

BASES DE DONNÉES IV
INTRODUCTION AU LANGAGE SQL
&
REQUÊTES ÉLÉMENTAIRES
(SÉANCE DE TP-COURS)

Laurent Kaczmarek

PCSI² 2013-2014
Lycée Louis Le Grand

Jeudi 05 juin 2014

**BASES DE
DONNÉES IV**
INTRODUCTION
AU LANGAGE
SQL
&
REQUÊTES
ÉLÉMENTAIRES
(SÉANCE DE
TP-COURS)

LAURENT
KACZMAREK

INTRODUCTION
AU LANGAGE
SQL

EXEMPLES DE
REQUÊTES SOUS
SQL*PLUS

REQUÊTES SOUS
PYTHON

SÉANCE DE TP

I

INTRODUCTION AU LANGAGE SQL

INTRODUCTION
AU LANGAGE
SQL

EXEMPLES DE
REQUÊTES SOUS
SQLITEMAN

REQUÊTES SOUS
PYTHON

SÉANCE DE TP

NOTRE FIL ROUGE

Nous illustrerons ce cours par la base de données à deux relations suivante :

► **population** :

ville	id	pop
Lille	01	225787
Douai	01	42621
Paris	02	2221000
Quimper	03	63235

► **region** :

id	nom
01	Nord-Pas-de-Calais
02	Ile-de-France
03	Bretagne

- En SQL, on distinguera les colonnes **id** de chacune de ces relations en écrivant **departement.id** et **population.id**.

ATTENTION, LES TABLES SONT TEMPORAIRES

Hormis les tables de la base de données, toutes les tables créées en SQL sont temporaires, on ne peut les enregistrer dans une variable. Il faudra donc effectuer chacune des requêtes en une seule fois.

LA PROJECTION

- C'est la commande **SELECT** qui permet d'effectuer une projection (et non une sélection).

- **SELECT ville, id FROM population :**

ville	id
Lille	01
Douai	01
Paris	02
Quimper	03

- On utilise **SELECT *** pour projeter sur toutes les colonnes.

LA PROJECTION (SUITE)

- ▶ Attention, la commande **SELECT** n'élimine pas les doublons. On obtiendra une sélection au sens de l'algèbre relationnelle en précisant **SELECT DISTINCT**.

EX : PROJECTION SUR **id** DANS POPULATION

SELECT id FROM population

id
01
01
02
03

**SELECT DISTINCT id FROM
population**

id
01
02
03

LA SÉLECTION

- ▶ On peut effectuer une sélection en ajoutant une clause **WHERE** à **SELECT**.
- ▶ **SELECT ville,pop FROM population WHERE id=01**

ville	pop
Lille	225787
Douai	42621

- ▶ On peut construire des conditions plus élaborées en utilisant les opérateurs logiques **AND**, **OR** et **NOT**.
- ▶ **SELECT ville,pop FROM population WHERE id=01 AND pop>45000**

ville	pop
Lille	225787

EX: SÉLECTION LES **id** AVEC VILLES DE **pop>70000**

► **SELECT DISTINCT id FROM population :**

id
01
02

RENOMMAGE

- C'est le mot-clé **AS** qui permet le renommage d'une colonne.
- **SELECT ville,id AS idd,pop FROM population**

ville	idd	pop
Lille	01	225787
Douai	01	42621
Paris	02	2221000
Quimper	03	63235

PRODUIT CARTÉSIEN

- ▶ On décrit un produit cartésien de tables en énumérant les tables après **FROM**.
- ▶ **SELECT * FROM population,region**

ville	population.id	pop	region.id	nom
Lille	01	225787	01	Nord-Pas-de-Calais
Lille	01	225787	02	Ile-de-France
Lille	01	225787	03	Bretagne
Douai	01	42621	01	Nord-Pas-de-Calais
Douai	01	42621	02	Ile-de-France
Douai	01	42621	03	Bretagne
Paris	02	2221000	01	Nord-Pas-de-Calais
Paris	02	2221000	02	Ile-de-France
Paris	02	2221000	03	Bretagne
Quimper	03	63235	01	Nord-Pas-de-Calais
Quimper	03	63235	02	Ile-de-France
Quimper	03	63235	03	Bretagne

- ▶ La présence du même attribut **id** dans les deux tables ne pose pas de problème car ils sont en fait manipulés comme **region.id** et **population.id**.

DIVISION CARTÉSIENNE

- Il n'existe aucune commande pour la division cartésienne en SQL; on peut la simuler par d'autres commandes.

JOINTURE

- Une jointure s'effectue au moyen de **JOIN ... ON**.
- **SELECT ville,pop,nom FROM (population JOIN region ON population.id=region.id)**

ville	pop	nom
Lille	225787	Nord-Pas-de-Calais
Douai	42621	Nord-Pas-de-Calais
Paris	2221000	Ile-de-France
Quimper	63235	Bretagne

- On peut aussi créer une jointure en parcourant un produit cartésien avec une clause **WHERE**. Cette méthode est plus lisible en cas de jointures multiples.
- **SELECT ville,nom FROM population,region WHERE population.id=region.id**

SÉLECTIONNER LES VILLES DE LA RÉGION NORD-PAS-DE-CALAIS

- ▶ **SELECT ville FROM population,region WHERE
population.id=region.id AND nom="Nord-Pas-de-Calais"**

<u>ville</u>
Lille
Douai

OPÉRATIONS ENSEMBLISTES

- ▶ L'opérateur **UNION** réalise la réunion de deux tables.
- ▶ **SELECT * FROM region UNION SELECT * FROM region**
- ▶ L'intersection n'existe pas sous SQL mais on peut la simuler à l'aide de jointures ou de sélections.
- ▶ Idem pour l'opération de différence.

COMPTAGE, MAX, MIN, SOMME ET MOYENNE

- ▶ Le comptage des enregistrements d'une table s'effectue au moyen de la fonction **COUNT**.
- ▶ **SELECT COUNT(*) AS total FROM region**

total
<hr/>
3
<hr/>

- ▶ **MAX, AVG, MIN** et **SUM** permettent de déterminer le maximum, la moyenne, le minimum et la somme d'une colonne.
- ▶ **SELECT MAX(pop) AS maximum FROM population**

maximum
<hr/>
2221000
<hr/>

GROUPEMENT

- ▶ On peut appliquer une fonction à un groupe de lignes en utilisant la commande **GROUP BY**.
- ▶ **SELECT nom,SUM(pop) as popu FROM population,region WHERE population.id=region.id GROUP BY nom**

nom	popu
Nord-Pas-de-Calais	268408
Ile-de-France	2221000
Bretagne	63235

- ▶ La clause **HAVING** remplace **WHERE** lorsque les colonnes intervenant dans la condition proviennent d'une fonction.
- ▶ **SELECT nom,SUM(pop) as popu FROM population,region WHERE population.id=region.id GROUP BY nom HAVING popu<65000**

nom	popu
Bretagne	63235

II

REQUÊTES SOUS SQLITEMAN

**BASES DE
DONNÉES IV**
INTRODUCTION
AU LANGAGE
SQL
&
REQUÊTES
ÉLÉMENTAIRES
(SÉANCE DE
TP-COURS)

**LAURENT
KACZMAREK**

INTRODUCTION
AU LANGAGE
SQL

EXEMPLES DE
REQUÊTES SOUS
SQLITEMAN

REQUÊTES SOUS
PYTHON

SÉANCE DE TP

ON CODE EN SQL DANS UN INTERPRÉTEUR

films.db - Sqliteman

Schema Pragmas

Base de données

- main
 - Tables (6)
 - acteur
 - entree
 - film
 - jeu
 - realisateur
 - utilisateur
 - Vues (0)
 - Catalogue système (1)

```
1 SELECT prenom FROM acteur;
```

Durée: 0.098 secondes * Col: 27 Ligne: 1/1

Vue complète Vue d'un élément Script de sortie

	prenom
1	Tim
2	Morgan
3	Bob
4	Marlon
5	Al
6	James

Requête OK
Ligne(s) retournée(s): 256 (Plus de lignes peuvent être récupérées. Faites défiler les résultats pour obtenir plus de lignes et/ou lisez la documentation.)

**BASES DE
DONNÉES IV**
INTRODUCTION
AU LANGAGE
SQL
&
REQUÊTES
ÉLÉMENTAIRES
(SÉANCE DE
TP-COURS)

**LAURENT
KACZMAREK**

INTRODUCTION
AU LANGAGE
SQL

EXEMPLES DE
REQUÊTES SOUS
SQLITEMAN

REQUÊTES SOUS
PYTHON

SÉANCE DE TP

III

REQUÊTES SOUS PYTHON

**BASES DE
DONNÉES IV**
INTRODUCTION
AU LANGAGE
SQL
&
REQUÊTES
ÉLÉMENTAIRES
(SÉANCE DE
TP-COURS)

**LAURENT
KACZMAREK**

INTRODUCTION
AU LANGAGE
SQL

EXEMPLES DE
REQUÊTES SOUS
SQLITEMAN

REQUÊTES SOUS
PYTHON

SÉANCE DE TP

LA BIBLIOTHÈQUE SQLITE3

- ▶ Il est possible de manipuler une base de données directement depuis un interpréteur Python.
- ▶ On utilise pour cela la bibliothèque sqlite3.
- ▶ On commence par créer *une connexion* à la base de données au moyen de la méthode connect :

```
conn=sqlite3.connect('nom.db')
```

- ▶ On crée alors *un curseur* au moyen de la méthode cursor :

```
c=conn.cursor()
```

- ▶ On peut alors utiliser du code SQL *encapsulé* au moyen de execute :

```
c.execute('SELECT...')
```

- ▶ On récupère le résultat au moyen de la méthode fetchall :

```
print(c.fetchall())
```


UN EXEMPLE

- ▶ `>>> import sqlite3`
- ▶ `>>> conn=sqlite3.connect('films.db')`
- ▶ `>>> c=conn.cursor()`
- ▶ `>>> c.execute('SELECT prenom FROM acteur')`
- ▶ `<sqlite3.Cursor object at 0xb582e760>`
- ▶ `>>> print(c.fetchall())`
- ▶ `[(u'Tim',), (u'Morgan',), (u'Bob',), ...]`

IV

SÉANCE DE TP

**BASES DE
DONNÉES IV**
INTRODUCTION
AU LANGAGE
SQL
&
REQUÊTES
ÉLÉMENTAIRES
(SÉANCE DE
TP-COURS)

**LAURENT
KACZMAREK**

INTRODUCTION
AU LANGAGE
SQL

EXEMPLES DE
REQUÊTES SOUS
SQL*MAN

REQUÊTES SOUS
PYTHON

SÉANCE DE TP

PLACE AU TP !

Nous allons illustrer le cours par une base de données sur le cinéma : **films.db**.

LA BASE DE DONNÉES FILMS

La base de données **films** est constituée de six relations. Voici son schéma relationnel :

- ▶ acteur[id,prenom,nom]
- ▶ entree[id_utilisateur,id_film,note]
- ▶ film[id,titre,annee,id_realisateur]
- ▶ jeu[id_film,id_acteur]
- ▶ realisateur[id,prenom,nom]
- ▶ utilisateur[id,prenom,nom]

À VOUS DE JOUER !