Introduction aux bases de données
Cours 5 : Algèbre relationnelle
sélections, union, intersection, produit, jointure

Jean Francis Michon [1]

[1]Université Nationale d'Economie, Kharkov

16 Décembre 2021

Opérations sur une table :

Sélection:

$$T' \subset T$$

On extrait des lignes satisfaisant un critère (ou prédicat) k :

```
SELECT * FROM T  WHERE k ;
```

Projection :

$$\pi : T \to T' \not\subset T$$

On extrait seulement certaines colonnes (on peut créer des doublons dans le résultat)

```
SELECT  nom , ville FROM  T;
```

(les champs NULL seront acceptés par cette commande)
L'opérateur SELECT réalise les sélections et les projections. Il peut aussi faire des permutations de colonnes !

Exemples de sélections

Créer une table `client2` identique à `client` (même schéma, mêmes enregistrements):

```
CREATE TABLE client2 AS SELECT * FROM client;
```

Créer une table `client2` identique à `client` mais sans doublons (même schéma, mêmes enregistrements):

```
CREATE TABLE client2 AS (SELECT DISTINCT * FROM client);
```

Créer une table `cliento` identique à `client` mais triée sur le champ `nom` (même schéma, mêmes enregistrements):

```
CREATE TABLE cliento AS
      (SELECT * FROM client ORDER BY nom) ;
```

Attention les clés primaires, index, contraintes ne sont pas copiés dans la nouvelle table !

Enchaînement de sélections

Il est possible de lancer un SELECT sur une sélection intermédiaire. Cela revient à **composer** les opérations de sélection.

```
SELECT ... WHERE ... FROM (SELECT ... WHERE... FROM)
```

Par exemple

```
SELECT COUNT(codepostal) FROM SELECT  DISTINCT codepostal FR
```

On compte le nombre de code postaux différents dans la tables clients.

Exemples de sélections

Si on veut copier la table **avec ses paramètres** on peut procéder en deux temps :
On recopie seulement la structure (avec les clés primaires, index, contraintes ...):

```
CREATE TABLE table2 LIKE table_source ;
```

puis on copie les données

```
INSERT INTO table2 SELECT * FROM table_source ;
```

Attention les clés étrangères ne seront pas copiées avec les commandes CREATE. Pour cela il faut **exporter** la table et la réimporter sous un autre nom .

Exercice

La table panier est un choix de 3 fruits au plus (l'ordre importe peu) :

f1	f2	f3
pomme	orange	raisin
framboise	citron	raisin
...		

Eliminez les répétitions de fruits :

$$(framboise, citron, citron) = (framboise, citron,)$$

et les choix identiques :

$$(framboise, citron,) = (citron, framboise,)$$

Opérations ensemblistes : union, intersection, différence

Si les deux tables T_1 et T_2 ont le même schéma, on peut définir les nouvelles tables :

$$T_1 \cup T_2,\ \ T_1 \cap T_2, T_1 - T_2$$

l'union, l'intersection et le complémentaire de T_2 dans T_1.
On a trois opérateurs de SQL correspondants: UNION, INTERSECT, EXCEPT.
L'opérateur UNION existe dans MySQL mais pas INTERSECT, ni DIFFERENCE. Heureusement INTERSECT et existent dans MariaDB.

Exemples d'union de tables

Créer une table `clients_tous` = `clientsa` $\cup$ `clientsb`

```
CREATE TABLE clients\_union AS SELECT * from clientsa
     UNION
SELECT * FROM clientsb;
```

Notez que si un engistrement est dans les deux tables, il sera copié **une seule fois par définition de l'union**. Sinon il faut utiliser UNION ALL.

Travailler avec 2 tables ou plus : les jointures

Théoriquement pour travailler avec deux tables indépendantes T_1 et T_2 il faudrait travailler dans l'ensemble $T_1 \times T_2$ (ou l'inverse). En général cet ensemble est très gros puisque sa taille est le produit des tailles des deux tables. Imaginez ce qui arrive avec 3 tables de 1000 enregistrements !

- clients
- livres
- On veut afficher une table de tous les couples (clients, livres).

```
SELECT * FROM  client, livre;
```

Dans les applications pratiques on a rarement besoin de cette construction.

Travailler avec deux tables : INNER JOIN

Lorsque les tables T_1 et T_2 ont un ou des champs communs, on utilise ces champs pour récupérer des informations, cela évite de construire tous les couples.

Exemple : La table `conducteurs` a pour schéma `nom, ville, type`. La table `voitures` a pour schéma `marque, type, carburant`. L'attribut type désigne le type du véhicule dans les deux tables.

Question : Qui a un véhicule électrique ?

Réponse :

```
SELECT nom FROM conducteurs
INNER JOIN voitures ON conducteurs.type=voitures.type
WHERE carburant ='électricité';
```

Les deux tables

conducteurs

nom	ville	type
ALIX	DIJON	S420
PERRAULT	MARSEILLE	T76
CALIVOT	ORLEANS	T850
PARENT	STRASBOURG	U56
PING	PARIS	T56
DURAND	BORDEAUX	T76
RATIER	PARIS	

voitures

type	carburant
S420	essence
T56	diesel
T76	électricité
T850	essence
U56	essence
V40	gaz

Comment fonctionne INNER JOIN

Chaque enregistrement de la table T_1 est comparé avec chaque enregistrement de T_2, et on vérifie que les conditions imposées dans la commande sont vérifiées pour ce couple. Si c'est le cas on affiche les champs demandés. Une jointure est donc une opération assez coûteuse.
Si le champ a le même nom dans les deux tables (comme dans notre exemple) on peut remplacer le ON `condition` par

```
USING (carburant)
```

Si on a besoin de 3 tables on rajoute une ligne INNER JOIN. On peut ordonner le résultat avec un ORDER BY en fin de commande.

LEFT JOIN et RIGHT JOIN

On peut avoir besoin de lister tous les éléments de la table T_1 et de mettre les champs qui ne correspondent pas dans T_2 en blanc.

```
SELECT nom, carburant FROM conducteurs
                LEFT JOIN voitures USING (type);
```

afffiche la liste des conducteurs avec le carburant utilisé. S'ils n'ont pas de véhicule référencé on affiche NULL.
Le RIGHT JOIN affiche tous les enregistrement de la table T_2.

Clé étrangère

Une colonne ou une famille de colonnes d'une table $\mathcal{A}$ qui est une clé primaire pour une autre table $\mathcal{B}$ s'appelle une **clé étrangère**. Dans notre exemple : supposons que dans T_2 on utilise l'attribut **type** comme **clé primaire**. le champ **type** de la table T1 peut être considéré comme une **clé étrangère** de T_1 . On peut réaliser cela en écrivant au moment de la création des tables :

```
CREATE TABLE T1 ... ,type VARCHAR(5) NOT NULL,...,
        FOREIGN KEY (type) REFERENCES T2(type) ;
CREATE TABLE T2 ... PRIMARY KEY (type);
```

On dit que T2 est un **parent** de T1 (et que T1 est un enfant de T2. Ces déclarations permettent de contrôler l'**intégrité référentielle** de la BD : si on insère une ligne dans T1 avec un type qui n'existe pas dans T2, on aura une erreur.

SELF JOIN

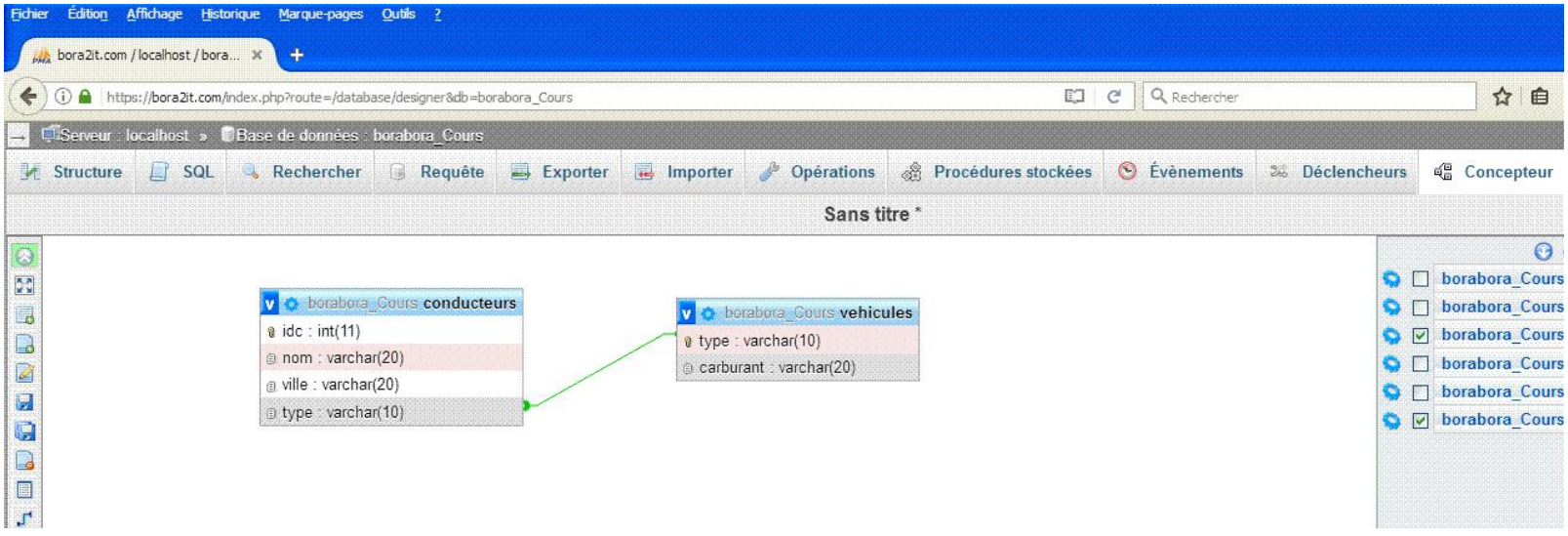
On peut créer une jointure d'une table avec elle même. On parle de d'auto-jointure mais il n'y a rien à écrire de spécial.
Cela peut être utile pour recherche des enregistrements qui partagent un champ identique par exemple, ou définir un lien hiérarchique dans un équipe.

```
SELECT  A.nom, B.nom
FROM T1 AS A, T1 AS B
WHERE A.nom <> B.nom AND  A.type =B.type;
```

extrait les clients qui ont le même type de voiture.

Représentations graphiques des BD et des clés étrangères

Il est très courant de représenter graphiquement les tables, les clés primaires, les clés étrangères etc ... par des graphiques, comme on le fait dans les diagrammes Entités-Relation en UML ou en Programmation Objet.. Il existe même des outils qui génère les instructions SQL de création de tables à partir des graphiques. Cela peut vous aider à faire des présentations attrayantes et à clarifier un problème. Vous pouvez visiter
http://www.databaseanswers.org/modelling_tools.htm.
Il existe un outil dans PhpMyAdmin :aller au niveau de la "Base de données" puis cliquer sur l'onglet "Concepteur".



Un exemple pour l'ajout d'une clé étrangère sur une table enfant **conducteurs** vers une table parent **vehicules**:

```
CONSTRAINT conducteurs FOREIGN KEY (type)
                REFERENCES vehicules  (type);
```

Attention ces commandes peuvent générer des erreurs facilement. Veillez à rentrer des données cohérentes.

Suppression en cascade

Les clés étrangères permettent de simplifier les suppression dans les tables reliées avec des instructions comme

```
CONSTRAINT conducteurs FOREIGN KEY (type)
        REFERENCES vehicules  (type)
       ON DELETE CASCADE;
```

Si vous supprimez (DELETE) un enregistrement de la table parent les enregistrement reliés seront automatiquement supprimés ! Il existe d'autres versions qui mettent les champs concernés à NULL.