

NF18 Projet – Partie 2

Adaptation et optimisation en noSQL

Nous avons choisi de modifier notre implémentation de base de données à deux endroits, en se servant du format JSON.

1) Véhicule - Modèle - Catégorie - Marque

a) Analyse du besoin :

- des véhicules sont souvent ajoutés à la bdd et il y en a donc beaucoup
- les bilans nécessitent de faire des jointures sur toutes ces classes, ce qui est assez coûteux
- une redondance des attributs (catégorie etc) n'est pas dérangeante

b) Travail sur les dépendances fonctionnelles et la normalisation :

- Véhicule (#immat:string, #modele ⇒ Modele(nom), carburant ⇒ TypeCarburant(nom), couleur:string, nb_km : float; agence ⇒ Agence(idAgence), agent_tech ⇒ AgentTechnique(idEmploye)); /// nb_km not null

Dépendances fonctionnelles :

- (#immat,#modele) → carburant
 - (#immat,#modele) → couleur
 - (#immat,#modele) → nb_km
 - (#immat,#modele) → agence
 - (#immat,#modele) → agent_tech
 - agent_tech → agence (par transitivité)
-
- 1NF - OUI : La table possède une clé (#immat,#modele) et tous les attributs de la table sont atomiques.

- 2NF - OUI : La clé #immat, #modele. Or, tous les attributs de la table peuvent être déduits de la sous-partie de la clé #immat (exemple: la DF #immat → couleur).
- 3NF - NON : Si on s'intéresse à la fermeture transitive des DF, AgentTechnique → Employe → Agence donc AgentTechnique → Agence par transitivité

→ Véhicule est 2NF

Afin de transformer la classe véhicule en 3NF, on pourrait tout simplement enlever l'attribut Agence de la relation que l'on peut retrouver en faisant une jointure entre les tables AgentTechnique et Employe puis entre les tables Employe et Agence.

- **Option (#nom:string);**
 - 3NF - OUI: La relation contient un seul attribut atomique clé.
 - Option est 3NF
- **AssociationOptionsVehicule (#vehicule ⇒ Vehicule(immat), #option ⇒ Option(nom));**
 - 3NF - OUI: La relation contient un seul attribut atomique clé.
 - AssociationOptionsVehicule est 3NF
- **TypeCarburant (#nom:string);**
 - 3NF - OUI: La relation contient un seul attribut atomique clé.
- **Marque (#nom:string);**
 - 3NF - OUI: La relation contient un seul attribut atomique clé.
- **Modele (#nom:string, #marque ⇒ Marque(nom), #categorie ⇒ Categorie(nom), nb_portes : int);**

Dépendances fonctionnelles :

 - (#nom, #marque, #categorie) → nb_portes
 - 1NF - OUI : La table possède une clé (#nom,#marque,#categorie) et tous les attributs de la table sont atomiques.
 - 2NF - OUI
 - 3NF - OUI

→ Modele est 3NF

- **Categorie(#nom: string, description : string);**

Dépendances fonctionnelles :

- (#nom) → description

- 1NF - OUI : La table possède une clé (#nom) et tous les attributs de la table sont atomiques.
- 2NF - OUI
- 3NF - OUI

→ Categorie est 3NF

c) Passage en noSQL des tables concernées :

Afin de limiter les jointures nous choisissons de transformer les relations Vehicule-Modele, Vehicule-Categorie en attributs de type JSON, grâce à l'imbrication :

```
CREATE TABLE Vehicule(
  immat VARCHAR(7) UNIQUE,
  modele JSON,
  liste_options JSON,
  carburant VARCHAR NOT NULL,
  couleur VARCHAR NOT NULL,
  nb_km DECIMAL NOT NULL,
  agence INTEGER NOT NULL,
  agent_tech INTEGER,
  PRIMARY KEY(immat),
  FOREIGN KEY(carburant) REFERENCES TYPECARBURANT(nom),
  FOREIGN KEY(agence) REFERENCES AGENCE(id_agence),
  FOREIGN KEY(agent_tech) REFERENCES AGENTTECHNIQUE(id_employe),
  CONSTRAINT check_immat CHECK(immat SIMILAR TO '[0-9A-Z]{7}'));
```

- Fichier Json Modèle correspondant : (utilisation d'une imbrication)

```
'{
  "nom":"C4",
  "nb_portes":4,
  "marque":{
    "nom":"citroen"
  }
  "categorie":{
    "nom":"berline",
    "description":"voiture de ville",
  }
}'
```

- Fichier Json liste_options correspondant : (utilisation du tableau car c'est un attribut multivalué)

```
'["climatisation","limiteur de vitesse","sieges massants"]'
```

- Insertion dans la table véhicule :

```
INSERT INTO Vehicule (immat, modele, liste_options, carburant, couleur, nb_km, agence, agent_tech)
VALUES (
'AA123AA',
'{"nom":"C4", "nb_portes":4, "marque":{"nom":"citroen"},
  "categorie":{"nom":"berline", "description":"voiture de ville"}}',
'["climatisation","limiteur de vitesse","sieges massants"]',
'essence',
'bleu',
3000,
1,
2
);
```

d) Exemples de requêtes et affichages

```
SELECT v.immat, opt.* -- permet d' afficher toute les options pour un même véhicule
FROM Vehicule v, JSON_ARRAY_ELEMENTS(v.liste_options) opt;
```

-- trouver la catégorie d un véhicule

```
SELECT v.immat, CAST (v.modele->'categorie'->'nom' AS VARCHAR) AS categorie, CAST(v.modele->'categorie'->'description' AS VARCHAR) AS description
FROM Vehicule v;
```

-- trouver le modele d un véhicule

```
SELECT v.immat, CAST(v.modele->'nom' AS VARCHAR) AS modele, CAST(v.modele->'nb_portes' AS INTEGER) AS nbre_portes
FROM Vehicule v;
```

--trouver la marque d'un véhicule

```
SELECT v.immat, CAST(v.modele->'marque'->'nom' AS VARCHAR) AS marque
FROM Vehicule v;
```

- résultat de la requête sur les options:

Data Output		
	immat [PK] character varying (7)	value json
1	AA123AA	"climatisation"
2	AA123AA	"limiteur de vitesse"
3	AA123AA	"sieges massants"

- résultat de la requête sur la catégorie:

Data Output			
	immat [PK] character varying (7)	categorie character varying	description character varying
1	AA123AA	berline	voiture de ville

- résultat de la requête sur le modèle:

Data Output

	immat [PK] character varying (7)	modele character varying	nbreportes integer
1	AA123AA	C4	4

- résultat de la requête sur la marque :

Data Output

	immat [PK] character varying (7)	marque character varying
1	AA123AA	citroen

e) Exemple de l'application python

Nous avons implémenté une fonction python avec la requête permettant de relever toute les options présentent sur une voiture:

```
from utils import *
import json

def option_vehicule():
    sql = "SELECT v.immat, opt.* FROM Vehicule v, JSON_ARRAY_ELEMENTS(v.liste_options) opt"
    curseur.execute(sql)
    raw = curseur.fetchone()
    while raw:
        print("immatriculation: ", raw[0], "option:", raw[1])
        raw = curseur.fetchone()
```

voici le résultat obtenu:

```
Sélectionnez une option :
1. Afficher liste des véhicules
2. Ajouter un véhicule
3. Ajouter une location
4. Annuler une location
5. Modifier une location
6. Valider une location
7. Payer une facturation
8. Contrôler un entretien
9. Bilan par client
10. Bilan par véhicule
11. Bilan par catégorie
12. Trace des agents
13. Afficher les options de tous les véhicules
0. Quitter le programme
> 13
immatriculation: AA123AA option: climatisation
immatriculation: AA123AA option: limiteur de vitesse
immatriculation: AA123AA option: sièges massants
```

2) Location - AgentCommercial - ValidationFinale

a) Analyse du besoin

Nous avons choisi de traduire la relation AgentCommercial - Location en classe d'association afin d'ajouter les attributs date_validation et resultat_validation, bien que ce soit une association 1:N.

Cependant, avec les moyens que nous connaissons aujourd'hui (attribut de type JSON), nous pouvons traduire l'information de validation finale comme attribut de la classe Location.

Cette nouvelle implémentation est plus simple, plus efficace (pas besoin de jointure), et plus fidèle à la réalité : en effet, il suffit de la donnée location pour déduire sa date de validation et le résultat de sa validation. Il suffit ensuite de rajouter une clé étrangère AgentCommercial à la classe Location.

b) Travail sur les dépendances fonctionnelles et la normalisation :

- **Location** (#idContrat:int, date_debut:date, date_fin:date, km_parcourus:float, vehicule=>Vehicule(immat), agent_com=>AgentCommercial(idEmploye), entretien=>Entretien(idEntretien), facturation=>Facturation(idFacturation));

DF: idContrat → date_debut
idContrat → date_fin
idContrat → km_parcourus
idContrat → vehicule
idContrat → entretien
idContrat → facturation

3NF: La relation contient une seule clé atomique et tous les autres attributs dépendant uniquement de cette clé

- **ValidationFinale**(#agent_com=>AgentCommercial(idEmploye), #location=>Location(idContrat), date_validation:date, resultat_validation:boolean);

DF:
#agent_com, #location → date_validation
#agent_com, #location → resultat_validation

1NF : Tous les attributs sont atomiques

2NF : Tous les attributs sont déterminés par la clé entière

3NF: il n'y a pas de dépendances entre attributs non clés

c) Passage en noSQL des tables concernées :

```
CREATE TABLE Location(  
  id_contrat SERIAL PRIMARY KEY,  
  date_debut DATE NOT NULL,  
  date_fin DATE,  
  km_parcourus DECIMAL NOT NULL,  
  vehicule_immat VARCHAR(7) NOT NULL,  
  entretien INTEGER UNIQUE NOT NULL,  
  facturation INTEGER NOT NULL,  
  agent_com INTEGER NOT NULL,  
  validationFinale JSON,  
  FOREIGN KEY(vehicule_immat) REFERENCES Vehicule(immat),  
  FOREIGN KEY(entretien) REFERENCES Entretien(id_entretien),  
  FOREIGN KEY(facturation) REFERENCES Facturation(idFacturation),  
  FOREIGN KEY(agent_com) REFERENCES AgentCommercial(id_employe)  
);
```

- Insertion dans la table véhicule :







```
INSERT INTO Location (date_debut, date_fin, km_parcourus, vehicule_immat, entretien, facturation, agent_com, validation_finale)  
VALUES (  
  '2021-09-18',  
  '2022-01-02',  
  1004,  
  'AA123AA',  
  1,  
  1,  
  3,  
  '{"date":"2021-06-15", "resultat":"TRUE"}'  
);
```

d) Exemples de requêtes

```
-- trouver la validation finale d'un vehicule  
SELECT L.id_contrat, CAST(L.validation_finale->>'resultat' AS VARCHAR) AS resultat,  
       CAST(L.validation_finale->>'date' AS DATE) AS date, E.nom, E.prenom  
FROM Location L join Employe E on L.agent_com=E.id_employe;
```

- Résultat :

Data Output

	 id_contrat integer 	resultat character varying 	date date 	nom character varying 	prenom character varying 
1	1	TRUE	2021-06-15	Lerner	Thomas