Marquer comme terminé

Compléments au cours de Base de données 2

1. jointures exotiques
2. dépendances fonctionnelles
3. diagrammes UML de séquences et de cas d'utilisation

1 Jointures exotiques

Les jointures sont utilisées pour associer plusieurs tables selon un critère, généralement l'égalité de clés dans ces tables.

Prenons l'exemple classique `clients - achats - produits` :

```
create table client(id varchar(10), primary key(id));
create table produit(id varchar(10), primary key(id));
create table achat(client varchar(10) references client, produit varchar(10) references produit);

select * from client;
    id
----------
 client 1
 client 2
 client 3

select * from produit;
    id
-----------
 produit 1
 produit 2
 produit 3

select * from achat;
  client  |  produit
----------+-----------
 client 1 | produit 1
 client 2 | produit 2
```

1.1 Jointure interne

On souhaite suivre les associations entre les tables et obtenir le combinaisons `client - produit` :

```
select * from client inner join achat on client.id = achat.client;
    id    |  client  |  produit
----------+----------+-----------
 client 1 | client 1 | produit 1
 client 2 | client 2 | produit 2
```

Rappelons qu'une jointure interne est l'enchaînement d'un produit cartésien et d'un filtre :

```
select * from client, achat, produit
where achat.produit = produit.id and client.id = achat.client;
```

1.2 Jointure externe

Si l'on souhaite obtenir la liste des clients et *potentiellement* (éventuellement, si c'est le cas) la liste des produits achetés, la jointure interne ne permet pas de répondre à la question car l'association `client - produit` n'est pas présente dans la relation `achat`.

SQL offre alors la possibilité d'effectuer des jointures externes, qui retournent les enregistrements même si le critère de jointure n'est pas vérifié :

?

```
select * from client left join achat on client.id = achat.client;
    id    |  client  |  produit
----------+----------+-----------
 client 1 | client 1 | produit 1
 client 2 | client 2 | produit 2
 client 3 |          |

select * from achat right join client on client.id = achat.client;
  client  |  produit  |    id
----------+-----------+----------
 client 1 | produit 1 | client 1
 client 2 | produit 2 | client 2
          |           | client 3

select * from produit left join achat on produit.id = achat.produit;
    id     |  client  |  produit
-----------+----------+-----------
 produit 1 | client 1 | produit 1
 produit 2 | client 2 | produit 2
 produit 3 |          |

select * from achat right join produit on produit.id = achat.produit;
  client  |  produit  |    id
----------+-----------+----------
 client 1 | produit 1 | produit 1
 client 2 | produit 2 | produit 2
          |           | produit 3
```

Pour aller plus loin : https://sql.sh/cours/jointures

2 Normalisation

Nous avons vu en cours comment passer d'un schéma UML à un modèle relationnel :

1. une relation par classe d'entité
2. une relation par classe d'association avec des clés étrangères
3. si l'association a une cardinalité finie, mettre la clé étrangère dans la relation de l'autre bout.

Historiquement, cette étape était connue sous le nom de *normalisation*. La normalisation fait appelle à des *formes normales* qui caractérisent certaines contraintes structurelles d'un schéma relationnelle.

2.1 Dépendances fonctionnelles

Pour comprendre la notion de forme normale, il est utile d'utiliser la notion de *dépendance fonctionnelle* :

Dans une relation R, on dit qu'il y a dépendance fonctionnelle entre un ensemble d'attributs A et un ensemble d'attributs B, ou que l'ensemble A d'attributs détermine l'ensemble B d'attributs (et on écrit $A \rightarrow B$) si quand deux n-uplets coïncident sur leurs attributs A, alors ils coïncident sur leurs attributs B.

Ceci est en lien avec la notion de *fonction*, qui à un antécédent associe une *unique* image. A et B sont en dépendance fonctionnelle s'il existe une fonction permettant de mettre en relation les éléments de A et de B.

Cette notion est déjà connue grâce à celle de *clé primaire* qui déterminer les valeurs des attributs selon une relation fonctionnelle : lorsqu'on connaît la clé, on connaît les autres attributs. Ou dit d'une autre façon, lorsque l'on s'interroge sur ce qui permet de concevoir une clé primaire : rend-elle unique chaque enregistrement ? ou Plusieurs enregistrement peuvent-ils avoir la même clé primaire ? Auquel cas ce ne serait pas une relation fonctionnelle (unicité de l'image pour un antécédent qui est la clé primaire).

2.2 Formes normales

Huit formes normales sont définies (voir https://fr.wikipedia.org/wiki/Forme_normale_(bases_de_donn%C3%A9es_relationnelles)), dans la pratique les trois premières sont prépondérantes :

1. Une relation est un 1FN si elle ne comporte que des attributs atomiques (qui ne peuvent être découpés en éléments porteurs d'information).

Par exemple, le nom d'une personne est atomique, mais pas son adresse complète, qui peut être découpé en n° de rue, rue, code postal, ville.

2. Une relation est un 2FN si :

- Elle est en 1FN
- Tout attribut n'appartenant pas à une clé ne dépend pas fonctionnellement d'une partie de la clé.

On interdit R(X1, X2, X3, X4, X5) si X3 ne dépend que de X2 : il faut une autre relation pour la dépendance entre X2 et X3.

?

Dit autrement : Un attribut non clé ne dépend pas d'une partie de la clé mais de toute la clé.

Le non-respect de la 2FN entraîne une redondance des données qui encombrent alors inutilement la mémoire et l'espace disque.

3. Une relation est un 3FN si :

- Elle est en 2FN
- Tout attribut n'appartenant pas à une clé ne dépend pas fonctionnellement d'un attribut non clé.

On interdit R(X1, X2, X3, X4, X5) si X4 ne dépend que de X3 : il faut une autre relation pour la dépendance entre X3 et X4 !

2.3 Normalisation à l'aide des dépendances fonctionnelles (DF)

- Pour décomposer en 2FN une relation en 1FN, on construit des relations différentes pour chaque sous-arbre du graphe des DF issus des clés multiples.
- Pour décomposer en 3FN une relation en 2FN, on construit des relations différentes pour chaque sous-arbre du graphe des DF de hauteur 1.

3 Compléments d'UML

Il est important de comprendre qu'UML ne fournit pas que des diagrammes de classes mais également :

- des diagrammes de cas d'utilisation
- des diagrammes de séquence.

Voir Polycopié de Jacques Madelaine pour un panorama exhaustif.

Modifié le: vendredi 25 mars 2022, 10:04

◄ TM4 - Optimisations avec PostgreSQL (avec corrigé)

Choisir un élément

Aller à...

Diagrammes UML ►