chapitre I

Bases de données

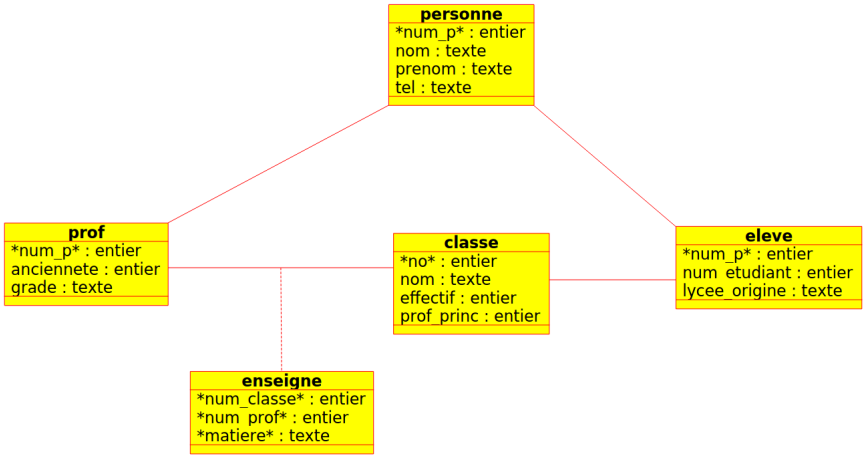
I Introduction

1 Nécessité de structures de données

Le lycée souhaite créer un fichier de renseignement sur les élèves. Sur chaque fiche, on inscrit des renseignements : nom, prénom, adresse, lycée d'origine, nom des professeurs, etc.

Problèmes :

- *homonymie* : si plusieurs élèves ont même nom, prénom ?
  Solution : ajouter un numéro d'élève.
- Comment trouver tous les élèves ayant un même enseignant ? d'une même classe ? Si on doit mettre à jour la classe ou le nom d'un professeur ? Faut-il tout refaire ?
  Solution : séparer les données.



[Il manque un num_classe dans la table eleve]

2 Principe des bases de données

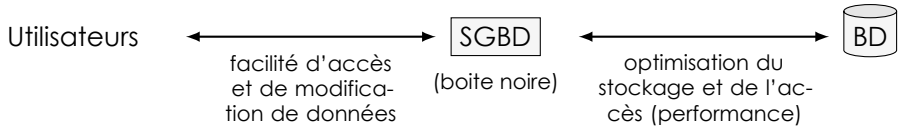
Une base de données est un ensemble d'informations qui doit être

- **cohérent** : élimination de redondances.
- **partagé** : utilisable par plusieurs utilisateurs.

Le cycle de vie d'une base de données se sépare en trois phases :

- **conception** : le plus difficile (Hors programme)
- **implémentation** : via un SGBD (système de gestion de base de données) : définition des tables et insertion des données. (Hors programme)
- **Utilisation** : extraction de données, mise à jour (insertion, modification, destruction).

Les deux derniers points se font à l'aide d'un langage appelé SQL (Structured Query Language).



3 Modèle entité-association

Les **entités** sont les éléments qui constituent la base de donnée (personne, prof, eleve, classe...), elles sont reliées par les **associations**.

On peut préciser sur le diagramme une cardinalité :

- `1` pour un seul
- `0, 1` pour au plus un
- `*` pour un nombre quelconque.
  On peut préciser `1, *` pour au moins un.

Exemple

Mais l'association `*-*` d'enseignement pose problème : on veut des attributs **mono-valués** : pas possible d'avoir plusieurs profs pour une classe ou plusieurs classes pour un prof.

Solution : créer une entité pour l'association `enseigner`.

Bilan : on peut transformer une association `*-*` problèmatique en deux associations `1-*`.

II Modèle relationnel

1 Définitions

Les objets et leurs associations sont représentés par la notion de **relations** (on dit aussi **tables**) qui sont des tableaux à deux dimensions.

Exemple

clé primaire ↓

personne :	**num_p**	nom	prenom	tel
	(entier)	(texte)	(texte)	(texte)
	6275	Dupont	Pierre	0645456789
	1234	Durand	Sylvain	0612253694

← Attributs ← Domaines ← Tuple

clé étrangère ↓

classe :	**no**	nom	effectif	prof_princ
	(entier)	(texte)	(entier)	(entier)
	12	MPSI	40	6275
	22	PC	30	5781
	41	MP	20	2517
	5	PSI*	22	1234

Définition

Étant donnée une relation (table), on appelle :
- **Attributs** : les colonnes qui la composent.
- **Domaine d'un attribut** : ensemble de valeurs que peut prendre un attribut (entier, réel, chaîne de caractères, etc.)
- **Tuple** ou **enregistrement** : Ligne d'une relation.
- ♡ **Clé primaire** : Ensemble minimal d'attributs qui permet d'identifier un tuple unique de la relation. On parle aussi d'identifiant.
- **Schéma** : Nom de la relation suivi de la liste des attributs et de leur domaine. Clé primaire à souligner (en gras dans ce poly).
- ♡ **Clé étrangère** : Ensemble d'attributs égal à la clé primaire d'une autre relation : un tuple de la relation ne peut exister que si sa clé étrangère est égale à la clé primaire d'un tuple dans l'autre relation.

Remarque

Une relation est donc mathématiquement un ensemble de n-uplets appartenant à $D_1 \times \cdots \times D_n$ où $D_1, \ldots, D_n$ sont les domaines de ses attributs, c'est-à-dire une partie de $D_1 \times \cdots \times D_n$.

Exemple : Schéma de la base de donnée Lycée (souligné pour les clés primaires, flèches pour les clés étrangères)

- personne
  - ⋆ num_p : entier
  - ⋆ nom : texte
  - ⋆ prenom : texte
  - ⋆ tel : texte
- eleve
  - ⋆ num_p : entier
  - ⋆ num_etudiant : entier
  - ⋆ lycee_origine : texte
  - ⋆ num_classe : entier
- classe
  - ⋆ no : entier
  - ⋆ nom : texte
  - ⋆ effectif : entier
  - ⋆ prof_princ : entier
- prof
  - ⋆ num_p : entier
  - ⋆ anciennete : entier
  - ⋆ grade : texte
- enseigne
  - ⋆ num_classe : entier
  - ⋆ num_prof : entier
  - ⋆ matiere : texte

Définition

La **population** d'une relation est l'ensemble des tuples de la relation.

Exemple : Populations

- classe :

no	nom	effectif	prof_princ
12	MPSI	40	6275
22	PC	30	5781
41	MP	20	2517
5	PSI*	22	1234

- prof :

num_p	anciennete	grade
6275	20	Agr
5781	5	Agr HC
1234	10	Agr CS

- enseigne :

num_classe	num_prof	matiere
12	6275	Maths
12	5781	Info
22	5781	Chimie
5	1234	Maths

- personne :

num_p	nom	prenom	tel
6275	Dupont	Pierre	0645456789
1234	Durand	Sylvain	0612253694
5781	Martin	Pierre	0635147630
2533	Galois	Evariste	0699999999
2517	Martin	Pierre	NULL
1254	Beauval	Elise	0678951354
5544	Martin	Filou	0612345678

- eleve :

num_p	num_etudiant	lycee_origine	num_classe
5544	12	Henry-le-petit	12
2533	45	Louis VI	5
2517	58	Leconte de Lisle	22
1254	22	Leconte de Lisle	12

Tous les attributs sont **mono-valués** : ils ne peuvent prendre qu'une seule valeur (exemple des professeurs d'une classe : on a créé une table `enseigne` plutôt que de les faire apparaître dans `classe`, exemple de plusieurs prénoms...)

Lorsque l'on ne peut pas préciser un attribut (exemple : nom de jeune fille), on lui attribue la valeur NULL (Hors programme).

⚠ un identifiant (clé primaire) ne peut pas être NULL !

Cela peut poser des problèmes.

Exemple

Soit une table Employé(**n°**, nom, salaire, commission) et deux tuples (7, DURAND, 1500, 200) et (6, DESCHAMPS, 1300, NULL).

A la fin du mois, on obtient pour le premier $1500+200=1700$ et pour le second 1300+NULL=NULL...

Difficulté : Construire le schéma d'une relation de manière optimale.

Bonne nouvelle : c'est hors programme.

2 Interrogation de la base de donnée

a SQL

Le SQL est le langage permettant d'interroger effectivement la base de donnée.

Les requêtes se présentent sous l'une des formes :

```
SELECT ...
FROM ...
WHERE ...
;
```

```
SELECT ...
FROM ... JOIN ... ON ...
WHERE ...
;
```

Par convention on notera en majuscules les instructions (mais le SQL n'est pas sensible à la casse.) Les instructions SQL se terminent toujours par un point-virgule.

L'instruction de base

```
SELECT * FROM eleve;
```

permet de renvoyer tous les tuples de la table `eleve`.

b Projection

On peut choisir de ne selectionner que certaines colonnes :

```
SELECT attribut_1, ..., attribut_p FROM table;
```

⚠ Pas d'ensemble en SQL (trop coûteux) : il peut y avoir des répétitions. Pour les éviter :

```
SELECT DISTINCT attribut_1, ..., attribut_p FROM table;
```

Exemple : Nom et prénom de toutes les personnes.

On peut aussi renommer les attributs et les tables. Par exemple,

```
SELECT p.nom AS "Nom de la personne", p.prenom AS "Prénom"
FROM personne p;
```

c Produit cartésien

Définition : Produit cartésien

Le **produit cartésien** de deux tables est la table obtenue en concaténant tout tuple de la première avec tout tuple de la deuxième.

Remarque

Si la première table a M tuples et la deuxième table a N tuples, alors le produit cartésien en a $M \times N$.

Exemples

E1 –

T_1 :	A	B
	0	1
	3	4

T_2 :	C	D
	5	6
	7	8

$T_1 \times T_2$:	A	B	C	D
	0	1	5	6
	3	4	5	6
	0	1	7	8
	3	4	7	8

En SQL :

```
SELECT * FROM table_1, table_2;
```

d Sélection

La clause `WHERE` permet d'ajouter une condition.

Exemple : Quelles sont les personnes nommées Martin ?

Liste non exhaustive de comparateurs :

=	!= ou <>	<	>
<=	>=	IS NULL	IS NOT NULL
AND	OR	NOT	BETWEEN ... AND ...

Citons aussi `LIKE '...'` (_ remplace un caractère, % remplace un nombre quelconque de caractères.)

Exemple

- `LIKE 'N%'` :
- `LIKE '%a%'` :
- `LIKE 'pa%on'` :
- numéros des personnes qui s'appellent Martin avec un prénom est différent de Pierre ou dont le nom commence par D :

e Jointures

Définition : Jointure

Une jointure entre deux tables est la table obtenue en concaténant les tuples des deux tables vérifiant l'égalité d'un ou plusieurs attributs.

Remarque

Seules ces jointures (appelée **équijointures**) sont au programme, mais il en existe bien d'autres. On peut par exemple mettre autre chose qu'une égalité dans la condition.

```
SELECT *
FROM table_1 t1 JOIN table_2 t2
        ON t1.X = t2.A;
```

```
SELECT *
FROM table_1 t1, table_2 t2
WHERE t1.X = t2.A;
```

Remarques

R 1 – Lorsque l'on fait une jointure avec :

```
SELECT ... FROM table_1 t1, table_2 t2 WHERE t1.X = t2.A;
```

on fait a priori une sélection sur le produit cartésien qui coûte cher à calculer.
On se doute que l'optimiseur du SGBD va faire en sorte de ne pas calculer complètement le produit cartésien, mais la formulation :

```
SELECT ... FROM table_1 t1 JOIN table_2 t2 ON t1.X = t2.A;
```

est souvent préférée pour lever toute ambiguïté et faire apparaître clairement les jointures.

Exemples

E 1 – Jointure de personne et prof sur num_p

personne.num_p	nom	prenom	tel	prof.num_p	anciennete	grade
6275	Dupont	Pierre	0645456789	6275	20	Agr
1234	Durand	Sylvain	0612253694	1234	10	Agr CS
5781	Martin	Pierre	0635147630	5781	5	Agr HC

E 2 – Donner le nom et le numéro de téléphone des étudiants dont l'effectif de la classe est inférieur ou égal à 35.

E 3 – Nom et prénom des élèves ayant le même prénom qu'un de leurs professeurs ?

E 4 – Nom et matière des professeurs enseignant dans une classe dont ils ne sont pas professeurs principaux ?

f Opérations ensemblistes

Définition : Opérations ensemblistes

Pour deux tables **de même schéma** (c'est-à-dire mêmes attributs), on définit les opérations ensemblistes de réunion, intersecton et différence, donnant chacune une table ayant encore le même schéma.

Exemple

T_1 :	nom	prenom
	Martin	Jean
	François	Marie

et

T_2 :	nom	prenom
	Martin	Jean
	François	François
	Marie	Luc

Alors

$T_1 \cup T_2$:	nom	prenom
	Martin	Jean
	François	Marie
	François	François
	Marie	Luc

$T_1 \cap T_2$:	nom	prenom
	Martin	Jean

$T_1 - T_2$:	nom	prenom
	François	Marie

En SQL :

```
SELECT ... { INTERSECT | EXCEPT | UNION | UNION ALL } SELECT ... ;
```

⚠ Pas de parenthèses autour des `SELECT` ! Dans certains SGBD (MySQL, Oracle, ...), on rencontre `MINUS` à la place d'`EXCEPT`.

`UNION ALL` permet de garder les répétitions lors du calcul de la réunion.

Exemple : Donner le nom de tous les élèves en spé

Exemple : Donner le nom de tous les élèves n'étant pas en PSI*

Exemple : Donner le nom des profs enseignant dans toutes les spé

g Fonctions d'agrégat

Il est possible d'appliquer des fonctions aux attributs d'une table au sens où on applique la fonction à cet attribut dans chaque tuple.

Exemple : conversion monétaire dans une table article(num, nom, prix_unitaire, nb)

```sql
SELECT nom, prix_unitaire * 1.11 AS prix_unitaire_usd FROM article;
```

Exemple : Prix total des articles :

```
SELECT nom, prix_unitaire * nb AS prix_total FROM article;
```

Il est également possible d'appliquer des fonctions utilisant toutes valeurs des tuples d'une table, appelées **fonctions d'agrégat** : somme, minimum, maximum, moyenne, décompte (et bien d'autres hors programme).

En SQL : `SUM, MIN, MAX, AVG, COUNT`.

⚠ L'argument de la fonction d'agrégation d'agrégat n'est pas le `SELECT` mais l'attribut. La fonction s'applique donc **dans la clause** `SELECT`.

```
SELECT FONCTION(attribut) FROM table;
```

Pour la fonction `COUNT`, il est courrant de l'appliquer sur l'argument $*$ pour compter le nombre total de tuples.

Exemple : Moyenne des anciennetés des professeurs

Exemple : Nombre d'élèves dont le nom est Payet

Remarque

Pour supprimer les doublons : `COUNT (DISTINCT ...)`.

3 Ordre, limite, offset

La clause `ORDER BY A`, utilisée en fin de requête, permet de trier les résultats par A croissants ; `ORDER BY A DESC` par A décroissants.

Exemple : Personnes triées par nom puis prénom, par ordre croissant

Exemple : Enseignants triées par ancienneté décroissante.

La clause `LIMIT n` permet de limiter aux n premiers tuples.
La clause `LIMIT n OFFSET m` permet de renvoyer les n premiers tuples suivant les m premiers tuples (donc du numéro $m+1$ au numéro $m+n$.)

Exemple : Deuxième plus ancien professeur, en supposant que chaque ancienneté n'est obtenue que par un professeur

4 Requêtes avec blocs imbriqués

Remarque

Sans renommage et en cas d'homonymie, les attributs sont ceux de la table la plus interne.

a Si la requête interne ne renvoie qu'un résultat

```
SELECT ... FROM ... WHERE expr { =, !=, <, >, <=, >= } (SELECT ...) ;
```

(SELECT ...) ne doit renvoyer qu'un résultat.

Exemple : Professeur(s) le(s) plus jeune(s)

b Si la requête interne renvoie plusieurs résultats

On utilise alors :

- `IN ...` ($\in$)
- `NOT IN ...` ($\notin$)
- `expr { =, !=, <, >, <=, >= } ANY ...` ($\exists$)
- `expr { =, !=, <, >, <=, >= } ALL ...` ($\forall$).

Remarques

R1 – `ALL` et `ANY` ne fonctionnement pas en `SQLite`.

R2 – `IN` $\iff$ `= ANY`.

R3 – `NOT IN` $\iff$ `!= ALL`.

R4 – `!=ANY` $\iff$ `NOT (... = ALL)` : ne s'utilise jamais !

Exemples

E1 – Numéros des professeurs de MPSI

E2 – numéro des prof qui enseignent la même matière que le professeur n° 1 mais dans une classe ayant plus d'élèves.

E3 – effectif des classes n° 1, 2, 3.

E4 – Professeurs les plus jeunes

ou encore

ou encore avec une jointure (à préférer)

Remarque

Un `EXISTS (...)` équivaut à un `0 != (SELECT COUNT(*) ...)`.

C Existence

```
SELECT ... FROM ... WHERE EXISTS (SELECT ...);
SELECT ... FROM ... WHERE NOT EXISTS (SELECT ...);
```

Exemple : Nom des élèves n'étant pas seuls issus d'un lycée

ou bien

Exemple : Nom des professeurs non professeurs principaux

ou bien

ou encore

5 Regroupement avant agrégation : GROUP BY et HAVING

a GROUP BY

Idée : regrouper les tuples ayant certains attributs en commun en vue d'utiliser une fonction d'agrégat.

```
SELECT ... FROM ... WHERE ... GROUP BY ...;
```

Résultat : Une seule ligne est renvoyée par groupe.

Cohérence : Il faut donc que le contenu du `SELECT` soit cohérent : soit des attributs du `GROUP BY` ou qui sont identiques pour chaque élément du groupe, soit des fonctions d'agrégat (qui porteront sur chaque groupe).

Exemples

E1 – Déterminer le nombre moyen d'élèves de chaque professeur.

Mais attention, les éventuels professeurs n'ayant pas d'élèves (???) n'apparaîtront pas.

Pour les voir, on peut par exemple faire :

E2 – Combien d'élèves au total pour chaque professeur ?

(À nouveau, sans les professeurs qui n'ont aucun élève...)

b HAVING

Idée : On peut imposer une condition sur les groupes définis par `GROUP BY` :

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ...;
```

Cohérence : Comme pour le `SELECT`, dans la condition du `HAVING` doivent apparaître soit des attributs du `GROUP BY` ou qui sont identiques pour chaque élément du groupe, soit des fonctions d'agrégat (qui porteront sur chaque groupe).

Exemple : Nom et nombre moyen d'élèves de chaque professeur qui enseigne dans au moins deux classes

C Agrégats de double niveau

Un `GROUP BY` renvoyant plusieurs résultats, on peut enchaîner avec une autre fonction d'agrégat.

Exemple : Moyenne du nombre total d'élèves par professeur (ayant des élèves)

Mais on a compté plusieurs fois une même classe si le professeur enseigne plusieurs matières dans la classe.
Pour éviter cela, on peut écrire :

6 Cas de la division cartésienne

La division cartésienne est une opération ensembliste qui est en quelque sorte l'opération inverse du produit cartésien.

Ainsi, dire que $x \in T_1 \div T_2$, c'est dire que $\forall\, y \in T_2,\ \ (x, y) \in T_1$. Cela se traduit en général par une requête formulée avec un « tous ».

Voici plusieurs traductions possibles

- Avec un `NOT EXISTS` et un `NOT IN` :

$$x \in R_1 \div R_2 \iff \forall\, y \in R_2,\ \ (x, y) \in R_1$$
$$\iff$$
$$\iff$$

- Avec deux `NOT EXISTS` imbriqués :

$$x \in R_1 \div R_2 \iff \forall\, y \in R_2,\ \ (x, y) \in R_1$$
$$\iff$$

- Avec `COUNT` : $x \in R_1 \div R_2$ si et seulement si le nombre de

- Avec `GROUP BY ... HAVING COUNT(...) = ...`

Exemple : Numéros des professeurs enseignant à tous les étudiants

-

-

-

III Exercices

Écrire en SQL les requêtes suivantes.

Exemples

E1 – Donner le nom et le numéro de téléphone de toutes les personnes enregistrées et qui ont bien un numéro de téléphone.

E2 – Donner le nom et le numéro de téléphone de tous les élèves.

E3 – Donner le nom et le prénom de tous les élèves dont le nom de la classe contient 'SI'.

E4 – Donner d'au moins deux autres façons différentes le nom et le prénom de tous les élèves de sup.

E5 – Donner les classes dont l'effectif est l'ancienneté d'un de ses professeurs, puis le double, puis le nom du professeur.

E6 – Déterminer les couples d'élèves venant d'un même lycée.

E7 – Donner le nom et prénom des élèves et des professeurs de MP2I.

E8 – Donner le nom et le prénom des élèves ayant le même prénom qu'un de leurs professeurs.

E9 – Donner le nom des élèves ayant un professeur avec moins de 15 ans d'ancienneté.

E10 – Donner le nom et la matière des professeurs enseignant dans une classe dont ils ne sont pas professeur principal.