

Les requêtes d'interrogation

Les requêtes de présentation des résultats

Sous-requêtes

Complément 1 : les requêtes de modification

Complément 2 : définition des données

Résumé

# Informatique en CPGE (2018-2019)

## Le langage SQL

S. B.

Lycée des EK

14 mai 2019

Le SQL (Structured Query Language = langage de requêtes structuré) est un langage informatique de dialogue avec une base de données relationnelle.

Une **relation** dans le modèle relationnel est une **table** dans le langage SQL.

Une requête est une question posée à une base de données. Nous allons voir comment sont écrites les requêtes de base en SQL.

## Les requêtes d'interrogation

Les requêtes de présentation des résultats

Sous-requêtes

Complément 1 : les requêtes de modification

Complément 2 : définition des données

Résumé

## La logique d'interrogation

Les opérations de base

Le champ calculé

Les fonctions d'agrégation

Les clauses de regroupement

Procédure	Opération relationnelle	Ordre SQL
Champs à afficher	Projection	SELECT
Tables concernées		FROM
Restrictions	Sélection Jointure	WHERE

Le mot `SELECT` indique la liste des champs à afficher ; pour afficher tous les champs on utilise `*`.

Le mot `FROM` indique à partir de quelles tables seront extraites les informations.

Toute requête SQL se termine par un point-virgule.

Par convention, les instructions SQL sont écrites en capitales dans le code d'un programme afin de les distinguer du langage de programmation.

## La projection

La projection permet de n'afficher qu'une partie des attributs ou champs (colonnes) d'une table.

Modèle relationnel : Eleves (Id, Nom, Prenom, Adresse, CP, Ville, Tel)

Clé primaire : Id

Requête 1 : Afficher toutes les informations concernant les élèves.

Requête SQL : **SELECT \* FROM Eleves ;**

Requête 2 : Afficher les noms et prénoms des élèves.

Algèbre relationnelle :  $\pi_{\text{Nom}, \text{Prenom}}(\text{Eleves})$

Requête SQL : **SELECT Nom, Prenom FROM Eleves ;**

Requête 3 : Afficher les villes dans lesquelles habitent des élèves.

Requête SQL : **SELECT DISTINCT Ville FROM Eleves ;**

Le mot clé Distinct permet de supprimer les doublons.

## La sélection

La sélection permet de n'afficher qu'une partie des lignes d'une table. On utilise le mot **WHERE** suivi du critère (ou des critères) de sélection.

Les opérateurs de comparaison utilisables sont :

**=** ou **!=** qui peuvent être utilisés avec tout type de données ;

**>**, **<**, **>=**, **<=** qui sont utilisables uniquement avec des données numériques ;

On peut aussi utiliser **LIKE** (Comme), **BETWEEN** (Entre), **IN**, **AND**, **OR**, **NOT**.

Pour des recherches sur des chaînes de caractères :  
% représente une chaîne de caractères quelconque ;  
\_ représente un caractère quelconque.

Champ au format texte ' ... '

Champ au format date 'mm/jj/aaaa'

Valeur de comparaison saisie par l'utilisateur [Texte à afficher]

Modèle relationnel : Eleves (Id, Nom, Prenom, Adresse, CP, Ville, Tel)

Clé primaire : Id

Requête 1 : Afficher les nom et prénom des élèves qui habitent Nice.

Algèbre relationnelle :  $\pi_{\text{Nom, Prenom}}(\sigma_{\text{Ville='Nice'}}(\text{Eleves}))$

Requête SQL : **SELECT Nom, Prenom FROM Eleves WHERE Ville= 'Nice' ;**

Requête 2 : Afficher le nom et le numéro de téléphone des élèves qui habitent à Nice ou à Cannes.

Requête SQL : **SELECT Nom, Tel FROM Eleves WHERE Ville LIKE 'Nice' OR Ville LIKE 'Cannes' ;**

Requête 3 : Afficher le nom et le prénom des élèves dont le numéro de téléphone commence par 06 et dont la première lettre du nom est comprise entre A et M.

Requête SQL : **SELECT Nom, Prenom FROM Eleves WHERE Tel LIKE '06%' AND Nom BETWEEN 'A' AND 'M' ;**

## La jointure

La jointure permet de mettre en relation plusieurs tables, par l'intermédiaire des liens qui existent en particulier entre la clé primaire de l'une et la clé étrangère de l'autre.

La jointure est une opération de sélection car elle permet de ne retenir que les enregistrements pour lesquels la valeur de la clé primaire d'une table correspond à la valeur de la clé étrangère d'une autre table.

Modèle relationnel :

Profs (Id, Nom, Prenom, Tel, Salle)

Clé primaire : Id

Eleves (Id, Nom, Prenom, Adresse, CP, Ville, Tel, Numprof)

Clé primaire : Id

Clé étrangère : Numprof en référence à Id de Profs

Algèbre relationnelle :  $\text{Eleves} \bowtie \text{Eleves.Numprof}=\text{Profs.Id} \text{ Profs}$   
 SQL : `Eleves JOIN Profs ON Eleves.Numprof=Profs.Id ;`

La jointure est équivalente à une sélection sur le produit cartésien :

Algèbre relationnelle :

$\sigma_{\text{Eleves.Numprof}=\text{Profs.Id}}(\text{Eleves} \times \text{Profs})$

SQL : `Eleves, Profs WHERE Eleves.Numprof=Profs.Id ;`

Requête : Afficher le nom des élèves et la salle où aura lieu le cours avec Monsieur Python.

Algèbre relationnelle :

$\pi_{\text{Eleves.Nom, Profs.Salle}}$

$(\sigma_{\text{Profs.Nom}='Python'}(\text{Profs} \bowtie_{\text{Eleves.Numprof}=\text{Profs.Id}} \text{Eleves}))$

Requête SQL : **SELECT Eleves.Nom, Profs.Salle FROM Eleves JOIN Profs ON Eleves.Numprof=Profs.Id WHERE Profs.Nom='Python' ;**

ou pour abrégé : **SELECT e.Nom, p.Salle FROM Eleves e  
JOIN Profs p ON e.Numprof=p.Id WHERE p.Nom='Python' ;**

Noter que c'est bien la même chose que l'extrait du produit  
cartésien Eleves  $\times$  Profs :

**SELECT Eleves.Nom, Profs.Salle FROM Eleves, Profs  
WHERE Eleves.Numprof=Profs.Id AND  
Profs.Nom='Python' ;**

On peut afficher des données résultant d'une ou plusieurs autres données et éventuellement d'un calcul. Ces nouvelles données sont affichées dans un nouveau champ créé pour l'occasion.

Modèle relationnel : Notes (Id, Maths, Physique, SI)

Clé primaire : Id

Requête 1 : Afficher l'identifiant des copies avec la note de Maths coefficientée par 5.

Requête SQL : **SELECT Notes.Id, Notes.Maths\*5 AS Points\_Maths FROM Notes ;**

Ces fonctions permettent d'effectuer des opérations mathématiques ou des calculs statistiques sur un ensemble d'enregistrements sélectionnés.

## Compter les enregistrements

On utilise la fonction COUNT.

Le résultat de l'opération sera affiché dans un nouveau champ.

Modèle relationnel : Notes (Id, Maths, Physique, SI)

Requête 1 : Compter le nombre de copies dont la note de Physique est supérieur à 8.

Requête SQL : **SELECT COUNT (Notes.Id) AS  
Nombre\_copies FROM Notes WHERE Notes.Physique > 8 ;**

## **Additionner les valeurs d'un champ numérique**

On utilise la fonction SUM comme la fonction COUNT avec la syntaxe SUM().

## **Calculer la moyenne des valeurs d'un champ numérique**

On utilise la fonction AVG (average=moyenne) comme la fonction COUNT avec la syntaxe AVG ().

## Afficher la valeur minimale d'un champ numérique

On utilise la fonction MIN comme la fonction COUNT avec la syntaxe MIN ().

## Afficher la valeur maximale d'un champ numérique

On utilise la fonction MAX comme la fonction COUNT avec la syntaxe MAX ().

Les clauses de regroupement permettent de réaliser des opérations sur des groupes d'enregistrements.

## La clause GROUP BY

La clause GROUP BY permet de créer des groupes d'enregistrements sur lesquels pourront être utilisées les fonctions d'agrégation. Elle est nécessaire dès lors que l'on souhaite afficher des données issues des tables et des données issues de fonctions d'agrégation.

Les champs de l'instruction SELECT doivent être repris dans la clause GROUP BY.

La syntaxe est : GROUP BY Champ1, Champ2, ...

Modèle relationnel : Notes (Id, Maths, Physique, SI, Classe)

Clé primaire : Id

Requête : afficher la classe avec la note maximale de Maths dans chaque classe.

Requête SQL :

```
SELECT Classe, MAX(Maths) FROM Notes GROUP BY Classe ;
```

## La clause HAVING

La clause HAVING permet d'appliquer des sélections sur les regroupements créés à l'aide de la clause GROUP BY.

Contrairement à l'ordre WHERE qui sélectionne des enregistrements, la clause HAVING sélectionne des résultats d'une fonction d'agrégation.

La syntaxe est : HAVING critères de sélection.

Requête 1 : afficher la classe avec la note maximale de Maths dans chaque classe où la moyenne est strictement supérieure à 10.

Requête SQL :

```
SELECT Classe, MAX(Maths) FROM Notes GROUP BY  
Classe HAVING AVG(Maths)>10 ;
```

Requête 2 : la même requête mais sur les élèves ayant une note de Physique strictement supérieure à 10.

Requête SQL :

```
SELECT Classe, MAX(Maths) FROM Notes WHERE  
Physique>10 GROUP BY Classe  
HAVING AVG(Maths)>10 ;
```

Attention à l'ordre : WHERE ... GROUP BY ... HAVING ...

La requête

```
SELECT Classe, MAX(Maths) FROM Notes WHERE  
Classe='PCSI' OR 'classe'='PTSI' GROUP BY Classe ;
```

est plus rapide et donc préférable à la requête

```
SELECT Classe, MAX(Maths) FROM Notes GROUP BY  
Classe HAVING Classe='PCSI' OR Classe='PTSI' ;
```

car on limite ainsi le nombre d'enregistrements qui devront être triés afin d'être groupés.

Il est possible de changer le nom d'une colonne qui est affichée :

Requête SQL : **SELECT moy AS 'moyenne informatique'**  
**FROM ... WHERE ... ;**

Le tri est une opération qui consiste à classer les enregistrements en fonction d'un ou plusieurs critères. La clause `ORDER BY` dispose de deux instructions de tri : `ASC` pour un tri dans l'ordre croissant et `DESC` pour un tri dans l'ordre décroissant.

Modèle relationnel : Elèves (Id, Nom, Prenom, Adresse, CP, Ville, Tel, Numprof)

Profs(Id, Nom, Prenom, Tel, Salle)

Requête : afficher les noms des élèves regroupés suivant le nom de leurs professeurs.

Requête SQL : **SELECT Eleves.Nom, Profs.Nom FROM Eleves, Profs WHERE Eleves.Numprof=Profs.Id ORDER BY Eleves.Numprof ASC ;**

On parle aussi de requêtes externes ou internes, de requêtes imbriquées.

Modèle relationnel : Notes (Id, Maths, Physique, SI, Classe)

Clé primaire : Id

Requête : afficher les identifiants des élèves ayant plus que la moyenne de tous les élèves en Maths.

Requête SQL :

```
SELECT id FROM Notes WHERE Maths > (SELECT  
AVG(Maths) FROM Notes);
```

Il est aussi possible d'utiliser le résultat d'une requête comme une nouvelle table à condition de lui donner un nom :

```
SELECT id FROM (SELECT * FROM Notes WHERE  
Physique<10) AS Newtab  
WHERE Maths > (SELECT AVG(Maths) FROM Notes) ;
```

Après FROM, il s'agit d'une table dérivée qui doit avoir son propre alias.

Il s'agit ici de modifier des données stockées dans les tables, c'est-à-dire des lignes ou n-uplets.

Les requêtes d'insertion permettent d'ajouter un enregistrement dans une table.

La syntaxe est : `INSERT INTO table (champ1, champ2, ...)  
VALUES ('valeur1', 'valeur2', ...);`

Modèle relationnel : Eleves (Id, Nom, Prenom, Adresse, CP, Ville, Tel, Numprof)

Requête SQL : **`INSERT INTO Eleves(Nom,Numprof)  
VALUES ('Toto','4');`**

Les requêtes de mise à jour permettent de modifier les données stockées dans les tables.

La syntaxe est : UPDATE table SET champs à modifier WHERE sélection ;

Modèle relationnel : Eleves (Id, Nom, Prenom, Adresse, CP, Ville, Tel, Numprof)

Requête : Modifier le numéro du professeur pour un élève donné.

Requête SQL : **UPDATE Eleves SET Eleves.Numprof='4' WHERE Eleves.Nom='Dede' ;**

Les requêtes de suppression permettent de supprimer des données stockées dans les tables. La suppression concerne l'intégralité de l'enregistrement.

La syntaxe est : **DELETE FROM <nom de table> WHERE <sélection> ;**

La création de tables ou de bases de données est complexe.  
Ceci sera étudié en TP avec l'importation et l'exportation de  
tables ou de bases.

Requête SQL : **DROP TABLE <nom table>** ;

Requête SQL : **ALTER TABLE <nom table> DROP COLUMN <nom attribut> ;**

L'ajout et l'augmentation de la taille d'un attribut seront étudiés en TP.

Nous étudions ici les requêtes d'interrogation.

**SELECT (DISTINCT) colonne1, colonne2, ... ou \*** pour tous les champs

**FROM table1, table2, ... , sous-requête1, sous-requête2, ...** ou jointure

**WHERE expression AND =** pour jointure si faite avec produit cartésien

**GROUP BY expression1, ...**

**HAVING expression**

nous pouvons compléter pour la présentation des résultats par

ORDER BY terme1 ordonnant,terme2 ordonnant ASC ou  
DESC

LIMIT expression

OFFSET expression