Merwan BELKHIRI-GUEGUEN
Benjamin ROECKER
Felix PILLOT
L3 - Groupe G

Rapport projet de Databases

Dans le cadre de l'enseignement Databases du Semestre 6, nous devions réaliser une application permettant d'aider les gestionnaires d'agences de location avec l'identification des clients et le suivi des locations, réservations et retours de véhicules selon un cahier des charges.

Ce dernier devait comporter une gestion des locations prenant en compte les clients, les véhicules et les employés, une gestion des devis, une gestion des ressources et une gestion des clients.

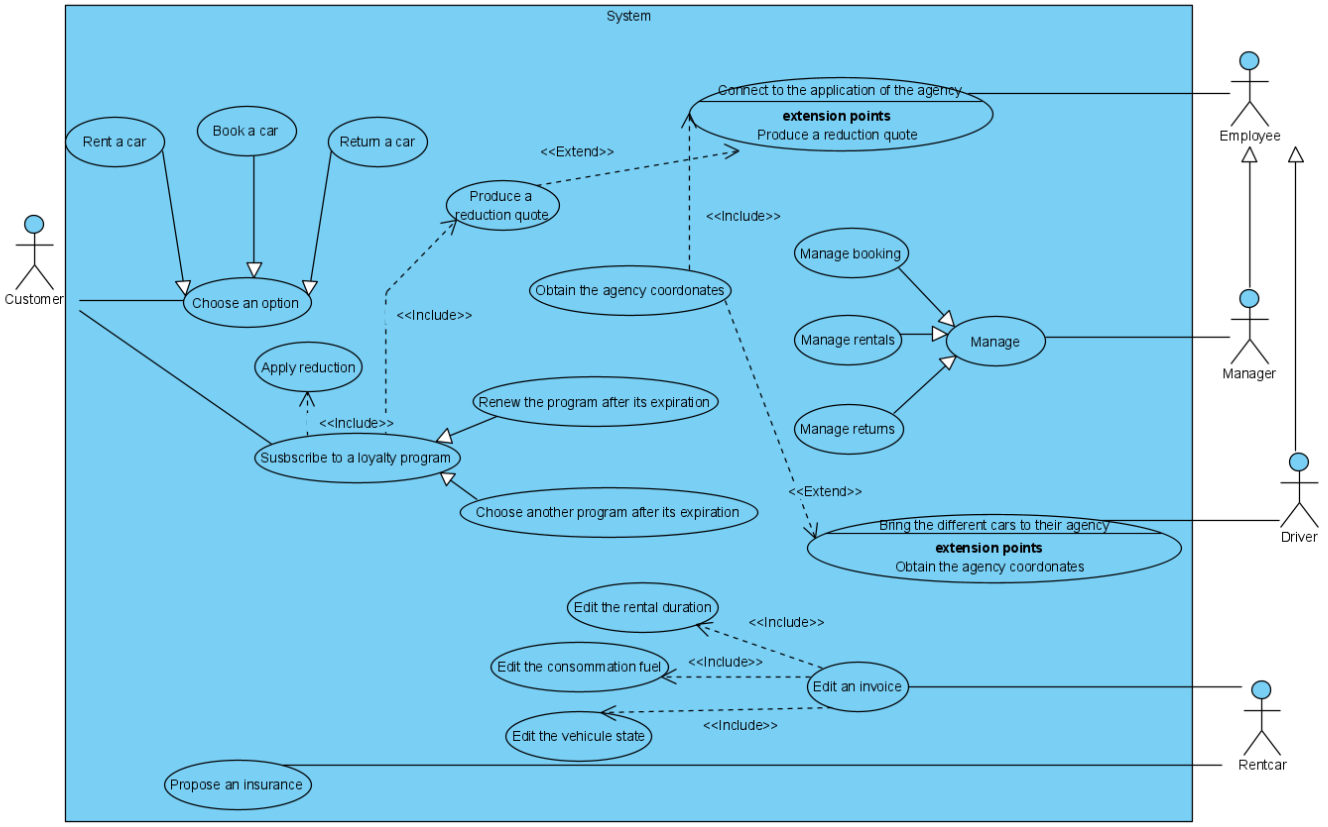
L'application devait être codée en Java tandis que la base de données devait être sous SQL

Sommaire :

Merwan BELKHIRI-GUEGUEN
Benjamin ROECKER
Felix PILLOT
L3 - Groupe G

1. Modélisation du problème

Face à ce projet, nous ne savions pas réellement par quoi commencer tellement le projet était dense.
Dès lors, nous avions commencé par modéliser les besoins de l'application Rentcar dans son ensemble, d'abord avec un cas d'utilisation

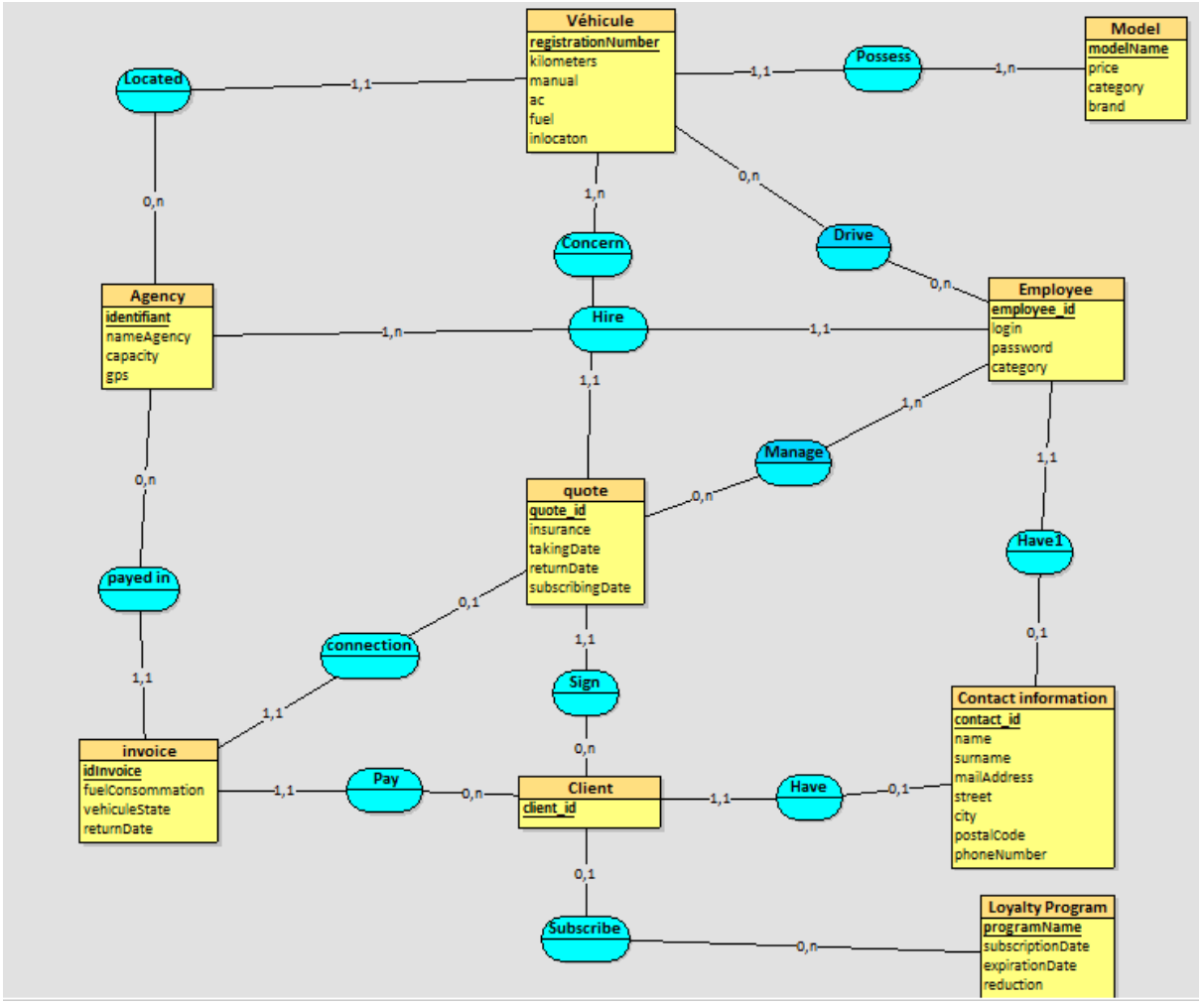


Cas d'utilisation pour modéliser les besoins de l'application Rentcar.

Le cas d'utilisation servait à avoir une idée plus globale et plus imagée de la situation et permettre d'identifier les différents acteurs et leurs tâches.

De ce fait, nous pouvions alors établir plus facilement le MCD que nous avons perfectionné jusqu' à avoir une base de données parfaite.

Merwan BELKHIRI-GUEGUEN
Benjamin ROECKER
Felix PILLOT
L3 - Groupe G



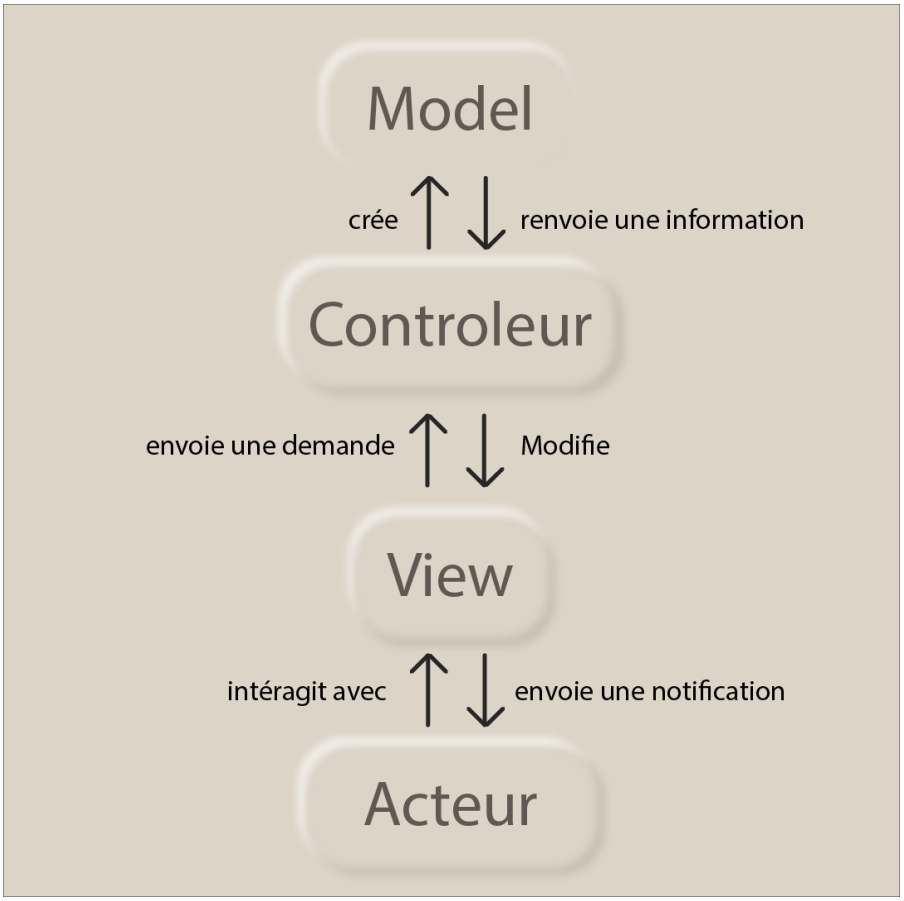
MCD de la base de données que nous utiliserons.

Dès lors, nous pouvons en déduire le MLD et le début du SQL.

Agency = (**identifiant** CHAR(50), nameAgency CHAR(50), capacity INT, gps VARCHAR(50));
Model = (**modelName** VARCHAR(25), price DECIMAL(3,2), category VARCHAR(10), brand VARCHAR(50));
Loyalty_Program = (**programName** VARCHAR(50), subscriptionDate DATE, expirationDate DATE, reduction INT, price DECIMAL(6,2));
Contact_information = (**contact_id** INT, name VARCHAR(50), surname VARCHAR(50), mailAddress VARCHAR(50), street CHAR(50), city CHAR(50), postalCode VARCHAR(5), phoneNumber VARCHAR(13));
Clients = (**No_Client** INT, *#programName*, **#contact_id***);
Véhicule = (**registrationNumber** VARCHAR(9), kilometers INT, manual LOGICAL, ac LOGICAL, fuel CHAR(50), inlocaton LOGICAL, *#identifiant, #modelName*);
Employee = (**Employe_id** VARCHAR(50), login VARCHAR(15), password VARCHAR(55), category LOGICAL, *#identifiant, **#contact_id***);
quote = (**quote_id** INT, insurance LOGICAL, takingDate DATE, returnDate DATE, subscribingDate DATE, *#registrationNumber, #No_Client*);
invoice = (**idInvoice** INT, fuelConsommation CHAR(50), vehiculeState VARCHAR(50), returnDate DATE, ***#quote_id**, #identifiant, #No_Client*);
Drive = (***#registrationNumber, #Employe_id***);
Manage = (***#Employe_id, #quote_id***);

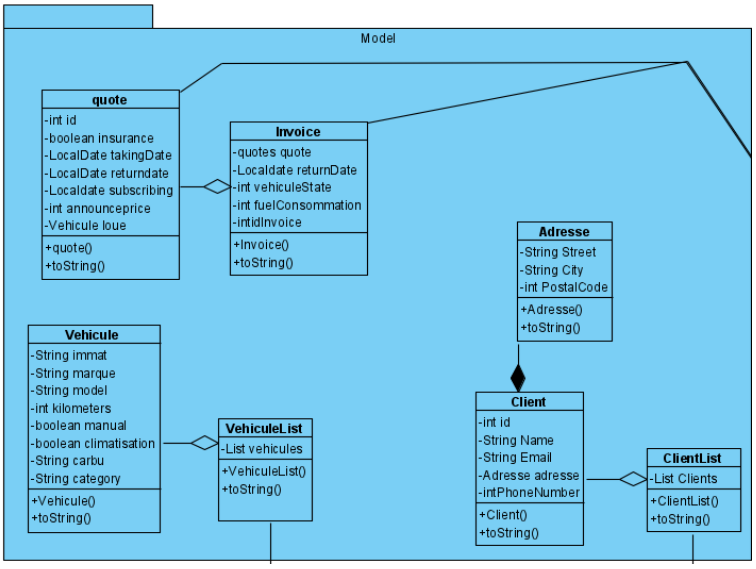
MLD de la base de données

Nous nous sommes aussi rapprochés d'un modèle en MVC qui existerait sur des diagrammes de classes et de séquences.

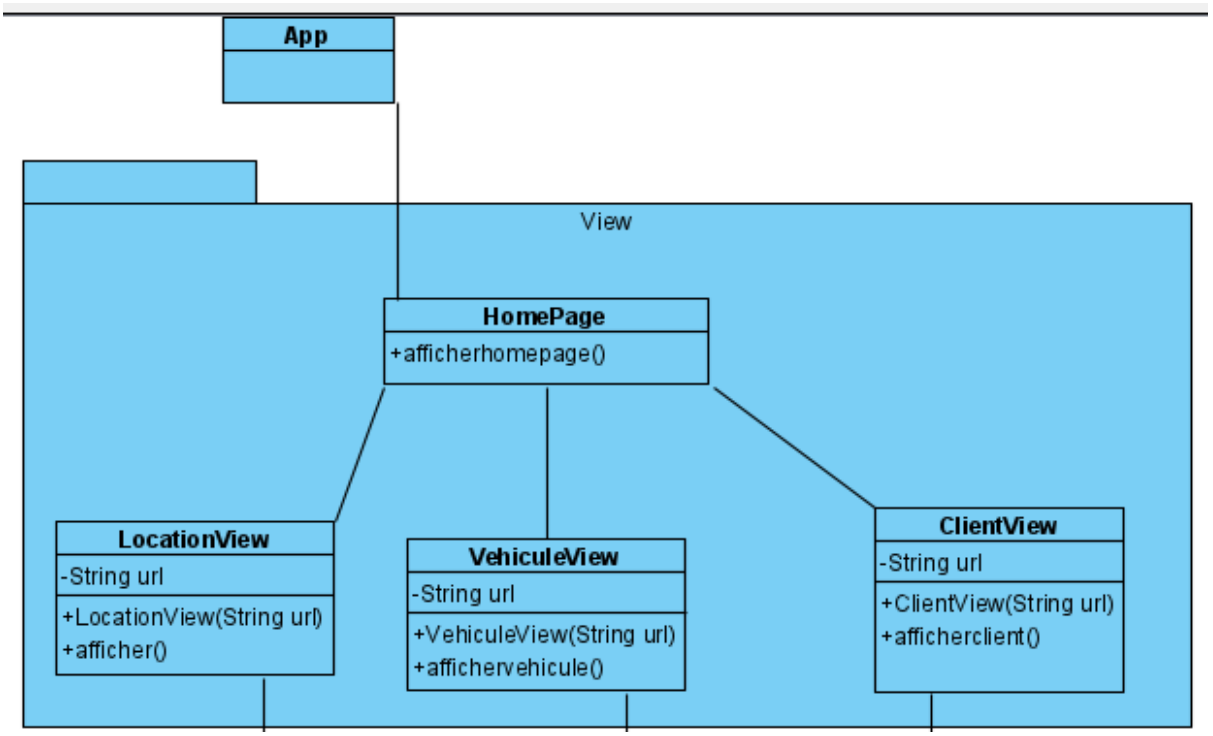


Modèle MVC

2. De la conception à la réalisation

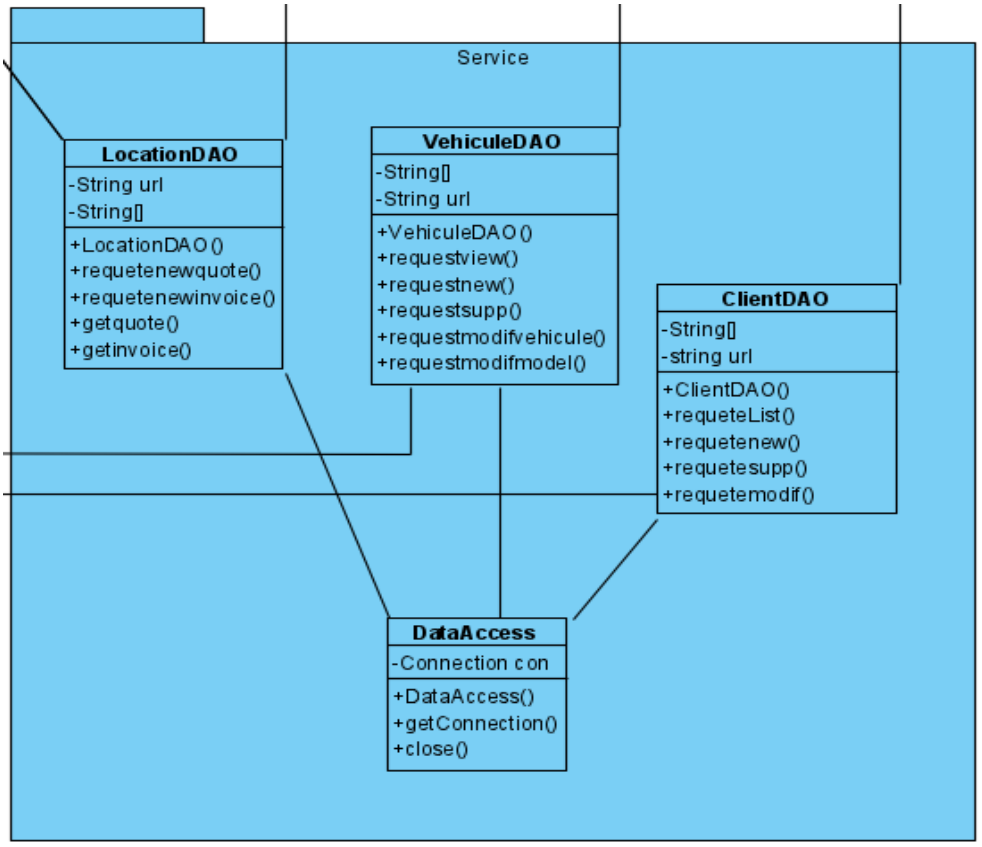


Voici le modèle de notre application, ce sont les objets que l'application manipule. Les fonctions toString nous permettent de les afficher à l'aide des différentes vues. Chaque attribut représente une donnée nécessaire.



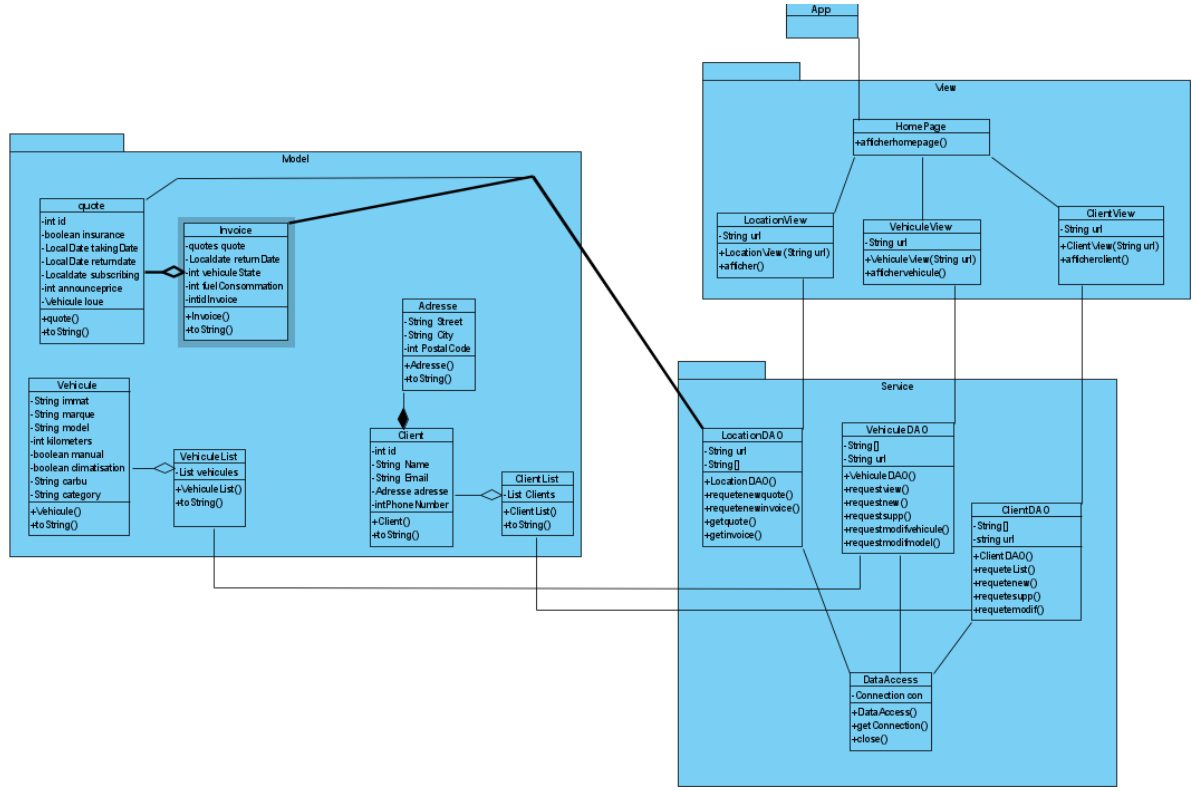
C'est dans les Vues que l'on fait l'affichage ainsi que la demande d'information pour remplir notre base de données.Nous n'avons pas fait d'interface graphique à l'aide de swing car le temps nous a manqué. Notre affichage n'est donc que sur console.

Voici les différentes vues de notre application, chaque vue nous permet de manipuler différents types de données. Clientview nous permet de manipuler les clients et leurs adresses. VehiculeView nous permet de manipuler les véhicules et Location View nous permet une gestion des devis et des factures.



Voici la partie service, elle remplace la partie controller. Chaque requête fait appelle à la base de données avec des requêtes stockées dans ces classes et des procédures stockées dans la base de données.

Chaque classe fait appel à la classe dataAccess pour obtenir un accès à la base de données et obtenir les informations voulues, pouvoir faire des ajouts et supprimer des données de notre table.



Voilà le classe diagramme et le lien entre toutes les classes.

Pour lancer le programme:

1. Ouvrir l'invite de commande.
2. Lancer le document rentcar.sql
3. se déplacer jusque dans le dossier
4. exécuter la ligne suivante dans l'invite de commande : java -jar rentcar.jar

Vous aurez auparavant besoin de l'url de votre base de données.

3. Difficultés rencontrées

La difficulté principale que nous avons rencontré et qui nous a réellement manqué est le temps. En effet, étant en pleine période de partiels et le projet étant relativement dense, nous étions à court de temps donc quelques fonctionnalités tel que la création et la gestion des employés, programmes et agences n'ont pas pu être créées.
Ceci étant dit, il s'agit d'un seul et même processus, donc nous savons le faire mais nous n'avions juste pas assez de temps.
De plus nous n'avons pas pu effectuer l'interface graphique car nous pensions que c'était moins important considérant le temps qui nous restait.

Dans le SQL, nous nous sommes heurtés à des problèmes de requêtes nombreuses et volumineuses.

Dès lors, nous avons utilisé des processus qui permettent de faire un simple appel de requêtes.

```
DELIMITER $ $ USE `rentcar` $ $ CREATE DEFINER = `root` @`localhost` PROCEDURE `new_vehicule`(
  registrationNumber VARCHAR(9),
  kilometers INT,
  manual boolean,
  ac boolean,
  fuel VARCHAR(50),
  inlocation boolean,
  identifiant VARCHAR(50),
  modelName VARCHAR(50),
  price int(3),
  category Varchar(10),
  brand Varchar(50)
) BEGIN DECLARE l,
i INT;
select
  count(modelName)
from
  Model
where
  Model.modelName = modelName into l;
if l = 0 then
insert into
  Model(ModelName, price, category, brand)
values(modelName, price, category, brand);
end if;
insert into
  v é hicule(
    registrationNumber,
    kilometers,
    manual,
    ac,
    fuel,
    inlocation,
    identifiant,
    modelName
  )
values(
    registrationNumber,
    kilometers,
    manual,
    ac,
    fuel,
    inlocation,
    identifiant,
    modelName
  );
END $ $ USE `rentcar`;
```

Exemple de processus que nous avons utilisé

CONCLUSION :

Malgré un manque de temps qui nous a obligé à passer certaines parties du projet, nous avons dans l'ensemble aimé ce projet car il regroupait des notions de base de données et des notions de code.
Nous sommes contents de ce que nous avons effectué dans un timing qui n'a pas été très bon. La taille de ce projet ne se prête pas à cette période de l'année avec de nombreux DE, projets et TP à rendre pour d'autres matières.