Projet de Base de Données - « Musiqlite »

par JAAFAR Joachim

Musiqlite est un organisme spécialisé dans le domaine musical. Une équipe composée de développeurs de Musiqlite ont reçu la tâche de créer une base de données se portant sur la réservation de billets pour les concerts qu'il sponsorise. Ce rapport décrit comment s'est déroulé la création de cette base de données.

Cet été, Musiqlite a obtenu 3 partenariats avec des concerts. Nous avons donc crée un système de réservation de billets pour eux grâce au langage de programmation Java et au langage de gestion de base de données SQLite.

Nous avons séparé la réservation, modification et suppression de billets en deux parties, chacune représentée par une interface graphique :

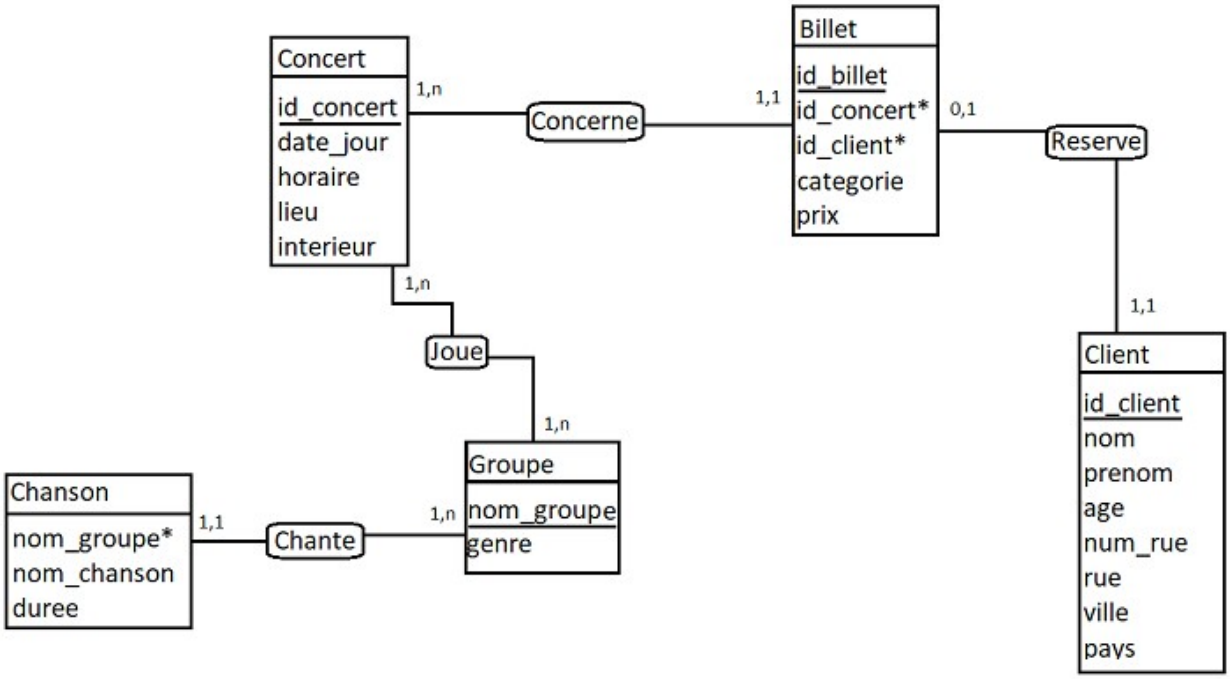
- côté client. Une personne peut réserver un billet en sélectionnant le concert ainsi que le type de place souhaités (fosse, standard ou VIP).
Il doit pour cela entrer ses coordonnées (prénom, nom, âge, adresse complète) afin d'être enregistré dans la base de données en tant que client (si le client existe déjà, c'est à dire si de nouvelles coordonnées renseignées correspondantes exactement à des coordonnées déjà enregistrées, la base de données reconnaît alors qu'il s'agit de la même personne). Puis il spécifie quel type de place (Fosse, Standard ou VIP) et quel concert il veut assister. La base de données vérifie s'il existe un billet de ce type encore disponible pour ce concert. Il avertit le client quand le compte à son nom vient d'être crée, si les informations entrées sont correctes, si le billet choisi est disponible et s'il a été crée.

- côté serveur. Le gérant de la base de données y a un accès total : il peut lister toutes les données disponibles, modifier certaines données d'un billet sur demande d'un client.

Exemple: un client veut annuler une réservation d'un billet car il a pris une place «Fosse» au lieu de «Standard». Si un billet correspondant à la demande du client existe, le gérant n'aura qu'à retirer l'identifiant du client de la table Billet, au tuple représentant le billet réservé (en réalité, il le changera par 0, correspondant par «null»). Le client n'aura plus qu'à recommander le billet qu'il souhaite, s'il en reste. Ce changement se fera grâce à des boutons, et non par entrée direct du code SQLite. Le gérant du côté serveur peut aussi ajouter, modifier et supprimer des groupes et des chansons si l'organisateur du concert le prévient. Par exemple, si un groupe vient de confirmer sa venue, il peut ajouter ce groupe à l'un des concerts et y ajouter leur chansons. Les clients auront une modification en temps réel sur leur interface.

Modele Relationnel

Billet (id_billet, id_concert*, id_client*, categorie, prix)
Client(id_client, nom, prenom, age, num_rue, rue, ville, pays)
Concert(id_concert, date_jour, horaire, lieu, interieur)
Groupe(nom_groupe, genre)
Chanson(nom_groupe*, nom_chanson, duree)
Joue(id_concert*, nom_groupe*)



Modele Entite Association

Types de chaque attribut, clés primaires, clés étrangères, contraintes :

```
create table Groupe (nom_groupe char(50) primary key NOT NULL,
                     genre char(50) NOT NULL DEFAULT 'Non Renseigne');

create table Chanson (nom_groupe char(50) REFERENCES Groupe(nom_groupe),
             nom_chanson char(50) primary key NOT NULL,
             duree time NOT NULL);

create table Concert(id_concert int primary key NOT NULL,
          date_jour date NOT NULL,
          horaire time NOT NULL,
          lieu char(50) NOT NULL,
          interieur int NOT NULL check (interieur in (0,1)));

create table Billet (id_billet int primary key NOT NULL,
      id_concert int NOT NULL REFERENCES Concert(id_concert),
      id_client int DEFAULT 0 REFERENCES Client(id_client),
      categorie char(10) NOT NULL,
      prix float DEFAULT NULL);

create table Client (id_client integer primary key AUTOINCREMENT NOT NULL,
       nom char(50) NOT NULL,
       prenom char(50) NOT NULL,
       age int NOT NULL,
       num_rue int NOT NULL,
       rue char(100) NOT NULL,
       ville char(100) NOT NULL,
       pays char(50) NOT NULL);

create table Joue (id_concert int NOT NULL REFERENCES Concert(id_concert),
   nom_groupe char(50) NOT NULL REFERENCES Groupe(nom_groupe),
   constraint pk_joue primary key(id_concert, nom_groupe));
```

Liste des déclencheurs :

```
-- Supprime chanson et joue lorque l'on supprime un groupe
create trigger del_groupe after delete on Groupe
begin
        delete from Joue where nom_groupe = old.nom_groupe;
        delete from Chanson where nom_groupe = old.nom_groupe;
end;

-- Update (réduit) le prix lorsqu'un client mineur commande un billet
create trigger upd_prix_billet_reduc after update on Billet when new.id_client > 0 and (select age from Client where id_client = new.id_client) < 18
begin
        update Billet set prix = prix-20 where id_billet = new.id_billet;
end;

-- Update (réinitialise) le prix lorsqu'un client mineur supprime un billet
create trigger upd_prix_billet_reinit after update on Billet when new.id_client = 0 and (select age from Client where id_client = old.id_client) < 18
begin
        update Billet set prix = prix+20 where id_billet = new.id_billet;
end;
```