BASES DE DONNÉES RELATIONNELLES ET SQL

I. GÉNÉRALITÉ SUR LES BASES DE DONNÉES

1. De quoi s'agit-il ?

Une **base de données** est une structure de données munie d'un langage pour accéder à ces données en lecture et en écriture. Les bases de données peuvent être divisées en deux catégories : **bases de données relationnelles** et non relationnelles (hors programme).

On agit sur une base de données en utilisant un **système de gestion de base de données**, **SGBD** ou **DBMS** en anglais, DataBase Management System. La plupart des SGBDR (R de relationnelle) utilisent le langage **SQL**, **Structured Query Language**.

Un SGBD permet le stockage des données, un traitement efficace des requêtes, la gestion des accès concurrents, par exemple lorsque deux personnes se connectent pour acheter le dernier billet d'avion, et la sécurisation des accès avec des droits différents selon les profils d'utilisateurs.

2. Les systèmes de gestion de base de données les plus connus

Principaux systèmes open-source

- MySQL : système open-source lancé en 1995 utilisé principalement par les applications web, comme Joomla et Wordpress, proposé par la plupart des hébergeurs comme serveur de base de données pour les sites web, utilisé par Facebook, Google, Twitter, YouTube. Propriété d'Oracle depuis 2008, MySQL existe également en version payante.
- MariaDB : système créé en 2009 par des développeurs de MySQL suite à l'achat de ce dernier par Oracle en 2008. MariaDB est très similaire à MySQL et utilisé par Google, Mozilla, Wikimedia …

  Les noms MySQL et Maria SQL viennent de My et Maria, les deux filles du développeur Michael "Monty" Widenius.
- PostgreSQL : SGBD open-source utilisé par exemple dans des applications de jeux en ligne, installé par défaut sur les serveurs MacOS. Le projet Postgres a été initié en 1985 par Michael Stonebraker pour remplacer la base de données Ingres, et sa première version a été publiée en 1996 sous le nom PostgreSQL.
- SQLite : SGBD léger, du domaine public, permet de manipuler localement des bases de données sans passer par un serveur (première version en 2000).

Principaux systèmes propriétaires

- Oracle : référence historique, puisque la première version du SGBD Oracle est sortie en 1979. Elle est développée par la société Oracle Corporation, crée en 1977.
- Microsoft SQL Server et Microsoft Access : les SGBD Microsoft. Access est un logiciel livré avec le pack MS Office permettant de manipuler localement les bases de données, comme SQLite, tandis que Microsoft SQL Server est un serveur SQL au même titre qu'Oracle ou MySQL.

Complément hors-programme

MongoDB et Google Bigtable sont deux exemples de bases de données non relationnelles.

Nous utiliserons principalement SQLite :

- en Python : import sqlite3,
- en ligne : https://sqliteonline.com/,
- ou en version portable : https://sqlitebrowser.org/dl/
- et éventuellement MySQL via un site web.

3. Architecture trois-tiers

Un logiciel permettant de créer et de gérer une base de données est appelé **serveur**. L'ordinateur équipé de ce logiciel, hébergeant la base de données, est également appelé **serveur**.
Un **client** est un logiciel, ou un ordinateur, permettant d'accéder à une base de données existante située sur un serveur.

Un système de base de données est divisé en trois couches (ou niveaux).

- Couche présentation ou premier niveau : partie visible, interface utilisateur graphique ou texte, transmet les requêtes de l'utilisateur à la couche métier, et les réponses de la couche métier à l'utilisateur.
- Couche métier ou second niveau : partie fonctionnelle de l'application, traitement des données, gestion des opérations, communique avec les deux autres couches.
- Couche accès aux données ou troisième niveau : gère l'accès aux données, qui peuvent être stockées localement ou externes, et les transmets à la couche métier.

Dans un système client-serveur, le client gère la couche présentation et une partie de la couche métier, le serveur gère la couche accès aux données et une partie de la couche métier.
Dans un système à trois niveaux, appelé architecture trois-tiers, chaque couche est gérée de manière indépendante, par trois ordinateurs (client, serveur d'application, serveur de données) ou par trois fonctions d'un même programme.

II. BASE DE DONNÉES RELATIONNELLE

1. Définitions

Une base de données relationnelle est constituée d'un ensemble de **tables** appelées aussi **relations** contenant le plus souvent des données du type texte ou numérique. Chaque table a un nom et est composée de plusieurs colonnes nommées elles aussi, appelées **attributs** ou **champs.** On préfèrera le terme attribut car selon le contexte, le terme « champ » peut désigner une colonne, le titre d'une colonne ou une case de la table. Deux colonnes d'une même table doivent avoir des noms différents.
Un **enregistrement** ou **tuple** est une ligne de la table.
Chaque **attribut** est déterminé par son **nom** et son **type** (entier, chaine de caractères…) ou son **domaine** (ensemble des valeurs prises : $\mathbb{N}$, un intervalle, une liste de noms…).
Le nom d'une relation avec ses attributs (noms et types ou domaines) constitue le **schéma relationnel** de la relation. On peut l'assimiler à la table vide.
L'ensemble des schémas relationnels des tables d'une base de données forme le **schéma relationnel** ou la **structure** de la base de données.
Une **clé** est constituée d'un ou plusieurs champs permettant d'identifier chaque enregistrement de manière unique. Une **clé primaire** est une clé choisie comme clé principale. Le plus souvent, on utilise un champ dédié du type entier qui s'auto-incrémente, un **identifiant**.

2. Exemple

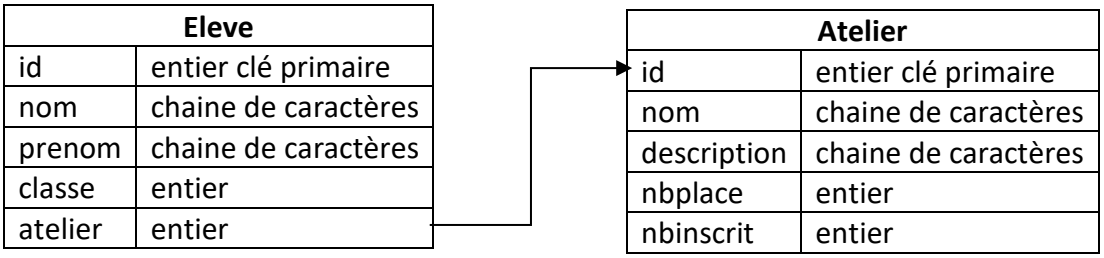
On a créé un site permettant aux élèves du lycée de s'inscrire à des ateliers.
Ci-après un extrait des tables Eleve et Atelier de la base de données du site.

id	nom	prenom	classe	atelier
1	BEAUBOURG	Alain	1	2
2	DUBERNARD	Luc	6	1
3	CASTRON	Alissa	4	3
4	GUEPPAN	Hélène	2	3

id	nom	description	nbplace	nbinscrit
1	TICE	Organisé par les …	20	2
2	English Project	Dans le cadre de …	18	1
3	Intercambio …	Echanges en espagnol …	12	3

La table Eleve est composée de 5 attributs. L'attribut nom est du type chaine de caractère, l'attribut classe du type entier. On peut affiner les types selon le SGBD utilisé. Par exemple, sur MySQL, l'attribut atelier pourra être de type tinyint (codé sur un octet), le nom de type varchar(30) si tous les noms font moins de 30 caractères.
Vu le grand nombre de données que peuvent contenir les tables, il est important de bien choisir les types de données à utiliser pour ne pas grossir inutilement la taille des tables. Rechercher "type de données SQL" sur internet pour plus d'informations.

On peut représenter le schéma relationnel de cette base de données comme suit.



3. Une vision algébrique

On peut définir chaque champ comme un ensemble, par son domaine :

- $D_1 = \mathbb{N}$ est le domaine des identifiants
- D_2 = {BEAUBOURG, DUBERNARD, …} ou D_2 = {chaines de 30 caractères ou moins} est le domaine des noms, idem pour D_3
- D_4 = {1, 2, 3, 4, 5, 6}, ou $D_4 = \mathbb{N}$ le domaine des classes, idem pour D_5.

Le produit cartésien de ces ensembles, noté en mathématiques $\prod_{i=1}^{5} D_i$ est l'ensemble des tuples construits en prenant un élément dans chaque ensemble. Il contient donc (1, BEAUBOURG, Alain, 1, 2), mais aussi (1, DUBERNARD, Alain, 1, 2), (1, DUBERNARD, Luc, 6, 1), (1, DUBERNARD, Luc, 6, 2) … toutes les combinaisons possibles.

Une relation (ou table) sur $\prod_{i=1}^{5} D_i$ est l'ensemble des combinaisons correctes : on ne conserve que les éléments qui ont du sens dans le produit cartésien.
Le schéma de la relation est l'ensemble {$A_1 : D_1$, …, $A_n : D_n$}, où A_1, … , A_n sont les noms des attributs.

Le schéma relationnel d'une base de données est l'ensemble des schémas des relations (liste des tables, de leurs attributs et de leurs domaines). Construire le schéma relationnel est la première étape dans la création d'une base de données.

La table Eleve précédente est une relation sur $\prod_{i=1}^{5} D_i$.
Le schéma de cette relation est {*id* : int, *nom* : varchar(30), *prenom* : varchar(30), *classe* : int, *atelier* : int}.
int désigne le type integer (entier).
varchar(30) désigne une chaine de 30 caractères maximum.

4. Opérations sur les relations

On interroge les bases de données avec différents opérateurs sur les relations : projection, sélection, produit cartésien, jointure, renommage, union, intersection, … Seuls les quatre premiers, ainsi que les opérateurs d'insertion, de modification et de suppression sont explicitement au programme de NSI.

4.1. Projection

Une projection d'une table est une copie de la table contenant seulement les attributs mentionnés. En d'autres termes, on sélectionne des colonnes.

Exemple : La projection de la table Eleve sur les attributs {nom, prenom} est une table contenant la liste de tous les élèves mais uniquement les colonnes nom et prenom.

4.2. Produit cartésien

Le produit cartésien de deux tables est une table contenant en colonne, toutes les colonnes des deux tables considérées, et en ligne toutes les concaténations possibles du type (tuple_table_1) + (tuple_table_2). En pratique, beaucoup de ces enregistrements n'auront pas de sens.

Exemple : Le produit cartésien des tables Eleve et Atelier va contenir par exemple
(1, BEAUBOURG, Alain, 1, 2, 1, TICE, Organisé par les …, 20, 2)
(1, BEAUBOURG, Alain, 1, 2, 2, English Project, Dans le cadre de …, 18, 1)
(1, BEAUBOURG, Alain, 1, 2, 3, Intercambio, Echanges en espagnol …, 12, 3)

Concrètement, on voudra garder seulement la deuxième ligne qui rassemble les données sur l'élève et sur l'atelier auquel il est inscrit. On effectue pour cela une sélection ou une jointure.

4.3. Sélection

Une sélection renvoie une table contenant seulement les enregistrements vérifiant une condition donnée. On sélectionne des lignes.

Exemple : La sélection de la table Eleve suivant le critère atelier=3 va renvoyer tous les enregistrements correspondant aux élèves inscrits dans l'atelier 3.

4.4. Jointure

La jointure est la composée d'un produit cartésien et d'une sélection. Elle sert à « coller » deux tables ayant des éléments en commun. Le critère de sélection sera généralement l'égalité entre deux clés.

Exemple : on effectue une jointure naturelle des tables Eleve et Atelier en prenant comme critère de sélection que l'atelier de la table Eleve est égal à l'identifiant de la table Atelier, ce que l'on note Eleve.atelier = Atelier.id.

III. STRUCTURED QUERY LANGUAGE (SQL)

1. De quoi s'agit-il ?

Le langage **SQL** permet d'effectuer des requêtes sur les bases de données.
En pratique, on peut programmer des requêtes SQL en Python, PHP … ou bien utiliser une interface graphique comme l'interface en ligne phpMyAdmin très utilisée pour travailler sur les bases de données MySQL des sites web.

2. Requêtes d'interrogations classiques SELECT … FROM … WHERE

La forme générale d'une requête SQL est :

SELECT attributs	*projection*
FROM relations	*produit*
[WHERE condition]	*sélection*

Elle s'exécute dans l'ordre FROM puis WHERE puis SELECT.

Exemples

SELECT nom, prenom FROM Eleve WHERE classe = 2

→ Renvoie une table avec les noms et prénoms des élèves dans la classe 2

Si certains élèves étaient inscrits dans plusieurs Ateliers, ils vont apparaitre plusieurs fois. On peut éviter ces doublons avec :

SELECT DISTINCT nom, prenom FROM Eleve WHERE classe = 2

SELECT Eleve.nom, prenom, Atelier.nom FROM Eleve, Atelier WHERE Eleve.atelier = Atelier.id

→ Renvoie une table avec les noms et prénoms des élèves et le nom des ateliers auxquels ils sont Inscrits

Cette dernière requête est une jointure « à l'ancienne ». La syntaxe actuelle est présentée dans le paragraphe suivant.

Sélection sans projection : le symbole * remplace tous les attributs des tables mentionnées dans FROM :
SELECT * FROM Eleve, Atelier WHERE Eleve.atelier = Atelier.id

Compléments sur le renommage

Comme les tables Eleve et Atelier ont toutes les deux des attributs nommés nom et id, ils sont appelés Eleve.nom, Atelier.nom, Eleve.id, Atelier.id.

On peut alléger la requête en renommant les tables :

SELECT E.nom, prenom, A.nom FROM Eleve AS E, Atelier AS A WHERE E.atelier = A.id

ou

SELECT E.nom, prenom, A.nom FROM Eleve E, Atelier A WHERE E.atelier = A.id

(le "AS" est facultatif)

Le renommage permet aussi de dupliquer une table :
SELECT E1.nom, E2.nom FROM Eleve E1, Eleve E2 WHERE E1.classe = E2.classe
→ Renvoie une table contenant les binômes d'élèves présents dans la même classe
La requête précédente fonctionne car on a effectué deux copies de la table Eleve nommées E1 et E2.

On va retrouver dans le résultat de cette requête le binôme (BEAUBOURG, DUBERNARD), mais aussi (DUBERNARD, BEAUBOURG) et (BEAUBOURG, BEAUBOURG).
On peut éviter les doublons et les binômes contenant deux fois le même élève avec :
SELECT E1.nom, E2.nom FROM Eleve E1, Eleve E2 WHERE E1.classe = E2.classe AND E1.nom < E2.nom

Critères de sélection
La clause WHERE est suivie d'une condition qui peut être exprimée à l'aide d'égalités, d'inégalités, des opérateurs AND, OR, NOT, BETWEEN, IN, LIKE, IS NULL, IS NOT NULL.

IS (NOT) NULL sert à tester si un attribut prend (ou pas) la valeur NULL.
On n'écrit jamais attribut = NULL ou attribut <> NULL. Différent en SQL s'écrit <>.

Exemples
SELECT * FROM Eleve WHERE classe BETWEEN 3 AND 6
→ Renvoie la liste des élèves des classes numérotées 3 à 6

SELECT * FROM Eleve WHERE NOT nom LIKE 'D%'
→ Renvoie la liste des élèves dont le nom ne commence pas par D.
LIKE 'nom' recherche le nom exact; _ peut remplacer un caractère unique et % une chaîne de caractères de longueur quelconque.

SELECT nom, prenom FROM Eleve WHERE atelier IN (SELECT id FROM Atelier WHERE nbplace = nbinscrit)
→ Renvoie la liste des élèves inscrits dans des Ateliers complets

SELECT nom, nbplace - nbinscrit FROM Atelier WHERE nbplace - nbincrit > 4
→ Renvoie le nom des ateliers et le nombre de places libres dans les ateliers pour lesquels il reste encore au moins 5 places

Comme dans la requête précédente, on peut effectuer les opérations arithmétiques (+, -, *, /) sur les attributs après le SELECT ou le WHERE.
SQL connait aussi les fonctions mathématiques classiques (EXP, SQRT…).

3. Jointures
On a déjà vu un exemple de jointure, qui renvoie un « collage » des tables Eleve et Atelier en faisant correspondre à chaque élève l'atelier qu'il a choisi :
SELECT Eleve.nom, prenom, Atelier.nom FROM Eleve, Atelier WHERE Eleve.atelier = Atelier.id

La syntaxe actuelle, à retenir est :
SELECT Eleve.nom, prenom, Atelier.nom FROM Eleve JOIN Atelier ON Eleve.atelier = Atelier.id

En pratique, les deux requêtes sont équivalentes, mais l'utilisation de JOIN permet de mettre en évidence l'opération de jointure, ce qui simplifie la compréhension du code et la maintenance. JOIN offre de plus d'autres possibilités hors programme (INNER JOIN, LEFT OUTER JOIN, …).

Remarque
Une jointure peut être suivie d'une clause WHERE pour effectuer une sélection en plus de la jointure.

4. Fonctions d'agrégation

Une fonction d'agrégation prend une collection de valeurs et retourne une valeur unique.
Les fonctions d'agrégation sont **AVG**, **COUNT**, **MAX**, **MIN**, **SUM**.
AVG et SUM s'appliquent uniquement à des données numériques.
COUNT(∗) renvoie le nombre de lignes dans la table, COUNT(*A*) renvoie le nombre de lignes dans la colonne *A* dont le contenu n'est pas NULL.
On peut utiliser DISTINCT à l'intérieur de Count : SELECT Count(DISTINCT nom) FROM table
Une fonction d'agrégation **s'utilise uniquement après SELECT** ou HAVING (hors-programme), mais on peut placer un SELECT … FROM … dans une condition WHERE puisque l'utilisation d'une fonction d'agrégation ne renverra qu'une valeur utilisable dans un test.

Exemples

```
SELECT AVG(nbinscrit) FROM Atelier
```

→ Renvoie le nombre moyen d'inscrits par Atelier.

```
SELECT nom
FROM Atelier
WHERE nbinscrit = (SELECT MAX(nbinscrit)
                   FROM Atelier)
```

→ Renvoie le nom de l'atelier ayant le plus d'inscrits

```
SELECT nom
FROM Atelier
WHERE nbplace - nbinscrit = (SELECT Max(plibres)
                             FROM  (SELECT nbplace - nbinscrit AS plibres
                                    FROM Atelier) AS Temp)
```

→ Renvoie le nom de l'atelier ayant le plus de places libres

5. Requêtes d'insertion, de modification et de suppression

5.1. Insertion

INSERT INTO *table* VALUES (…) permet de rajouter des enregistrements à une table.

Exemple

```
INSERT INTO Eleve VALUES (12, 'POTTER', 'Harry', 5, 4)
```

On peut préciser les noms des attributs. On peut alors les saisir dans le désordre ou ne pas entrer ceux prenant une valeur NULL.

```
INSERT INTO Eleve (id, nom, prenom, classe, atelier)
VALUES (12, 'POTTER', 'Harry', 5, 4)
```

5.2. Modification

UPDATE *table* SET *attribut1* = *valeur1*, *attibut2* = *valeur2*, … WHERE *condition* permet de modifier la valeur d'un ou plusieurs attributs.

Exemple

```
UPDATE Atelier SET nbplace = 15
WHERE nom = 'English Project'
```

5.3. Suppression

DELETE FROM *table* WHERE *condition* permet d'effacer des enregistrements.

Exemple

```
DELETE FROM Atelier
WHERE nbinscrit = 0
```

6. Contraintes d'intégrité

Les contraintes d'intégrité sont des règles que l'on impose aux champs de la base de données afin de garantir leur validité.

6.1. Contrainte de domaine

Toute valeur prise par un attribut doit appartenir au domaine choisi pour cet attribut. Ainsi, le nombre de place d'un Atelier doit être un entier du type choisi (int, tinyint…).

On peut aussi imposer une contrainte supplémentaire, par exemple que tous les Ateliers aient entre 10 et 20 places :

- au moment de la création de la table en rajoutant une contrainte CHECK dans la définition de l'attribut nbplace :

  ```
  nbplace tinyint CHECK (nbplace >= 10 AND nbplace <= 20)
  ```

  Sur MySQL, rajouter une virgule avant le CHECK.
- ou après la création de la table avec la requête ALTER TABLE :

  ```
  ALTER TABLE Atelier ADD CHECK (nbplace >= 10 AND nbplace <= 20)
  ```

On peut nommer les contraintes (le système donne un nom par défaut dans le cas contraire) :

- lors de la création de la table :

  ```
  CONSTRAINT chk_nbplace CHECK (nbplace >= 10 AND nbplace <= 20)
  ```
- ou après la création de la table :

  ```
  ALTER TABLE Atelier
  ADD CONSTRAINT chk_nbplace CHECK (nbplace >= 10 AND nbplace <= 20)
  ```

Le nommage est obligatoire pour les contraintes portant sur plusieurs attributs :

```
ALTER TABLE Atelier
ADD CONSTRAINT places_inscrits CHECK (nbinscrit <= nbplace)
```

ALTER TABLE Atelier DROP CONSTRAINT chk_nbplace permettra ensuite de supprimer la contrainte.
En MySQL, c'est ALTER TABLE Atelier DROP CHECK chk_nbplace .

On peut imposer qu'un champ soit renseigné : nbplace tinyint NOT NULL

6.2. Contrainte de clé primaire

On définit généralement une clé primaire dans chaque relation. Dans la table Eleve, c'est l'attribut id, mais on aurait aussi pu prendre les attributs (nom, prenom), sauf si deux élèves ont les mêmes noms et prénoms :

- PRIMARY KEY(id) ou PRIMARY KEY (nom, prenom) à la création de la table
- CONSTRAINT nom_contrainte PRIMARY KEY(id) pour une contrainte nommée
- ALTER TABLE Eleve ADD CONSTRAINT C1 PRIMARY KEY(id) pour un ajout ultérieur

Le SGBD va alors refuser l'insertion d'un nouvel élève s'il a la même clé primaire qu'un autre élève déjà enregistré.

On peut définir une seule clé primaire, mais on peut rajouter des clés secondaires avec UNIQUE, qui imposera des valeurs différentes de ces clés pour chaque enregistrement : UNIQUE(nom) dans la table atelier empêchera que deux ateliers portent le même nom.

6.3. Contrainte de clé étrangère (intégrité référentielle)

On peut définir une clé étrangère dans une table, c'est-à-dire choisir un ou plusieurs attributs d'une table qui forment une clé primaire d'une autre table :

```
ALTER TABLE Eleve
CONSTRAINT FK_Eleve_Atelier
FOREIGN KEY(atelier) REFERENCES Atelier(id)
```

FOREIGN KEY empêche :

- d'ajouter un enregistrement à la table Eleve où le numéro d'atelier n'existe pas dans la table Atelier
- de supprimer un atelier contenant des élèves inscrits

Il faut que la clé choisie ait bien été déclarée comme une clé primaire dans la table de référence.

Une variante : la suppression d'un atelier de la table Atelier va alors supprimer tous les élèves inscrits dans la table Eleve :

```
ALTER TABLE Eleve
CONSTRAINT FK_Eleve_Atelier
FOREIGN KEY(atelier) REFERENCES Atelier(id) ON DELETE CASCADE
```

7. Compléments utiles

LIMIT

LIMIT *n* à la fin d'une requête renvoie uniquement les *n* premiers résultats.
LIMIT *n* OFFSET *p* renvoie *n* résultats à partir du (*p*+1)-ième (en sautant les *p* premiers).

ORDER BY

ORDER BY à la fin d'une requête permet de trier suivant les attribut(s) dans l'ordre croissant (ASC ou par défaut) ou décroissant (DESC).
Exemple : SELECT * FROM Eleve ORDER BY classe, nom DESC

EXISTS/NOT EXISTS

EXISTS (requête) renvoie True si la requête contient des enregistrements, False sinon et s'utilise de la manière suivante :

(requête 1) WHERE EXISTS (requête 2)

La requête 1 ne renvoie que les enregistrements pour lesquels la requête 2 contient des enregistrements.

(requête 1) WHERE NOT EXISTS (requête 2)

Ici, la requête 1 ne renvoie que les enregistrements pour lesquels la requête 2 n'en contient pas.

Exemple : SELECT * FROM Atelier WHERE EXISTS (SELECT * FROM Eleve WHERE Eleve.atelier = Atelier.id)

→ Renvoie la liste des ateliers contenant des élèves inscrits

8. Quelques sites

Un bon site d'initiation et d'entrainement qui dépasse le cadre du programme	http://sqlzoo.net
Un bon site d'entrainement programme de NSI	http://colbert.bzh/sql/
SQLiteOnLine	https://sqliteonline.com/
Formatage des requêtes SQL	https://sqlformat.org/
SQL Murder Mystery	https://mystery.knightlab.com/