LE LANGAGE SQL

Présentation

Le langage **SQL** (***Structured Query Language***) est un standard reconnu par les **SGBD** (***Systèmes de Gestion de Bases de Données***), mais aussi par les applications plus généralistes utilisant une base relationnelle en leur cœur et par les langages autour des pages Web (*PHP*, *ASP*, *JSP*, etc).

Si certaines fonctionnalités permettent de constituer graphiquement une interrogation de base de données (***QBE*** ou Query By Example), elles présentent l'inconvénient de n'être utilisables que dans un logiciel déterminé. Lors d'une migration, toutes les interrogations qui auront ainsi pu être développées devront être reconstituées dans le nouvel environnement. Alors qu'une requête écrite en SQL reste utilisable en l'état avec n'importe quel *SGBD*.

Le langage SQL découle directement de la structure relationnelle des bases de données qu'il interroge. En effet, ces bases sont constituées selon des principes standards dont, principalement :

- Chaque table (issue d'une relation du modèle relationnel) est identifiée par un champ nommé **clé primaire**, qui aura une valeur unique pour chaque ligne de la table
- Les liens entre les tables sont réalisés par la présence d'une **clé étrangère** dans une table X référençant la ***clé primaire*** dans une table Y. Cela signifie que les seules valeurs autorisées pour la ***clé étrangère*** de la table X sont les valeurs préexistantes pour la ***clé primaire*** de la table Y. C'est une ***contrainte d'intégrité référentielle***.

I La structure de la requête SELECT

Le langage SQL permet d'interroger des bases de données sans maîtriser l'implantation des informations (notion de fichiers). Il propose la clause SELECT.

Celle-ci a pour vocation de retenir des informations à afficher (opération de PROJECTION) à partir du contenu de certaines tables. On limitera la recherche selon certaines conditions (opération de SELECTION). En outre, on devra reformuler les contraintes d'intégrité référentielles qui ont été paramétrées à partir des dépendances fonctionnelles du modèle relationnel (opération de JOINTURE) : pour cela, on indiquera quels champs doivent être mis en correspondance entre deux tables (la clé étrangère et la clé primaire).

Le langage propose quelques clauses permettant de structurer cette interrogation :

CLAUSE		Explication
SELECT		Liste les champs ou calculs à afficher ou restituer
FROM		Liste les tables nécessaires à l'interrogation
WHERE		Liste des conditions limitant la recherche d'information. N'apparaît qu'une fois
	AND	Précise que les conditions se cumulent
	OR	Précise des conditions alternatives
	NOT	Exprime le contraire d'une condition
ORDER BY		Effectue un tri croissant (ASC) ou décroissant (DESC) sur certains champs

CLAUSE	Explication
GROUP BY	Effectue un tri puis réalise des calculs par sous-catégories. Inclut donc la notion d'ORDER BY.
HAVING	Précise des conditions sur les résultats des calculs

1.1 Construire une requête simple

1. La construction d'une requête commence par l'identification des informations à afficher ⇒ Clause SELECT
2. On détermine ensuite les restrictions nécessaires à l'obtention du résultat ⇒ clause WHERE
3. On repère ensuite dans le modèle relationnel ou physique les tables concernées par l'ensemble des champs mis en œuvre dans les points 1 et 2 : on en déduit les tables nécessaires ⇒ clause FROM
4. On retrace sur le modèle relationnel l'ensemble des liens permettant de joindre les tables retenues au 3. Si nécessaire, on ajoutera à la clause FROM les tables traversées et non présentes au 3.
5. Enfin, on indique les jointures (égalités entre champs clé étrangère et la clé primaire référencée, pour reconstituer la contrainte d'intégrité) dans la clause WHERE.
6. On triera enfin grâce à la clause ORDER BY éventuelle.

1.2 Construire une requête avec calcul

On appliquera le déroulement ci-dessus, en ajoutant les étapes suivantes :

1. identifier le calcul à réaliser et les champs sur lesquels il porte: moyenne (AVG), somme de valeurs numériques (SUM), comptage de lignes (COUNT), plus grande valeur (MAX), plus petite valeur (MIN), etc. Ajouter ce calcul à la clause SELECT, en ajoutant éventuellement les tables à la clause FROM et les jointures à la clause WHERE
2. identifier les éventuels champs sur lesquels réaliser des regroupements clause GROUP BY
3. ajouter les champs de la clause GROUP BY à la clause SELECT (pas toujours obligatoire, mais plus pertinent d'un point de vue affichage)
4. Ajouter tout champ complémentaire de la clause SELECT à la clause GROUP BY
5. Traiter les restrictions éventuelles sur les calculs clause HAVING
6. Il n'y a pas de ORDER BY, la clause GROUP BY réalisant un tri.

1.3 Quelques règles de base

- Pour n tables dans la clause FROM, on trouvera la plupart du temps (n-1) jointures dans la clause WHERE.
- Tout champ affiché (SELECT) en plus d'un calcul (AVG, SUM, COUNT, etc) doit faire partie de la clause GROUP BY. Il peut y avoir des champs hors calculs dans le GROUP BY qui ne sont pas dans le SELECT mais jamais l'inverse
- Les restrictions portant sur les fonctions de calcul ne peuvent apparaître qu'après avoir effectué le regroupement, c'est à dire dans la clause HAVING
- La somme (SUM) ou la moyenne (AVG) portent nécessairement sur un champ numérique précis (ou un calcul à partir de champs numériques), alors que le comptage (COUNT) porte sur un identifiant ou les lignes d'une table

II LE LANGAGE SQL

2.1 Interrogations

Elles se font toutes à partir de l'instruction

```
SELECT ... FROM ... WHERE
```

2.1.1 Sélection, projection, jointure

Une projection correspond à l'information que l'on souhaite voir afficher au résultat. Elle est donc incluse dans la clause SELECT.

```
SELECT A1, A3 FROM R1
```

Une sélection correspond à des conditions exprimées sur certains champs. Ces conditions sont exprimées dans la clause WHERE.

```
FROM R1 WHERE A2=cond2 AND ...
```

Les jointures sont nécessaires pour établir les liens entre les diverses tables de l'interrogation. Elles apparaissent aussi dans la clause WHERE.

```
FROM R1, R2 WHERE R1.A3 = R2.B1
```

Remarque : Les jointures peuvent aussi s'exprimer grâce aux clauses IN ou EXISTS dans des sous requêtes (voir plus loin)

Jointures et ambiguïté

Lorsque l'on construit les tables des bases de données (ou lorsqu'on les génère avec un Atelier de génie Logiciel – AGL), il est coutumier de donner à la clé étrangère le même nom que la clé primaire qu'il référence.

Pour le langage SQL, chaque champ interrogé doit être distinguable sans ambiguïté, c'est à dire qu'il ne peut pas utiliser deux champs ayant exactement le même nom, ne sachant pas dans quelle table aller la chercher.

Aussi, lorsque l'on doit citer deux champs ayant le même nom dans des tables distinctes, il est impératif de préfixer chaque champ par le nom de la table auquel il appartient.

```
SELECT NomTable.NomChampAmbigu
```

2.1.2 Alias, jokers et conditions particulières

Il est possible, pour des raisons diverses, de renommer un champ ou une table (facilité d'écriture,

notamment).

- Renommer un champ :

```
SELECT NomDeChampTropLong AS Chp
```

- Renommer une table :

```
FROM TableDontLeNomEstTropLong    T
```

On peut aussi utiliser des jokers pour remplacer tout (*) ou une partie (?) de texte (ou de champ).

- Sélectionner tous les champs :

```
SELECT *
```

- Remplacer tous les caractères :

```
WHERE Nom LIKE "S*" ou WHERE Nom LIKE "S%"
```

- Remplacer certains caractères :

```
WHERE Code LIKE "ABC?5?"
```

Enfin, certaines conditions font appel à des clauses spécifiques pour gérer les intervalles et les ensembles

- Appartenance à un intervalle :

```
WHERE Qté BETWEEN 1 AND 10
```

- Appartenance à un ensemble :

```
WHERE Code IN ('5','10','15','20')
```

2.1.3 Tris et élimination des doublons

Pour trier un résultat, on utilisera la clause :

```
ORDER BY champ1 [ASC/DESC], champ2 [ASC/DESC], ...
```

Cette clause est placée après la dernière condition du WHERE et ne doit pas apparaître avec un GROUP BY..

Pour éviter qu'un résultat ne reprenne plusieurs fois les mêmes informations, on utilisera la clause DISTINCT

```
 SELECT DISTINCT champ1, champ2...
```

2.1.4 Fonctions statistiques et de regroupement

L'affichage de données à plat ne satisfait pas les besoins d'exploitation des bases de données. En effet, plus qu'une simple restitution de l'information telle qu'elle a été saisie, il est important de pouvoir réaliser des croisements, des statistiques, des éléments de synthèse à partir de l'information.

Les fonctions statistiques suivantes sont utilisables en SQL :

- Somme, moyenne, maximum et minimum d'une valeur numérique :

```
SELECT SUM(Champ), AVG(Champ), MAX(Champ), MIN(Champ)...
```

- Comptage des lignes ou des valeurs obtenues :

```
SELECT COUNT(*), SELECT COUNT(Champ)
```

Il est possible d'utiliser simplement ces fonctions (calculs sur toute les valeurs disponibles comme par exemple quel est le nombre de référence dans la base), ou de les associer à un regroupement pour chacune des valeurs d'un champ ou d'un ensemble de champs (nombre de titres par artiste, prix des achats par date et magasin...). La clause à utiliser est GROUP BY.

```
SELECT COUNT(*), Champ1, Champ2
FROM ...
WHERE ...
GROUP BY Champ1, Champ2
```

Conditions sur les résultats de calculs

Enfin, on peut être amené à exprimer des conditions particulières sur les fonctions statistiques (liste des artistes ayant plus de trois disques dans la base) </code>SELECT Count(*), Champ1, Champ2 FROM ... WHERE ... GROUP BY Champ1, Champ2 HAVING Count(*) > valeur</code>

Remarque : La clause GROUP BY/HAVING est toujours la dernière clause d'une requête.

2.1.5 Sous-requêtes

Les fonctions les plus complexes consistent à imbriquer les requêtes les unes dans les autres. L'idée est qu'une décomposition est nécessaire pour obtenir certains résultats utilisant, par exemple, deux fois la même table, des comparaisons avec des ensembles non prédéfinis, avec des fonctions statistiques... Une requête avec sous-requête est de la forme :

```
SELECT ....
FROM ...
WHERE ...
AND ...UnChamp.. COMPARAISON (SELECT MonChamp
                    FROM... WHERE....)
```

où COMPARAISON peut être :

- les signes =, >, <, NOT =... si l'on est certain que le résultat de la sous-requête donnera une valeur unique pour le champ MonChamp
- IN (ou NOT IN) si l'on souhaite que la valeur de Unchamp apparaisse (n'apparaisse pas) dans l'ensemble des valeurs du champ MonChamp retenues dans la sous-requête
- > ALL (ou < ALL) si l'on souhaite que la valeur de UnChamp soit supérieure (ou inférieure) à chacune des valeurs retenue pour le champ MonChamp

ou encore de la forme

```
SELECT ....
FROM ...
WHERE ...
AND NOT EXISTS (SELECT *
            FROM... WHERE.... AND .
    .        UnChamp.. = ..MonChamp..)
```

Cette forme permettra d'exprimer des questions du type : liste des élèves qui n'ont jamais été interrogés (tables élèves et interroger), liste des personnels sans affectation (tables personnel et affecter), liste des candidats non encore inscrits au brevet (tables candidats et inscription), liste des élèves ayant été absents au premier semestre (tables élèves et absence avec EXISTS).

2.2 Actions sur les données

Une dernière partie du langage de manipulation de données permet d'avoir une action dynamique sur la base : la possibilité de modifier le contenu des tables, d'ajouter de nouvelles informations ou de supprimer des lignes.

2.2.1 Modifications

Il s'agit de pouvoir mettre à jour le contenu d'un champ, en appliquant éventuellement des restrictions (sélections). Syntaxe :

```
UPDATE TABLE SET champ1 = valeur1 [,champ2 = valeur2...] WHERE condition
```

Exemples :

```
UPDATE Produit SET Pro_PrixUnit = Pro_PrixUnit * 10, Pro_DateTarif =
27/04/99 ;
        UPDATE Relance SET Rel_rappel = TRUE WHERE Rel_payé = FALSE ;
```

2.2.2 Ajouts

On ajoute, dans une table existante ou non des données à la main ou récupérées d'une autre table. Syntaxe 1 :

```
INSERT INTO TABLE [(champ1, champ3...)] VALUES (valeur1, valeur2...) ;
```

Exemples 1 :

```
INSERT INTO Produit VALUES ("PR055","Vis chantournée 2mm",12.10,100) ;
        INSERT INTO Produit (Pro_Code, Pro_Desgin) VALUES ("PR056","Clou
3mm");
```

Syntaxe 2 :

```
INSERT INTO TABLE [(champ1, champ3...)] SELECT....
```

Exemples 2 :

```
INSERT INTO Produit SELECT * FROM AnciensProduits ;
        INSERT INTO BonClients SELECT Ach_CliNum  FROM Acheter WHERE ach_CA
> 10000
```

2.2.3 Suppressions

Il s'agit de supprimer les lignes d'une table, avec la possibilité de les choisir selon certains critères.
Syntaxe :

```
DELETE [*] FROM TABLE WHERE condition
```

Exemple :

```
DELETE FROM Produit WHERE Pro_Type LIKE "*Clou*"
```

2.3 L'administration de bases de données

Toutes les manipulations évoquées précédemment supposent qu'une base existe et que l'on possède le droit d'effectuer les consultations et modifications proposées par le langage SQL. Dans la réalité, il est d'abord nécessaire de construire les tables pour pouvoir y insérer les données, et toute donnée ne doit pas être accessible à n'importe qui. Il faut donc définir les autorisations correspondant aux manipulations que l'on souhaite donner.`

2.3.1 Création d'une structure de base de données (version basique)

Création des comptes utilisateur

Avant de créer une table, il faudra choisir qui en est le propriétaire. Sous les environnements de type Access, OpenOffice ou Approach (SGBD micro), l'utilisateur par défaut est l'administrateur. Dans les SGBD plus sécurisés, il s'agit de l'utilisateur connecté. La création d'un utilisateur (le terme exact est schéma mais relève d'explications trop particulières) est établie par l'ordre SQL suivant :

```
CREATE USER nom_utilisateur IDENTIFIED BY mot_de_passe
…
```

Attribution des droits de gestion

L'administrateur devra alors donner à l'utilisateur un certain nombre de privilèges pour définir ce qu'il a le droit de faire sur la base (se connecter, créer des tables, ...). L'ordre SQL correspondant est GRANT, avec des privilèges système (c'est à dire sur le SGBD).

```
GRANT privilège [, privilège2…] TO nom_utilisateur
```

Parmi les privilèges système, on trouvera :

CREATE SESSION	Pour avoir les droits d'ouvrir une session
CREATE objet	Pour la création de tables (CREATE TABLE), synonymes (CREATE SYNONYM)...
ALTER objet	Pour la modification de la structure des objets
DROP objet	Destruction d'un objet

Création de la structure Pour créer la structure d'une table, l'ordre SQL est de la forme :

```
CREATE TABLE nom_table
(nom_champ1        TYPE        nullité,
 nom_champ2     TYPE        nullité,
 ….
 CONSTRAINT nom_contrainte PRIMARY KEY (nom_champ_clé_primaire) )
```

Les types de données dépendent du SGBD. On trouve généralement :

Entier	INTEGER
Nombre à virgule	NUMBER(x,y) pour x chiffres dont y après la virgule, DOUBLE ou SINGLE
Texte	VARCHAR(taille)

Gestion des contraintes

Il est possible de rajouter des contraintes d'intégrité référentielle sur les clés étrangères :

```
ALTER TABLE nom_table_ayant_clé_étrangère (
ADD CONSTRAINT nom_contrainte FOREIGN KEY (nom_champ_clé_étrangère)
REFERENCES table_ayant_clé_primaire (nom_clé_primaire) )
```

Exemple :

```
ALTER TABLE produit (ADD CONSTRAINT FK_Famille FOREIGN KEY (fam_code)
REFERENCES Famille(Fam_Code)
```

Attribution des droits de manipulation

Une fois toute la structure de la base créée, il faut définir les autorisations d'accès aux tables pour les utilisateurs (schémas). L'instruction GRANT est à nouveau utilisée mais dans un format différent, et cette fois avec des privilèges objet (c'est à dire sur les objets de la base comme les tables, les contraintes...)

```
GRANT privilège1 [,privilège2…] ON nom_table TO nom_utilisateur
```

Les privilèges objet sont entre autres :

SELECT	droit de lecture des données
UPDATE	droit de modification des données
INSERT	droit d'ajout
DELETE	droit de suppression

Organisation avec les rôles

Pour éviter d'avoir à réécrire les ordres un par un, objet par objet, on peut créer des rôles qui englobent un certain nombre de privilèges système ou objet.

```
CREATE ROLE nom_role
```

Qui sera pourra être suivi par l'attribution de privilèges :

- système : GRANT privilege1, privilege2, ... TO nom_role
- Objet : GRANT privilege1,privilege2, ON [nom_base.]nom_table/nom_colonne TO nom_Role.

On pourra alors attribuer un rôle à un ou plusieurs utilisateurs par GRANT nom_role TO nom_Utilisateur.

Des rôles prédéfinis existent pour les privilèges système. Par exemple sous Oracle, on dispose des rôles :

CONNECT	droits de connexion, de création de synonymes...
RESOURCE	droits de lecture de table...
DBA	droits d'administration de base de données.