Requêtes sur une table

Lycée Thiers

Crédit

- Un cours de Quentin Fortier

Crédit

- Un cours de Quentin Fortier
- Le tuto SQL du W3C (indispensable)

Crédit

- Un cours de Quentin Fortier
- Le tuto SQL du W3C (indispensable)
- Un cours en français ici

Langages de requêtes

- On accède à des informations d'une base de donnée avec un *langage de requêtes*.

Langages de requêtes

- On accède à des informations d'une base de donnée avec un *langage de requêtes*.
- On n'utilise ni variable ni boucle contrairement aux langages de programmation habituels.

Langages de requêtes

- On accède à des informations d'une base de donnée avec un *langage de requêtes*.
- On n'utilise ni variable ni boucle contrairement aux langages de programmation habituels.
- On écrit (dans un certain langage) ce qu'on veut obtenir mais pas comment l'obtenir. On laisse le SGBD se débrouiller.

Somme des capacités de salles

D'après un exemple de Q. Fortier.
On cherche le nombre de places dans des salles de cinéma climatisées à Marseille.

- En Python

```
somme = 0
for salle in liste_des_salles:
    si climatisé(salle) and situé_à_Marseille(salle):
        somme+=capacité(salle)
```

Somme des capacités de salles

D'après un exemple de Q. Fortier.
On cherche le nombre de places dans des salles de cinéma climatisées à Marseille.

- En Python

```
somme = 0
for salle in liste_des_salles:
    si climatisé(salle) and situé_à_Marseille(salle):
        somme+=capacité(salle)
```

- Dans un langage de requête :

Calculer la somme des capacités des salles de cinéma climatisées à Marseille

SQL

Le langage de requêtes le plus utilisé est SQL (Structured Query Langage)
Plusieurs implémentations (avec des nuances dans la syntaxe)

- En **MPI**, nous utilisons MySQL open source, gratuit.

SQL

Le langage de requêtes le plus utilisé est SQL (Structured Query Langage)
Plusieurs implémentations (avec des nuances dans la syntaxe)

- En **MPI**, nous utilisons MySQL open source, gratuit.
- Oracle Database : propriétaire, payant

SQL

Le langage de requêtes le plus utilisé est SQL (Structured Query Langage)
Plusieurs implémentations (avec des nuances dans la syntaxe)

- En **MPI**, nous utilisons MySQL open source, gratuit.
- Oracle Database : propriétaire, payant
- PostgreSQL : open source, gratuit.

SQL

Le langage de requêtes le plus utilisé est SQL (Structured Query Langage)
Plusieurs implémentations (avec des nuances dans la syntaxe)

- En **MPI**, nous utilisons MySQL open source, gratuit.
- Oracle Database : propriétaire, payant
- PostgreSQL : open source, gratuit.
- Pour cours d'ITC, nous utilisons SQLite et le navigateur léger DB Browser for SQLite.

Syntaxe SQL

- Chaque requête est terminée par un point virgule « **;** »
- SQL n'est pas sensible à la casse (majuscules/minuscules) mais il est d'usage d'écrire les mots clés en majuscules et les noms de tables et colonnes en minuscules.

Types SQL

Les attributs peuvent être de type

- `INT` entier

Types SQL

Les attributs peuvent être de type

- `INT` entier
- `CHAR(k)` chaîne d'au plus k caractères

Types SQL

Les attributs peuvent être de type

- `INT` entier
- `CHAR(k)` chaîne d'au plus k caractères
- `FLOAT` (nombre flottant)

Types SQL

Les attributs peuvent être de type

- `INT` entier
- `CHAR(k)` chaîne d'au plus k caractères
- `FLOAT` (nombre flottant)
- `BOOLEAN` : bouléen (en fait 0 ou 1)

Types SQL

Les attributs peuvent être de type

- `INT` entier
- `CHAR(k)` chaîne d'au plus k caractères
- `FLOAT` (nombre flottant)
- `BOOLEAN` : bouléen (en fait 0 ou 1)
- D'autres types existent comme un type `TIME` mais le programme se limite aux 4 précédents.

Types SQL

Les attributs peuvent être de type

- `INT` entier
- `CHAR(k)` chaîne d'au plus k caractères
- `FLOAT` (nombre flottant)
- `BOOLEAN` : bouléen (en fait 0 ou 1)
- D'autres types existent comme un type `TIME` mais le programme se limite aux 4 précédents.
- Pour les dates, conformément au programme, on utilise des chaînes de caractères au format AAAA-MM-JJ : l'ordre lexicographique correspond alors à l'ordre chronologique.
  De même, les horaires sont écrits au format HH-MM-SS.

Création

La syntaxe de création de table n'est pas au programme.

- On crée une table **Utilisateur** avec une clé primaire **id** qui est incrémentée automatiquement à chaque nouvel utilisateur :

Création

La syntaxe de création de table n'est pas au programme.

- On crée une table **Utilisateur** avec une clé primaire **id** qui est incrémentée automatiquement à chaque nouvel utilisateur :
- Exemple en MYSQL pour le cours de **MPI** :

```
1     CREATE TABLE utilisateur
2   ( id INT AUTO_INCREMENT,
3     PRIMARY KEY(id),
4     nom VARCHAR(100),
5     prenom VARCHAR(100),
6     date_naissance DATE,
7     pays VARCHAR(255),
8     code_postal INT(5) );
```

Création

La syntaxe de création de table n'est pas au programme.

- On crée une table **Utilisateur** avec une clé primaire **id** qui est incrémentée automatiquement à chaque nouvel utilisateur :
- Exemple en MYSQL pour le cours de **MPI** :

```
1     CREATE TABLE utilisateur
2   ( id INT AUTO_INCREMENT,
3     PRIMARY KEY(id),
4     nom VARCHAR(100),
5     prenom VARCHAR(100),
6     date_naissance DATE,
7     pays VARCHAR(255),
8     code_postal INT(5) );
```

- **MPI** : Le type `VARCHAR(100)` indique que les chaînes de caractères ont au plus 100 lettres contrairement à `CHAR(100)` dont l'occupation en mémoire est figée.

Insertion

Syntaxe hors programme

```
INSERT INTO `utilisateur`
(`nom`, `prenom`, `date_naissance`, `pays`,
`ville`, `habite_marseille`)
VALUES
('DUPONT','Pierre','2002:11:30','UK','LONDON','0'),
('CAGOLE','MARIE','2001:02:23','F','ISTRES','1')
```

Contexte

- Une fois la table créée et remplie, on pose des questions sur son contenu.
- Pour faire tourner les exemples, se rendre sur W3C

Fonction identité

Afficher toutes les colonnes de la table client

```
SELECT * FROM Customers;
```

Choix de colonnes

Projection

On ne conserve que certaines colonnes, par exemple le nom de client et sa ville :

```
SELECT CustomerName, City FROM Customers;
```

Il s'agit d'une *projection* sur une ou plusieurs colonnes.

Filtrer des lignes

Clause WHERE (Sélection)

On ne conserve que certaines lignes dont les caractéristiques sont filtrées dans la clause WHERE. Il s'agit d'une *Sélection*.
Donner tous les renseignement sur les clients mexicains :

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

Filtrer des colonnes et des lignes

On peut enlever des lignes ET des colonnes.

Donner toutes les villes ou vivent des clients en Grande-Bretagne :

```
SELECT City FROM Customers
WHERE Country='UK';
```

City
London
London
London
London
Cowes
London
London

Enlever des doublons

`DISTINCT`

- On a vu que le SGBD travaille avec des multi-ensembles. Il n'est donc pas rare que les requêtes de projection renvoient des doublons (contrairement aux projections du cours de maths).

Enlever des doublons

`DISTINCT`

- On a vu que le SGBD travaille avec des multi-ensembles. Il n'est donc pas rare que les requêtes de projection renvoient des doublons (contrairement aux projections du cours de maths).
- Le mot clé `DISTINCT` supprime les doublons.

Enlever des doublons

DISTINCT

- On a vu que le SGBD travaille avec des multi-ensembles. Il n'est donc pas rare que les requêtes de projection renvoient des doublons (contrairement aux projections du cours de maths).
- Le mot clé DISTINCT supprime les doublons.
- Donner sans doublon les villes des clients anglais.

```
1 SELECT DISTINCT City FROM Customers
2 WHERE Country='UK';
```

City
Cowes
London

FIGURE – Une seule fois Londres

Opérateurs de comparaison

- = (et surtout pas ==)

Opérateurs de comparaison

- = (et surtout pas ==)
- <, <=

Opérateurs de comparaison

- = (et surtout pas ==)
- <, <=
- != (ou son équivalent <>)

Opérateurs de comparaison

- `=` (et surtout pas `==`)
- `<`, `<=`
- `!=` (ou son équivalent `<>`)
- `AND`, `OR`, `NOT`

Opérateurs de comparaison

- `=` (et surtout pas `==`)
- `<`, `<=`
- `!=` (ou son équivalent `<>`)
- `AND`, `OR`, `NOT`
- `LIKE` (voir plus loin)

Opérateurs de comparaison

- `=` (et surtout pas `==`)
- `<`, `<=`
- `!=` (ou son équivalent `<>`)
- `AND`, `OR`, `NOT`
- `LIKE` (voir plus loin)
- `IS NULL` (pour repérer les cases vides ou non renseignées) ; `IS NOT NULL` (pour repérer les cases non vides) ;

Calcul avec des colonnes

Renommage

- Mot clé `AS`

Calcul avec des colonnes

Renommage

- Mot clé `AS`
- La somme de la colonne Quantité avec le numéro de produit (ce qui ne signifie rien, bien sûr)

```
1 SELECT ProductID + Quantity AS Somme_DEBILE
2 FROM OrderDetails;
```

Le renommage permettra d'utiliser ce résultat dans des requêtes plus complexes.

Calcul de carré ou de racine

```
SELECT POW(4, 2), POW(4, 0.5), 4 * 4;
```

Observons que dans ce cas précis, on veut juste un résultat numérique sans lien avec aucune table. D'où l'absence de `FROM`

`POW` n'est pas un attendu du programme.

Comparaison

Opérateur LIKE

LIKE est utilisé pour chercher un motif particulier dans une colonne dont le domaine est CHAR.
Deux jokers sont utilisés en conjonction avec LIKE

- Le signe de pourcentage % représente zéro, un ou plusieurs caractères.
- L'underscore _ représente un seul caractère.

```
-- clients dont le nom commence par a
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';

/*clients dont le nom a n pour
2eme lettre et se termine par s*/
SELECT * FROM Customers
WHERE CustomerName LIKE '_n%s';
```

Trier avec `ORDER BY`

- Pour trier le résultat attendu par ordre croissant (par défaut -mot clé `ASC`) ou décroissant (mot clé `DESC`).

Trier avec `ORDER BY`

- Pour trier le résultat attendu par ordre croissant (par défaut -mot clé `ASC`) ou décroissant (mot clé `DESC`).
- Quand plusieurs colonnes sont indiquées, le tri se fait par ordre lexicographique.

Trier avec ORDER BY

- Pour trier le résultat attendu par ordre croissant (par défaut -mot clé ASC) ou décroissant (mot clé DESC).
- Quand plusieurs colonnes sont indiquées, le tri se fait par ordre lexicographique.

```
1 SELECT column1, column2, ...
2 FROM table_name
3 ORDER BY column1, column2, ... ASC|DESC;
```

Trier avec ORDER BY

- Pour trier le résultat attendu par ordre croissant (par défaut -mot clé ASC) ou décroissant (mot clé DESC).
- Quand plusieurs colonnes sont indiquées, le tri se fait par ordre lexicographique.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

- Trier les clients par ordre décroissant de pays et croissant de nom :

```
SELECT * FROM Customers
ORDER BY Country DESC, CustomerName ASC;
```

Trier avec ORDER BY

- Pour trier le résultat attendu par ordre croissant (par défaut -mot clé ASC) ou décroissant (mot clé DESC).
- Quand plusieurs colonnes sont indiquées, le tri se fait par ordre lexicographique.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

- Trier les clients par ordre décroissant de pays et croissant de nom :

```
SELECT * FROM Customers
ORDER BY Country DESC, CustomerName ASC;
```

- Coût d'un tri en $O(n \log n)$.

Limiter le nombre de lignes

LIMIT pour MYSQL

Attention, ce code SQL fonctionne avec **MySQL** mais pas **SQL SERVER**. Bien indiquer sur la copie qu'on travaille en MySQL ! ! Voir ce qu'en dit le W3SCHOOL

- Syntaxe en **MYSQL** et **SQLite** (diffère de **ORACLE**)

```
1 SELECT column_name(s)
2 FROM table_name
3 WHERE condition
4 LIMIT number;
```

Limiter le nombre de lignes

LIMIT pour MYSQL

Attention, ce code SQL fonctionne avec **MySQL** mais pas **SQL SERVER**. Bien indiquer sur la copie qu'on travaille en MySQL ! ! Voir ce qu'en dit le W3SCHOOL

- Syntaxe en **MYSQL** et **SQLite** (diffère de **ORACLE**)

```
1 SELECT column_name(s)
2 FROM table_name
3 WHERE condition
4 LIMIT number;
```

- Seulement les 3 premières lignes de la réponse :

```
1 SELECT * FROM Customers LIMIT 3;
```

Affiche les lignes 0,1,2. Essayer ici.

Limiter le nombre de lignes

Donner un point de départ : OFFSET

- Seulement 3 lignes après la sixième (incluse) :

```
1 SELECT * FROM Customers LIMIT 3 OFFSET 6;
```

Cela affiche les lignes 7,8,9. Noter la syntaxe : d'abord `LIMIT` ensuite `OFFSET`.

Limiter le nombre de lignes

Donner un point de départ : OFFSET

- Seulement 3 lignes après la sixième (incluse) :

```
1 SELECT * FROM Customers LIMIT 3 OFFSET 6;
```

Cela affiche les lignes 7,8,9. Noter la syntaxe : d'abord `LIMIT` ensuite `OFFSET`.

- Même chose avec une syntaxe abrégée :

```
1 SELECT * FROM Customers LIMIT 6, 3
```

Noter qu'on met alors le `OFFSET` avant le nombre de lignes. Essayer ici.

Limiter le nombre de lignes

Donner un point de départ : OFFSET

- Seulement 3 lignes après la sixième (incluse) :

```
1 SELECT * FROM Customers LIMIT 3 OFFSET 6;
```

Cela affiche les lignes 7,8,9. Noter la syntaxe : d'abord `LIMIT` ensuite `OFFSET`.

- Même chose avec une syntaxe abrégée :

```
1 SELECT * FROM Customers LIMIT 6, 3
```

Noter qu'on met alors le `OFFSET` avant le nombre de lignes. Essayer ici.

- `LIMIT k OFFSET 0` est équivalent à `LIMIT k`

Cases vides

NULL

- NULL est un mot clé indiquant une case vide.

Cases vides

NULL

- NULL est un mot clé indiquant une case vide.
- Deux opérateurs y sont associés :

Cases vides

NULL

- NULL est un mot clé indiquant une case vide.
- Deux opérateurs y sont associés :
  - IS NULL (pour repérer les cases vides ou non renseignées) ;

Cases vides

NULL

- NULL est un mot clé indiquant une case vide.
- Deux opérateurs y sont associés :
  - IS NULL (pour repérer les cases vides ou non renseignées) ;
  - IS NOT NULL (pour repérer les cases non vides) ;

Cases vides

NULL

- NULL est un mot clé indiquant une case vide.
- Deux opérateurs y sont associés :
  - IS NULL (pour repérer les cases vides ou non renseignées) ;
  - IS NOT NULL (pour repérer les cases non vides) ;
- Les noms de clients, celui de leur contact et leur adresse pour les clients dont le champ **Adress** est non vide

```
1 SELECT CustomerName, ContactName, Address
2 FROM Customers
3 WHERE Address IS NOT NULL;
```

Application d'une fonction d'agrégation

Il y en a 5 à connaître :

Correspondances	
Algèbre relationnelle	SQL
comptage ou *cardinal*	COUNT
max	MAX
min	MIN
somme	SUM
moyenne	AVG (pour « average »

Application d'une fonction d'agrégation

Il y en a 5 à connaître :

Correspondances	
Algèbre relationnelle	SQL
comptage ou *cardinal*	COUNT
max	MAX
min	MIN
somme	SUM
moyenne	AVG (pour « average »)

- Ces fonctions peuvent être utilisées pour des informations statistiques sur TOUTE la table ou bien les mêmes informations mais sur les éléments d'une PARTITION de la table (ce qu'on appelle des *agrégats*)

MIN,MAX,SUM

- Syntaxe :

```
1   SELECT MAX(nom_colonne) FROM table
```

Retourne une table d'une seule ligne et une seule colonne dont le nom est **MAX(nom_colonne)**

MIN,MAX,SUM

- Syntaxe :

```
1   SELECT MAX(nom_colonne) FROM table
```

Retourne une table d'une seule ligne et une seule colonne dont le nom est **MAX(nom_colonne)**

- Donner le prix minimum parmi les produits et renommer le résultat

```
1 SELECT MIN(Price) AS SmallestPrice
2 FROM Products;
```

Exo : idem avec prix maximum

MIN,MAX,SUM

- Syntaxe :

```
1  SELECT MAX(nom_colonne) FROM table
```

Retourne une table d'une seule ligne et une seule colonne dont le nom est **MAX(nom_colonne)**

- Donner le prix minimum parmi les produits et renommer le résultat

```
1  SELECT MIN(Price) AS SmallestPrice
2  FROM Products;
```

Exo : idem avec prix maximum

- Donner la somme des prix unitaires de la table produit.

```
1  SELECT SUM(Price) AS Somme
2  FROM Products;
```

MIN,MAX,SUM

- Syntaxe :

```
1  SELECT MAX(nom_colonne) FROM table
```

Retourne une table d'une seule ligne et une seule colonne dont le nom est **MAX(nom_colonne)**

- Donner le prix minimum parmi les produits et renommer le résultat

```
1  SELECT MIN(Price) AS SmallestPrice
2  FROM Products;
```

Exo : idem avec prix maximum

- Donner la somme des prix unitaires de la table produit.

```
1  SELECT SUM(Price) AS Somme
2  FROM Products;
```

- Donner les produits dont le prix unitaire est maximum (plusieurs réponses possibles) (attendre d'avoir vu les requêtes imbriquées)

Moyenne

Le prix unitaire moyen des produits :

```
SELECT AVG(Price) AS Moyenne
FROM Products
;
```

Compter

La fonction `COUNT` a un comportement particulier

- `COUNT(a)` : Compte le nombre de fois que a est différent de `NULL`.

Compter

La fonction `COUNT` a un comportement particulier

- `COUNT(a)` : Compte le nombre de fois que a est différent de `NULL`.
- Souvent on compte le nombre total d'enregistrements avec `COUNT(*)`.
  Exemple : donner le nombre de clients :

```
1 SELECT COUNT(*) AS Nombre_de_clients
2 FROM Customers;
```

Le mot clé IN

Un raccourci pour éviter de multiples conditions **OR**

Pas explicitement au programme (mais pas explicitement interdit)

- Syntaxe 1 :

```
1 SELECT column_name(s)
2 FROM table_name
3 WHERE column_name IN (value1, value2, ...);
```

Le mot clé IN

Un raccourci pour éviter de multiples conditions **OR**

Pas explicitement au programme (mais pas explicitement interdit)

- Syntaxe 1 :

```
1 SELECT column_name(s)
2 FROM table_name
3 WHERE column_name IN (value1, value2, ...);
```

- Syntaxe 2 : Dans des requêtes imbriquées (patience)

```
1 SELECT column_name(s)
2 FROM table_name
3 WHERE column_name IN (SELECT STATEMENT);
```

Le mot clé IN

- Tous les clients allemands, français et anglais :

```
1 SELECT * FROM Customers
2 WHERE Country IN ('Germany', 'France', 'UK');
```

Le mot clé IN

- Tous les clients allemands, français et anglais :

```
1 SELECT * FROM Customers
2 WHERE Country IN ('Germany', 'France', 'UK');
```

- Tous les clients qui sont dans le même pays qu'au moins un fournisseur : (patience, on en parlera quand on verra les requêtes imbriquées)