Chapitre 3. Le Langage SQL

Olivier Terraz

- Plan du chapitre 3 :

1. Langage de manipulation de données (LMD)
2. Langage de définition de données (LDD)

Le langage SQL

L'ANSI a choisi SQL comme langage standard d'accès aux bases de données relationnelles. C'est également une norme ISO.

En fait, SQL est composé de 2 langages :

- Un langage de manipulation de données (LMD). Celui ci permet la modification et l'interrogation des données.
- Un langage de définition de données (LDD) ; Qui permet, notamment, la création des tables et la définition des droits d'accès sur les tables.

Il est utilisé dans la plupart des SGBD relationnels.

Le langage SQL

En fait il existe plusieurs normes SQL : Évolution d'un simple langage d'accès à une BD relationnelle (SQL1) vers un langage de programmation de Bases de Données Relationnelles-Objets (SQL3).

- La norme SQL1 (SQL86) : les bases de l'accès mais beaucoup de notions de base absentes (mise à jour du schéma, ...).

Le langage SQL

- La norme SQL2 (SQL92) :
  - Renommage de colonnes résultats (c.f. as ...)
  - Interfaces SQL pour Ada et C
  - Nouveaux types de données : Date, Time, ..
  - Support complet de l'algèbre relationnelle : intersect, except, ..
  - Ordres de modification de schéma : alter, drop
  - Fonctions de scroll curseur : first, next, last, ..
  - Gestion de relations temporaires
  - Support complet de l'intégrité (référentielle, ...).
  - Nouveaux types de données : chaîne de bits, ..
  - etc ..

Le langage SQL

- La norme SQL3 (SQL99) :
  - Extensions procédurales de SQL => notion de procédures stockées
  - Fonctionnalités orientées objet
  - Construction de nouveaux types de données
  - Support de l'héritage entre tables
  - Support de règles déclenchées par des événements BD => mécanisme de "trigger »

- La Norme SQL4 (SQL2003) :
  - Introduction de fonctions pour XML
  - …

- Et d'autres en 2011, 2016

Le langage SQL

Dans les faits relativement peu de SGBD implémentent SQL2 en entier et peu ou pas de SGBD implémentent SQL3 de manière complète (surtout en ce qui concerne la partie « objet » de SQL3.)

Il existe 2 types de commandes de **manipulation de données** :

- Les commandes de sélection des données.
- Les commandes de modification des données.

Pour illustrer la présentation des commandes SQL nous nous servirons de la base de données comportant les 3 relations suivantes :

- **Etudiants** (net, nom, age, ville)

  Relation donnant des renseignements sur les étudiants avec leur numéro, nom âge et ville de résidence.

- **Suit** (net, nmod, moy)

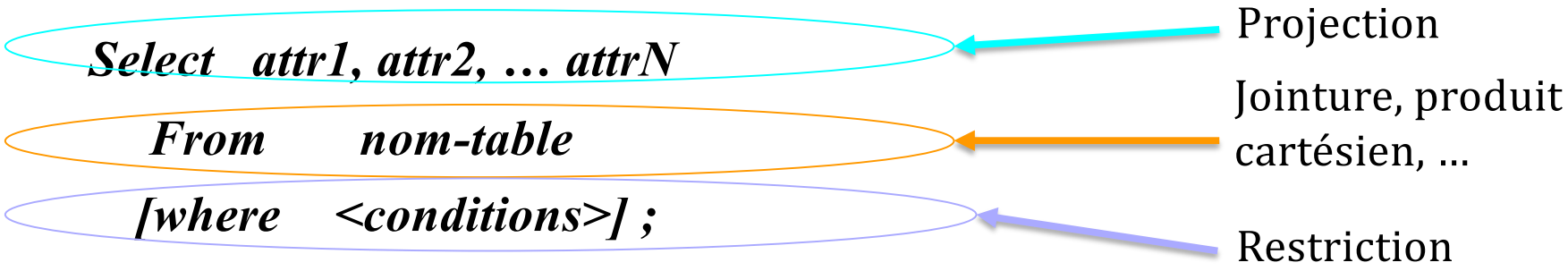
  Relation donnant des renseignements sur les cours que suivent les étudiants avec le numéro de l'étudiant, le numéro du module et la moyenne obtenue par l'étudiant dans ce module.

- **Cours** (nmod, intitulé, responsable)

  Relation donnant des renseignements sur les cours avec le numéro du module, l'intitulé du cours et le nom du responsable de ce module.

Les commandes de sélection des données

En fait, il s'agit d'une seule commande un certain nombre d'opérations.



où <conditions> est un ensemble de conditions simples séparées par des opérateurs logiques And et Or.

Projection

Définition : Opération portant sur une relation R1 et consistant à construire une deuxième relation R2 en enlevant à la relation initiale tous les attributs non mentionnés en opérandes.

Etudiants

net	nom	age	ville
E1	Smith	20	Limoges
E2	Dupond	21	Limoges
E3	Durand	22	Poitiers

SELECT net, nom FROM Etudiants

NET	NOM
E1	Smith
E2	Dupond
E3	Durand

Restriction (sélection de certaines lignes)

Définition: Opération portant sur une relation R1 et consistant à construire une deuxième relation R2 mais comportant les seuls n-uplets qui vérifient la condition précisée en argument

Etudiants

net	nom	age	ville
E1	Smith	20	Limoges
E2	Dupond	21	Limoges
E3	Durand	22	Poitiers

SELECT net, nom FROM Etudiants WHERE ville = 'Limoges'

NET	NOM
E1	Smith
E2	Dupond

Les commandes de sélection des données

Une condition simple peut porter :

- Sur une comparaison avec les **opérateurs classiques** : <, >, >=, <=, =, <>

  Exemple : Numéro des étudiants ayant une moyenne >= 10 en BD1.

  Select net

  From Suit

  Where nmod = 'BD1' AND note >= 10;

- Sur l'appartenance à un ensemble avec **l'opérateur in**

  Exemple : Liste des noms des étudiants habitant Limoges, Poitiers ou Brive.

  Select nom

  From Etudiant

  Where ville in ('Limoges', 'Poitiers', 'Brive');

Les commandes de sélection des données

Une condition simple peut porter :

- Sur l'appartenance à un intervalle avec l'opérateur **between**

  Exemple : Noms des étudiants ayant entre 20 et 24 ans.

  Select nom

  From Etudiant

  Where age between 20 and 24;

- Sur la présence de certains caractères dans une chaîne, avec l'opérateur **like 'chaîne'**.
- Dans la chaîne le caractère * remplace n'importe quelle chaîne de caractère et le caractère ? remplace n'importe quel caractère.

  Exemple : Nom des étudiants habitant dans une ville commençant par un 'B'.

  Select nom

  From étudiant

  Where ville like 'B*';

Les commandes de sélection des données

La négation d'une condition se traduit par not. Par exemple not in ou not like.

Exemple : Noms et villes des étudiants n'habitant pas Limoges ou Poitiers.

```
Select nom, ville
    From Etudiant
    Where ville not in ('Limoges', 'Poitiers') ;
```

Les commandes de sélection des données

On rajoute la possibilité d'obtenir les résultats sous forme ordonnée au moyen de la clause order by.

Exemple : Liste des noms et villes des étudiants n'habitant pas Limoges, ordonnée par ordre alphabétique des villes, puis des noms (pour les étudiants d'une même ville).

```
Select nom, ville
    From Etudiants
    Where ville <> 'Limoges'
    Order by ville, nom ;
```

on obtient un ordre décroissant en rajoutant la mention desc à la fin du order by.

Fonctions de calcul

La possibilité d'effectuer des calculs sur les attributs existe également:

Exemple : Liste des numéros d'étudiant et des notes majorées de 1 point pour les étudiants ayant eu moins de 5 en BDR.

```
Select net, moy+1
    From Suit
    Where nmod ='BDR' and moy < (4 + 1) ;
```

Exo 1

Fonctions de calcul en colonne

On peut effectuer des opérations sur les colonnes des tables.

Ces fonctions sont les suivantes :

- Sum (col) somme des valeurs de la colonne;
- Avg (col) moyenne des valeurs de la colonne;
- Max (col) valeur maximale de la colonne;
- Min (col) valeur minimale de la colonne;
- Count (col) nombre de valeur non nulle dans la colonne.

Fonctions de calcul en colonne

Exemple 1 : Plus petite, plus grande moyenne et moyenne générale dans l'UE BDR.

```
Select Min(moy), Max(moy), Avg(moy)
    From Suit
    Where nmod = 'BDR';
```

Exemple 2 : Nombre d'étudiant suivants le module BDR.

```
Select Count(*)
    From Suit
    Where nmod = 'BDR';
```

Agrégats

- On peut réaliser des agrégats (partitionnement horizontal d'une relation).
- Les agrégats sont en règle générale utilisés pour effectuer des calculs en utilisant les fonctions de calculs sur ensemble vues ci-dessus.
- C'est la clause group by qui permet ces agrégats.
- Les attributs figurant dans la clause Group by, doivent obligatoirement figurer dans la clause Select.

Agrégats

Exemple: Noms des villes et pour chacune de celles-ci nombre d'étudiants y habitant.

Select ville, count(*)

From Etudiant

Group by ville;

On peut également faire des sélections en fonction des résultats des fonctions de calcul sur ensemble à l'aide de la clause **Having**.

Exemple: Noms des villes où habitent plus de 10 étudiants.

Select ville, count(*)

From Etudiant

Group by ville

Having count(*) > 10 ;

Exo 2

Requêtes imbriquées

Une condition peut également contenir des requêtes (on parle alors de sous requête).

Exemple: Nom des étudiants habitant la même ville que l'étudiant Dupond .

```
Select nom
    From étudiant
    Where ville in ( Select ville
                     From étudiant
                     Where nom = "Dupond" ) ;
```

Jointures

Définition: Opération consistant à rapprocher selon une condition les n-uplets de deux relations R1 et R2 afin de former une troisième relation R3 qui contient l'ensemble de tous les n-uplets obtenus en concaténant un n-uplet de R1 et un n-uplets de R2 vérifiant la condition de rapprochement.

L'opération de jointure s'effectue en mettant dans la clause From la liste des relations que l'on veut joindre et dans la clause Where les conditions de jointure.

Exemple: Nom des étudiants et notes qu'ils ont obtenues dans les différentes matières.

```
Select nom, nmod, moy
    From Etudiant, Suit
    Where Etudiant.net = Suit.net ;
```

Jointure : Exemple

Etudiants

net	nom	age	ville
E1	Smith	20	Limoges
E2	Dupond	21	Limoges

Suit

net	nmod	moy
E1	BDR	18
E2	BDR	5
E1	ProgJava	15

Select nom, nmod, moy From Etudiant, Suit
Where Etudiant.net = Suit.net ;

Nom	nmod	moy
Smith	BDR	18
Dupond	BDR	5
Smith	ProgJava	15

Jointures

Autre écriture des jointures : on peut également utiliser la clause JOIN afin de mettre la condition de jointure au niveau de la clause FROM.

- Ainsi l'exemple précédent peut s'écrire :

  Select nom, nmod, moyenne

  From Etudiant JOIN Suit ON (Etudiant.net = Suit.net);

- ou encore :

  Select nom, nmod, moyenne

  From Etudiant JOIN Suit USING (net);

- ou encore :

  Select nom, nmod, moyenne

  From Etudiant NATURAL JOIN Suit;

Exo 3

Attention : pour la jointure naturelle il y a jointure pour tous les attributs ayant le même nom. On évitera donc de l'utiliser notamment en TP car MySql s'y perd un peu.

Toutes ces jointures sont appelées **jointure interne**

Jointures externes

- LEFT OUTER JOIN renvoie toutes les lignes dans le produit cartésien qualifié
  - i.e. toutes les lignes combinées qui passent la condition ON
  - plus une copie de chaque ligne dans le côté gauche de la table pour lequel il n'y a pas de côté droit qui satisfasse la condition ON (avec des NULL pour les valeurs manquantes).

- À l'inverse, RIGHT OUTER JOIN retourne toutes les lignes jointes, plus une ligne pour chaque ligne côté droit non appariée (étendue avec des NULL côté gauche).

Jointures externes

A

NUM	REF
1	R1
2	R2
3	R3

B

NUM	NOM
1	NA
2	NB
4	NC

Select * From A LEFT OUTER JOIN B ON (A.Num = B.Num)

NUM	REF	NOM
1	R1	NA
2	R2	NB
3	R3	Null

Jointure interne

Jointure externe

Requêtes quantifiées: clauses ANY et ALL

- Il est aussi possible de vouloir comparer une expression de valeurs à tous les résultats d'une sous requête ou à seulement l'une quelconque des valeurs résultantes. Pour cela, SQL propose l'usage de sous requête quantifiées par quel que soit (ALL) ou il existe (ANY).
- Plus précisément (Θ étant l'un des opérateurs de comparaison <, >, =, …):
  - La condition f Θ ANY <sous requête> est vraie ssi la comparaison f Θ V est vraie au moins pour une valeur V du résultat de la sous requête;
  - La condition f Θ ALL <sous requête> est vraie ssi la comparaison f Θ V est vraie pour toutes les valeurs V du résultat de la sous requête.
- Remarques:

  Les prédicats IN et = ANY sont équivalents;

  Les prédicats NOT IN et <> ALL sont équivalents.

Requêtes quantifiées : clause EXISTS

- SQL offre une autre possibilité de quantification pour tester si le résultat d'une sous requête est vide ou non. Il s'agit du prédicat d'existence EXISTS.

  EXISTS <sous requête> est vrai ssi le résultat de la sous requête est non vide.

Exemple: Etudiants qui suivent au moins un cours.

```
Select nom
    From Etudiant
    Where EXISTS  ( Select *
                    From Suit
                    Where net = Etudiant.net );
```

Exo 4

Opérateurs de conjonction

Enfin, on dispose de différents opérateurs de conjonction entre les requêtes. Ce sont les opérateurs suivant:

- UNION;
- INTERSECT;
- EXCEPT (MINUS pour Oracle).

Correspondant aux opérations d'union, d'intersection et de différence de l'algèbre relationnelle.

Opérateurs de conjonction

Exemple : Numéro des étudiants suivant les cours de M1 et de M2 (ceux qui suivent les deux).

Select net

From Suit

Where nmod = 'M1'

INTERSECT

Select net

From Suit

Where nmod = 'M2' ;

Exo 5

Vues et relations temporaires

- Une vue est une relation virtuelle au sens où ses instances n'existent pas physiquement mais sont calculées à chaque invocation de la vue.
- Une vue est définie par une requête qui utilise des relations ou des vues existantes.
- La forme générale est la suivante :

Create view nom_de_la_vue ***as*** (clause select ...)

Vues et relations temporaires

Exemple: Vue représentant les étudiants de Limoges.

```
Create view EtudiantsLimoges
    as ( Select *
    From Etudiant
    Where ville = 'Limoges' );
```

- En interrogation, une vue est utilisée comme toute autre relation.

Exemple: Etudiants de Limoges agés de 22 ans.

```
Select nom, prénom
    From EtudiantsLimoges
    Where age = 22;
```

Les commandes de modification de données

- Ajout de données dans une table

  Insert into nom-table [(colonne1, …, colonneN)] values (v1, …, vN) ;

  Ajoute une ligne à la table "nom-table". On peut préciser la valeur de certaines colonnes uniquement, dans ce cas la valeur pour les autres colonnes sera la valeur nulle.

- On peut se servir du résultat d'une requête pour ajouter des données. La commande s'écrit alors :

  Insert into nom-table [(colonne1, …, colonneN)]

  Select … ;

Les commandes de modification de données

- Modification de données dans une table

  Update nom-table ***set*** colonne1 = epr1, …

  [***where*** condition];

  Modifie la valeur des colonnes dont on spécifie les noms pour toutes les lignes vérifiant la condition.

Exemple : Modification du nom de la ville de l'étudiant numéro 1234.

Update étudiant ***set*** ville = 'Limoges' ***where*** net = 1234 ;

- Suppression de lignes dans une table

  Delete from nom-table [***where*** condition] ;

  Supprime toutes les lignes vérifiant la condition.

Les commandes de définition des données

Remarque: Il faut être administrateur de la base de données pour avoir le droit d'exécuter ces commandes.

Création d'une base de données

Create database nom_base [paramètres];

Créé une base de données identifiée par nom_base et tout son environnement système.

Les commandes de définition des données

- Création d'une table

```
Create table nom_table
 (   attribut1      type(longueur)  [contraintes],
     attribut2      type(longueur)  [contraintes],
     ......
     attributn      type(longueur)  [contraintes]      )
                        [paramètres optionnels ];
```

Crée une table, identifiée par nom_table.

Les commandes de définition des données

On précise les noms des attributs (ou colonnes) de celle-ci et pour chaque attribut son type, sa longueur et éventuellement des contraintes sur la valeur de ces colonnes.

Le type est par exemple:

varchar(n) où n est la longueur de la chaîne de caractères

numeric(n1, n2) où n1 est le nombre de chiffre total + la virgule et n2 le nombre de chiffre après la virgule

Date, ……

Les contraintes peuvent être, par exemple :

not null qui indique que l'on interdit les n-uplets dans lesquels la valeur de cet attribut peut être nulle

Primary key qui indique que cet attribut est une clé primaire.

Foreign key qui indique que cet attribut est une clé secondaire (attribut servant dans une jointure avec une clé primaire d'une autre table).

Exemples

Création d'une table Stock(num_p, num_dep, quantite) indiquant où (dans quel dépôt) sont stockés les produits et en quelle quantité.

```
Create Table Stock
    ( num_p         Number(5),
      num_dep       Number(5) Check (num_dep Between 10 and 50)
      quantite      Number(4) Default 0 Check (quantite >= 0)
      Primary Key (num_p, num_dep),
                    // La clé primaire est le couple d'attribut
     Foreign Key (num_p) References Produit(num_p)
     Foreign Key (num_dep) References Depot(num_dep)
    );
```

Suppression d'une table

- Suppression d'une table :

  Drop table nom_table ;

  Efface toutes les données et la définition de la structure d'une table, en la supprimant des catalogues systèmes.

Création d'un index

Les index sont des tables permettant d'optimiser l'accès aux données.

Ces tables associent un attribut (en général la clé) et l'emplacement de la ligne correspondante dans la table.

Create index ***nom_index*** ***on*** ***nom_table (Colonne 1….) ;***

Crée un index sur la table dont on précise le nom et les attributs pour lesquels on veut gérer une table d'index. On peut supprimer une table d'index par la commande ***Drop index***