Conception d'une base de données de gestion d'une concession automobile

21.09.2021

CUINET Maxence / RADJAH Sofiane
Toulon

Cahier des Charges

Contexte

Suite à l'augmentation des ventes de voitures, les concessionnaires automobiles se doivent de répondre à la demande des clients.

L'objectif est donc de proposer une application Web permettant la gestion d'une concession automobile à travers l'utilisation d'une base de données PostgreSQL.

Cette base de données leur permettra d'ajouter et de connaître l'ensemble des informations sur les véhicules neufs ou d'occasions dont dispose l'agence, les caractéristiques et options des voitures mais aussi les informations clients et vendeurs.

Problématique

1. Mettre en place une base de données PostgreSQL répondant au mieux au besoin.
2. Mettre en place une application Web (PHP) permettant d'accéder aux données stockées.

Besoins

I. Besoins fonctionnels

Accéder aux données stockées et les afficher.

Enregistrer de nouvelles données.

Permettre la gestion de la concession via l'application.

Historiser les actions effectuées dans la concession par les vendeurs ou clients

II. Contraintes techniques

Base de donnée PostgreSQL

Langage de programmation PHP

III. Livrables

- Une application Web en PHP permettant la gestion de la concession utilisant une base de donnée PostgreSQL

Conception

Conception

I. Dictionnaire des données

Voici le dictionnaire regroupant toutes les données que nous aurons à conserver dans notre base (et qui figureront donc dans le MCD).

Les données sont les suivantes :

1. id_cl
2. nom_cl
3. prenom_cl
4. pays_cl
5. ville_cl
6. adresse_cl
7. id_vend
8. nom_vend
9. prenom_vend
10. pays_vend
11. ville_vend
12. adresse_vend
13. id_cmd
14. date_cmd
15. prix_total_cmd
16. id_voit
17. prix_voit
18. date_construct_voit
19. energie_voit
20. kilometrage_voit
21. id_voit_type
22. nom_voit_type
23. couleur_base
24. prix_base
25. immatriculation
26. couleur
27. date_premiere_mise_circulation
28. id_marq
29. nom_marq
30. id_mod
31. nom_mod
32. id_opt
33. nom_opt
34. prix_opt

II. Modèle Entité Association (MEA) et schéma relationnel

➢ Schéma relationnel

Le modèle relationnel est une manière de modéliser les relations existantes entre plusieurs informations, et de les ordonner entre elles. On identifie les clés primaires de chaque table en les soulignant et les clés étrangère sont précédées du symbole “#”

CLIENT (id_cl, nom_cl, prenom_cl, pays_cl, ville_cl, adresse_cl)

VENDEUR (id_vend, nom_vend, prenom_vend, pays_vend, ville_vend, adresse_vend)

COMMANDE (id_cmd, date_cmd, prix_total_cmd, #id_cl)

VOITURE (id_voit, prix_voit, date_construct_voit, energie_voit, kilometrage_voit, #id_cl, #id_cmd, #id_voit_type, #id_mod)

VOITURE_TYPE (id_voit_type, nom_voit_type)

NEUVE(couleur_base, prix_base)INHERITS VOITURE

OCCASION (immatriculation, couleur)INHERITS VOITURE

OPTION (id_opt, nom_opt, prix_opt)

MODELE (id_mod, nom_mod, #id_marq)

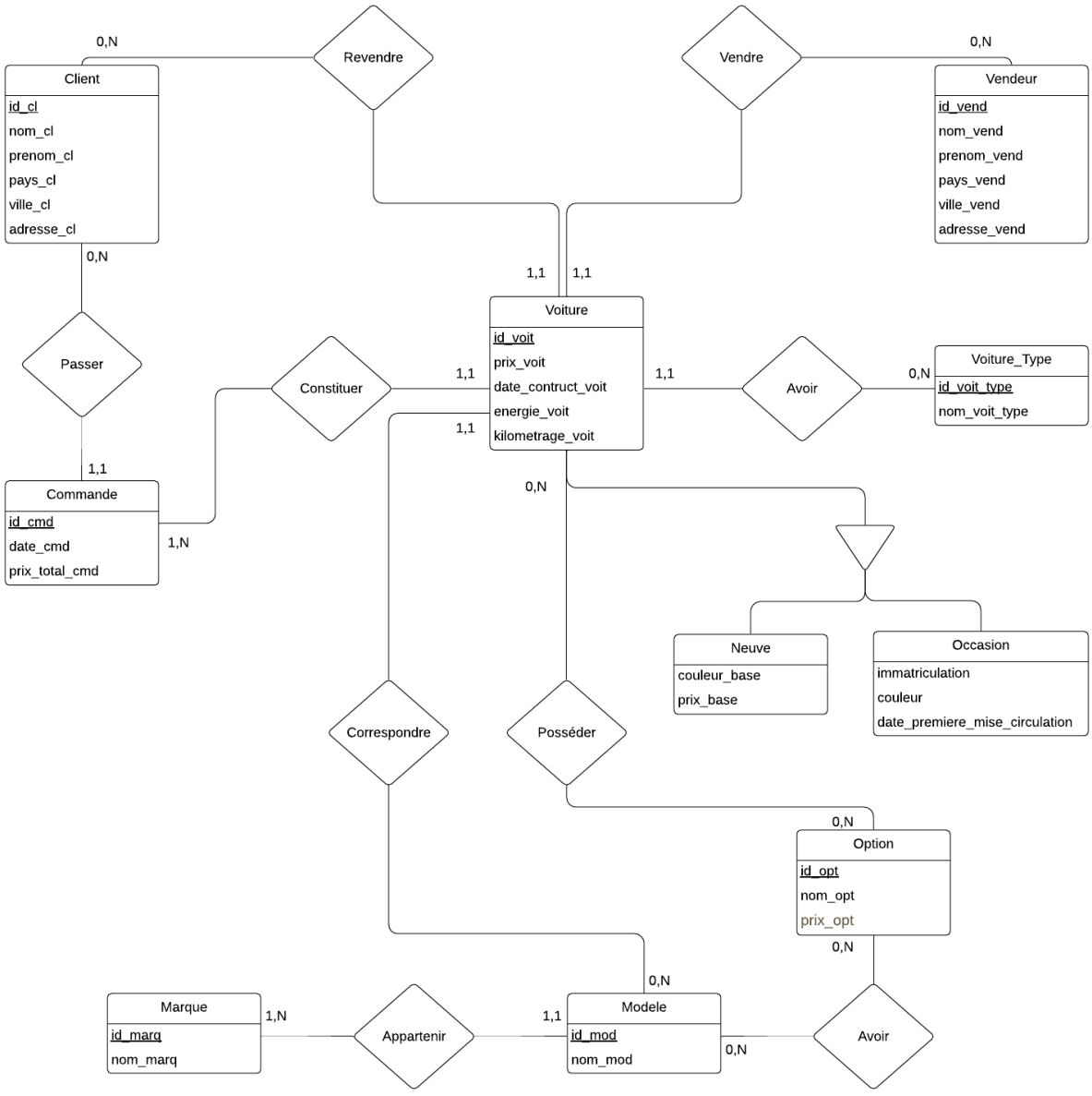
MARQUE (id_marq, nom_marq)

VOITURE_HAS_OPTION (#id_voit, #id_opt)

MODELE_HAS_OPTION (#id_opt, #id_mod)

➢ Modèle Entité Association (MEA)

Le diagramme entité-association est un type d'organigramme illustrant la façon dont des « entités » telles que des personnes, objets ou concepts sont liées les unes aux autres.



III. Contraintes particulières (NOT NULL ; valeur > 0 ; domaines de valeurs ; ...)

SQL permet de définir des contraintes sur les colonnes et les tables. Les contraintes donnent autant de contrôle sur les données des tables qu'un utilisateur peut le souhaiter. En effet, si un utilisateur tente de stocker des données dans une colonne en violation d'une contrainte, une erreur est levée. Cela s'applique même si la valeur vient de la définition de la valeur par défaut.

Une contrainte NOT NULL indique simplement qu'une colonne ne peut pas prendre la valeur NULL. Dans ce cas d'étude l'ensemble des champs des tables de la base ont une contrainte NOT NULL appliquée, hormis les clés étrangères "id_cli" et "id_vend" dans la table VOITURE. En effet si "id_cli" est NULL il s'agira d'une voiture vendue par un vendeur et à l'inverse si "id_vend" est NULL alors il s'agira d'une vente faite par le client.

Une contrainte de type clé primaire indique qu'une colonne, ou un groupe de colonnes, peut être utilisée comme un identifiant unique de ligne pour cette table. Ceci nécessite que les valeurs soient à la fois uniques et non NULL. Ainsi chaque table comporte une clé primaire identifiée dans le schéma relationnel par le soulignement de celle-ci.

Une contrainte de clé étrangère stipule que les valeurs d'une colonne (ou d'un groupe de colonnes) doivent correspondre aux valeurs qui apparaissent dans les lignes d'une autre table. On dit que cela maintient l'*intégrité référentielle* entre les deux tables. Ici toutes les tables de la base contiennent des clés étrangères désignées par le symbole "#" qui les précède à l'exception des tables CLIENT, VENDEUR, OPTION et MARQUE qui n'ont pas de clés étrangères.

Ici, pour notre cas d'étude, un problème se pose pour la mise en place de notre base de données sous PostgreSQL. En effet, la modélisation des tables VOITURE, NEUVE et OCCASION est correcte mais les contraintes de clés étrangères sur la table VOITURE_HAS_OPTION ne seront pas implémenté sur la base de donnée PostgreSQL à cause de la spécificité suivante :

L'héritage ne fait pas en sorte que les données soient présentes dans la table mère et la table fille. Si on réalise un SELECT sur la table mère, on aura dans le résultat de la requête les enregistrements de la table mère et de la table fille. Or si on ajoute la clause ONLY, les enregistrements présents dans la table fille n'apparaîtront pas. En effet, car selon la document de PostgreSQL : les clés étrangères ne vérifient que la table visée, pas les tables filles. C'est un des inconvénients majeurs du partitionnement avec PostgreSQL.

IV. Liste des fonctionnalités prises en charge par votre BD (fonctions, triggers)

La base de gestion d'une concession prend en charge des fonctionnalités permettant le bon fonctionnement de la base et permet de garder une certaine logique dans sa conception.

Les fonctions SQL permettent d'effectuer des requêtes plus élaborées.

De plus, les triggers, également appelés déclencheurs, permettent d'exécuter un ensemble d'instructions SQL juste après un événement.

- **Fonction calculant le kilométrage moyen du parc automobile de la concession**

```
CREATE FUNCTION KilometrageMoyen(numeric) RETURNS numeric
AS $$
    DECLARE kilometrage_moyen NUMERIC;
BEGIN
    SELECT round(avg(kilometrage_voit),2) FROM VOITURE WHERE id_cmd
    IS NULL INTO kilometrage_moyen;
    RETURN kilometrage_moyen;
END
    $$ LANGUAGE plpgsql;
```

- **Fonction calculant les ventes de voitures entre deux dates**

```
CREATE FUNCTION VenteVoiture(date_début date, date_fin date) RETURNS
numeric
AS $$
     DECLARE stat_voiture NUMERIC;
BEGIN
     SELECT COUNT(*) FROM COMMANDE WHERE date_cmd BETWEEN date_debut
     AND date_fin INTO stat_voiture;
     RETURN stat_voiture;
END
     $$ LANGUAGE plpgsql;
```

- **Fonction affichant l'ensemble des options disponibles pour un modèle**

```
CREATE FUNCTION AllOption(id_modele INTEGER)
AS $$
BEGIN
     RETURN(SELECT public."Option".id_opt FROM public."Option" INNER
     JOIN public."Modele_has_Option" ON public."Option".id_opt =
     public."Modele_has_Option".id_opt WHERE id_mod = id_modele);
END
     $$ LANGUAGE plpgsql;
```

- **Trigger calculant le prix total d'une commande en fonctions des options ajoutées**

```
CREATE TRIGGER AjoutOption AFTER INSERT
ON VOITURE_HAS_OPTION FOR EACH ROW
DECLARE id_cmd, prix_opt;
BEGIN
      SELECT id_cmd FROM VOITURE WHERE id_voit = NEW.id_voit INTO
      id_cmd;
      SELECT prix_opt FROM VOITURE_HAS_OPTION INNER JOIN OPTION ON
      VOITURE_HAS_OPTION.id_opt = OPTION.id_opt WHERE id_voit =
      NEW.id_voit INTO prix_opt;
      IF id_cmd IS NOT NULL
            THEN
                  UPDATE COMMANDE SET prix_total_cmd = prix_total_cmd +
                  prix_opt WHERE COMMANDE.id_cmd = id_cmd;
      END IF;
END
```