Informatique en CPGE (2018-2019)
Le langage SQL

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.

1 Les requêtes d'interrogation

1.1 La logique d'interrogation

Procédure	Opération relationnelle	Ordre SQL
Champs (colonnes) à afficher	Projection	SELECT
Tables concernées		FROM
Conditions de restriction à l'affichage	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.

1.2 Les opérations de base

1.2.1 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 : $\pi_{\text{Nom, Prenom}}\left(\text{Eleves}\right)$
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.

1.2.2 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]

Exemples :
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 : $\pi_{\text{Nom, Prenom}}\left(\sigma_{\text{Ville='Nice'}}\left(\text{Eleves}\right)\right)$
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' ;**

1.2.3 La jointure

La jointure permet de mettre en relation plusieurs tables, par l'intermédiaire des liens qui existent en particuler 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 : Eleves $\bowtie_{\text{Eleves.Numprof=Profs.Id}}$ Profs
SQL : Eleves JOIN Profs ON Eleves.Numprof=Profs.Id ;

La jointure est équivalente à une sélection sur le produit cartésien :
Algèbre relationnelle : $\sigma_{\text{Eleves.Numprof=Profs.Id}}\left(\text{Eleves} \times \text{Profs}\right)$
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 :
$\pi_{Eleves.Nom,Profs.Salle}\left(\sigma_{Profs.Nom='Python'}(\text{Profs} \bowtie_{Eleves.Numprof=Profs.Id} \text{Eleves})\right)$

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

ou pour abréger : **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' ;

1.3 Le champ calculé

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 ;

1.4 Les fonctions d'agrégation

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

1.4.1 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 ;

1.4.2 Additionner les valeurs d'un champ numérique

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

1.4.3 Calculer la moyenne des valeurs d'un champ numérique

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

1.4.4 Afficher la valeur minimale d'un champ numérique

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

1.4.5 Afficher la valeur maximale d'un champ numérique

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

1.5 Les clauses de regroupement

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

1.5.1 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 ;

1.5.2 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.

2 Les requêtes de présentation des résultats

2.1 Renommage de colonne

Il est possible de changer le nom d'une colonne qui est affichée :
Requête SQL : **SELECT moy AS 'moyenne informatique' FROM ... WHERE ... ;**

2.2 Le tri

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 ;

3 Sous-requêtes

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.

4 Complément 1 : les requêtes de modification

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

4.1 Les requêtes d'insertion de données

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') ;**

4.2 Les requêtes de mise à jour de données

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' ;**

4.3 Les requêtes de suppression de données

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 table> WHERE <sélection> ;**

5 Complément 2 : définition des données

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.

5.1 Suppression d'une table

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

5.2 Suppression d'un attribut

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.

6 Résumé

Nous étudions ici les requêtes d'interrogation.

6.1 Forme générale

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

Toute requête SQL se termine par un point-virgule. Par convention, les instructions SQL sont écrites en majuscule dans le code d'un programme afin de les distinguer du langage de programmation.

7 Utilisation de Python

Voici comment utiliser Python pour créer ou interroger une base avec des requêtes SQL (les réponses seront affichées dans l'interpréteur) :

Création d'une base :

```
import sqlite3 # importation du module

# création et connexion à la base concours blanc
conn=sqlite3.connect('ccb.db')
```

```
cur=conn.cursor() # création d'un curseur

# création de la première table (table Eleves)
cur.execute("""
CREATE TABLE IF NOT EXISTS Eleves(
     id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
     nom TEXT, prenom TEXT, maths REAL DEFAULT 0.00,
     physique REAL DEFAULT 0.00, classe TEXT
)
""")
conn.commit() # pour transmettre à la base de données

# insertion des données (type tuple)
data=[(1,'AAA','Toto',12,14,'psi'),
(2,'III','Tata',8,6,'psi'),
(3,'EEE','Titi',9,5,'psi'),
(4,'UUU','Tutu',11,8,'pt'),
(5,'OOO','Tyty',12,13,'pt')]

for d in data:
    cur.execute("""
    INSERT INTO Eleves(id,nom,prenom,maths,physique,
                       classe)
    VALUES(?, ?, ?, ?, ?, ?)""",d)
conn.commit()

# création de la seconde table (table Professeurs)
cur.execute("""
CREATE TABLE IF NOT EXISTS Professeurs(
     id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
     nom TEXT,
     classe TEXT,
     matiere TEXT
)
""")
conn.commit()

# insertion des données
data=[(1,'Java','psi','maths'),
(2,'Ceplus','psi','physique'),
(3,'Camel','pt','maths'),
(4,'Boa','pt','physique') ]

for d in data:
    cur.execute("""
    INSERT INTO Professeurs(id,nom,classe,matiere)
    VALUES(?, ?, ?, ?) """,d)
conn.commit()

cur.close()
conn.close() # déconnexion
```

Il est possible d'ajouter des données à la base à partir du fichier d'un tableur.

Pour cela, on enregistre le fichier au format csv en utilisant le point-virgule comme séparateur. On obtient un fichier texte :

nom ;prenom ;maths ;physique ;classe

;;;;

Aaa ;Toto ;8,5 ;9 ;psi
Iii ;Tata ;15 ;15 ;psi
Eee ;Titi ;8 ;7 ;psi
Uuu ;Tutu ;17,5 ;13 ;pt
Ensuite, on utilise le code Python suivant :

```
import sqlite3

conn=sqlite3.connect('ccb.db')
cur=conn.cursor()

fic=open('notes.csv','r')
fic.readline() # élimination des premières lignes inutiles
fic.readline()
donnees=[]
for ligne in fic:
    donnees.append(ligne.rstrip().split(';'))

# l'id eleve s'auto incrémente
for d in donnees:
    cur.execute("""
    INSERT INTO Eleves(nom,prenom,maths,physique,classe)
    VALUES(?, ?, ?, ?, ?)""",d)
conn.commit()

# test
R=" SELECT * FROM Eleves; "
cur.execute(R)
for t in cur:
    print(t)

cur.close()
conn.close()
```

Questions.

1. Ecrire une requête en langage SQL qui récupère depuis la table Eleves toutes les données de la classe PSI qui correspondent à des élèves ayant eu une note supérieure ou égale à 10 en Physique.
2. Ecrire une requête en langage SQL qui renvoie la plus haute des moyennes obtenues en mathématiques dans la classe.
3. Ecrire une requête en langage SQL qui renvoie pour chaque classe le nom de la classe et la moyenne des notes de mathématiques obtenue dans la classe.
4. Ecrire une requête en langage SQL qui détermine le nom du professeur de mathématiques donc la classe a eu la meilleure moyenne en mathématiques.

```
import sqlite3

conn=sqlite3.connect('ccb.db') # connection

cur=conn.cursor() # création d'un curseur

R1="""SELECT * FROM Eleves WHERE Classe='psi' AND physique>=10;"""
cur.execute(R1)
```

```
print("requête 1")
for t in cur:
    print(t)

R2="""SELECT Classe, AVG(maths) FROM Eleves GROUP BY Classe;"""
cur.execute(R2)
print("requête 2")
for t in cur:
    print(t)

R3="""SELECT MAX(m) FROM (SELECT AVG(maths) AS m
    FROM Eleves GROUP BY Classe); """
cur.execute(R3)
print("requête 3")
for t in cur:
    print(t)

R4="""SELECT Professeurs.nom FROM Professeurs WHERE matiere='maths'
    AND Classe=(SELECT Classe FROM
        (SELECT Classe, AVG(maths) AS m FROM Eleves
        GROUP BY Classe) AS tab
        WHERE m= (SELECT MAX(moy) FROM (SELECT AVG(maths)
        AS moy FROM Eleves GROUP BY Classe))); """
cur.execute(R4)
print("requête 4")
for t in cur:
    print(t)

cur.close() # déconnexion
conn.close()
```