RAPPORT DE PROJET BD6

Charly Martin Avila et Ana Parres Cerezo

1. Modèle entité-relation

1.1 UTILISATEUR

L'entité *Utilisateur* est représenté comme l'entité mère de *Assosiation*, *Groupe* et *Personne.* On avait decidé de les séparés suivant les specifications donnés sur le sujet.

Des relation d'*amitié* (symetrique) et de *follow* relie les utilisateurs entre eux.

Et *Playlist* et *Avis* sont relié avec *Utilisateur* par un entité faible, puisque chaque playlist et avis, appartiennet à un seul utilisateur.

IMPLEMENTATION:

Finalement lors de l'implementation on a décidé que les utilisateurs qui seront present sur l'entité *organise* seraient donc un organisateur, laisons ainsi la definition des rols aux activités réalisés par l'utilisateur sur le réseau social, et non pas imposer des rols à des utilisateurs.

De même pour personne, qui seraient celles qui sont présent dans la relation. La seule fille de l'entité mère survivante à l'implementation sont les groupes (on considere comme groupe les artistes, même en solo). On aurai voulu faire en sorte que n'importe qui étant l'auteur d'un morceau soit un artiste, et donc laissé la classe *Utilisateur* en entière sans distinctions de rols (sans entités filles). Cependant le fait que les tags puisse être que assignés à des groupes, et non pas aux autres utilisateurs, nous a obligé à garder l'entité *Groupe* sous forme de table.

CONTRAINTES EXTERNES: On n'a pa pu délimiter sur le SGBD le nombre de Playlist pour chaque utilisateur (10 au maximum). Ainsi que le nombre de morceaux maximum sur un Playlist (20). Cela sera à l'application de le faire

De même façon, que la PlayList d'un groupe contiennent que des morceaux du groupe, c'est une constrainte externe.

1.2 EVENEMENT

L'entité *Événement* est rattaché aux utilisateurs (*Assosiation*, *Personne*, *Groupe*) à travers les relations *organise, s'interese ou participe* (lors de l'implementation devuenu *interaction*) et *performance* (lors de l'implementation devuenu *concert*)

Elle est rattaché à l'entité *Lieu* par une relation *localisation.*

Evenement Passé herite d'*Évenement* , sur lequel on peut donner un *Avis* (*Avis_Concert*)

IMPLEMENTATION:

Une amelioration qu'on à realisé c'est au lieu de mettre un attribut line up, avec les groupes qui vont participer à l'événement, on utilise la table concert pour determiner les groupes qui participent à un concert.

1.3 AVIS

Un Avis doit avoir un id, un commentaire qui peut-être null et une note (de 1 à 5).

Avis possede des entités filles speccifique de chaque sujet (Groupe, Morceaux, Concert, Lieu) qui heritent d'elle.

Un avis possede l'id de l'*Utilisateur* et l'id du sujet particulier, exemple, *Avis_Morceau*, possede l'id du morceau. Le couple de ces id forment un couple unique, ce qui fait que une personne puisse donner un seul avis sur chaque sujet (comme Google Maps).

Avis et Tag sont relié par la relation *contient* (appelée relation_ta l'heure de l'implementation), qui permet avoir plusieurs tag dans un seul avis, et plusieurs avis contenent le même tag.

IMPLEMENTATION:

Verification de la note comprise entre 1 et 5.

1.4 GENRE

On a décidé d'implementer les genres comme l'exemple donné dans cours, sur les parent et les enfants (ref. Associations réflexives, page 42, 4.Modelisation). Alors il y a une relation *derive* qui tiens compte du genre principal et de ses sous-genres. Un sous-genre peut avoir plusieurs genre d'origine. Le rockpop apparaitra deux fois comme sous-genre, dans la table *derive* ayant une fois rock comme principal, et une deuxiemme fois pop comme principal.

2. Schema relationnel

Utilisateur(**id_user**, name_user, email)

Groupe(**id_user**, status_g)

Lieu(**id_place**, name_p, city_p, cp_p)

Evenement(**id_event**, id_p, date_ef, name_ef, prix_ef, kids_ef, ext_ef)

Evenement (**id_event**, #id_p, kids_ep, ext_ep)

Evenement_Passe (**id_event**, date_ep, name_ep, mult_ep)

Evenement_Futur (id_event, date_ef, name_ef)

Morceau(**id_s**, id_g, name_s, album, n_order)

Playlist(**id_pl**, name_pl, desc_pl)

Avis(**id_avis**)

Avis_Groupe(**id_avis**, #id_user, #id_grp, commentary, note_g)

Avis_Morceau(**id_avis**, #id_user, #id_song, commentary, note_m)

Avis_Evenement(**id_avis**, #id_user, #id_event, commentary, note_event)

Avis_Lieu(**id_avis**, #id_user, #id_place, commentary, note_l)

Tag(**id_tag**)

Tag_General(**id_tag**, text)

Tag_Lieu(**id_tag**, #id_p)

Tag_Groupe(**id_tag**, #id_grp)

Genre(**id_g**, #name_g)

concert(#**id_grp**, #**id_event**)

organise(#**id_user**,# **id_event**)

interaction(**id_i**, #id_user, #id_event, status_s)

follows(#**id_userFollows**, #**id_userFollowed**)

amitie(#**id_userA**, #**id_userB**)

pl_belongs(#**id_s**, #**id_pl**)

relation_ta(#**id_t**, #**id_avis**)

derive_g(#**id_gp**,# **id_ge**)

3. Requetes

- ***Une requête qui porte sur au moins trois tables:***

  Organisateurs bénéboles:
  Les noms de ceux qui ont organisé des événements moins chers de 10€

```
PREPARE recherche_par_organise_evenement_benevole as
      SELECT u.nom
      FROM Utilisateur u, organise o, Evenement e
      WHERE u.id_user=o.id_user
      AND o.id_event=e.id_event AND e.prix<10
```

```
Lines up: liste tous les groupes qui jouent sur un evenement

      PREPARE line_up_concert(INTEGER) as
            SELECT g.name_user
            FROM Groupe g, concert c, Evenement e
            WHERE c.id_grp=g.id_user AND c.id_event=e.id_event AND e.id_event=$1
```

- ***Une 'auto jointure' ou 'jointure réflexive' (jointure de deux copies d'une même table)***

Playlist ayant le même nom:

```
PREPARE recherche_par_nom_playlist(VARCHAR) as
      SELECT p2.id_pl FROM Playlist p1, Playlist p2
      WHERE p1.name_pl= p2.name_pl AND  p1.name_pl=$1
```

- ***Une sous-requête corrélée ;***

Noms des utilisateurs qui ont donné un avis sur un même groupe groupe:

```
PREPARE utilisateur_avis_groupe(INTEGER) as
SELECT name_user
FROM Utilisateur u
WHERE EXISTS (
      SELECT *
      FROM Avis_Groupe ag
      WHERE ag.id_user = u.id_user AND ag.id_grp=$1
);
```

- ***Une sous-requête dans le FROM ;***

```
SELECT u.name_user, a.total_avis
FROM Utilisateur u
JOIN (
      SELECT id_user, COUNT(*) AS total_avis
      FROM Avis
      GROUP BY id_user
) a ON u.id_user = a.id_user;
```

Cette requête effectue une agrégation sur la table Avis pour compter le nombre total d'avis pour chaque utilisateur, puis jointe les résultats avec la table Utilisateur pour récupérer les noms associés à chaque utilisateur.

- ***une sous-requête dans le WHERE ;***

Les meilleurs villes :

```
SELECT city_p
FROM Lieu
WHERE id_place IN (
      SELECT id_place
      FROM Avis_Lieu
      WHERE note_l >= 4
);
```

Cette requête sélectionne tous les enregistrements de la table *Lieu* où l'identifiant du lieu est présent dans les résultats d'une sous-requête. La sous-requête récupère les identifiants des lieux à partir de la table *Avis_Lieu* où la note attribuée au lieu est supérieure ou égale à 4.

- ***Deux agrégats nécessitant GROUP BY et HAVING ;***

Chanson populaires:

```
SELECT id_song, AVG(note_m) AS moyenne_note
FROM Avis_Morceau
GROUP BY id_song
HAVING AVG(note_m) > 4;
```

Cette requête calcule la moyenne des notes des morceaux dans la table *Avis_Morceau* et sélectionne uniquement les morceaux ayant une note moyenne supérieure à 4

Lieux populaires:

```
SELECT id_place, COUNT(id_event) AS nb_evenements
FROM Evenement
GROUP BY id_place
HAVING COUNT(id_event) > 20;
```

Cette requête compte le nombre d'événements pour chaque lieu dans la table *Evenement* et sélectionne uniquement les lieux ayant plus de 2 événements.

- ***une requête impliquant le calcul de deux agrégats***

```
SELECT id_user, MAX(note_g) AS max_note_g, AVG(note_g) AS avg_note_g
FROM Avis_Groupe
GROUP BY id_user;
```

Cette requête calcule la note maximale (MAX) et la note moyenne (AVG) attribuées à chaque groupe dans la table Avis_Groupe. Les résultats sont regroupés par groupe et le résultat inclut les colonnes id_user, max_note_g (note maximale) et avg_note_g (note moyenne).

- ***une jointure externe (LEFT JOIN, RIGHT JOIN ou FULL JOIN) ;***

```
SELECT Utilisateur.id_user, Utilisateur.name_user, Avis.commentary
FROM Utilisateur
LEFT JOIN Avis ON Utilisateur.id_user = Avis.id_user
```

Cette requête effectue une jointure externe (LEFT JOIN) entre la table *Utilisateur* et la table *Avis* sur la colonne "id_user". Elle récupère l'ID de l'utilisateur, son nom associé et le commentaire de l'avis correspondant. Les utilisateurs qui n'ont pas encore laissé d'avis auront la valeur NULL pour le commentaire dans les résultats.

- ***Deux requêtes équivalentes exprimant une condition de totalité, l'une avec des sous requêtes corrélées et l'autre avec de l'agrégation ;***

Playlist completes:

```
SELECT id_pl, name_pl
FROM Playlist
WHERE (
      SELECT COUNT(id_s)
      FROM pl_belongs
      WHERE pl_belongs.id_pl = Playlist.id_pl
```

```
) = 20
```

Cette requête récupère les identifiants (id_pl) et les noms (name_pl) des playlists pour lesquelles le nombre de morceaux associés dans la table pl_belongs est égal à 20.

```
SELECT p.id_pl, p.name_pl
FROM Playlist p, pl_belongs b
WHERE p.id_pl = b.id_pl
GROUP BY p.id_pl, p.name_pl
HAVING COUNT(b.id_s) = 20
```

Cette requête utilise l'agrégation pour compter le nombre de morceaux pour chaque playsliste, en utilisant la table *pl_belongs*. Ensuite, elle compare ce nombre est egal à 20.

- deux requêtes qui renverraient le même résultat si vos tables ne contenaient pas de nulls

```
SELECT *
FROM Avis_Morceau WHERE commentary LIKE %

SELECT *
FROM Avis_Morceau
```

- ***une requête récursive***

Tous les genres

```
WITH RECURSIVE derivation(id_cours, id_prereq) AS
(
      SELECT id_gp, id_ge FROM derive

      UNION

      SELECT prereq.prereq_id, derivation. id_ge
      FROM derive, derivation
      WHERE derive. gp = derivation. id_ge
)
SELECT *
FROM derivation ;
```

- ***une requête utilisant du fenêtrage***

Toutes les morceaux dans d'un utilisateur

```
CREATE VIEW Morceaux_Playlist (name) AS (

            SELECT m.name_s, p.id_pl
            FROM Morceaux m, pl_belongs b, Playlist p
            WHERE b.id_s=m.id_s AND b.id_pl=p.id_pl
);
```

```
PREPARE all_songs(VARCHAR) as
SELECT p.name
FROM Morceaux_Playlist p, Utilisateur u, Playlist p
WHERE u.id_user=pl.id_user AND p.id_pl=pl.id_pl AND u.name_user=$1;
```

SOURCES:

https://www.britannica.com/topic/list-of-bands-2026814