Bases de données 2
Gestion d'une société de location de véhicules

Quadrinôme

Alpha DIALLO, Kevin TASSI, Mamadou DIALLO et Mamadou Oury DIALLO

Groupe : 684J

L3 INFORMATIQUE
UNIVERSITÉ DE NANTES

Table des matières

Introduction

Pour ce projet de base de données, nous avons choisi comme thème la gestion d'une société de location de véhicules.
Dans la première partie de notre rapport nous allons définir une table contenant tous les tuples de notre base et de cette table nous allons tirer les dépendances fonctionnelles et par la suite la décomposer en plusieurs tables à l'aide de différents algorithmes.
Ensuite dans la deuxième partie du rapport nous allons à l'aide des fonctions , des triggers , et des vues essayer de mieux optimiser notre base de données.

1 - Déroulement du processus et distribution des tâches

Pour s'organiser dans notre travail, nous avons créé un espace de travail en ligne (googledoc et UNCLOUD), pour avoir un fichier commun de travail afin d'y inscrire l'avancement des normalisations, des dépendances fonctionnelles ainsi que de la base de données en elle-même.

Nous nous sommes aussi rapidement rassemblés pour réfléchir ensemble sur le sujet, au contenu des tables et aux dépendances fonctionnelles qui constituent notre base de données.

Dans la première partie du projet, Alpha et Mamadou Oury se sont occupés des normalisations avec les deux algorithmes proposés, de Bernstein et par décomposition.
Kevin quant à lui s'est chargé de la représentation du modèle relationnel de notre base de données avec les clés primaires et étrangères et de l'ajout de dépendances fonctionnelles pour permettre de calculer la couverture minimale de notre ensemble de dépendances fonctionnelles. Mamadou lui s'est occupé de la définition des tuples contenus dans notre table et de la définition de la couverture minimale de nos dépendances fonctionnelles.

Première table contenant tous les attributs

R (id_ag, ville, id_resp, nom_resp, pren_resp, tel_resp,id_veh, marque_veh, type_veh, cout_jours, id_clt, nom_clt, prenom_clt, adr_clt, tel_clt, date_loc, date_retour, imm_veh_veh)

2 - Dépendances fonctionnelles

DF = {	id_ag	→ ville_ag, id_resp, nom_resp, prenom_resp	(1)
	Id_ag, id_veh	→ marque_veh, type_veh, cout_jour	(2)
	id_resp	→ nom_resp, prenom_resp, tel_resp, id_ag	(3)
	id_veh	→ marque_veh, type_veh, cout_jour	(4)
	type_veh, marque_veh	→ cout_jour	(5)
	id_clt	→ nom_clt, prenom_clt, adr_clt, tel_clt	(6)

id_clt, nom_clt	→ prenom_clt	(7)
id_clt, id_veh	→ marque_veh	(8)
date_loc, date_retour, id_clt	→ id_ag	(9)
date_loc, date_retour, id_ag, id_veh	→ id_clt, id_resp	(10)
im_veh	→ type_veh, id_ag	(11)

}

Quelques tuples du schéma < R, DF >

id_ag	ville_ag	im_veh	nom_clt	pren_clt	adr_clt	id_clt	tel_clt	id_veh	marque	type_veh	id_resp	nom_resp	pren_resp	tel_resp	date_loc	date_retour	coût
A01	Nantes	im005	Foyer	Zadi	Brest	clt055	06070502	veh001	Toyota	Utilitaire	resp01	Tavis	Jose	0712364	10/12/2017	15/01/2018	80
A02	Paris	im045	Balde	Thomas	Lille	clt124	07530102	veh001	Toyota	Utilitaire	resp02	Ferdi	Cesar	0695312	10/12/2017	15/02/2018	90
A03	Angers	im451	Balde	Thomas	Lille	clt124	07530102	veh003	Renault	Tourisme	resp03	Wayne	Tom	0607434	03/10/2017	13/05/2018	125
A01	Nantes	im122	Sylla	kevin	Nantes	clt230	06478645	veh001	Toyota	Utilitaire	resp01	Tavis	Jose	0712364	10/02/2017	25/02/2017	80
A04	Lyon	im634	Maalox	mouji	Marseille	clt017	02212325	veh002	Mercedes	Tourisme	resp04	Piro	Naldo	0777725	14/06/2017	12/07/2018	55
A05	Rennes	im412	Amdu	Chacul	Lyon	clt111	06539603	veh002	Mercedes	Tourisme	resp05	Galtier	Romeo	0777725	25/12/2017	14/11/2018	100
A02	Paris	im199	Kiffi	Baloi	Paris	clt369	06257493	veh001	Peugeot	Utilitaire	resp02	Ferdi	Cesar	0695312	01/02/2018	13/05/2018	210
A01	Nantes	im269	Sow	Nicolas	Nantes	clt458	09523684	veh003	Nissan	Tourisme	resp01	Tavis	Jose	0712364	30/012017	11/03/2018	70
A03	Angers	50	Kodis	Malta	Angers	clt499	06521493	veh004	Mercedes	Transport	resp03	Wayne	Tom	0607434	14/01/2018	16/11/2018	250
A05	Rennes	im816	Cenic	John	Rennes	clt363	07526348	veh003	Peugeot	Transport	resp05	Galtier	Romeo	0777725	05/02/2018	25/10/2018	230

Ces quelques tuples permettent de se rendre compte de certaines relations entre attributs et d'avoir un échantillon des valeurs possibles des différents attributs. On peut aussi s'imaginer certains déclencheurs ou changements automatiques de valeurs qui seraient implémentés en PL/SQL, par exemple le changement du prix de location d'un type de véhicule ou de l'achat d'un nouveau véhicule qui doit être ajouter dans la relation.

Nous nous rendons compte des dfs assez intuitivement, pour les voir plus simplement, il faut imaginer ces tuples comme la jointure naturelle de notre schéma final de tables.
Les autres dfs sont plutôt intuitives et concernent surtout un id et des valeurs qui y sont associées.

2 - 1 - Couverture minimale

Décomposition des dépendances fonctionnelles pour qu'elles soient élémentaires

DF = {

id_ag	→ ville_ag
id_ag	→ id_resp
id_ag	→ nom_resp
id_ag	→ prenom_resp
Id_ag, id_veh	→ marque_veh
Id_ag, id_veh	→ type_vehi
Id_ag, id_veh	→ cout_jour
id_resp	→ nom_resp,
id_resp	→ prenom_resp,
id_resp	→ tel_resp,
id_resp	→ id_ag
id_veh	→ marque_vehi
id_veh	→ type_veh
id_veh	→ cout_jour
type_veh, marque_veh	→ cout_jour
id_clt	→ nom_clt
id_clt	→ prenom_clt
id_clt	→ adr_clt
id_clt	→ tel_clt
id_clt, nom_clt	→ prenom_clt
id_clt, id_veh	→ marque_veh
date_loc, date_retour, id_clt	→ id_ag
date_loc, date_retour, id_ag, id_veh	→ id_clt
date_loc, date_retour, id_ag, id_veh	→ id_resp
im_veh	→ type_veh
im_veh	→ id_ag

}

Élimination des attributs superflus à gauche

Prenons par exemple cette df : {id_ag, id_veh → marque_veh }
Vérifions que les attributs à gauche ne sont pas superflus en calculant leur fermeture :

{ id_ag } +
Id_ag
Ville_ag_ag
Id_resp
Nom_resp
Prenom_resp
Tel_resp

{ id_veh } +
id_veh
marque_veh
type_veh
cout_jour

Ici nous remarquons que dans { id_veh }+ on retrouve l'attribut de droite, donc id_ag est superflu à gauche. Ce qui nous amène à le supprimer.
Faisons de même avec cet ensemble : { date_loc, date_retour, id_clt → id_ag }

{ date_loc } +
date_loc

{ date_retour } +
date_retour

{ id_clt } +
id_clt
nom_clt
prenom_clt
tel_clt
adr_clt

Nous n'obtenons ni l'autre attribut de gauche, ni les attributs de droite, ces deux attributs ne sont donc pas superflus. On ne supprime aucun attribut.
On fait de même pour toutes les autres dépendances qui ont plus d'un attribut à gauche.

- Élimination des dfs redondantes

Enfin nous vérifions qu'aucune df ne soit redondante en calculant la fermeture de l'attribut à gauche sans tenir compte de cette df en cours d'analyse :
Prenons par exemple l'ensemble de dfs suivant : { id_veh → cout_jour }

{ id_veh } +
Type_veh
Cout_jour
Marque_veh

Nous retrouvons les attributs droites dans la fermeture, donc cette df est donc redondante. Elle est supprimée.

De même avec : { date_loc, date_retour, id_ag, id_veh → id_clt}

{ date_loc, dat_retour, id_ag, id_veh} +
date_loc, dat_retour, id_ag, id_veh
ville_ag, id_resp, nom_resp, prenom_resp
marque_veh, type_veh, cout_jour

L'attribut de droite n'apparaît pas dans le résultat, donc elle est nécessaire.
On fait pareil pour toutes les autres dépendances et finalement on obtient la couverture minimale suivante :

CV(DF) = CV(DF) = {

1. id_ag → ville_ag;
2. id_ag → nom_resp;
3. id_ag → pren_resp;
4. id_ag → tel_resp;
5. Id-veh → marque_veh;
6. Id-veh → type_veh;
7. Id-veh → cout_jour;
8. id_clt → nom_clt;
9. id_clt → pren_clt;
10. id_clt → tel_clt;
11. Id_clt → adr_clt;
12. Date_loc , date_retour , imm_veh_veh → id_clt;
13. Imm_veh_veh → id_veh
14. Imm_veh_veh →id_ag

}

2 – 2 - Détermination de la clé

Après analyse de la couverture de l'ensemble { Date_loc , date_retour ,imm_veh_veh}, nous constatons qu'il détermine l'univers, il est unique et minimal, donc il est l'unique clé de notre relation.

Vérifions si c'est exact en calculant la fermeture de cet ensemble :

{ date_loc, date_retour, imm_veh_veh } +
id_ag, viile, id_resp, nom_resp, pren_resp,
tel_resp,id_veh, marque_veh, type_veh,
cout_jours, id_clt, nom_clt, prenom_clt,
adr_clt, tel_clt, date_loc, date_retour, imm_veh_veh

{ date_loc, date_retour, imm_veh_veh }+ = U, de plus cet ensemble est minimal, { date_loc, date_retour, imm_veh_veh } est bien la clé.

2 – 3 - Algorithme de Bernstein

- La couverture minimale a déjà été faite précédemment.

- Partition de CV(DF) en 5 groupes DFi (1 i 5) :

En regroupant les dfs ayant la même partie gauche, on obtient :

DF1={**id_ag** → ville_ag, id_resp, nom_resp, pren_resp, tel_resp}
DF2={**id_veh**, marque_veh, type_veh, cout_jour}
DF3={**(id_client**, nom_client, prenom_client, adr_client, tel_client}
DF4={**date_loc, date_retour, imm_veh_veh**, id_client}
DF5={**imm_veh_veh**, id_veh, id_ag}

- Construction des schémas <Ri(Ui), DFi> (1 i 6) :

< R1 = { idequipe, nomequipe, nbvictoire, nbdefaite, nbnul } , DF1 >
<R1={**id_ag**, ville, id_resp, nom_resp, pren_resp, tel_resp}, DF1>
<R2={**id_veh**, marque_veh, type_veh, cout_jour}, DF2>
<R3={**id_client**, nom_client, prenom_client, adr_client, tel_client}, DF3>
<R4={**date_loc, date_retour, imm_veh**, id_client}, DF4>
<R5={**imm_veh**, id_veh, id_ag}, DF5>

Ajout d'un schéma contenant la clé.

Ici on a pas besoin de créer une nouvelle relation pour contenir la clé, car elle est apparaît dans a relation R4.

2 – 4 - Algorithme par décomposition

- Principe :

Nous partons de la couverture minimale des dépendances fonctionnelles CV(Df) et de la relation R.
1°) Dans R, on extrait les attributs de la dépendance fonctionnelle DF1 pour constituer une table nommée R1 ayant pour clé, celle de la dépendance fonctionnelle DF1 (ici **id_ag**).

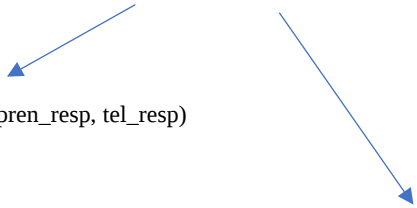
2°) **id-ag** étant la partie gauche de DF1, alors nous le ramenons dans R pour obtenir une nouvelle relation que nous avons appelée R' qui elle est constituée des attributs des autres DFs et de la clé (**id-ag**) de DF1.

En itérant cette opération pour les dépendances fonctionnelles DF2, DF3, DF4 et DF5 nous avons obtenu respectivement les tables R2, R3, R4 et R5.

A noter également, lors de cette décomposition, R4 contient la clé { **date_loc, date_retour, imm_veh_veh** } de la relation , du coup il n'y a pas eu de la création d'une nouvelle table contenant uniquement ces attributs avec DF = {}.

Nous obtenons ainsi la décomposition ci-dessous. Pour les DFi confère à la page 5 et voir CV(DF)

R(**id_ag**, ville, id_resp, nom_resp, pren_resp, tel_resp,**id_veh**, marque_veh, type_veh, cout_jours, **id_client**, nom_client, prenom_client, adr_client, tel_client, **date_loc, date_retour, imm_veh**)



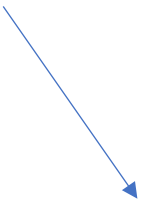
DF1

Latin

R1(**id_ag**, ville, id_resp, nom_resp, pren_resp, tel_resp)

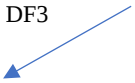
R'(**id_ag, id_veh**, marque_veh, type_veh, cout_jours, **id_client**, nom_client, prenom_client, adr_client, tel_client, **date_loc, date_retour, imm_veh**)

DF2

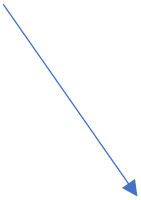


R2(**id_ag, id_veh**, marque_veh, type_veh, cout_jours)

R''(**id_ag, id_veh, id_client**, nom_client, prenom_client, adr_client, tel_client, **date_loc, date_retour, imm_veh**)

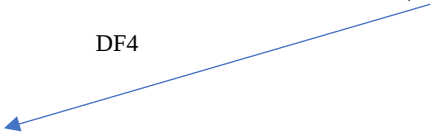


DF3

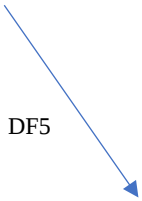


R3(**id_client**, nom_client, prenom_client, adr_client, tel_client)

R'''(id_ag, id_veh, date_loc, date_retour, imm_veh, id_client)



DF4



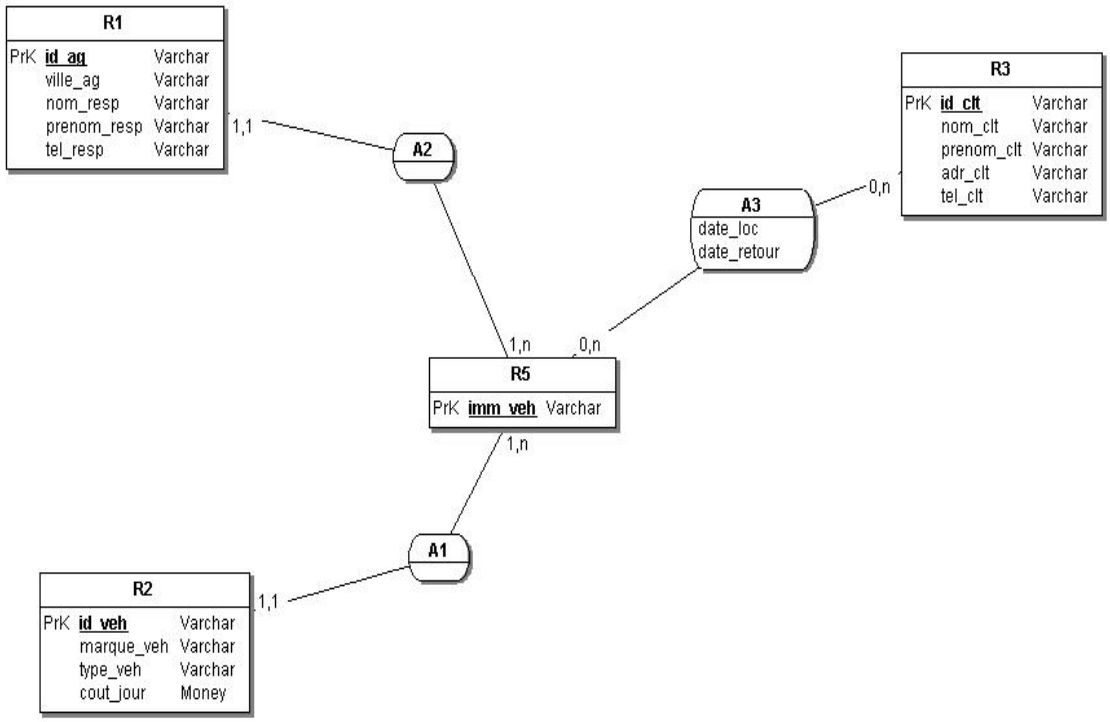
R4(**date_loc, date_retour, imm_veh**, id_client)

DF5

R5(**imm_veh**, id_ag, id_veh)

L'algorithme par décomposition nous permet de dégager 5 tables identiques à celles obtenues par l'algorithme de Bernstein.

3 - Modèle Relationnel de notre base de données



4 - Explication des fonctions, triggers, vues

4 – 1 - Création des rôles

Pour la création des rôles nous sommes partis sur le modèle RBAC.
En considérant l' administrateur disposant de tous les privilèges, nous avons créé trois 3 rôles que nous avons appelés :

R_alpha :	CREATE ROLE R_alpha;
R_kevin:	CREATE ROLE R_kevin;
R_oury :	CREATE ROLE R_oury;

A ces rôles nous avons attribués les privilèges suivants :

R_alpha : SELECT et INSERT sur toutes les tables

GRANT SELECT, INSERT ON Locations, Vehicules, TypesVehicule, Agences, Client TO R_alpha;

R_kevin : SELECT et DELETE sur toutes les tables

GRANT SELECT, DELETE ON Locations, Vehicules, TypesVehicule, Agences, Client TO R_kevine;

R_oury: SELECT et UPDATE sur toutes les tables
GRANT SELECT, UPDATE ON Locations, Vehicules, TypesVehicule, Agences, Client TO R_oury;

Attribution des rôles aux différents utilisateurs:

Alpha: GRANT R_alpha TO L3_24;
Kevin: GRANT R_kevin TO L3_35;
Oury: GRANT R_oury TO L3_22;

4 – 2 - Fonction

Cette procédure permet à un client qui a déjà effectué une location de modifier les dates de début et de retour de location.
Pour cela nous vérifions dans un premier temps que le client a bien loué le véhicule dont il souhaite modifier les dates de location et sinon on affiche un message d'erreur.

- procedure dates_update

```
CREATE OR REPLACE PROCEDURE dates_update is
  nb integer;
  new_date_deb Locations.date_loc%TYPE;
  new_date_ret Locations.date_retour%TYPE;
  matricule_veh Locations.imm_veh%TYPE;
  id_clt Locations.id_client%TYPE;
  error_modif EXCEPTION;
  BEGIN
    id_clt := '&id_client';
    new_date_deb := '&date_loc';
    new_date_ret := '&date_retour';
    matricule_veh := '&imm_veh';
    SELECT COUNT(*) INTO nb FROM Locations
    WHERE imm_veh = matricule_veh AND id_client = id_clt;
    IF(nb = 0) THEN RAISE error_modif;
    ELSE
      UPDATE Locations
      SET date_loc = new_date_deb , date_retour = new_date_ret
      WHERE id_client = id_clt AND imm_veh = matricule_veh;
    END IF;
    EXCEPTION
      WHEN error_modif THEN
        DBMS_OUTPUT.PUT_LINE('Ce client n a effectue aucune Locations');
  END dates_update;
/
SHOW ERROR
EXEC dates_update;
```

4 – 3 - Vues

Une vue est une requête stockée qui est interrogée comme une table. Elle assure une indépendance entre la couche SGBD et la couche applicative. (cf. Cours)
Pour ce projet, on a jugé nécessaire de créer deux vues, « historique_clients » et « Facture_location ».

- vue « historique_clients »:

Cette vue nous permet de stocker l'historique de tous les clients avec les informations des véhicules loués et les dates (location et retour).

```
CREATE OR REPLACE VIEW historique_clients AS
SELECT id_client,nom_client,prenom_client,imm_veh,date_loc,date_retour
FROM Client NATURAL JOIN Locations
ORDER BY id_client;
```

- vue « Facture_location » :

Au moment de l'analyse de l'existant, pour une optimisation de la mémoire, on a pris le soin de ne pas stocker le coût total d'une location. Le coût total étant le coût par jour d'un type donné de véhicule multiplié par la durée de la location en jour. C'est un élément calculable, donc pas besoin de le stocker.
On a fait cette vue pour faciliter aux utilisateurs (éviter d'utiliser une calculatrice). Cette vue permettra d'afficher les clients, leurs locations et le prix total.

```
CREATE OR REPLACE VIEW Facture_location AS
SELECT nom_client,prenom_client, id_veh, cout_jour, TO_DATE(date_retour, 'DD/MM/YYYY') TO_DATE(date_loc,'DD/MM/YYYY') as
Nb_Jour, (TO_DATE(date_retour, 'DD/MM/YYYY') - TO_DATE(date_loc, 'DD/MM/YYYY'))* cout_jour as Montant_total
FROM TypesVehicule NATURAL JOIN Vehicules NATURAL JOIN Locations NATURAL JOIN Client;
```

4 - 4 - Triggers

- trigger_location_invalid_date

TRIGGER qui assure que dans la relation Locations, on empêche l'insertion ou la modification d'un tuple si la date de location d'un véhicule est supérieure à la date de retour.

```
CREATE OR REPLACE TRIGGER trigger_location_invalid_date
    BEFORE INSERT OR UPDATE ON Locations
    FOR EACH ROW
      DECLARE
      invalid_date EXCEPTION;
    BEGIN
       IF :new.date_loc > :new.date_retour THEN RAISE invalid_date; END IF;
       EXCEPTION
       WHEN invalid_date THEN
       RAISE_APPLICATION_ERROR(-20001, 'Inserer une date de retour superieure a la date de Location');
  END;
/
SHOW ERROR
```

- trigger_vehicule_indisponible

TRIGGER qui assure que dans la relation Locations, on empêche la location d'un Véhicule si celui-ci est en cours de location.
Pour cela, pour un Véhicule donné, on vérifie si sa date de retour est supérieure à la nouvelle date de location entrée, si c'est le cas alors nous levons une exception.

```
CREATE OR REPLACE TRIGGER trigger_vehicule_indisponible
  BEFORE INSERT OR UPDATE ON Locations
  FOR EACH ROW
  DECLARE
  veh_indisponible EXCEPTION;
  cursor c1 is SELECT imm_veh,date_retour FROM Locations;
  ligne c1% rowtype;
  BEGIN
    FOR ligne in c1 LOOP
      IF ligne.imm_veh = :new.imm_veh AND ligne.date_retour > :new.date_loc THEN
        RAISE veh_indisponible;
      END IF;
    END LOOP;
    EXCEPTION
      WHEN veh_indisponible THEN
        RAISE_APPLICATION_ERROR(-20001, 'Ce Vehicule est en cours de location');
  END;
/
SHOW ERROR
```

5 - Atouts, améliorations possibles et critiques de notre travail

5 – 1 - Atouts

Notre base de données offres plusieurs atouts dont entre autres :

- Faire l'historique des clients qui ont effectué une location ;
- D'empêcher de louer un véhicule qui est déjà en cours de location ;
- D'empêcher une location dont la date de début est supérieure à celle de retour ;
- Possibilité d'établir une facture résumant le coût total et la durée de location d'un véhicule par un client.

5 – 2 – Améliorations

Comme amélioration de notre travail , par exemple On pourrait faire des statistiques pour voir les clients qui ont effectué le plus de locations pour pouvoir leur proposer des promotions ou tout autres avantages.
Également nous aurions pu créer un index qui nous permettra d'optimiser les recherches dans les différents objets de notre base de données.

Conclusion

Ce projet nous a permis de découvrir les différentes étapes de réalisation d'une base de données et de se familiariser avec les outils (trigger, vue, ...) du SGBD ORACLE. Dans ce projet nous avons aussi approfondi nos connaissances théoriques déjà acquises en les mettant en pratique.

Par la suite nous nous sommes aussi rendu compte que certaines choses qui avaient été choisies au début du projet n'étaient pas forcément utile, et que, dans le cas contraire, il nous manquait certains choses (ex : des triggers), montrant la constante évolution d'une base de donnée au cours d'un projet.