Le langage SQL

Introduction

- SQL (*Structured Query Language*)
- langage de communication avec une BDD relationnelle
- recherche, ajout, modification, suppression de données
- créé en 1974
- normalisé en 1986

SQL et les SGBD

- Implémenté dans la plupart des SGBD
- Mais étendu différemment par chaque SQBD
- => Cours sur SQL de base (avec extension PostgreSQL si besoin)

Figure 1: Logo PostgreSQL

SQL SELECT

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande SELECT, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table. Commande basique

L'utilisation basique de cette commande s'effectue de la manière suivante:

SELECT nom_du_champ FROM nom_du_tableau

Cette requête SQL va sélectionner (SELECT) le champ "nom_du_champ" provenant (FROM) du tableau appelé "nom_du_tableau". Exemple

Imaginons une base de données appelée “client” qui contient des informations sur les clients d’une entreprise.

Table client :

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

Si l’ont veut avoir la liste de toutes les villes des clients, il suffit d’effectuer la requête SQL ci-dessous :

```
SELECT id, ville FROM client
```

De cette manière on obtient le résultat suivant :

id	ville
1	Paris
2	Nantes
3	Lyon
4	Marseille
5	Grenoble

Obtenir toutes les colonnes d’un tableau

Il est possible de retourner automatiquement toutes les colonnes d’un tableau sans avoir à connaître le nom de toutes les colonnes. Au lieu de lister toutes les colonnes, il faut simplement utiliser le caractère “*” (étoile). C’est un joker qui permet de sélectionner toutes les colonnes. Il s’utilise de la manière suivante:

```
SELECT * FROM client
```

Cette requête SQL retourne exactement les mêmes colonnes qu’il y a dans la base de données.

Avantages du caractère étoile

Ce caractère peut lister toutes les colonnes sans avoir besoin de connaître leurs noms. C’est très pratique pendant une phase de tests ou de débug pour voir ce que contient un tableau rapidement et facilement.

Une requête utilisant ce caractère à moins de chance d’échouer si les tables évolues. Puisque les noms des champs ne sont pas spécifiés dans la sélection,

la requête continuera de fonctionner si un nom de champ est modifié ou si une colonne est supprimée. A l'inverse, une requête qui spécifie la lecture des champs "nom" et "ville" échouera si le champ "ville" est remplacé par le champ "code_postal".

Inconvénients du caractère étoile

Bien que ce caractère semble magique pour les débutants, il y a tout de même des inconvénients pour l'utiliser dans une application. Son utilisant implique que le système de gestion de base de données retourne toutes les colonnes. Or, si une application désire seulement le prénom et le nom du client, il est inutile de lire d'autres données telles que la ville, le nombre d'achat ...

Utilisé à mauvais escient, le caractère étoile peut avoir de mauvaises conséquences sur les performances d'une application. Pour optimiser les performances, il convient de lire seulement les données qui seront utilisées par l'application.

Par ailleurs, lorsque le sélecteur étoile est utilisé, les colonnes sont retournées dans le même ordre qu'elles sont présentes en base. Si une application utilise le champ étoile et qu'elle lit les informations en fonction de l'ordre dans lequel les colonnes sont retournées, il y aura une grosse erreur si la base de données est modifiée (modification d'un champ, modification de l'ordre des champs ...).

SQL WHERE

La commande WHERE dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition. Cela permet d'obtenir uniquement les informations désirées.

Syntaxe

La commande WHERE s'utilise en complément à une requête utilisant SELECT. La façon la plus simple de l'utiliser est la suivante:

```
SELECT nom_colonnes FROM nom_table WHERE condition
```

Exemple

Imaginons une base de données appelée "client" qui contient le nom des clients, le nombre de commandes qu'ils ont effectués et leur ville:

id	nom	nbr_commande	ville
1	Paul	3	paris

id	nom	nbr_commande	ville
2	Maurice	0	rennes
3	Joséphine	1	toulouse
4	Gérard	7	paris

Pour obtenir seulement la liste des clients qui habitent à Paris, il faut effectuer la requête suivante:

SELECT * FROM client WHERE ville = 'paris'

Cette requête retourne le résultat suivant:

id	nom	nbr_commande	ville
1	Paul	3	paris
4	Gérard	7	paris

Attention: dans notre cas tout est en minuscule donc il n'y a pas eu de problème. Cependant, si un table est sensible à la casse, il faut faire attention aux majuscules et minuscules.

Opérateurs de comparaisons

Il existe plusieurs opérateurs de comparaisons. La liste ci-jointe présente quelques uns des opérateurs les plus couramment utilisés.

Opérateur	Description
=	Égale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée (utile pour les nombres ou dates)
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot.
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

Attention: il y a quelques opérateurs qui n'existe pas dans des vieilles versions de système de gestion de bases de données (SGBD). De plus, il y a de nouveaux opérateurs non indiqués ici qui sont disponibles avec certains SGBD. N'hésitez pas à consulter la documentation de MySQL, PostgreSQL ou autre pour voir ce qu'il vous est possible de faire.

SQL AND & OR

Une requête SQL peut être restreinte à l'aide de la condition WHERE. Les opérateurs logiques AND et OR peuvent être utilisées au sein de la commande WHERE pour combiner des conditions.

Syntaxe d'utilisation des opérateurs AND et OR

Les opérateurs sont à ajoutés dans la condition WHERE. Ils peuvent être combinés à l'infini pour filtrer les données comme souhaités.

L'opérateur AND permet de s'assurer que la condition1 ET la condition2 sont vrai :

```
SELECT nom_colonnes FROM nom_table WHERE condition1 AND condition2
```

L'opérateur OR vérifie quant à lui que la condition1 OU la condition2 est vrai :

```
SELECT nom_colonnes FROM nom_table WHERE condition1 OR condition2
```

Ces opérateurs peuvent être combinés à l'infini et mélangés. L'exemple ci-dessous filtre les résultats de la table "nom_table" si condition1 ET condition2 OU condition3 est vrai :

```
SELECT nom_colonnes FROM nom_table WHERE condition1 AND (condition2
OR condition3)
```

Attention : il faut penser à utiliser des parenthèses lorsque c'est nécessaire. Cela permet d'éviter les erreurs car et ça améliore la lecture d'une requête par un humain.

Exemple de données

Pour illustrer les prochaines commandes, nous allons considérer la table "produit" suivante :

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	clavier	informatique	32	35

id	nom	categorie	stock	prix
3	souris	informatique	16	30
4	crayon	fourniture	147	2

Opérateur AND

L'opérateur AND permet de joindre plusieurs conditions dans une requête. En gardant la même table que précédemment, pour filtrer uniquement les produits informatique qui sont presque en rupture de stock (moins de 20 produits disponible) il faut exécuter la requête suivante :

```
SELECT * FROM produit WHERE categorie = 'informatique' AND stock
< 20
```

Cette requête retourne les résultats suivants :

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
3	souris	informatique	16	30

Opérateur OR

Pour filtrer les données pour avoir uniquement les données sur les produits "ordinateur" ou "clavier" il faut effectuer la recherche suivante :

SELECT * FROM produit WHERE nom = 'ordinateur' OR nom = 'clavier'

Cette simple requête retourne les résultats suivants:

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	clavier	informatique	32	35

Combiner AND et OR

Il ne faut pas oublier que les opérateurs peuvent être combinés pour effectuer de puissantes recherche. Il est possible de filtrer les produits "informatique" avec un stock inférieur à 20 et les produits "fourniture" avec un stock inférieur à 200 avec la recherche suivante :

```
SELECT * FROM produit WHERE ( categorie = 'informatique' AND
stock < 20 ) OR ( categorie = 'fourniture' AND stock < 200 )
```

Cela permet de retourner les 3 résultats suivants :

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	souris	informatique	16	30
4	crayon	fourniture	147	2

SQL ORDER BY

La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

Syntaxe

Une requête où l'on souhaite filtrer l'ordre des résultats utilise la commande ORDER BY de la sorte :

```
SELECT colonne1, colonne2 FROM table ORDER BY colonne1
```

Par défaut les résultats sont classés par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne. Par ailleurs, il est possible de trier sur plusieurs colonnes en les séparant par une virgule. Une requête plus élaborée ressemblerait à cela :

```
SELECT colonne1, colonne2, colonne3 FROM table ORDER BY colonne1
DESC, colonne2 ASC
```

A noter : il n'est pas obligé d'utiliser le suffixe "ASC" sachant que les résultats sont toujours classés par ordre ascendant par défaut. Toutefois, c'est plus pratique pour mieux s'y retrouver, surtout si on a oublié l'ordre par défaut.

Exemple

Pour l'ensemble de nos exemples, nous allons prendre une base "utilisateur" de test, qui contient les données suivantes :

id	nom	prenom	date_inscription	tarif_total
1	Durand	Maurice	2012-02-05	145
2	Dupond	Fabrice	2012-02-07	65
3	Durand	Fabienne	2012-02-13	90
4	Dubois	Chloé	2012-02-16	98
5	Dubois	Simon	2012-02-23	27

Pour récupérer la liste de ces utilisateurs par ordre alphabétique du nom de famille, il est possible d'utiliser la requête suivante :

```
SELECT * FROM utilisateur ORDER BY nom
```

Résultat :

id	nom	prenom	date_inscription	tarif_total
4	Dubois	Chloé	2012-02-16	98
5	Dubois	Simon	2012-02-23	27
2	Dupond	Fabrice	2012-02-07	65
1	Durand	Maurice	2012-02-05	145
3	Durand	Fabienne	2012-02-13	90

En utilisant deux méthodes de tri, il est possible de retourner les utilisateurs par ordre alphabétique ET pour ceux qui ont le même nom de famille, les trier par ordre décroissant d'inscription. La requête serait alors la suivante :

```
SELECT * FROM utilisateur ORDER BY nom, date_inscription DESC
```

Résultat :

id	nom	prenom	date_inscription	tarif_total
5	Dubois	Simon	2012-02-23	27
4	Dubois	Chloé	2012-02-16	98
2	Dupond	Fabrice	2012-02-07	65
3	Durand	Fabienne	2012-02-13	90
1	Durand	Maurice	2012-02-05	145

SQL GROUP BY

La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat. Sur une table qui contient toutes les ventes d'un magasin, il est par exemple possible de liste regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client.

Syntaxe d'utilisation de GROUP BY

De façon générale, la commande GROUP BY s'utilise de la façon suivante

```
SELECT colonne1, fonction(colonne2) FROM table GROUP BY colonne1
```

A noter : cette commande doit toujours s'utiliser après la commande WHERE et avant la commande HAVING.

Exemple d'utilisation

Prenons en considération une table "achat" qui résume les ventes d'une boutique :

id	client	tarif	date
1	Pierre	102	2012-10-23
2	Simon	47	2012-10-27
3	Marie	18	2012-11-05
4	Marie	20	2012-11-14
5	Pierre	160	2012-12-03

Ce tableau contient une colonne qui sert d'identifiant pour chaque ligne, une autre qui contient le nom du client, le coût de la vente et la date d'achat.

Pour obtenir le coût total de chaque client en regroupant les commandes des mêmes clients, il faut utiliser la requête suivante :

```
SELECT client, SUM(tarif) FROM achat GROUP BY client
```

La fonction SUM() permet d'additionner la valeur de chaque tarif pour un même client. Le résultat sera donc le suivant :

client	SUM(tarif)
Pierre	262
Simon	47
Marie	38

La manière simple de comprendre le GROUP BY c'est tout simplement d'assimiler qu'il va éviter de présenter plusieurs fois les mêmes lignes. C'est une méthode pour éviter les doublons.

Juste à titre informatif, voici ce qu'on obtient de la requête sans utiliser GROUP BY.

Requête :

SELECT client, SUM(tarif) FROM achat

Résultat :

client	SUM(tarif)
Pierre	262

client	SUM(tarif)
Simon	47
Marie	38
Marie	38
Pierre	262

Utilisation d'autres fonctions de statistiques

Il existe plusieurs fonctions qui peuvent être utilisées pour manipuler plusieurs enregistrements, il s'agit des fonctions d'agrégations statistiques, les principales sont les suivantes :

- **AVG()** pour calculer la moyenne d'un set de valeur. Permet de connaître le prix du panier moyen pour de chaque client
- **COUNT()** pour compter le nombre de lignes concernées. Permet de savoir combien d'achats a été effectué par chaque client
- **MAX()** pour récupérer la plus haute valeur. Pratique pour savoir l'achat le plus cher
- **MIN()** pour récupérer la plus petite valeur. Utile par exemple pour connaître la date du premier achat d'un client
- **SUM()** pour calculer la somme de plusieurs lignes. Permet par exemple de connaître le total de tous les achats d'un client

SQL LIMIT

La clause LIMIT est à utiliser dans une requête SQL pour spécifier le nombre maximum de résultats que l'ont souhaite obtenir. Cette clause est souvent associé à un OFFSET, c'est-à-dire effectuer un décalage sur le jeu de résultat. Ces 2 clauses permettent par exemple d'effectuer des système de pagination (exemple : récupérer les 10 articles de la page 4).

La syntaxe commune aux principales système de gestion de bases de données est la suivante :

```
SELECT *
FROM table
LIMIT 10
```

Cette requête permet de récupérer seulement les 10 premiers résultats d'une table. Bien entendu, si la table contient moins de 10 résultats, alors la requête retournera toutes les lignes.

Bon à savoir : la bonne pratique lorsque l'ont utilise LIMIT consiste à utiliser également la clause ORDER BY pour s'assurer que quoi qu'il en soit ce sont toujours les bonnes données qui sont présentées. En effet, si le système de tri

est non spécifié, alors il est en principe inconnu et les résultats peuvent être imprévisible.

Limit et Offset

L'offset est une méthode simple de décaler les lignes à obtenir. La syntaxe pour utiliser une limite et un offset est la suivante :

```
SELECT *
FROM table
LIMIT 10 OFFSET 5
```

Cette requête permet de récupérer les résultats 6 à 15 (car l'OFFSET commence toujours à 0). A titre d'exemple, pour récupérer les résultats 16 à 25 il faudrait donc utiliser: LIMIT 10 OFFSET 15

A noter : Utiliser OFFSET 0 reviens au même que d'omettre l'OFFSET.

SQL INSERT INTO

L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO. Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup.

Insertion d'une ligne à la fois

Pour insérer des données dans une base, il y a 2 syntaxes principales :

- Insérer une ligne en indiquant les informations pour chaque colonne existante (en respectant l'ordre)
- Insérer une ligne en spécifiant les colonnes que vous souhaiter compléter. Il est possible d'insérer une ligne renseignant seulement une partie des colonnes

Insérer une ligne en spécifiant toutes les colonnes

La syntaxe pour remplir une ligne avec cette méthode est la suivante :

```
INSERT INTO table VALUES ('valeur 1', 'valeur 2', ...)
```

Cette syntaxe possède les avantages et inconvénients suivants :

- Obliger de remplir toutes les données, tout en respectant l'ordre des colonnes

- Il n'y a pas le nom de colonne, donc les fautes de frappe sont limitées. Par ailleurs, les colonnes peuvent être renommées sans avoir à changer la requête
- L'ordre des colonnes doit resté identique sinon certaines valeurs prennent le risque d'être complétée dans la mauvaise colonne

Insérer une ligne en spécifiant seulement les colonnes souhaitées

Cette deuxième solution est très similaire, excepté qu'il faut indiquer le nom des colonnes avant "VALUES". La syntaxe est la suivante :

```
INSERT INTO table (nom_colonne_1, nom_colonne_2, ...)  VALUES
('valeur 1', 'valeur 2', ...)
```

A noter : il est possible de ne pas renseigner toutes les colonnes. De plus, l'ordre des colonnes n'est pas important.

Insertion de plusieurs lignes à la fois

Il est possible d'ajouter plusieurs lignes à un tableau avec une seule requête. Pour ce faire, il convient d'utiliser la syntaxe suivante :

```
INSERT INTO client (prenom, nom, ville, age) VALUES ('Rébecca',
'Armand', 'Saint-Didier-des-Bois', 24), ('Aimée', 'Hebert',
'Marigny-le-Châtel', 36), ('Marielle', 'Ribeiro', 'Maillères',
27), ('Hilaire', 'Savary', 'Conie-Molitard', 58);
```

A noter : lorsque le champ à remplir est de type VARCHAR ou TEXT il faut indiquer le texte entre guillemet simple. En revanche, lorsque la colonne est un numérique tel que INT ou BIGINT il n'y a pas besoin d'utiliser de guillemet, il suffit juste d'indiquer le nombre.

Un tel exemple sur une table vide va créer le tableau suivant :

id	prenom	nom	ville	age
1	Rébecca	Armand	Saint-Didier-des-Bois	24
2	Aimée	Hebert	Marigny-le-Châtel	36
3	Marielle	Ribeiro	Maillères	27
4	Hilaire	Savary	Conie-Molitard	58

SQL UPDATE

La commande UPDATE permet d'effectuer des modifications sur des lignes existantes. Très souvent cette commande est utilisée avec WHERE pour spécifier

sur quelles lignes doivent porter la ou les modifications.

Syntaxe

La syntaxe basique d'une requête utilisant UPDATE est la suivante :

```
UPDATE table SET nom_colonne_1 = 'nouvelle valeur' WHERE condition
```

Cette syntaxe permet d'attribuer une nouvelle valeur à la colonne nom_colonne_1 pour les lignes qui respectent la condition stipulé avec WHERE. Il est aussi possible d'attribuer la même valeur à la colonne nom_colonne_1 pour toutes les lignes d'une table si la condition WHERE n'était pas utilisée.

A noter, pour spécifier en une seule fois plusieurs modification, il faut séparer les attributions de valeur par des virgules. Ainsi la syntaxe deviendrait la suivante :

```
UPDATE table SET colonne_1 = 'valeur 1', colonne_2 = 'valeur 2',
colonne_3 = 'valeur 3' WHERE condition
```

Exemple

Imaginons une table "client" qui présente les coordonnées de clients.

Table client :

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	France
2	Pierre	18 Rue de l'Allier	Ponthion	51300	France
3	Romain	3 Chemin du Chiron	Trévérien	35190	France

Modifier une ligne

Pour modifier l'adresse du client Pierre, il est possible d'utiliser la requête SQL suivante :

```
UPDATE client SET rue = '49 Rue Ameline', ville = 'Saint-Eustache-la-Forêt',
code_postal = '76210' WHERE id = 2
```

Cette requête sert à définir la colonne rue à "49 Rue Ameline", la ville à "Saint-Eustache-la-Forêt" et le code postal à "76210" uniquement pour ligne où l'identifiant est égal à 2.

Résultats :

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	France

id	nom	rue	ville	code_postal	pays
2	Pierre	49 Rue Ameline	Saint-Eustache-la-Forêt	76210	France
3	Romain	3 Chemin du Chiron	Trévérien	35190	France

Modifier toutes les lignes

Il est possible d'effectuer une modification sur toutes les lignes en omettant d'utiliser une clause conditionnelle. Il est par exemple possible de mettre la valeur "FRANCE" dans la colonne "pays" pour toutes les lignes de la table, grâce à la requête SQL ci-dessous.

```
UPDATE client SET pays = 'FRANCE'
```

Résultats :

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	FRANCE
2	Pierre	49 Rue Ameline	Saint-Eustache-la-Forêt	76210	FRANCE
3	Romain	3 Chemin du Chiron	Trévérien	35190	FRANCE

SQL DELETE

La commande DELETE en SQL permet de supprimer des lignes dans une table. En utilisant cette commande associé à WHERE il est possible de sélectionner les lignes concernées qui seront supprimées.

Syntaxe

La syntaxe pour supprimer des lignes est la suivante :

```
DELETE FROMtableWHERE condition
```

Attention : s'il n'y a pas de condition WHERE alors **toutes** les lignes seront supprimées et la table sera alors vide.

Exemple

Imaginons une table "utilisateur" qui contient des informations sur les utilisateurs d'une application.

Table "utilisateur" :

id	nom	prenom	date_inscription
1	Bazin	Daniel	2012-02-13
2	Favre	Constantin	2012-04-03
3	Clerc	Guillaume	2012-04-12
4	Ricard	Rosemonde	2012-06-24
5	Martin	Natalie	2012-07-02

Supprimer une ligne

Il est possible de supprimer une ligne en effectuant la requête SQL suivante :

DELETE FROM `utilisateur` WHERE `id` = 1

Une fois cette requête effectuée, la table contiendra les données suivantes :

id	nom	prenom	date_inscription
2	Favre	Constantin	2012-04-03
3	Clerc	Guillaume	2012-04-12
4	Ricard	Rosemonde	2012-06-24
5	Martin	Natalie	2012-07-02

Supprimer plusieurs lignes

Si l'ont souhaite supprimer les utilisateurs qui se sont inscrit avant le **10/04/2012**, il va falloir effectuer la requête suivante :

```
DELETE FROMutilisateur WHERE date_inscription < '2012-04-10'
```

La requête permettra alors de supprimer les utilisateurs “Daniel” et “Constantin”. La table contiendra alors les données suivantes :

id	nom	prenom	date_inscription
3	Clerc	Guillaume	2012-04-12
4	Ricard	Rosemonde	2012-06-24
5	Martin	Natalie	2012-07-02

Il ne faut pas oublier qu'il est possible d'utiliser d'autres conditions pour sélectionner les lignes à supprimer.

Supprimer toutes les données

Pour supprimer toutes les lignes d'une table il convient d'utiliser la commande **DELETE** sans utiliser de clause conditionnelle.

```
DELETE FROM utilisateur
```

Supprimer toutes les données : DELETE ou TRUNCATE

Pour supprimer toutes les lignes d'une table, il est aussi possible d'utiliser la commande **TRUNCATE TABLE** de la façon suivante :

```
TRUNCATE TABLE utilisateur
```

Cette requête est similaire. La différence majeure étant que la commande TRUNCATE va ré-initialiser l'auto-incrémente s'il y en a un. Tandis que la commande **DELETE** ne ré-initialise pas l'auto-incrément.