4.3 Mise à jour

Puisque les tables ont des dépendances entre eux, nous avons besoin d'implémenter des déclencheurs *"triggers"* qui vont gérer les dépendances entre les tables de données et faire les modifications possibles sur les autres tables pour que les contraintes ne soient pas violées, dans le cas d'une modification, ajout ou suppression d'enregistrements. Ces Déclencheurs sont placés dans notre base de données dans le fichier `mise-a-jour.sql`

Parmi les déclencheurs qui gèrent les opérations de mise à jour dans notre base, il y a l'exemple du déclencheur `AJOUT_EMPRUNT` qui permet, à chaque ajout d'un emprunt, de changer la clé étrangère du vélo associé à l'emprunt (sa valeur devient nulle car l'adhérent ne connais pas la station d'arrivée) et nous retirons du solde de l'adhérent le prix de l'emprunt du vélo.

```
create or replace trigger AJOUT_EMPRUNT
AFTER insert on EMPRUNTS
for each row
begin
    update VELOS
    set NUMERO_STATION = :new.NUMERO_STATION_DEPART
    where NUMERO_REFERENCE = :new.NUMERO_REFERENCE;
    update ADHERENTS
    set SOLDE = (select SOLDE - PRIX_HORAIRE from VELOS where
    NUMERO_REFERENCE =:new.NUMERO_REFERENCE)
    where NUMERO_ADHERENT = :new.NUMERO_ADHERENT;
end;
/
```

5 Interface graphique

5.1 Environnement d'exécution et *JDBC*

Le projet a été réalisé sur les machines de l'ENSEIRB-MATMECA. Afin de construire notre interface, nous avons utilisé *JDBC* (Java Database Connectivity) pour connecter celle-ci à notre base de données.

5.2 Description

L'exécution du programme se fait sur un terminal. Le menu principal représenté dans la figure 4 apparaît avec 3 choix : consultation, statistiques, mise à jour.

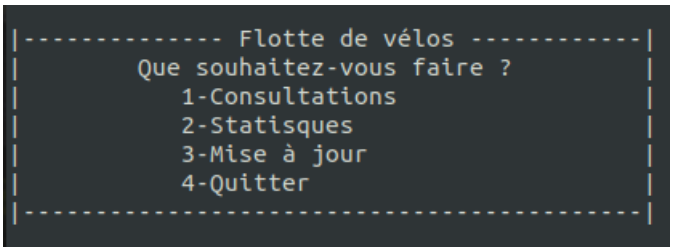

FIGURE 4 – Menu principale.

Selon le choix effectué, un nouveau menu apparaît (*cf* l'exemple du menu de consultation sur la figure 5).
Dans le cas d'une consultation, et selon le choix effectué, un fichier d'extension html est crée dans le répertoire html (créé aussi dynamiquement à la racine du dépôt) et qui contient le résultat de l'opération sous forme d'un table *cf* la figure 6. Le choix statistiques permet de visualiser quelques statistiques de notre base de données. Le dernier choix Mise à jour permet d'effectuer 3 opérations : la première est d'ajouter un nouvel enregistrement en demandant à l'utilisateur de fournir toutes les données dans la ligne de commandes, en veillant à le prévenir d'insérer des données valides (insertion d'un numéro de station dans VELOS qui n'existe pas dans la table STATIONS par exemple). La deuxième opération est la suppression d'un enregistrement. Le programme demande à l'utilisateur de choisir la table de laquelle il veut supprimer l'enregistrement en question, puis de fournir le numéro (clé primaire) de ce dernier (cela nous garantit de supprimer que cet enregistrement par unicité des clés primaires). D'autres données peuvent dépendre de l'enregistrement supprimé, mais comme mentionné précédemment, les déclencheurs permettent de régler ce problème. La 3ème opération : modification d'un enregistrement. Le programme demande à l'utilisateur dans un premier temps de choisir dans quelle table il veut modifier l'enregistrement en question, et puis de fournir les nouvelles informations.

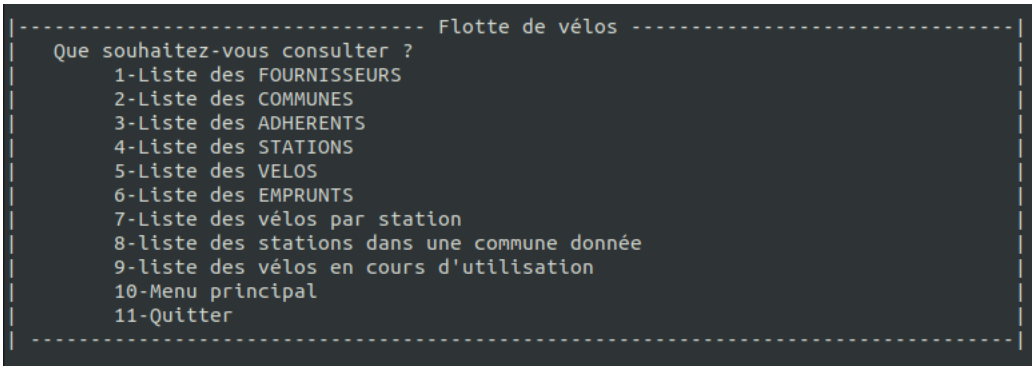

FIGURE 5 – Menu consultation.

Liste des VELOS

NUMERO_REFERENCE	MODEL	DATE_MISE_SERVICE	KILOMETRAGE	ETAT	PRIX_HORAIRE	NIVEAU_DE_BATTERIE	NUMERO_FOURNISSEUR	NUMERO_STATION
1	24SEVEN INSPIRON 10	2012-01-11 00:00:00	6986	bon	1	85	2	0
2	24SEVEN INSPIRON 10	2012-01-11 00:00:00	5184	bon	1	43	2	1
3	24SEVEN INSPIRON 10	2012-01-11 00:00:00	3896	bon	1	64	2	1
4	24SEVEN INSPIRON 10	2012-01-11 00:00:00	3679	bon	1	29	2	1
5	24SEVEN INSPIRON 10	2012-01-11 00:00:00	6158	bon	1	27	2	1
6	24SEVEN INSPIRON 10	2012-01-11 00:00:00	6397	bon	1	29	2	0
7	24SEVEN INSPIRON 10	2012-01-11 00:00:00	4824	bon	1	39	2	1
8	24SEVEN INSPIRON 10	2012-01-11 00:00:00	5858	bon	1	78	2	1
9	24SEVEN INSPIRON 10	2012-01-11 00:00:00	5971	bon	1	52	2	1
10	24SEVEN INSPIRON 10	2012-01-11 00:00:00	4157	bon	1	92	2	1
11	24SEVEN INSPIRON 10	2012-01-11 00:00:00	3397	bon	1	87	2	1
12	24SEVEN INSPIRON 10	2012-01-11 00:00:00	6601	bon	1	55	2	0
13	24SEVEN INSPIRON 10	2012-01-11 00:00:00	6714	bon	1	62	2	1
14	24SEVEN PAVILLION	2014-02-28 00:00:00	4195	moyen	1	17	2	2
15	24SEVEN PAVILLION	2014-02-28 00:00:00	3397	moyen	1	27	2	2
16	24SEVEN PAVILLION	2014-02-28 00:00:00	4547	moyen	1	21	2	2
17	24SEVEN PAVILLION	2014-02-28 00:00:00	5794	moyen	1	65	2	0

FIGURE 6 – Tableau généré pour la consultation des vélos.

5.3 Utilisation

Pour exécuter notre programme, il faut d'abord se connecter sur la machine virtuelle d'*Oracle*, puis se déplacer dans le répertoire sql, et exécuter les scripts `suppression, creation, peuplement, mise-a-jour` dans cet ordre sur *sqlplus*. Ensuite, il faut se déplacer dans le répertoire `Application` et changer le username et le mot de passe dans Main.java. Finalement, il faut se déplacer à la racine du dépôt et exécuter la commande :

```
# make && make run
```

Pour supprimer les fichiers java compilés et les fichiers html générés il suffit d'exécuter la commande :

```
# make clean
```

6 Conclusion

Ce projet était l'occasion de mettre ce que nous avons appris à propos des systèmes de gestion de base de données en pratique. Il nous a permis de pousser notre réflexion pour avoir un schéma entité-association le plus adéquat possible. De plus, ce projet nous a permis une grande marge de flexibilité pour définir les hypothèses que nous avons utilisées pour construire notre modèle conceptuel et également notre base de données. Concernant l'interface graphique, c'est sur ce côté que nous avons eu le plus de problèmes, car aucun membre du groupe ne disposait de connaissances préalables en *php* ou le développement web en général. D'ailleurs, c'est la raison principale de notre utilisation de *JDBC*. L'utilisation de ce dernier était énormément tordu, surtout qu'à chaque modification, il faut se connecter sur une des machines de l'école en ssh pour compiler, et avec les différentes pannes qui ont atteintes les serveurs de bordeaux-inp ces deux dernières semaines, le travail était plus difficile. La deuxième difficulté que nous avons dû faire face est la manipulation des dates complètes avec l'heure et la minute sous *Oracle* et *sqlplus*. Après plusieurs tentatives, nous avons réussi à écrire les requêtes nécessaires, par contre nous n'avons pas pu l'écrire dans le fichier .java pour l'utiliser avec *JDBC* et l'afficher sur notre interface des choix.