TP8

# Bases de données

L'administration, les banques, les assurances, les secteurs de la finance utilisent des bases de données, systèmes d'informations qui stockent dans des fichiers les données nombreuses qui leur sont nécessaires. Une base de données relationnelle permet d'organiser, de stocker, de mettre à jour et d'interroger des données structurées volumineuses utilisées simultanément par différents programmes ou différents utilisateurs. Un logiciel, le système de gestion de bases de données (SGBD), est utilisé pour la gestion (lecture, écriture, cohérence, actualisation...) des fichiers dans lesquels sont stockées les données. L'accès aux données d'une base de données relationnelle s'effectue en utilisant un langage informatique qui permet de sélectionner des données spécifiées par des formules de logique, appelées requêtes d'interrogation et de mise à jour. L'objectif est de présenter une description applicative des bases de données en langage de requêtes SQL (Structured Query Language). Il s'agit de permettre d'interroger une base présentant un très grand nombre de données à travers plusieurs relations.

## 8.1 Modèle relationnel

DÉFINITION 8.1

Une **base de données** est une collection de données, de n'importe quel type, qui vont être partagées entre plusieurs services, serveurs, utilisateurs.

Une base de données pourra contenir aussi bien du texte que des valeurs numériques. Elle peut être de très grande taille et contenir des milliers, voire des millions de données. L'important est de pouvoir organiser, classifier, de manière lisible et rigoureuse ces données afin notamment de permettre une recherche efficace au sein de cette base de données.
Il existe plusieurs modèles d'organisation décrivant les différents éléments et les relations qui existent entre eux, nous nous restreindrons cette année au **modèle relationnel** : dans une base de données organisée suivant le modèle relationnel, les données sont organisées sous forme de tables, encore appelées relations, qui sont associées entre elles en respectant certaines règles.
Les enjeux associés à la gestion d'un très grand nombre de données sont complexes : il s'agit de faciliter l'insertion de nouvelles données ou la recherche de données existantes, mais également d'assurer l'accès à ces données par différents utilisateurs qui peuvent utiliser cette base de données potentiellement simultanément.

DÉFINITION 8.2

Une **relation** est une table qui contient des données. Chaque colonne de cette table correspond à un **attribut** de celle-ci, qui permet d'identifier une information stockée dans cette relation. Une ligne de la relation est appelée un **enregistrement**.

DÉFINITION 8.3

On appelle **domaine** d'un attribut contenu dans une table le type de données qu'il contient. Les principaux domaines sont les entiers (`integer`), les chaînes de caractères (`text`), les nombres décimaux (`float`), etc...

EXEMPLE.
On s'intéresse à différents indicateurs géographiques et démographiques par communes en France. Pour cela, on pourra regrouper les données dans une relation que l'on appellera tout naturellement `communes`. Cette relation contiendra 36699 enregistrements (les 36699 villes de France et Outre-Mer) et se présentera de la manière suivante :

Table | Formulaire

Nombre de lignes chargées : 36699

| | num_departement | nom | canton | population_2010 | population_1999 | surface | longitude | latitude | zmin | zmax |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 01 | Ozan | 26 | 618 | 469 | 6.6 | 2866 | 51546 | 170 | 205 |
| 2 | 01 | Cormoranche-sur-Saône | 27 | 1058 | 903 | 9.85 | 2772 | 51379 | 168 | 211 |
| 3 | 01 | Plagne | 03 | 129 | 83 | 6.2 | 3769 | 51324 | 560 | 922 |
| 4 | 01 | Tossiat | 25 | 1406 | 1111 | 10.17 | 3309 | 51268 | 244 | 501 |
| 5 | 01 | Pouillat | 33 | 88 | 58 | 6.23 | 3435 | 51475 | 333 | 770 |
| 6 | 01 | Torcieu | 28 | 698 | 643 | 10.72 | 3398 | 51025 | 257 | 782 |
| 7 | 01 | Replonges | 02 | 3500 | 2841 | 16.6 | 2833 | 51456 | 169 | 207 |
| 8 | 01 | Corcelles | 06 | 243 | 222 | 14.16 | 3597 | 51151 | 780 | 1081 |
| 9 | 01 | Péron | 12 | 2143 | 1578 | 26.01 | 3989 | 51322 | 411 | 1501 |
| 10 | 01 | Relevant | 30 | 466 | 367 | 12.38 | 2903 | 51211 | 235 | 282 |
| 11 | 01 | Chaveyriat | 10 | 927 | 810 | 16.87 | 3026 | 51331 | 188 | 260 |
| 12 | 01 | Vaux-en-Bugey | 17 | 1169 | 1003 | 8.22 | 3352 | 51031 | 252 | 681 |
| 13 | 01 | Maillat | 22 | 668 | 664 | 11.31 | 3557 | 51255 | 497 | 825 |
| 14 | 01 | Faramans | 19 | 681 | 591 | 11.22 | 3099 | 51002 | 255 | 306 |
| 15 | 01 | Béon | 09 | 373 | 364 | 10.3 | 3798 | 50950 | 228 | 1412 |
| 16 | 01 | Saint-Bernard | 34 | 1375 | 1281 | 3.15 | 2667 | 51050 | 167 | 198 |
| 17 | 01 | Rossillon | 36 | 148 | 147 | 8.07 | 3619 | 50924 | 324 | 1022 |

La relation `communes` contient ici 10 attributs : le numéro du département, le nom, le canton, la population en 2010, la population en 1999, la superficie, la longitude, la latitude, la hauteur minimale et la hauteur maximale. On représente cette relation à l'aide d'un **schéma de relation** :

`communes(num_departement,nom,canton,population_2010,population_1999,surface,longitude,latitude,zmin,zmax)`.

De plus, on a la structure suivante où l'on voit le type des différents attributs :

Structure | Données | Contraintes | Index | Déclencheurs | DDL

geograph | Nom de la table : communes | WITHOUT ROWID | STRICT

| | Nom | Type de données | Clé primaire | Clé étrangère | Unique | Contrôle | Non NULL | Collecter | Généré | Valeur par défaut |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | num_departement | varchar (3) | | | | | | | | NULL |
| 2 | nom | varchar (45) | | | | | | | | NULL |
| 3 | canton | varchar (4) | | | | | | | | NULL |
| 4 | population_2010 | mediumint (11) | | | | | | | | NULL |
| 5 | population_1999 | mediumint (11) | | | | | | | | NULL |
| 6 | surface | float | | | | | | | | NULL |
| 7 | longitude | INTEGER | | | | | | | | NULL |
| 8 | latitude | INTEGER | | | | | | | | NULL |
| 9 | zmin | mediumint (4) | | | | | | | | NULL |
| 10 | zmax | mediumint (4) | | | | | | | | NULL |

Les différents enregistrements peuvent avoir des points communs mais ils doivent être différents. Il est nécessaire d'avoir un critère permettant de les identifier sans risque d'erreur, ce qui se fait par l'intermédiaire de la notion de clé primaire.

DÉFINITION 8.4

Une **clé primaire** (`PRIMARY KEY`) est un attribut, ou un ensemble d'attributs, permettant d'identifier chaque enregistrement de manière unique.

REMARQUE **8.1.**
Dans l'exemple précédent, aucun attribut n'est une clé primaire simple puisqu'aucun des attributs ne permet d'identifier de la relation `communes` ne permet d'identifier un enregistrement de manière unique.

EXEMPLE.
Prenons la relation `departements` dont le schéma de relation est :

`departements(num_departement,num_region,nom)`.

Dans ce cas les attributs `num_departement` et `nom` peuvent jouer le rôle de clefs primaires. En revanche l'attribut `num_region` ne peut pas servir de clé primaire.
Dans le schéma de relation, on convient de souligner la clé primaire. Par exemple :

`departements(num_departement,num_region,nom)`.

Pour qu'il n'y ait pas d'incohérence dans la base de données, les relations qu'elle contient doivent respecter quelques règles élémentaires.

**PROPRIÉTÉ 8.5**

Règles assurant la cohérence d'une relation :

- Un attribut ne peut prendre, pour un enregistrement donné, qu'une unique valeur.
- Il n'y a qu'un nombre fini d'enregistrements
- Les enregistrements doivent tous être différents.
- Toute relation possède une clé primaire.

Une base de données contient plusieurs relations, certaines étant liées entre elles par le fait qu'elles possèdent des attributs communs. Ceci conduit à la notion de clé étrangère :

**DÉFINITION 8.6**

Une **clé étrangère** (`FOREIGN KEY`) pour une relation est un attribut qui n'est pas une clé primaire de cette relation mais qui est la clé primaire d'une autre relation. On convient de repérer une clé étrangère en plaçant un ♯ devant l'attribut concerné.

EXEMPLE.
Imaginons que notre base de données contienne les deux schémas de relation suivants :

`departements(num_departement,♯num_region,nom)`.
`regions(num_region,nom)`

Dans ce cas l'attribut `num_region` est une clé primaire pour la relation `regions` et cet attribut est également présent dans la relation `departements`, pour lequel il n'est pas une clé primaire (des départements sont associés au même code de région). Dans cette relation `departements`, il s'agira d'une clé étrangère.

Il est possible d'adopter une représentation plus lisible permettant de visualiser les différentes relations d'une base de données, et les liens qui existent entre-elles.

**DÉFINITION 8.7**

Le **schéma relationnel** d'une base de données correspond à l'ensemble des relations qu'elle contient. On réunit tous les schémas de relation de cette base en incluant le nom de chaque relation suivie de ses attributs, la clé primaire étant soulignée et les clés étrangères précédées d'un ♯. On convient de relier par une flèche chaque clé étrangère d'une relation à la clé primaire de l'autre relation à laquelle elle correspond.

**EXERCICE 8.1.** Une agence immobilière gère la location d'appartements et de maisons pour le compte de propriétaires. Cette agence maintient une base de données relationnelle dans laquelle chaque propriétaire (nom, prénom, adresse, téléphone) remet en gestion à l'agence plusieurs appartements ou maisons (taille, adresse, prix de location) selon un contrat (durée, pourcentage) propre à chaque bien. Chaque appartement ou maison est donnée en location par un bail identifié par un numéro et pour une période déterminée, à un locataire dont on enregistre le nom, prénom, adresse, téléphone.
Réaliser le schéma relationnel d'une telle base de données.

## 8.2 Bases de données

### 8.2.1 Vocabulaire

La partie précédente présente le cadre théorique général pour construire une base de données selon le modèle relationnel. Lorsqu'on travaille avec une base de données concrète, le vocabulaire précédemment introduit s'adapte pour se rapprocher de la terminologie employée pour les tableaux, selon les correspondances suivantes :

| Modèle relationnel | Relation | Attribut | Domaine | Schéma relationnel | Enregistrement |
|---|---|---|---|---|---|
| Base de données | Table | Colonne ou Champ | Type de données | Schéma de tables | Enregistrement ou Ligne |

Pour travailler avec une base de données, il faudra soit la créer de toute pièce, ce que nous verrons un peu plus loin, soit l'importer si elle a déjà été créée. Ceci se fait informatiquement par l'intermédiaire d'une lecture de fichier. Ici c'est le logiciel (SGBD) qui se chargera de cette opération de lecture, comme de toutes les autres opérations.

### 8.2.2 Opérations

Il peut arriver que certaines données se déduisent d'autres par simple opération mathématique. Pour cela, le SGBD offre des possibilités d'utilisation des opérateurs arithmétiques usuels $+$, $-$, $*$, à conditions qu'ils portent sur des données de type numérique. Par exemple, considérons une entreprise qui utilise une base de données pour garder la trace de ses ventes, cette base possédant une table `vente` avec pour colonnes `produit_vendu`, `client`, `prix_unitaire`, `quantite_commandee`, `prix_total`. Il est alors clair que le prix total pourra se déduire du prix unitaire et de la quantité commandée par une simple opération arithmétique.
De plus, lors de la consultation d'une base de données on peut être amené à comparer des données, qu'il s'agisse de données numériques ou de texte. Pour cela les opérateurs de comparaison suivants existent :

| égalité | supérieur strict | supérieur ou égal | inférieur strict | inférieur ou égal | différent |
|---|---|---|---|---|---|
| = | > | >= | < | <= | <> |

Enfin, il est possible d'enchaîner ces opérateurs de comparaison en utilisant les opérateurs logiques :

| et | ou | non |
|---|---|---|
| AND | OR | NOT |

## 8.3 Le langage SQL

Le langage le plus répandu pour l'exploitation des bases de données est le langage SQL (Structured Query Language). C'est donc un langage de requêtes structurées qui permet de définir, de manipuler et de contrôler les données d'un SGBD. Le langage SQL sert donc à dialoguer et interagir avec une base de données, en réalisant des **requêtes** SQL. Les commandes SQL permettant d'exécuter ces requêtes seront usuellement écrites en majuscules (même si le langage SQL n'est pas sensible à la casse, l'emploi des majuscules est destiné à faciliter la lisibilité).

### 8.3.1 Création de tables

Cela se fait via la commande suivante :

```
CREATE TABLE nom_de_table;
```

On ajoute ensuite, entre parenthèses, la liste des attributs (les colonnes que contiendra la table) et leur domaine (type de données), selon le modèle suivant :

```
CREATE TABLE nom_de_table(
  attribut_1 TYPE_1,
  attribut_2 TYPE_2,
  .....,
  PRIMARY KEY(attribut_1)
  );
```

On note que dans la dernière ligne on spécifie la clé primaire, qui doit être l'un des attributs précédemment inclus. Cette commande permet uniquement de créer la relation avec ses attributs, elle ne contient alors aucune donnée.
Pour insérer des données (et donc créer des enregistrements), on utilise la commande :

```
INSERT INTO nom_de_table
```

On peut alors insérer un seul enregistrement (c'est-à-dire une ligne dans la table) :

```
INSERT INTO nom_de_table VALUES (valeur1, valeur2, ...);
```

Sans précision supplémentaire, il faudra bien être vigilant sur l'ordre des colonnes et s'assurer que `valeur1` correspond bien à l'attribut `attribut_1` et ainsi de suite. On peut sinon redonner après `nom_de_table` la liste (entre parenthèses) des attributs dans l'ordre correspondant à celui de l'enregistrement que l'on veut ajouter. Il faut noter que les valeurs numériques seront entrées telles quelles, en revanche le texte devra être entouré par des apostrophes '...' ou des guillemets "...".
On peut également insérer plusieurs enregistrements en une seule commande, ce que l'on n'hésitera pas à faire :

```
INSERT INTO nom_de_table
VALUES (valeur1_1, valeur1_2, ...),
 (valeur2_1, valeur2_2, ...),
 .....  ;
```

**EXERCICE 8.2.** Écrire les commandes permettant de créer une table destinée à contenir toutes les informations vitales concernant les étudiants de ECG2, à savoir le nom, le prénom, l'âge, adresse. On n'oubliera pas de se poser la question de la clé primaire.
Écrire la commande permettant d'insérer un enregistrement correspondant à votre cas personnel.

Une base de données est évolutive, on est amené à insérer de nouveaux enregistrements mais également à en supprimer. Ceci se fait grâce à la requête :

```
DELETE FROM nom_de_table;
```

Cette instruction est assez radicale car elle effacera l'intégralité des données présentes dans la table. Il est possible de n'effacer qu'une partie des données, correspondant à une (ou des) condition(s), grâce à l'instruction `WHERE` :

```
DELETE FROM nom_de_table WHERE condition;
```

EXEMPLE. Dans le cas de l'exercice précédent, on peut choisir de supprimer tous les étudiants qui ont moins de 20 ans par la requête :

```
DELETE FROM etudiants WHERE age<20;
```

### 8.3.2 Lecture de données

Une fois les tables créées et complétées, ou récupérées par lecture d'un fichier, il s'agit de pouvoir extraire des données de cette table correspondant à certains critères souhaités. Pour cela, la commande SQL adaptée est :

```
SELECT nom_de_champ FROM nom_de_table;
```

Cette requête va extraire de la table s'appelant `nom_de_table` toutes les données de la colonne `nom_de_champ`, et les afficher. Cela est intéressant mais reste limité, on souhaite pouvoir exécuter des requêtes bien plus complexes pour extraire des données vérifiant certaines conditions. Pour cela on complétera de la sorte :

```
SELECT nom_de_champ FROM nom_de_table WHERE condition;
```

Si on ne souhaite pas limiter la requête à une certaine colonne particulière mais afficher toutes les colonnes, on écrira * à la place du `nom_de_champ`.

**EXERCICE 8.3.** Revenons à l'exemple de la relation `communes`.

1. Que produit selon vous la requête suivante ?

```
SELECT * FROM communes WHERE (numdepartement=13)
```

2. Même question pour cette requête :

```
SELECT nom FROM departements WHERE (population_2010>10000)
```

Enfin, on peut extraire et regrouper des données provenant de plusieurs tables, pourvu qu'elles aient au moins un point en commun. C'est ce qu'on appelle la **jointure** et qui s'écrit de la façon suivante :

```
SELECT * FROM nom_de_table1 INNER JOIN nom_de_table 2
 ON (nom_de_table1.nom_de_champ1=nom_de_table2.nom_de_champ2,...);
```

Cette commande va parcourir les deux tables et pour chaque enregistrement de la table `nom_de_table1` et de la table `nom_de_table2` qui possèdent en commun la colonne `nom_de_champ1` et `nom_de_champ2` (respectivement), ces enregistrements seront mis bout-à-bout et affichés.

**EXERCICE 8.4.**
Toujours autour de l'exercice 2, on suppose qu'on dispose d'une table `admissibilites` avec les colonnes `id`, `nom_ecole`, `moyenne`, `id_etudiant` rassemblant les moyennes obtenues par les étudiants aux écrits des différentes écoles auxquelles ils sont inscrits. On exécute la requête :

```
SELECT etudiants.nom,etudiants.prenom,admissibilites.nom_ecole,
   admissibilites.moyenne FROM etudiants INNER JOIN admissibilites
 ON (etudiants.id=admissibilites.id_etudiant);
```

Décrire le résultat de cette requête.

### 8.3.3 Mise à jour de données

Les enregistrements d'une base de données peuvent être à tout moment modifiés grâce à une requête SQL du type :

```
UPDATE nom_de_table
SET nom_de_champ1=valeur1, nom_de_champ2=valeur2, ...
WHERE condition
```

Par cette commande, les enregistrements vérifiant la condition seront tous modifiés, les valeurs des colonnes citées étant mises à jour.

REMARQUE **8.2.**
Sans condition, tous les enregistrements du champ seront modifiés?

**EXERCICE 8.5.** Dans le contexte de l'exercice 2 :

1. Écrire une requête permettant de mettre à jour votre enregistrement en remplaçant votre adresse par celle du lycée.
2. Pour des raisons de confidentialité et de protection de la vie privée, on décide d'attribuer la valeur *inconnue* à toutes les adresses des étudiants.

   Écrire une requête permettant de mettre à jour la base de données.