E6 - Compétence

SQL

Introduction

Le SQL ou Structured Query Language, (langage de requête structurée), est un langage informatique qui permet d'exploiter des bases de données relationnelles. La manipulation des données en SQL permet d'effectuer des rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.

En outre ce langage permet plusieurs choses comme :

- La manipulation des données.
- La définition des données qui nous permet de créer et de modifier l'organisation des données.
- La partie langage de contrôle et de transaction qui permettent de commencer et de terminer des transactions
- La partie contrôle de données permet d'autoriser ou d'interdire l'accès à certaines données à certaines personnes.

Historique

En juin 1970, Edgar Frank Codd publia un l'article dans Communications of the AMC ''A Relational Model of Data for Large Shared Data Banks'' traduit par ''Un référentiel de données relationnel pour de grandes banques de données partagées''. Ce référentiel a été fondé sur la logique des prédicats (toute variable est un terme), cela a était reconnu comme un modèle très intéressant pour interroger les bases de données. Cela inspira le développement du langage Structured English Query Language (SEQUEL) (« langage d'interrogation structuré en anglais »), renommé en SQL pour cause de conflit de marque déposée.

Le SQL a été développer chez IBM en 1970 par Donald CHAMBERLIN et Raymond BOYCE. La première version a été conçue pour manipuler et éditer des données stockées dans les bases de données. Le système de la gestion de la base de chez IBM et System R.

En 1979, Relational Software, Inc. (Oracle Corporation) présenta la première version commercialement disponible de SQL, rapidement imité par d'autres fournisseurs. Le SQL a été adopté comme recommandation par l'Institut de normalisation américaine (ANSI) en 1986, puis comme norme internationale par l'ISO en 1987 sous le nom de ISO/CEI 9075 - Technologies de l'information - Langages de base de données - SQL.

L’utilisation

Le SQL s’utilise de différentes manières :

La première utilisation c’est de l’inclure dans un programme écrit dans un langage de programmation donné qui utilisera l'interface du SGBD pour lui transmettre des instructions en langage SQL.

La seconde utilisation est d’utilisé la technique dite embedded SQL, ce sont des instructions en SQL qui sont incorporées dans le code source d'un programme écrit dans un autre langage.

La troisième utilisation se base sur les procédures stockées, ce sont des fonctions écrites en SQL, elles sont enregistrées dans la base de données en vue d'être exécutées par le SGBD. Cette technique est utilisée pour les triggers les procédures et ainsi les déclenchées automatiquement sur modification du contenu a effectué dans la base de données.

Création des requêtes

Les requêtes SQL ont pour but de d’ajouter, sélectionner, modifier et supprimer des données qui sont stocké dans la base de données. Elles nous permettent de manipulez les données et d’affichée les données sélectionner.

SELECT / WHERE

```
SELECT nom_du_champ FROM nom_du_tableau WHERE notre condition ;
```

L’outil SELECT nous permet de sélectionner les attributs qui composent la table. Le FROM indique le nom de la table sélectionné et le WHERE est la condition pour l’exécution de la requête.

```
SELECT * FROM client WHERE ville = 'paris';
```

L’étoile (*) signifie que l’on sélectionne toutes les données de la table client selon la condition ou la ville doit être égale à paris. Cette requête nous affichera tous les clients habitant à Paris.

AND

On peut ajouter d’autre condition après le WHERE avec la commande AND.

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

OR

On peut aussi utiliser le OR qui sélectionnera ou notre condition1 ou la condition2.

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

Après un AND, on peut ajouter une condition avec des opérateurs tel que l'égalité, inférieur, supérieur, différent, strictement supérieur, strictement inférieur.

ORDER BY

Le ORDER BY permet de filtrer nos données dans un ordre croissant (ASC) ou décroissant (DESC).

```
SELECT colonne1, colonne2 FROM nom_table WHERE condition1 ORDER BY colonne1 DESC, colonne2 ASC
```

LIMIT

On peut aussi limiter la quantité de donnée que l'on sélectionne.

```
SELECT * FROM nom_table WHERE nom_colonne = condition1 LIMIT 5;
```

Les données de la table seront limitées à 5 résultats selon notre condition que l'on a fixés c'est à dire que l'on affichera uniquement cinq résultats.

BETWEEN

La fonctionnalité BETWEEN permet de sélectionner un intervalle de données. Entre la valeur1 et la valeur2 le AND et obligatoirement présent.

```
SELECT * FROM nom_table WHERE nom_colonne BETWEEN 'valeur1' AND 'valeur2'
```

LIKE

LIKE s'utilise dans la clause WHERE des requêtes SQL. Il permet d'effectuer des recherches en fonction d'un modèle.

```
SELECT * FROM nom_table WHERE colonne LIKE modele
```

Modèles doit être remplacé par :

LIKE '%a' : le caractère "%" est un caractère joker qui remplace tous les autres caractères. Il permet d'effectuer une recherche sur toutes les chaines de caractères finissant par ''a''.

LIKE 'a%' : il effectue la recherche de toutes les lignes commençant par "a".

LIKE '%a%' : il effectue la recherche de tous les enregistrements qui ont un "a".

LIKE 'pa%on' : il effectue la recherche de toutes les chaines qui commence par "pa" et qui se terminent par "on".

HAVING

La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX().

```
SELECT colonne1, SUM(colonne2) FROM nom_table GROUP BY colonne1 HAVING fonction(colonne2) operateur valeur
```

Cela permet donc de SÉLECTIONNER les colonnes DE la table “nom_table” en GROUPANT les lignes qui ont des valeurs identiques sur la colonne “colonne1” et que la condition de HAVING soit respectée.

Important : HAVING est très souvent utilisé en même temps que GROUP BY bien que ce ne soit pas obligatoire.

GROUP BY

La commande GROUP BY est utilisée pour grouper plusieurs résultats et les utiliser en une seule fonction. Cela permet d’éviter de faire une multitude de requetés différentes

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

AVG, COUNT, MAX, MIN, SUM,

Nous pouvons aussi établir des moyennes ainsi que sélectionner uniquement la plus grande des valeurs de la table Avec des fonctionnalités comme AVG, COUNT, MAX, MIN et SUM. Il se place toujours après le SELECT avec des paramètres pour savoir qu’elle nom de colonnes doit savoir si c’est la somme maximal, minimal, ou une moyenne.

```
SELECT AVG(nom_colonne) FROM nom_table
```

UPDATE

UPDATE nous permet d’effectuer des modifications sur des données existante.

```
UPDATE nom_table SET nom_colonne_1 = nouvelle valeur WHERE condition
```

INSERT INTO

INSERT INTO nous permet d’insérer des données récupéreration de formulaire et les insérer dans la base de données.

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

DELETE

DELETE permet de supprimer des lignes dans une table. Avec un WHERE on peut choisir la ligne à supprimer.

```
DELETE FROM nom_table WHERE condition
```

Jointure

Les jointures SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter les bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace. Cela permet d'éviter les croisements de données et permet aussi l'élimination de requêtes superflue.

Les jointures consistent à associer les lignes de deux tables en associant l'égalité des valeurs d'une colonne d'une première table par rapport à la valeur d'une colonne d'une seconde table.

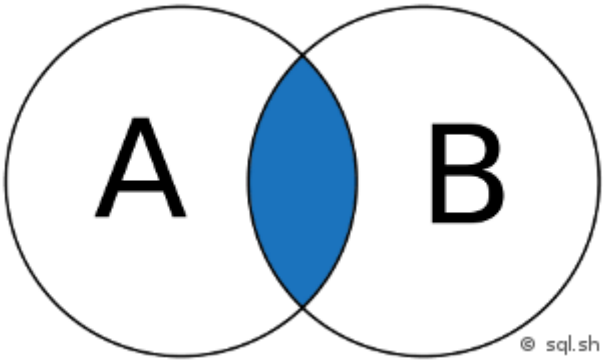
Prenons comme exemple qu'une base de données possède une table utilisateur et une autre table adresse. La table adresse, contient les adresses de ces utilisateurs. Avec une jointure, il est possible d'obtenir les données de l'utilisateur et de son adresse en une seule requête.

Il existe d'autres types de jointures, incluant des jointures sur la même table ou des jointures d'inégalité. Ces cas étant assez particulier, et rare d'en croiser.

Types de jointures

Il y a plusieurs façons pour associer 2 tables ensemble.

INNER JOIN : ou jointure interne pour retourner les enregistrements quand la condition est vraie dans les 2 tables. C'est l'une des jointures les plus communes et utilisée.

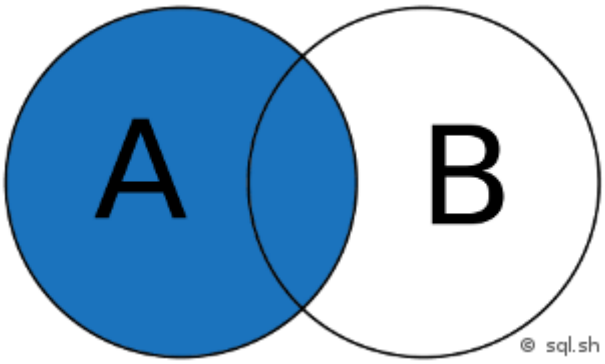



```
SELECT * FROM A INNER JOIN B ON A.key = B.key
```

CROSS JOIN : ou jointure croisée elle permet de faire un produit cartésien de 2 tables. En d'autres mots, elle permet de joindre chaque ligne d'une table avec chaque ligne d'une seconde table.

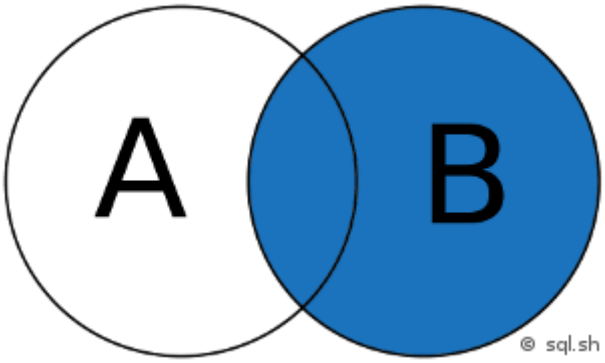
```
SELECT * FROM table1 CROSS JOIN table2
```

LEFT JOIN (ou LEFT OUTER JOIN) : ou jointure externe elle permet de retourner tous les enregistrements de la table de gauche même si la condition n'est pas vérifiée dans l'autre table.



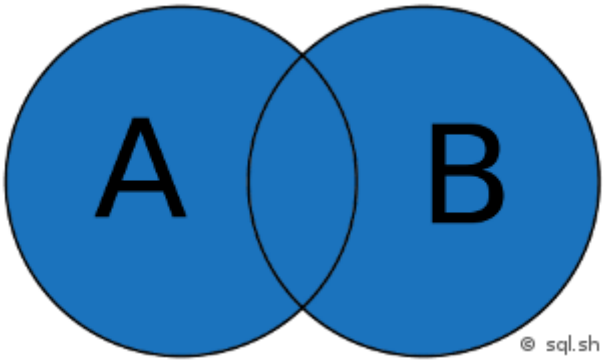

```
SELECT * FROM A LEFT JOIN B ON A.key = B.key
```

RIGHT JOIN (ou RIGHT OUTER JOIN) : ou jointure externe elle permet de retourner tous les enregistrements de la table de droite même si la condition n'est pas vérifiée dans l'autre table.




```
SELECT * FROM A RIGHT JOIN B ON A.key = B.key
```

FULL JOIN (ou FULL OUTER JOIN) : ou jointure externe elle permet de retourner les résultats quand la condition est vraie dans au moins une des 2 tables.




```
SELECT * FROM A FULL JOIN B ON A.key = B.key
```

SELF JOIN : ou jointure sur soi-même elle permet d'effectuer une jointure d'une table avec elle-même comme si c'était une autre table.

```
SELECT `t1`.`nom_colonne1`, `t1`.`nom_colonne2`, t2`.`nom_colonne1`,
`t2`.`nom_colonne2`
FROM `table` as `t1`
LEFT OUTER JOIN `table` as `t2`
ON `t2`.`fk_id` = `t1`.`id`
```

NATURAL JOIN : ou jointure naturelle entre 2 tables s'il y a au moins une colonne qui porte le même nom entre les 2 tables SQL.

```
SELECT * FROM table1 NATURAL JOIN table2
```

Les scripts de création de base de données

La création d'une base de données peut se faire avec des commande SQL. On appelle cela un script SQL, la plupart du temps il les base de données sont créés avec des SGBD. Mais toutes les commandes SQL sont possibles pour créer modifier, supprimer notre base de données.

```
CREATE TABLE IF NOT EXISTS `nom_de_la_table` (
  `id_colonne` int(11) NOT NULL AUTO_INCREMENT,
  `colonne1` varchar(100) NOT NULL,
  `colonne2` varchar(255) NOT NULL,
  `colonne3` datetime NULL,
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

On définira le type des colonnes (Integer, Varchar, Date time, etc…) on définit ensuite la longueur de la colonne, si elle peut être nulle ou non. Pour les id on peut définir directement l'AUTO_INCREMENT c'est-à-dire que cette colonne augmentera automatiquement à chaque nouvelle ligne crée dans la base de données. On spécifiera le moteur que l'on utilisera ici c'est InnoDB puis il suffira de définir le CHARSET (jeu de caractères).

```
ALTER TABLE `bien`
  ADD CONSTRAINT `bien_ibfk_1` FOREIGN KEY (`id_client`) REFERENCES
`client` (`id_client`),
  ADD CONSTRAINT `bien_ibfk_2` FOREIGN KEY (`id_type`) REFERENCES
`type` (`id_type`);
```

Pour la création des contrainte des clefs étrangères on est obligé d'utiliser un ALTER TABLE pour indiquer quelle table doit-on modifier et quelle est sa contrainte. On doit alors préciser le type de la contrainte dans l'exemple c'est une FOREIGN KEY de qui vient de la table Client vers la table Bien. Cela permet de connaitre par la suite quel client à déposer quel bien. Ce script est exécutable directement dans notre base de données sous PHPMyAdmin dans l'onglet SQL ou alors en important directement la base de données depuis sont chemin qui se trouvera sur notre ordinateur.

Avantages et Inconvénients

Avantages :

- Grande documentation sur sql.sh Et une communauté importante disponible pour nous aider.
- Utilisable dans plusieurs langages (PHP, C#, etc…).
- Souvent disponible sur les plateformes d'hébergement public.
- Très bien intégré dans l'environnement Apache et PHP.
- Facilité à prendre en main.
- Divers SGBD disponible avec de nombreuses différences ce qui permet de répondre à de nombreuses questions spécifiques.

Inconvénients :

- Tous les SGBD ne supportent pas de procédure stockée.
- Pas d'héritage de table.
- Manque de robustesse avec de fortes volumétries.

Conclusion

Nombreuses sont les possibilités offerte par le SQL pour manipulé les données. Il permet de crée une base de données, récupéré des informations ainsi que de nombreuse interaction entre notre base et notre code. Le SQL permet de faire cette liaison entre la base et le code cela permet d'affiché, supprimé, modifié ainsi que d'ajouter des données (SELECT, DELETE, UPDATE, INSERT INTO).

Les requêtes nous offre diverses possibilisées pour modifié tous les données que l'on à notre disposition dans notre base.

Bibliographie :

https://fr.wikipedia.org/wiki/Structured_Query_Language
https://sql.sh/cours/select
https://sql.sh/cours/where
https://sql.sh/cours/where/and-or
https://sql.sh/cours/order-by
https://sql.sh/cours/limit
https://sql.sh/cours/where/between
https://sql.sh/cours/where/like
https://sql.sh/cours/having

SQL INNER JOIN
SQL CROSS JOIN
SQL LEFT JOIN
SQL RIGHT JOIN
SQL FULL JOIN
SQL SELF JOIN
SQL NATURAL JOIN
SQL CREATE DATABASE
SQL DROP DATABASE
SQL CREATE TABLE
SQL PRIMARY KEY
SQL AUTO_INCREMENT