Introduction aux Bases de Données

1) Qu'est-ce qu'une base de données ? A quoi cela sert-il ?

Nous allons baser l'essentiel de cette introduction sur l'étude d'un exemple.

Pour la gestion du lycée, on veut regrouper des informations concernant le fonctionnement de l'établissement. On aimerait ainsi pouvoir savoir que :
- MA Emeline est en TSI1 dont les délégués sont ANGELIQUE Marie-Christina et BREDAS Willy
- Le professeur de maths de la TES1 est M.TEFIT, il en est d'ailleurs le professeur principal
- Il y a 27,3 élèves en moyenne pas classe de Seconde

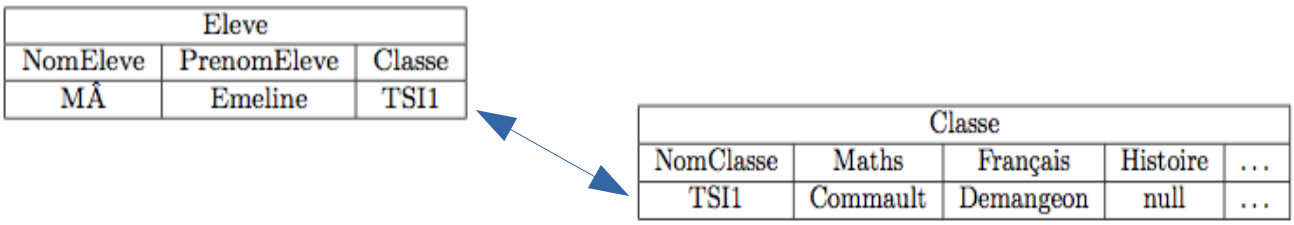
Regrouper ces informations est créer une **base de données**.

La première idée que l'on pourrait avoir est de créer un (grand) tableau :

Nom Elève	Prénom Elève	Classe	Prof Principal	Prof Maths	Prof Français	Prof Histoire	...
MÂ	Emeline	TSI1	null	Commault	Demangeon	null	
...							

Ce type de structure, dite « plate », a une défaut majeur : il y a beaucoup de doublons, c'est-à-dire de données stockées de façon répétées. Par exemple, tous les élèves en TSI1 ont les mêmes enseignants, les mêmes délégués…

On préférera un **schéma relationnel** :



Un **Système de Gestion de Base de Données** (SGBD) est un logiciel qui implémente les bases de données et permet de travailler avec elles. Nous utiliserons **MySQL**, qui est un logiciel libre (pour le moment), mais il en existe d'autres : Oracle Database, MS Access, SQLite, FileMaker Pro...

Un SGBD a plusieurs fonctions sur lesquelles nous reviendrons :
- **stocker** les données
- **sécuriser** les données
- gérer les **droits** utilisateurs

SQL (pour Structured Query Language) est le langage informatique que nous utiliserons pour travailler avec les bases de données.

2) Vocabulaire des bases de données

On se base sur l'exemple précédent. Une BDD est constituée de **tables**, par exemple la table Eleve.

Eleve		
NomEleve	PrenomEleve	Classe
MÂ	Emeline	TSI1

Cette table a trois **attributs** (ou **champs**) : NomEleve, PrenomEleve et Classe.

Attention : Classe désigne un attribut de la table Eleve mais également une autre table de la base. Pour éviter les confusions, on peut noter (syntaxe SQL) : Eleve.Classe pour désigner l'attribut.

Les données stockées dans les attributs sont d'un **type** défini lors de la création de l'attribut. Il existe de nombreux types différents :

- des types de nombres : SMALLINT, MEDIUMINT, DECIMAL, FLOAT...
- des types de chaînes de caractères : CHAR, VARCHAR, TEXT, MEDIUMTEXT...
- des types temporels : DATE, DATETIME, TIME...
- d'autres familles de types (binaires par exemple)

Ces nombreux types permettent de gagner en ressource et en efficacité de traitement. Par exemple, si l'on stocke une chaîne de caractères, on ne monopolisera pas a priori la même quantité de mémoire selon qu'on doive stocker un nom de famille, une phrase ou un texte entier.

La ligne (MÂ, Emeline, TSI1) s'appelle un **enregistrement** (ou **tuple**, ou **triplet** -car la table a trois attributs).
Pour retrouver un enregistrement dans une table, il est nécessaire qu'il soit identifié de façon certaine par une **clé primaire**.
Choisir NomEleve comme clé primaire ne serait pas prudent : on pourrait avoir deux élèves avec le même nom. De même, en prenant le couple (NomEleve,PrenomEleve), il y aurait encore un risque (certes moindre) de doublons. Souvent, on rajoute un attribut numérique Identifiant :

Eleve			
Id	NomEleve	PrenomEleve	Classe
1	MÂ	Emeline	TSI1
2	BREDAS	Willy	TSI1

On pourrait s'inquiéter que les élèves ne soient pas classés par ordre alphabétique, mais cet ordre n'a d'importance qu'au moment où l'on interroge la base. Ainsi, si l'on veut la liste des élèves par classe, il faut d'abord classer selon le attribut Eleve.Classe. Si l'on veut avoir la liste des votants pour l'élection des représentants des élèves au Conseil d'Administration au lycée, la liste alphabétique des élèves du lycée sera plus judicieuse.

En règle générale, pour schématiser une table, on procède de la façon suivante :

Eleve
Id
NomEleve
PrenomEleve
Classe

Dans la première ligne : le **nom** de la table.

Dans les lignes suivantes : les **attributs** parmi lesquels est souligné la **clé primaire.**

Dans le schéma relationnel présenté sur la première page, la flèche correspond à une **jointure** entre les tables Classe et Elève.

3) Introduction à la syntaxe SQL

Pour créer la table Eleve en langage SQL on écrira :

```
CREATE TABLE Eleve(
Id INT NOT NULL AUTOINCREMENT,
NomEleve VARCHAR,
PrenomEleve VARCHAR,
Classe VARCHAR,
PRIMARY KEY(Id) ) ;
```

Pour chaque attribut on spécifie le type des données, ainsi que d'autres qualités optionnelles : NOT NULL signifiera que l'attribut est nécessairement rempli, AUTO INCREMENT signifie que chaque nouvel enregistrement augmente de 1 la valeur de cet attribut. Enfin, on précise la clé primaire en indiquant quel(s) attribut(s) est(sont) concerné(s).

Remarquez que les mots-clés du langage sont en majuscule, ce n'est pas obligatoire mais c'est une convention qui facilite la lecture du code.

Pour ajouter un enregistrement à la base, on peut saisir :

```
INSERT INTO Eleve
(NomEleve, PrenomEleve, Classe) VALUES ('ELANA','Delio', 'TSI1') ;
```

Remarquez que la valeur de l'attribut Id n'a pas été précisée : c'est possible car on a indiqué qu'il s'auto-incrémente (l'Id de cet enregistrement sera augmenté de 1 par rapport à l'Id de l'enregistrement précédent).

Ce fonctionnement est intéressant : si l'on rajoute un nouvel élève dans la base de donnée, il n'est pas nécessaire de le « positionner » par ordre alphabétique ou dans sa classe. Il correspond juste à un nouvel enregistrement.

4) Introduction à l'algèbre relationnelle

L'algèbre relationnelle a été inventée en 1970 par Edgar Franck Codd qui travaillait pour IBM. Elle donne un cadre théorique qui permet d'implémenter les bases de données.

L'algèbre relationnelle est proche de la théorie des ensembles, certaines notions ensemblistes d'y retrouveront.

a) Opérations ensemblistes

- Union
- Intersection
- Différence
- Produit Cartésien

Exemple :

Nous disposons de trois tables :

PSG		
Numéro	Nom	Capitaine
30	Sirigu	Non
19	Aurier	Non
2	Silva	Oui
32	Luiz	Non
17	Maxwell	Non
	⋮	

Bresil		
Numéro	Nom	Capitaine
1	Jefferson	Non
4	Alves	Non
2	Silva	Oui
3	Luiz	Non
4	Marcelo	Non
	⋮	

Liste Courses	
Objet	Quantité
Jus	3
Pain	1
Lessive	1

Donner les tables $PSG \cup Bresil$, $PSG \cap Bresil$, $PSG - Bresil$, $PSG \times Liste\,Courses$

b) Opérateurs relationnels

- **Sélection** d'une partie d'une table, selon une condition (qui peut être composée à l'aide d'opérateurs logiques).

Par exemple :

$\sigma_{\text{Quantité}=1}(\text{Liste Courses})$ renverra :

- **Projection** de certains attributs d'une table.

Par exemple :

$\pi_{\text{Objet}}(\text{Liste Courses})$ renverra :

- **Jointure** entre deux tables selon un attribut : on crée (virtuellement) la table obtenue en recollant les deux tables selon l'attribut indiqué.

Que créera l'appel suivant ? $\text{Eleve} \underset{\text{Eleve.Classe}=\text{Classe.NomClasse}}{\bowtie} \text{Classe}$

Remarques :

- attention : la commande pour la projection en SQL est SELECT
- la table créée par une jointure est virtuelle en ce sens qu'elle n'est pas sauvegardée

Exercices

A l'aide de l'algèbre relationnelle comment obtenir :

- Les élèves de TSI1
- Les professeurs principaux des classes dont le professeur d'espagnol est M.Mercier
- Le professeur principal de l'élève Jean Leopoldie

Remarque : le langage SQL a pour objectif de mettre en œuvre les formules de l'algèbre relationnelle.

5) Clés étrangères

On reprend l'exemple d'une base de données pour l'établissement.

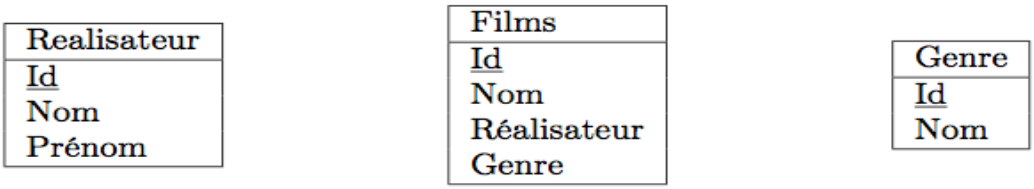
On souhaite qu'il y ait une correspondance entre les attributs *Eleve.Classe* et *Classe.NomClasse*. C'est d'ailleurs sur cette correspondance que l'on fera une jointure.

Du point de vue de la cohérence de la base de données, il ne faut donc pas qu'il y ait une valeur de *Eleve.Classe* qui n'apparaisse pas dans l'attribut *Classe.NomClasse*. Pour se prémunir de ce type d'erreur (erreur de saisie, corruption des données…) on va avoir contrainte ou **clé étrangère** sur les valeurs de l'attribut *Eleve.Classe* qui devront apparaître dans les valeurs de *l'attribut Classe.NomClasse.*

On dit qu'on a alors une « *liaison 1:n* », c'est-à-dire qu'un enregistrement de la table *Eleve* correspondra à un enregistrement de la table *Classe* **mais** qu'un enregistrement de la table *Classe* pourra correspondre à plusieurs enregistrements de la table *Eleve*.

On peut aussi envisager des « *liaisons 1:1* » ainsi que des « *liaisons n:n* » qui donnent lieu à des *tables de correspondances.*

Exemple
On considère la base CollectionFilms constituée des trois tables ci-dessous :



a) Faire apparaître les liaisons entre ces tables.

b) Quel risque a-t-on en faisant une jointure entre Films.Réalisateur et Réalisateur.Nom ?

c) Que proposez-vous pour l'éviter ?

d) Titanic est du genre « Drame » ou bien « Romantique » ou encore « Historique » ? Quel problème cela pose-t-il et comment le résoudre ?

e) Les liaisons entre les tables sont-elles 1:1, 1:n ou n:n ?

6) Langage SQL

a) Sélection et projection

La commande `SELECT` permet de faire une projection :

```
SELECT champ_1, … champ_n FROM table
```

Tout appel SQL correspond à une projection, quitte à projeter tous les champs. Pour obtenir la table entière :

```
SELECT * FROM table
```

Si l'on souhaite sélectionner des lignes, on utilise `WHERE` :

```
SELECT champ_1, … champ_n FROM table WHERE condition
```

Par exemple,
`SELECT NomClasse FROM Classe` renverra la liste des classes du lycée ;
`SELECT NomClasse FROM Classe WHERE Maths='Dupond'` renverra la liste des classes dont le professeur de mathématiques est Mme Dupond.

Notez qu'une condition peut utiliser les opérateurs logiques classiques : `AND`, `OR` et `NOT`.

On peut également, lorsque plusieurs valeurs sont identiques, demander à n'en garder qu'une. Ainsi, pour avoir la liste des professeurs de mathématiques du lycée :

```
SELECT DISTINCT maths FROM Classe
```

b) Jointures

On utilise la syntaxe suivante :

```
table_1 JOIN table_2 ON cle_1=cle_2
```

Si l'on souhaite obtenir la liste des films dont le réalisateur est Luc Besson on utilisera :

```
SELECT Film.Nom
FROM Film JOIN Realisateur
ON Film.realisateur=Realisateur.Id
WHERE Realisateur.Nom='Besson' AND Realisateur.Prenom='Luc'
```

Si l'on souhaite, en plus que cette liste soit par ordre alphabétique, on précisera (à la fin) :

```
ORDER BY Film.Nom ASC
```

- `ASC` pour ascendant peut être remplacé par `DESC`.
- On peut trier sur plusieurs colonnes : en cas d'égalité sur le premier champ on passera au second et ainsi de suite.

Exercice :
On considère la base constituée des deux tables suivantes :

communes
Id : int
dep : int
nom : varchar(50)
pop : int

departements
Id : int
nom : varchar(50)

Donner les commandes SQL qui permettent d'obtenir la liste des noms/population des communes de Martinique puis la liste des communes de Guadeloupe qui ont plus de 10 000 habitants.

7) Fonctions d'agrégation

Reprenons l'exemple précédent : comment obtenir le nombre de communes de chaque département ?
Il faut commencer par regrouper les données de la table commune lorsque le champ `dep` est le même, puis compter le nombre de lignes de chaque groupe ainsi obtenu.

```
SELECT dep, COUNT()
FROM communes
GROUP BY dep
```

Il existe d'autres fonctions d'agrégation que `COUNT()` :

- `SUM(champ)` va sommer un champ sur toutes les lignes du groupe ;
- `MAX(champ)` et `MIN(champ)` ;
- `AVG(champ)` donne la moyenne d'un champ sur un groupe

Par exemple, pour obtenir la liste nom/population totale de chaque département :

```
SELECT departements.nom, SUM(pop)
FROM communes JOIN departements
ON dep=departements.id
GROUP BY dep
```

Si l'on souhaite la même liste, par ordre décroissant de population :

```
SELECT departements.nom, SUM(pop) AS popDep
FROM communes JOIN departements
ON dep=departements.id
GROUP BY dep
ORDER BY popDep DESC
```

Notez qu'on a utilisé la commande `AS` pour renommer une quantité qui allait resservir. Cela s'appelle utiliser un *alias* (et c'est facultatif).

Exercices :

1) Donner la commande SQL qui fournit la liste département/population moyenne par commune, par ordre alphabétique des départements.

2) Donner la commande SQL qui fournit la liste département/commune la plus peuplée du département par ordre alphabétique des départements.

3) Donner la commande SQL qui donne la liste des départements/nombre de communes du département ordonnée par nombre de commune.