Bases de données

Introduction

Une base de données (BD) est un fichier constitué d'un ensemble organisé et structuré d'informations sous la forme de tables (tableaux) indépendants et dont le but global est de modéliser un système réel. L'objectif est de gérer de manière efficace et structurée des données qui peuvent être de taille très importante.
Pour gérer les données enregistrées dans une base, on utilise un système de gestion de bases de données (SGBD). Ce sont des logiciels complexes qui permettent à des utilisateurs ou des programmes d'exprimer des requêtes pour interroger des bases de données ou pour les modifier.
Nous nous focaliserons ici sur les plus répandus d'entre ces systèmes, les systèmes relationnels, parmi lesquels nous trouvons des logiciels propriétaires (ORACLE, Microsoft SQL server, ...) et des logiciels libres très utilisés (MySQL, SQLite, ...). Python propose également un module (sqlite3) qui permet d'interroger des BD.
De gros efforts de standardisation ont été effectués et un langage de requêtes commun aux différents SGDB est né : **le langage SQL** (Structured Query Language). C'est ce langage que nous utiliserons;

1 Description d'une base de données

1.1 Schéma général

Le modèle relationnel est le principal modèle employé par les SGBD. Le but du modèle relationnel est de percevoir une réalité sous la forme de tableaux (tables) de valeurs à deux dimensions appelées relations :

1. une relation (table) a un nom ;
2. une colonne d'une relation est un **attribut** ;
3. une ligne (on dit aussi enregistrement) d'une relation est un **n-uplet** ou **tuple** ;

Pour illustrer nos propos, nous nous appuieront sur la base de données `commerce` contenant les trois tables `clients`, `commandes` et `pizzas`, que l'on figure ci-dessous :

Table `clients` :

Cle	Nom	Prenom	Adresse	Ville
100	Dubois	Pierre	18 rue des Roses	Dijon
101	Dubois	Amélie	18 rue des Roses	Dijon
102	Albert	Alberic	25 rue Devosges	Dijon
...	...	...	...	...

Table `pizzas` :

Cle	Nom	Prix
01	Margarita	12,5
02	Calzone	14
...	...	...

Table `commandes` :

Cle	Client	Pizza	Date	Livre	Paye
1	102	8	12/07/2015	1	1
2	107	5	12/05/2015	1	1
3	115	7	09/04/2015	1	0
...	...	...	...	...	...

1.2 Les attributs et leurs domaines

Seuls les attributs sont nommés : dans la table (on dit aussi relation) `pizzas` les noms des trois attributs sont `Cle` , `Nom` et `Prix` .
Les différentes valeurs prises par un même attribut appartiennent toutes à un même ensemble, appelé le **domaine de l'attribut**. Le schéma relationnel d'une table est la donnée de ses attributs et de leurs domaines.
Suivant son domaine, chaque attribut est représenté par un type. Les principaux types rencontrés sont les entiers (`INT`), les flottants (`REAL`) et les chaînes de caractères (`CHAR` ou `VARCHAR(20)` , l'entier précisant la longueur maximale). Celui de la table `commandes` est le suivant (on a fait figuré les types plutôt que les domaines) :

```
(Cle : INT ;  Client : INT ; Pizza : INT  ; Date : DATE ; Livre : INT ; Paye : INT  )
```

et celui de la table `pizzas` :

```
(Cle : INT ; Nom : CHAR ; Prix : FLOAT).
```

Les attributs ne sont pas ordonnés : on ne peut pas demander le " premier attribut" de la table. Par contre chaque attribut possède un nom qui lui est propre et qui permet d'identifier l'attribut de façon univoque (deux attributs distincts d'une même table ne peuvent pas avoir le même nom).

1.3 Clé primaire et clé étrangère

Dans la relation, l'ordre des lignes n'est pas important. En revanche, chaque tuple doit être unique. Pour les identifier, on définit une **clé primaire** qui peut être constituée d'un ou plusieurs attributs. En pratique la clé primaire est souvent limitée à un seul attribut qui prend des valeurs distinctes pour chaque entrée de la table.

Exemples : Dans la table `pizzas`, l'attribut `Cle` joue le rôle de clé primaire.

Quand il existe plusieurs clés possibles (par exemple dans la table `pizzas` , l'attribut `Nom` pourrait aussi l'être), on parlera de clé primaire (ou clé principale) pour la clé choisie et de clé secondaire pour les autres clés possibles. Pour des raisons d'efficacité lors de l'interrogation d'une table, il est nécessaire de disposer d'une clé la plus simple possible.
Une colonne (ou un ensemble de colonne) dont le rôle est de référencer une colonne d'une autre table est dénommé **clé étrangère**.

Exemples : Dans la table `commandes`, l'attribut `Pizza` est une clé étrangère qui référence la clé primaire `Cle` de la table `pizzas`. Ces clés sont essentielles lorsque l'on souhaite obtenir des informations qui nécessitent la lecture de plusieurs tables. Elles sont représentées par des flêches entre les tables (à faire apparaître sur le schéma). La donnée des tables d'une base de donnée et des clés étrangères qui les relient les unes aux autres constituent le schéma relationnel global de la base de données.

1.4 En pratique

Au lycée nous utiliserons le logiciel `SQLite browser`. Ce logiciel est disponible à l'adresse `https://sqlitebrowser.org/`.
On peut également utiliser `SQLite` en ligne à l'adresse `https://sqliteonline.com/`. Il faut alors charger la base de données que l'on souhaite manipuler.

2 SQL : un langage de définition et de manipulation de données

2.1 Définition de la structure de la base

Le langage de définition permet la modification du schéma d'une base de données. Il propose trois opérations : la création (`CREATE`), la suppression (`DROP`) et la modification (`ALTER`).
Par exemple, la création d'une table a pour syntaxe (ce qui apparaît entre crochet dans la syntaxe est facultatif) :

```
CREATE TABLE table ( définitions des colonnes [ contraintes de table ])
```

Dans la partie `définition des colonnes` de la commande `CREATE TABLE`, on introduira **le nom de chaque attribut suivi de son type**, chaque colonne sera séparée par une virgule.

Exemple : Création de la table `pizzas`

```
CREATE TABLE pizzas (
   Cle INTEGER,
   Nom CHAR,
   Prix FLOAT
   ) ;
```

2.2 Manipulation de données

Maintenant que le schéma de la BD a été défini (cf. paragraphe précédent), il reste à ajouter les données proprement dites dans la BD. C'est le rôle du langage de manipulation de données.

2.2.1 Insertion

Une insertion de n-uplets a la syntaxe suivante :

```
INSERT INTO table [( liste d'attributs )] VALUES ( liste de valeurs )
```

On peut préciser la liste des attributs à renseigner : les autres valeurs sont alors fixées à la valeur par défaut de l'attribut ou à `NULL`.
Exemple :

```
INSERT INTO Pizzas (Cle,Nom,Prix)
VALUES   (10, '4 Fromages', 15.8)
```

2.2.2 Suppression

L'instruction `DELETE` supprime des n-uplets d'une table : `DELETE FROM table [WHERE condition ]` .
La clause `WHERE` précise les n-uplets à supprimer. Sans clause `WHERE`, tous les n-uplets de la table sont supprimés.
Exemple :

```
DELETE FROM Pizzas
WHERE Prix > 17
```

2.2.3 Mise à Jour

L'instruction pour une mise à jour est `UPDATE`.
Exemple :

```
UPDATE  Pizzas
SET Prix = 13
WHERE Nom = 'Margarita'
```

3 Interrogation d'une base de données en langage SQL

Dans cette, nous allons détailler l'écriture de requêtes permettant d'interroger une BDD dans le langage SQL.

3.1 La projection et la sélection

La projection consiste à sélectionner certaines colonnes d'une table. Sélectionner les attributs A_i et A_j de la table R_1 s'écrit en SQL : `SELECT` A_i, A_j `FROM` R_1 ;
Exemple : Requête donnant le nom et le prénom de chaque client de la table `clients`.

```
SELECT Nom , Prenom FROM clients;
```

Remarques :

- La commande pour sélectionner la totalité des attributs et donc visualiser la totalité de la table R_1 s'écrit : `SELECT` * `FROM` R_1 ;
- On utilise pour éviter les doublons la commande : `SELECT DISTINCT`.
  Exemple : pour obtenir la liste des villes où habitent des clients à partir de la table `clients` :

  ```
  SELECT DISTINCT  Ville FROM  clients ;
  ```

La sélection consiste à sélectionner certaines lignes (sous-ensemble de tuples) qui vérifient une condition. Le schéma de la table résultat est identique à celui de la table sur laquelle porte la sélection.
Pour exprimer la condition, on pourra utiliser les opérateurs suivants : =,<> (différent), >, <, <=, >=, AND, OR, NOT.
Sélectionner les tuples tel que l'attribut A de la relation R_1 vérifie la condition c sécrit : `SELECT` * `FROM` R_1 `WHERE` $A = c$;
La plupart du temps, on veut faire à la fois une sélection et une projection.
Exemple : Requête retournant le nom des clients habitant Dijon.

```
SELECT  Nom FROM clients WHERE Ville='Dijon';
```

3.2 Tris des résultats d'une requête

La commande `ORDER BY` permet de trier les résultats d'une requête selon un critère.
Exemple : Requête permettant d'obtenir un tableau contenant la liste des pizzas par ordre de prix.

```
SELECT  *  FROM pizzas ORDER BY Prix ASC;
```

Ou pour avoir par ordre de prix decroissant :

```
SELECT  *  FROM pizzas ORDER BY Prix DESC;
```

Par défaut l'ordre est ascendant.

3.3 Jointure

3.3.1 Renommage

Un attribut "A" peut être renommé en "B". Le schéma du résultat est identique à celui sur lequel porte le renommage. Ce renommage n'est valable que pour la requête dans lequel il est défini, son but est principalement de faciliter la lecture et l'écriture des requêtes un peu complexes.
Un renommage s'écrit : `SELECT` A `AS` B `FROM` R_1 ;
On verra des exemples dans les paragraphes suivants.

3.3.2 Jointure

Une jointure consiste à combiner une paire de tuples de deux relations en un seul tuple, en regroupant les tuples qui ont un point en commun.
L'écriture d'une jointure où l'attribut A_i de la relation R_1 est égal à l'attribut A_j de la relation R_2 est :
`SELECT` * `FROM` R_1 `INNER JOIN` R_2 `ON` $R_1.A_i = R_2.A_j$;
Exemple : requête permettant d'obtenir un tableau contenant la liste des commandes (leur numéro) et le nom de la pizza commandée.

```
SELECT  commandes.Cle,pizzas.Nom
FROM commandes  INNER JOIN pizzas
               ON commandes.pizza=pizzas.Cle;
```

ou, en utilisant le renommage :

```
SELECT  com.Cle,piz.Nom
FROM commandes  AS com INNER JOIN pizzas  AS piz
               ON com.pizza=piz.Cle;
```

3.4 Les fonctions d'agrégation

Les fonctions d'agrégation dans le langage SQL permettent d'effectuer des opérations statistiques sur un ensemble de tuples. Les principales sont :

1. `AVG()` pour calculer la moyenne sur un ensemble de tuples
2. `COUNT()` pour compter le nombre de tuples sur une table ou une colonne distincte
3. `MAX()` pour récupérer la valeur maximum d'un attribut sur un ensemble de lignes. Cela s'applique à la fois pour des données numériques ou alphanumériques
4. `MIN()` pour récupérer la valeur minimum de la même manière que `MAX()`
5. `SUM()` pour calculer la somme sur un ensemble de tuples

Exemples : requête qui permet d'établir le prix moyen d'une pizza.

```
SELECT  AVG(Prix) FROM pizzas   ;
```

Requête qui permet d'obtenir le nombre de villes dont proviennent les clients.

```
SELECT  COUNT  (DISTINCT Ville) FROM clients   ;
```

On peut également utiliser les opérations numériques usuelles (+,-,*,/).
Exemples : requête qui permet d'obtenir l'amplitude de prix des pizzas.

```
SELECT  Max(Prix)-MIN(Prix) FROM pizzas   ;
```

4 Exercice

Écrire les requêtes qui permettent de répondre aux questions suivantes (et les tester) :

1. Quel est le prix de la pizza Margarita ? **Corrigé :**

```
SELECT Prix
FROM pizzas
WHERE Nom="Margarita";
```

2. (a) Quelles sont les pizzas dont le prix est compris entre 15 et 18 euros ? **Corrigé :**

```
SELECT nom
FROM pizzas
WHERE prix>15 AND prix<18;
```

   (b) Quel est le prix moyen d'une pizza ? **Corrigé :**

```
SELECT AVG(Prix)
FROM pizzas ;
```

3. (a) Donner les noms des villes d'où viennent les clients (sans redondance).
   Corrigé :

```
SELECT  DISTINCT Ville
FROM clients ;
```

   (b) Donner le nombre de clients venant de Dijon. **Corrigé :**

```
SELECT  COUNT(*)
FROM clients
WHERE Ville='Dijon';
```

   (c) Donner le nombre de villes où habitent les clients. **Corrigé :**

```
SELECT  COUNT(DISTINCT Ville)
FROM clients ;
```

4. (a) Déterminer les numéros des commandes effectuées pour le 12/07/2015.

   (b) Donner le nombre de commandes effectuées pour le 12/07/2015. **Corrigé :**

```
SELECT  COUNT(*)
FROM commandes
WHERE Date='12/07/2015';
```

   (c) Donner le nombre de commandes effectuées pour le 12/07/2015 ou le 13/07/2015. **Corrigé :**

```
SELECT  COUNT(*)
FROM commandes
WHERE Date='12/07/2015' OR Date='13/07/2015';
```

   (d) Donner le nombre de commandes effectuées pour le 12/07/2015 et qui ont été payées. **Corrigé :**

```
SELECT  COUNT(*)
FROM commandes
WHERE Date='12/07/2015'  AND Paye=1;
```

5. Jointures.

   (a) Quels sont les noms et prénoms des clients ayant une commande impayée ?
   Corrigé :

```
SELECT   DISTINCT Cl.Nom, Cl.Prenom
FROM commandes AS Co JOIN clients AS Cl
ON Co.Client=Cl.Cle
WHERE  Paye=0;
```

   (b) Quels sont les noms et prénoms des clients ayant commandé une Margarita ?
   Corrigé :

```
SELECT  DISTINCT Cl.Nom,Cl.Prenom
FROM (commandes AS Co JOIN clients AS Cl ON Co.Client=Cl.Cle)
      JOIN pizzas AS Pi ON Co.Pizza=Pi.Cle
WHERE  Pi.Nom='Margarita';
```

6. Mises à jour.

   (a) Écrire une commande qui insère dans la table `Clients` un nouveau client portant vos nom, prénom, ville. L'attribut `Cle` sera fixé à 200 et l'adresse ne sera pas renseignée.
   Vérifier par la commande `SELECT * FROM Clients`.

   (b) Écrire une commande qui enlève votre nom de la table `Clients`
   Vérifier par la commande `SELECT * FROM Clients`.

   (c) Écrire une commande qui augmente de 10% le prix des pizzas dont le prix est inférieur à 15 euros.
   Vérifier le résultats.