Synthèse du langage SQL

Les données constituant la **base** sont organisées en tables. Une table contient des enregistrements composés de différents champs. Le type associé au champ défini le domaine sur lequel le champ pourra prendre ses valeurs (date, entier, chaîne de caractères).

Chaque **table** est désignée de manière unique par un identificateur (son nom) au sein de la base de données de même que chaque champ au sein d'une table.

Usuellement on représente une table par un tableau dont les colonnes correspondent aux champs et les lignes aux enregistrements :

La **clé primaire** d'une table est un champ (ou un ensemble de champs) qui identifie de manière unique chaque enregistrement dans la table. Chaque table devrait avoir une clé primaire.

Une **clé étrangère** dans une table est un champ (ou un ensemble de champs) qui fait référence à un champ (ou un ensemble de champs) d'une autre table (généralement la clé primaire).

Les **requêtes** permettent d'interroger une base de données et d'en modifier les informations.

Les requêtes

I Définitions

I.1 Valeur NULL

Un champ qui n'est pas renseigné, donc vide, contient la valeur NULL. Cette valeur est différente de zéro et représente l'absence de valeur.

I.2 Expressions

Les expressions valides mettent en jeu des noms de champs, le mot clé « * » (qui signifie « tous les champs »), des constantes, des fonctions et des opérateurs arithmétiques classiques. Il existe des fonctions arithmétiques, de manipulations de chaînes, de dates et des fonctions agrégat. Les fonctions agrégat permettent de calculer un résultat atomique (un entier ou un réel) à partir d'un ensemble de valeurs. Les agrégats sont au nombre de 5 : COUNT, SUM, MIN, MAX, AVG.

II Définition des données

II.1 Création

CREATE TABLE *table* (*col1 type1, col2 type2…*) AS SELECT…

table est le nom donné à la nouvelle table.
col1, col2,… sont les noms des colonnes.
type1, type2,… sont les types des données contenues dans les colonnes.

II.2 Modification de la structure

Moyennant quelques contraintes, il est possible de modifier la structure d'une base de données existante :

ALTER TABLE *table* ADD (*col1 type1, col2, type2…*)

ajoute les colonnes spécifiées à une table existante.

ALTER TABLE *table* MODIFY (*col1 type1, col2 type2…*)

modifie la définition des colonnes spécifiées.

ALTER TABLE *table* DROP *col*

supprime la colonne *col.*

II.3 Contraintes d'intégrité

Lors de la création ou la modification de la structure d'une table, il est possible de spécifier des contraintes d'intégrité c'est-à-dire des conditions que devront vérifier les enregistrements. Elles sont de deux types :

- Contrainte sur une colonne : suit la définition d'une colonne.
- Contrainte sur une table (une ou plusieurs colonne) : apparaît au même niveau que les définitions des colonnes.

CONSTRAINT *nom_contrainte contrainte*

définit et nomme une contrainte sur une ou plusieurs colonnes.

Les types de contraintes sont :

PRIMARY KEY (*col1, col2,…*)	(contrainte sur une table)
PRIMARY KEY (contrainte sur une colonne)	

indique la clé primaire de la table . Aucune des colonnes de cette clé ne doit avoir une valeur NULL.

UNIQUE (*col1, col2,...*) (contrainte sur une table) UNIQUE (contrainte sur une colonne)

interdit qu'une colonne (ou la concaténation de plusieurs colonnes) contienne deux valeurs identiques.

FOREIGN KEY (*col1, col2,...*) REFERENCES *table* [(*col'1,col'2,...*)] (contrainte sur une table) REFERENCES *table* [(*col1*)] (contrainte sur une colonne)

indique que la concaténation de col1, col2,… est une clé étrangère faisant référence à la concaténation de col'1, col'2,… de table. Si col'1,col'2,… sont omises, la clé primaire de table est prise par défaut.

CHECK (*condition*)

donne une condition devant être vérifiée par une ou plusieurs colonnes.

II.4 Destruction

DROP TABLE *table*

supprime la définition d'une table et par conséquent l'ensemble des enregistrements qu'elle contient.

III Interrogation d'une base

La syntaxe générale est la suivante :

SELECT … FROM… WHERE… GROUP BY… HAVING… ORDER BY…

III.1 SELECT...

SELECT [DISTINCT] *expr1* [[AS] *nom1*], *expr2* [[AS] *nom2*],…

réalise une projection.

Exrp1, expr2,... indiquent quelles expressions devront être renvoyées, par exemple des noms de champs. S'il y a une ambiguïté (cas de deux tables contenant des champs de même nom), il est nécessaire de préfixer le nom du champ par celui de la table et d'un point.

Nom1, nom2,... sont des noms facultatifs qui constitueront les titres des colonnes renvoyées.

Le mot clé DISTINCT permet de supprimer les doublons.

III.2 FROM...

FROM *table1* [*alias1*], *table2* [*alias2*],…

donne la liste des tables participant à l'interrogation. *alias1*, *alias2*,… sont des alias facultatifs attribués aux tables pour le temps de la requête. Quand une table se voit attribuer un alias, elle n'est plus reconnue sous son nom d'origine dans la requête.

III.3 WHERE...

WHERE *prédicat*

permet d'effectuer une restriction, c'est-à-dire de spécifier quels enregistrements sélectionner dans une table ou un produit cartésien de tables.

III.3.1 Prédicats simples

Un prédicat simple est la comparaison de plusieurs expressions au moyen d'un opérateur logique :

WHERE *expr1* = / != / < / > / <= / >= *expr2* (opérateurs classiques) WHERE *expr1* BETWEEN *expr2* AND *expr3* (appartenance à un intervalle bornes incluses) WHERE *expr1* [NOT] LIKE *expr2* (utilisation des caractères joker _ et % dans *expr2*) WHERE *expr1* [NOT] IN (*expr2,expr3,...*) (appartenance à la liste d'expressions) WHERE *expr1* IS [NOT] NULL (valeur NULL)

III.3.2 Prédicats composés

Les opérateurs logiques AND et OR permettent de combiner plusieurs prédicats. AND est prioritaire par rapport à OR mais l'utilisation de parenthèses permet de modifier l'ordre d'évaluation.

III.3.3 Sous-requêtes

Le critère de recherche employé dans une clause WHERE (l'expression à droite d'un opérateur de comparaison) peut être le résultat d'un SELECT. Dans le cas des opérateurs classiques, la sous–interrogation ne doit ramener qu'une ligne et une colonne. Elle peut ramener plusieurs lignes à la suite de l'opérateur [NOT] IN, ou à la suite des opérateurs classiques moyennant l'ajout d'un mot clé (ANY ou ALL).

ANY : la comparaison est vraie si elle est vraie pour au moins un élément de l'ensemble (donc fausse si l'ensemble des enregistrements est vide).

ALL : la comparaison est vraie si elle est vraie pour tous les éléments de l'ensemble (donc vraie si l'ensemble des enregistrements est vide).

```
WHERE expr1 = / != / < / > / <= / >= expr2ALL (SELECT…)
WHERE expr1 = / != / < / > / <= / >= expr2ANY (SELECT…)
```

III.3.4 Jointures

III.3.4.1 Jointures naturelles

```
SELECT expr1, expr2,…
FROM table1, table2
WHERE table1.champ1 = table2.champ2
```

Lorsque la clause FROM contient plusieurs tables, on en obtient le produit cartésien. La clause WHERE permet d'effectuer une restriction. Si cette clause est une égalité, on dit que l'on réalise une jointure naturelle ou équi-jointure.

III.3.4.2 Jointures interne

```
SELECT table.expr1, alias.expr2
FROM table, table alias
WHERE table.champ1 = alias.champ2
```

Il s'agit d'une jointure naturelle mettant deux fois en jeu la même table. Ceci permet de rassembler venant d'un enregistrement d'une table avec les informations venant d'un autre enregistrement. A noter qu'il est obligatoire de spécifier deux fois la table source dans la clause FROM, en attribuant un alias à l'une des occurrences.

III.3.4.3 Jointures externes

```
SELECT expr1, expr2,…
FROM table1 LEFT / RIGHT / FULL OUTER JOIN table2
ON table1.champ1 = table2.champ2
```

Lorsqu'on effectue une jointure naturelle sur deux tables, il est possible qu'un enregistrement d'une table n'ait pas de correspondant dans l'autre. Dans ce cas, l'enregistrement en question n'est pas affiché. La jointure externe permet de résoudre ce problème par l'ajout de lignes fictives, qui réalisent la correspondance avec les enregistrements de l'autre table n'ayant pas de correspondant réel (LEFT OUTER JOIN correspond à l'ajout d'enregistrements fictifs dans *table1*).

III.4 GROUP BY…

```
GROUP BY expr1, expr2,…
```

permet de subdiviser la table en groupes, chaque groupe étant l'ensemble des enregistrements ayant une valeur commune pour les expressions spécifiées. Les champs de la clause SELECT doivent alors être des fonctions agrégat ou des expressions figurant dans la clause GROUP BY.

III.5 HAVING…

```
HAVING prédicat
```

sert à préciser quels groupes doivent être sélectionnés. Cette clause se place après GROUP BY et le prédicat ne peut porter que sur des fonctions agrégat ou des expressions figurant dans la clause GROUP BY.

III.6 ORDER BY…

```
ORDER BY expr1 [DESC], expr2 [DESC],…
```

classe les enregistrements retournés selon l'ordre croissant de *expr1* puis *expr2*. La clause facultative DESC inverse l'ordre de classement. Pour préciser lors d'un tri sur quelle expression va porter le tri, il est possible de donner sa position dans la liste des expressions de la clause SELECT ou encore le nom qu'on lui a attribué.

III.7 Les opérateurs ensemblistes

```
requête1 UNION / INTERSECT / MINUS requête2 …
```

permettent de réaliser des opérations ensemblistes sur les résultats de plusieurs interrogations. Les champs renvoyés par les requêtes impliquées doivent être identiques. Les opérations sont évaluées de gauche à droite mais l'utilisation de parenthèses permet de modifier cet ordre.

IV Manipulation des données

IV.1 Insertion

```
INSERT INTO table [(col1, col2,…)] VALUES (val1, val2,…)
INSERT INTO table [(col1, col2,…)] SELECT…
```

insère un nouvel enregistrement dans *table*. Si (*col1, col2,…*) est omise, l'ordre utilisé par défaut est celui spécifié lors de la création de la table. A l'inverse, si cette liste est donnée, les colonnes n'y figurant pas auront la valeur NULL.

IV.2 Mise à jour

```
UPDATE table SET col1 = expr1, col2 = expr2,… WHERE prédicat
UPDATE table SET (col, col2,…) = (SELECT…) WHERE prédicat
```

modifie les enregistrements de *table* qui vérifient *prédicat*. Si *prédicat* est omis, **tous** les enregistrements sont mis à jour.

IV.3 Suppression

DELETE FROM *table* WHERE *prédicat*

supprime les enregistrements de *table* qui vérifient *prédicat*. Si *prédicat* est omis, **tous** les enregistrements sont supprimés.

V Contrôle de l'accès aux données

GRANT *privilege* ON *table* TO *utilisateur* [WITH GRANT OPTION]

permet au propriétaire de *table* de donner à *utilisateur* des droits d'accès sur celle-ci. Si l'option WITH GRANT OPTION est spécifiée, *utilisateur* pourra à son tour transmettre ces droits.

REVOKE *privilege* ON *table* FROM *utilisateur*

permet de reprendre un privilège à un utilisateur.

Vues : un ordre SELECT renvoie une table temporaire. Il est cependant possible d'enregistrer le résultat d'un tel ordre dans une vue. Ceci permet de découpler la structure réelle de la base de données de ce que voient les utilisateurs.

Transactions : Une transaction est un ensemble de modifications de la base qui forment un tout. Ces modifications doivent être effectuées entièrement ou pas du tout sous peine de laisser la base dans un état incohérent. Une commande (COMMIT) permet de valider la session en cours, une autre (ROLLBACK) permet de l'annuler.

Procédures stockées : Une procédure stockée est un programme qui comprend des instructions SQL pré compilées et qui est enregistré dans la base de données. Ceci permet d'accélérer les requêtes puisque les optimisations ont lieu lors de l'enregistrement de celles-ci.

Triggers : Ce sont des procédures stockées déclenchées automatiquement par des événements liés à des actions sur la base.

Références Web

http://www.postgresql.org (site de PostgreSQL) , http://www.mysql.com (site de MySQL)

Documentation en français : http://dev.mysql.com/doc manuel PDF format A4 de 6, 75 Mo et 1483 pages !