Numérique et Sciences Informatiques
Chapitre VI - Base de données relationnelles

Classe de terminale

Le développement des traitements informatiques nécessite la manipulation de données de plus en plus nombreuses. Leur organisation et leur stockage constituent un enjeu essentiel de performance. Le recours aux **bases de données** est aujourd'hui une solution très répandue. Ces bases de données permettent d'organiser, de stocker, de mettre à jour et d'interroger des données structurées volumineuses utilisées simultanément par différents programmes ou différents utilisateurs. Cela est impossible avec les représentations tabulaires étudiées en classe de première (comme le fichier csv).
Il existe différents types de base de données. Nous étudierons ici les bases de données relationnelles. Comme leurs noms l'indiquent, elles s'appuient sur le modèle relationnel.

I. Modèle relationnel

Une **relation** est considérée comme une table à 2 dimensions.
Elle contient des colonnes et des lignes.

Un **attribut** correspond à une colonne. A chaque attribut, on définit un **domaine**, c'est-à-dire l'ensemble des valeurs possibles prises par l'attribut.

Un **enregistrement** est un p-uplet mettant en relation les attributs. C'est donc une des lignes du tableau.

Une **entrée** (ou valeur) est une case du tableau.

Exemple

Voici un tableau à deux dimensions représentant une relation.

Database: test » Table: auteurs

id	annee	name	order	active
2	2012	Gazzotti	Gazzotti	1
3	2012	Dodier	Dodier	1
4	2012	Janry	Janry	1
5	2012	Fourquemin	Fourquemin	1
6	2012	Clarke	Clarke	1
7	2012	Mig	Mig	1
8	2012	Poipoi	Poipoi	1
9	2012	Darlot	Darlot	1
10	2012	Pilet	Pilet	1
11	2012	Mary Pumpkins	Mary Pumpkins	1
12	2012	Ernst	Ernst	1
13	2012	Walthéry	Walthéry	1
14	2012	Hamo	Hamo	1
15	2012	Carrère	Carrère	1
16	2012	Loyvet	Loyvet	1
17	2012	Krings	Krings	1
18	2012	Bodart	Bodart	1
19	2012	Carpentier	Carpentier	1
20	2012	Mabesoone	Mabesoone	1
21	2012	Juncker	Juncker	1

Cette relation s'appelle `auteurs`. Elle est composée de 5 attributs : `id`, `annee`, `name`, `order` et `active`.

Le domaine de l'attribut `id` est l'ensemble des nombres entiers (`integer`) alors que le domaine de l'attribut `name` est l'ensemble des chaines de caractères (`string`).

L'entrée de l'attribut `name` de l'enregistrement dont l'`id` vaut 4 est "Janry".

Un **schéma relationnel** est l'ensemble des relations d'une base de données.
Les liens entre ces relations sont stockées via des clés primaires et étrangères de 2 relations.

Une **clef primaire** (Primary Key) identifie de manière unique un enregistrement d'une relation. Elle ne peut être vide ('NULL'). Elle peut être composée d'un ou plusieurs attributs.

Une **clef étrangère** (Foreign Key) référence la clé primaire (donc un ou plusieurs attributs) d'une autre relation. Les entrées des attributs référencés doivent déjà exister.

Dans ce cours, nous nous restreindrons à des clefs primaires composées d'un seul attribut.

Exemple

Voici la relation `albums` de la même base de données que la relation `ia_auteur` précédente.

Base de données: [redacted] » Table: albums

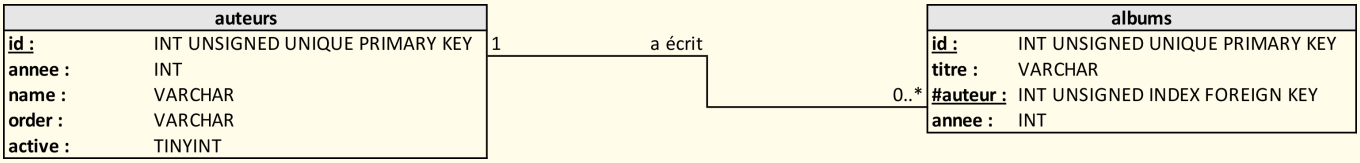
id	titre	auteur	annee
1	Soda	2	1991
6	Seuls	2	2006
7	Le petit Spirou	4	1990
8	Passe moi l'ciel	4	1999
9	Special Branch	14	2011
10	Le bon petit Henri	14	2016

L'attribut `auteur` de la relation `albums` est une clé étrangère et fait référence à l'attribut `id` de la relation `auteur`.

On peut établir le schéma relationnel suivant :

- auteurs : id (INT), annee (INT), name (VARCHAR), order (VARCHAR), active (TINYINT)
- albums : id (INT), titre (VARCHAR), #auteur (INT), annee (INT)

Ce schéma relationnel peut être représenté comme ci-dessous.



La relation `albums` possède une clef étrangère qui référence la clé primaire de la relation `auteurs`. Il y a donc une **association** ("a écrit") entre les relations `albums` et `auteurs`.
Sur cette association, deux informations numériques :

- une du côté de la relation `auteurs` : 1
- une du côté de la relation `albums` : 0..*

Ces informations sont toujours de la forme $x..y$, x et y étant des nombres avec $x \leq y$. Cela signifie n'importe quel nombre entre x et y. Dans le cas où $x = y$, on peut juste écrir x. Enfin, le symbole $*$ signifie "pas de limite".
Dans notre exemple, il peut y avoir des associations "a écrit" entre 1 enregistrement de la relation `auteurs` et au moins 0 enregistrement de la relation `albums`.
La création d'associations entre différentes relations permet d'éviter la redondance d'informations.

II. Base de données relationnelle

II.1. Généralités

Une **base de données** relationnelles est une collection d'informations organisées en tables, sur des sujets connexes. Ces dernières peuvent être associées ou non.

Un **système de gestion de bases de données relationnelles** (SGBDR), comme son nom l'indique, est un logiciel qui permet de gérer des bases de données relationnelles. On pourra notamment :

- créer une base de données
- créer ces tables
- modifier des tables (ajout ou suppression d'attributs)

Il rend différent services comme la garantie de :

- la qualité des informations et donc l'efficacité des traitements des requêtes
- la pérennité (persistance) des informations
- la confidentialité des informations et donc la sécurisation des accès
- la gestion des accès concurrents

Des systèmes de gestion de bases de données de très grande taille (de l'ordre du pétaoctet) sont au centre de nombreux dispositifs de collecte, de stockage et de production d'informations.

Nous utiliserons le SGBDR *phpmyadmin*.

II.2. Le langage SQL

L'accès aux données d'une base de données relationnelle s'effectue grâce à des **requêtes** d'interrogation et de mise à jour qui peuvent par exemple être rédigées dans le langage **SQL** (Structured Query Language).

Les requêtes que nous verrons permettent de :

- rechercher des entrées : `SELECT`
- ajouter des entrées dans une relation : `INSERT`
- modifier des entrées d'un enregistrement : `UPDATE`
- supprimer des enregistrements : `DELETE`

Pour la suite du cours, reprenons les relations précédentes.

Database: test » Table: auteurs

id	annee	name	order	active
2	2012	Gazzotti	Gazzotti	1
3	2012	Dodier	Dodier	1
4	2012	Janry	Janry	1
5	2012	Fourquemin	Fourquemin	1
6	2012	Clarke	Clarke	1
7	2012	Mig	Mig	1
8	2012	Poipoi	Poipoi	1
9	2012	Darlot	Darlot	1
10	2012	Pilet	Pilet	1
11	2012	Mary Pumpkins	Mary Pumpkins	1
12	2012	Ernst	Ernst	1
13	2012	Walthéry	Walthéry	1
14	2012	Hamo	Hamo	1
15	2012	Carrère	Carrère	1
16	2012	Loyvet	Loyvet	1
17	2012	Krings	Krings	1
18	2012	Bodart	Bodart	1
19	2012	Carpentier	Carpentier	1
20	2012	Mabesoone	Mabesoone	1
21	2012	Juncker	Juncker	1

Base de données: [illegible] » Table: albums

id	titre	auteur	annee
1	Soda	2	1991
6	Seuls	2	2006
7	Le petit Spirou	4	1990
8	Passe moi l'ciel	4	1999
9	Special Branch	14	2011
10	Le bon petit Henri	14	2016

II.2.a. Les tests conditionnels

Comme dans tout langage, les tests conditionnels prennent une place prépondérante.
En SQL, ils s'utilisent ainsi :

```
1 attribut opérateur valeur
```

Ils s'utilisent la plupart du temps après une commande `WHERE` ou `JOIN` (voir plus loin).

Les opérateurs possibles sont les suivants :

Opérateur	Description
=	est égal à, a pour valeur
<>	est différent de
!=	est différent de
>	est supérieur à
<	est inférieur à
>=	est supérieur ou égal à
<=	est inférieur ou égal à
IN	appartient à *une liste*
BETWEEN	appartient à *un intervalle*
LIKE	ressemble à, on spécifie le début la fin ou le mileu d'un mot, en utilisant les symboles `%` ou `_`
IS NULL	la valeur est nulle. *Remarque : ici il y a l'opérateur et la valeur.*
IS NOT NULL	la valeur n'est pas nulle. *Remarque : ici il y a l'opérateur et la valeur.*

II.2.b. La requête SELECT

Pour rechercher tous les enregistrement de la relation auteurs :

```
1 SELECT *
2 FROM auteurs;
```

2012	Gazzotti	Gazzotti	1	2
2012	Dodier	Dodier	1	3
2012	Janry	Janry	1	4
2012	Fourquemin	Fourquemin	1	5
2012	Clarke	Clarke	1	6
2012	Mig	Mig	1	7
2012	Poipoi	Poipoi	1	8
2012	Darlot	Darlot	1	9
2012	Pilet	Pilet	1	10

Pour chercher uniquement les auteurs présents aux festivals BD :

```
1 SELECT name
2 FROM auteurs;
```

name
Gazzotti
Dodier
Janry
Fourquemin
Clarke
Mig
Poipoi
Darlot
Pilet
Mary Pumpkins
Ernst
Walth
Hamo
Carr

On s'aperçoit cependant qu'il ne sont pas forcément rangés dans l'ordre alphabétique, ce qui n'est pas très lisible.

Pour l'ordre croissant :

```
1 SELECT name
2 FROM auteurs
3 ORDER BY name;
```

ou

```
1 SELECT name
2 FROM auteurs
3 ORDER BY name ASC;
```

name ▲ 1
Achd
Adeline Blondieau
Adeline Blondieau
Alain Dodier
Alain Dodier
Alain Dodier
Alain Dodier
Albert Uderzo
Alessandro Calore
Alexe
Alexe
Am
Ana
Andr

Pour l'ordre décroissant :

```
1 SELECT name
2 FROM auteurs
3 ORDER BY name DESC;
```

Remarque : l'ordre croissant ou décroissant se fait par rapport à l'interclassement choisi.

On s'aperçoit maintenant que certains noms apparaissent plusieurs fois. On peut demander de n'afficher les doublons(répétitions) qu'une fois.

```
1 SELECT DISTINCT name
2 FROM auteurs
3 ORDER BY name;
```

name ▲ 1
Achd
Adeline Blondieau
Alain Dodier
Albert Uderzo
Alessandro Calore
Alexe
Am
Ana
Andr
Annabel
Anne-Claire Jouvray
Antonio Lapone
Arleston
Arnaud Floc'h

Dans chacune de ces requêtes, nous n'avons demandé qu'un attribut mais nous pouvons en demander plusieurs, tous séparés par une virgule.

```
SELECT DISTINCT name, annee
FROM auteurs
ORDER BY name;
```

name ▲ 1	annee
Achd	2013
Adeline Blondieau	2014
Adeline Blondieau	2015
Alain Dodier	2013
Alain Dodier	2014
Alain Dodier	2015
Alain Dodier	2016
Albert Uderzo	2011
Alessandro Calore	2018
Alexe	2016
Alexe	2019
Am	2018
Ana	2020
Andr	2015

Certains noms apparaissent plusieurs fois malgré la commande `DISTINCT`. En effet, nous demandons des couples $(auteur, annee)$. Ces couples sont tous distincts. Il faut comprendre que chaque ligne est distincte de chaque autre.

Dans certains cas, on veut récupérer les entrées d'une table, liées aux entrées d'une autre. Pour cela, on utilise la commande `JOIN ... ON ...` :

```
SELECT DISTINCT auteurs.name, albums.titre , albums.annee
FROM auteurs

JOIN albums
ON albums.auteur=auteurs.id

ORDER BY name;
```

Remarquez qu'il est préférable de noter la relation à laquelle appartient l'attribut demandé : ceci permet de ne pas avoir d'erreur si on demande des attributs de même nom dans des tables différentes.

name ▲ 1	titre	annee
Gazzotti	Soda	1991
Gazzotti	Seuls	2006
Hamo	Special Branch	2011
Hamo	Le bon petit Henri	2016
Janry	Le petit Spirou	1990
Janry	Passe moi l'ciel	1999

Peut-être voudriez-vous les albums édités à partir de l'année 2000 ?

```
SELECT DISTINCT auteurs.name, albums.titre, albums.annee
FROM auteurs

JOIN albums
ON albums.auteur=auteurs.id

WHERE albums.annee>=2000
ORDER BY name;
```

name ▲ 1	titre	annee
Gazzotti	Seuls	2006
Hamo	Le bon petit Henri	2016
Hamo	Special Branch	2011

Peut-être voudriez-vous les albums édités à partir de l'année 2000 et dont le titre commence par un s ?

```
SELECT DISTINCT auteurs.name, albums.titre, albums.annee
FROM auteurs

JOIN albums
ON albums.auteur=auteurs.id

WHERE albums.annee>=2000 AND albums.titre LIKE 's%'
ORDER BY name;
```

name ▲ 1	titre	annee
Gazzotti	Seuls	2006
Hamo	Special Branch	2011

Si on veut généraliser la requête `SELECT`, on peut l'écrire ainsi :

```
SELECT attribut1, attribut2, ...
FROM relation

JOIN une_autre_relation
ON association_entre_les_relations

WHERE test_conditionnel1 (AND/OR test_conditionnel 2 AND/OR ...)
ORDER BY attribut1, attribut2, ...;
```

II.2.c. La requête INSERT

La requête `INSERT` permet d'ajouter un enregistrement dans une relation.
Elle s'utilise ainsi :

```
INSERT INTO relation
VALUES (les_entrées_séparées_par_des_virgules);
```

ou

```
INSERT INTO relation (liste_nom_attributs_à_remplir)
    VALUES (liste_des_valeurs_à_insérer_dans_ordre_liste_colonnes);
```

Pour insérer un nouvel album BD dans la relation `albums`, on écrit :

```
INSERT INTO albums VALUES (NULL, 'Sam Lawry',6,2001);
```

Remarquez que l'on demande d'enregistrer `NULL` pour l'attribut `id`. Cela vient du fait que cette entrée est auto-incrémentée et donc que le SGBDR indiquera automatiquement le bon nombre.

II.2.d. La requête UPDATE

La requête `UPDATE` permet de modifier un enregistrement déjà existant dans une relation.
Elle s'utilise ainsi :

```
UPDATE relation SET nom_attribut1=valeur1, nom_attribut2=valeur2, ...
WHERE test_conditionnel1 (AND/OR test_conditionnel2 AND/OR ...);
```

Dans l'exemple précédent, j'ai inséré le BD « Sam Lawry » dans la relation `albums` mais je me suis trompé de date et d'auteur.

```
UPDATE albums SET annee=2002, auteur=7
WHERE titre='Sam Lawry';
```

II.2.e. La requête DELETE

La requête `DELETE` permet de supprimer un enregistrement déjà existant dans une relation.
Elle s'utilise ainsi :

```
DELETE FROM relation
WHERE test_conditionnel1 (AND/OR test_conditionnel2 AND/OR ...);
```

Pour supprimer l'album « Sam Lawry », on indiquera :

```
DELETE FROM albums WHERE titre='Sam Lawry';
```

II.2.f. Remarques

Il faut respecter l'intégrité des données lorsqu'on insère un enregistrement dans une relation :

- On doit respecter le type de données (INT, CHAR, BOOLEAN, etc.)
- Une clé primaire doit être unique et non NULL :
  - On ne peut pas insérer une ligne avec une clé primaire qui existe déjà.
  - On ne peut pas modifier la valeur d'une clé primaire en une autre valeur qui existe déjà.
- Une clé étrangère doit référencer une clé primaire existante :
  - Il faut créer la ligne contenant la clé primaire avant une ligne contenant une clé étrangère la référençant.
  - On ne peut pas modifier une clé primaire si elle est déjà référencée.
  - On ne peut pas effacer une ligne contenant une clé primaire déjà référencée.

III. Interrogation de la base de données dans différents langages de programmation

Lorsqu'on crée un programme/logiciel, on peut avoir recours à une base de données. Il faut donc bien qu'il y ait une connexion entre le langage de programmation et la base de données.

Il faut alors créer des requêtes comme nous l'avons vu dans le paragraphe précédent, interroger la base de données et enfin envoyer la réponse dans un type de donnée connu du langage de programmation.

III.1. SQL et python

III.1.a. Connexion

Dans un premier temps, on se connecte à la base de données. Il aura fallu bien évidemment installer le paquet `mysql.connector`.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  port=8889,
  user="root",
  password="root",
  database="test"
)
```

On crée ensuite un curseur :

```
curseur = mydb.cursor(buffered=True)
```

Un curseur est un objet qui interagit avec le serveur MySQL en utilisant la connexion précédente.

Ensuite, on écrit les instructions du programme et notamment les requêtes à la base de donnée.

Enfin, on cloture le curseur et la connection.

```
curseur.close()
mydb.close()
```

III.1.b. Exécution d'une requête

Pour exécuter une requête, on utilise la commande `execute()`.

```
requete="SELECT * FROM albums;"
curseur.execute(requete)
```

La réponse de la requête est alors stockée dans la variable `curseur` sous forme d'une liste de p-uplets.

Pour accéder à l'enregistrement suivant (le premier si on utilise l'instruction pour la première fois), on utilise l'instruction `curseur.fetchone()`.
Pour accéder à plusieurs enregistrements, on utilise l'instruction `curseur.fetchmany(nb_enregistrements)`.
Pour accéder à tous les enregistrements, on utilise l'instruction `curseur.fetchall()`.

Les instructions `fetchmany()` et `fetchall()` renvoie une liste de tuple alors que l'instruction `fetchone()` renvoie un tuple.

instructions :

```
ligne=curseur.fetchone()
print(ligne)
```

Affichage :

```
(1, 'Soda', 2, 1991)
```

Instructions :

```
lignes=curseur.fetchmany(3)
print(lignes)
```

Affichage :

```
[(1, 'Soda', 2, 1991), (6, 'Seuls', 2, 2006), (7, 'Le petit Spirou', 4, 1990)]
```

Instructions :

```
lignes=curseur.fetchall()
print(lignes)
```

Affichage :

```
[(1, 'Soda', 2, 1991), (6, 'Seuls', 2, 2006), (7, 'Le petit Spirou', 4, 1990), (8, "Passe moi l'ciel",
(9, 'Special Branch', 14, 2011), (10, 'Le bon petit Henri', 14, 2016)]
```

Dans le cas de requêtes qui modifient la base de données (`INSERT` , `UPDATE`) il ne faut pas oublier de mettre à jour la base de données en écrivant l'instruction suivante :

```
mydb.commit()
```

III.2. SQL et php

Un fichier **PHP** est un fichier qui s'exécute sur le serveur. Ce dernier renvoie la réponse sous forme HTML au client. On peut donc noter que dans un même fichier, on peut écrire en PHP et en HTML. Le tout sera lisible par un navigateur.

III.2.a. Connexion

Dans un premier temps, on se connecte à la base de données.

```
<?php
    $link_sql = mysqli_connect ('localhost:8889', 'root', 'root', 'test');
    if (!$link_sql) {
        die('Erreur de connexion (' . mysqli_connect_errno() . ') '. mysqli_connect_error());
    }
?>
```

Ensuite, on écrit les instructions du programme et notamment les requêtes à la base de donnée.

Enfin, on cloture le curseur et la connection.

```
<?php
 mysqli_close($link_sql);
?>
```

III.2.b. Exécution d'une requête

```php
<?php
$requete="SELECT * FROM albums;"
$result = mysqli_query($link_sql,$query) or die ("requete invalide" . mysqli_error($link_sql));
?>
```

La réponse de la requête est alors stockée dans la variable `$result` sous forme d'une collection de tableaux.

En effet, la suite d'instructions suivante le montre.

```php
<?php
while ($element = mysqli_fetch_row($result)) {
    print_r($element);
    echo '<br/>';
}
?>
```

Affichage :

```
Array ( [0] => 1 [1] => Soda [2] => 2 [3] => 1991 )
Array ( [0] => 6 [1] => Seuls [2] => 2 [3] => 2006 )
Array ( [0] => 7 [1] => Le petit Spirou [2] => 4 [3] => 1990 )
Array ( [0] => 8 [1] => Passe moi l'ciel [2] => 4 [3] => 1999 )
Array ( [0] => 9 [1] => Special Branch [2] => 14 [3] => 2011 )
Array ( [0] => 10 [1] => Le bon petit Henri [2] => 14 [3] => 2016 )
```

Remarque

Dans l'exemple précédent, lorsqu'on rentre dans la boucle, la variable `$element` prend une nouvelle valeur sous forme d'un tableau, correspondant à un enregistrement de la relation *albums*.
Ainsi le premier élément de ce tableau correspond à l'`id`, le deuxième à `titre`, etc.
Pour récupérer le titre de l'album, on peut donc écrire l'instruction suivante :

```php
$element[1]
```

Il est donc important de connaître l'ordre des attributs dans la réponse à la requête.