1 Bases de données

Extrait du programme officiel :

On se limite volontairement à une description applicative des bases de données en langage SQL. Il s'agit de permettre d'interroger une base présentant des données à travers plusieurs relations. On ne présente pas l'algèbre relationnelle ni le calcul relationnel.

Notions	**Commentaires**
Vocabulaire des bases de données : tables ou relations, attributs ou colonnes, domaine, schéma de tables, enregistrements ou lignes, types de données.	On présente ces concepts à travers de nombreux exemples. On s'en tient à une notion sommaire de domaine : entier, flottant, chaîne ; aucune considération quant aux types des moteurs SQL n'est au programme. Aucune notion relative à la représentation des dates n'est au programme ; en tant que de besoin on s'appuie sur des types numériques ou chaîne pour lesquels la relation d'ordre coïncide avec l'écoulement du temps. Toute notion relative aux collations est hors programme ; on se place dans l'hypothèse que la relation d'ordre correspond à l'ordre lexicographique usuel. `NULL` est hors programme.
Clé primaire.	Une clé primaire n'est pas forcément associée à un unique attribut même si c'est le cas le plus fréquent. La notion d'index est hors programme.
Entités et associations, clé étrangère.	On s'intéresse au modèle entité–association au travers de cas concrets d'associations $1 \text{ — } 1$, $1 \text{ — } *$, $* \text{ — } *$. Séparation d'une association $* \text{ — } *$ en deux associations $1 \text{ — } 1$ et $1 \text{ — } *$.
Requêtes `SELECT` avec simple clause `WHERE` (sélection), projection, renommage `AS`. Utilisation des mots-clés `DISTINCT`, `LIMIT`, `OFFSET`, `ORDER BY`.	Les opérateurs au programme sont `+`, `-`, `*`, `/` (on passe outre les subtilités liées à la division entière ou flottante), `=`, `<>`, `<`, `<=`, `>`, `>=`, `AND`, `OR`, `NOT`.
Opérateurs ensemblistes `UNION`, `INTERSECT` et `EXCEPT`, produit carté sien.	
Jointures internes T_1 `JOIN` T_2 ...`JOIN` T_n `ON` ϕ. Autojointure.	On présente les jointures en lien avec la notion de relations entre tables. On se limite aux équijointures : ϕ est une conjonction d'égalités.
Agrégation avec les fonctions `MIN`, `MAX`, `SUM`, `AVG` et `COUNT`, y compris avec `GROUP BY`.	Pour la mise en œuvre des agrégats, on s'en tient à la norme SQL99. On présente quelques exemples de requêtes imbriquées.
Filtrage des agrégats avec `HAVING`.	On marque la différence entre `WHERE` et `HAVING` sur des exemples.
Mise en œuvre	
La création de tables et la suppression de tables au travers du langage SQL sont hors programme. La mise en œuvre effective se fait au travers d'un logiciel permettant d'interroger une base de données à l'aide de requêtes SQL. Récupérer le résultat d'une requête à partir d'un programme n'est pas un objectif. Même si aucun formalisme graphique précis n'est au programme, on peut décrire les entités et les as- sociations qui les lient au travers de diagrammes sagittaux informels. Sont hors programme : la notion de modèle logique vs physique, les bases de données non relation- nelles, les méthodes de modélisation de base, les fragments DDL, TCL et ACL du langage SQL, les tran- sactions, l'optimisation de requêtes par l'algèbre relationnelle.	

Table des matières

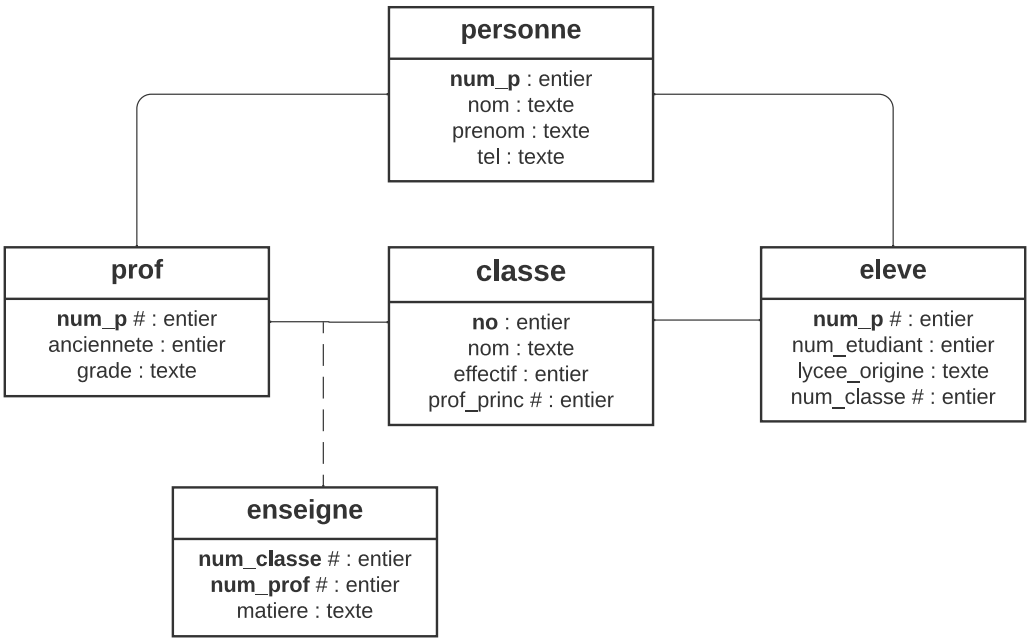
0 Introduction

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.

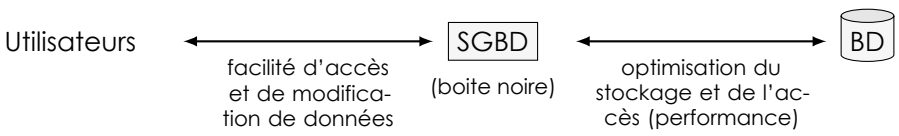


2 Principe des bases de données

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).



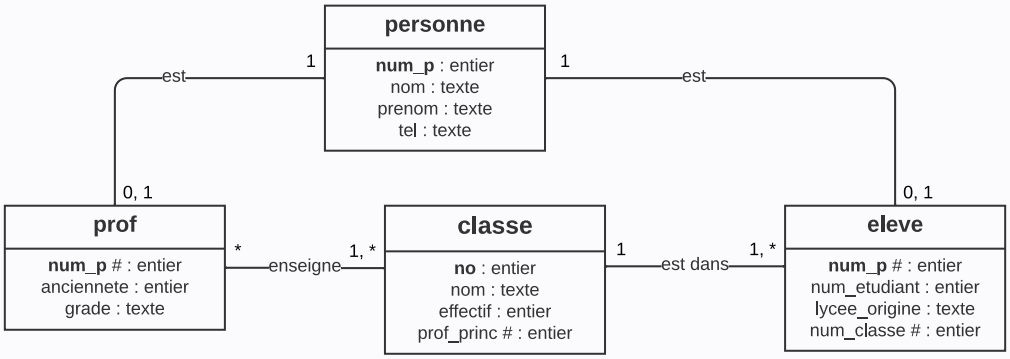
3 Modèle entité-association

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 :

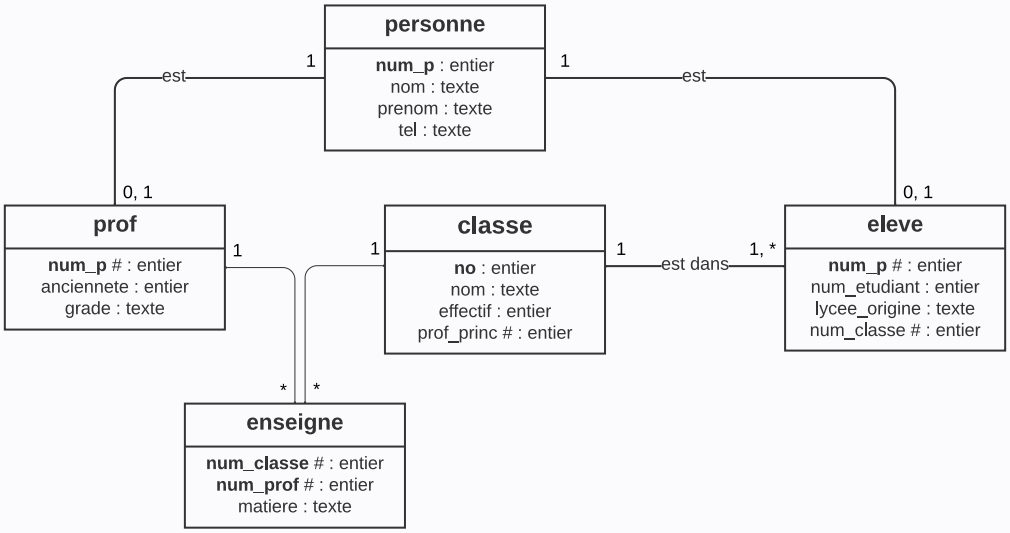
- `1` pour un seul
- `0, 1` pour au plus un
- $*$ pour un nombre quelconque.
  On peut préciser `1, *` pour au moins un.

Exemple 1



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 $*$ — $*$ problématique 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 2

clé primaire ↓

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

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 1

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.

Remarques 1

R 1 – 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 $D_1 \times \cdots \times D_n$.

R 2 – On retiendra que les clés étrangères, suivant qu'elles soient clé primaire de leur table ou non, permettent d'obtenir des associations de cardinalité `1-1` ou `1-*`.

Exemple 3 : Schéma de la base de donnée Lycée (gras pour les clés primaires, flèches pour les clés étrangères)

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

Définition 2

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

Exemple 4 : Populations

- classe :

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

- prof :

num_p	ancienneté	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 5

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 :

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

```SQL
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

```SQL
SELECT * FROM eleve;
```

b Projection

On peut choisir de ne sélectionner que certaines colonnes :

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

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

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

Exemple 6 : Nom et prénom de toutes les personnes

```sql
SELECT nom, prenom FROM personne;
```

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

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

c Produit cartésien

Définition 3 : 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.

Remarques 2

R 1 – Si la première table a M tuples et la deuxième table a N tuples, alors le produit cartésien en a $M \times N$.

R 2 – On ne se sert que très rarement d'un simple produit cartésien.

Exemples 1

E 1 – T_1 :

nom	prenom
Martin	Jean
François	Marie

T_2 :

salaire	prime
1000	200
2000	300

$T_1 \times T_2$:

nom	prenom	salaire	prime
Martin	Jean	1000	200
Martin	Jean	2000	300
François	Marie	1000	200
François	Marie	2000	300

En SQL :

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

d Sélection

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

Exemple 7 : Quelles sont les personnes nommées Payet ?

```sql
SELECT * FROM personne
WHERE nom = 'Payet';
```

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 8

- `LIKE 'N%'` : commence par N.
- `LIKE '%a%'` : contient a.
- `LIKE 'pa%on'` : commence par pa, finit par on (paon, pantalon, passion, etc.)
- numéros des personnes qui s'appellent Martin avec un prénom différent de Pierre, ou dont le nom commence par D :

```sql
SELECT num_p FROM personne
WHERE (nom = 'Martin' AND prenom != 'Pierre')
      OR (nom LIKE 'D%');
```

e Jointures

Définition 4 : 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 1

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.

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

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

Remarques 3

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

```sql
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 :

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

Exemples 2

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

```sql
SELECT *
FROM personne pe JOIN prof pr ON pe.num_p = pr.num_p;
```

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

```sql
SELECT p.nom, tel
FROM classe c JOIN eleve e JOIN personne p
  ON num_classe = no AND p.num_p = e.num_p
WHERE effectif <= 35;
```

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

```sql
SELECT pe.nom, pe.prenom
FROM personne pe JOIN eleve e JOIN enseigne en JOIN personne pp
  ON e.num_p = pe.num_p AND en.num_classe = el.num_classe AND pp.prenom = pe.prenom
  AND pp.num_p = num_prof;
```

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

```sql
SELECT p.nom, matiere
FROM personne p JOIN prof pr JOIN enseigne JOIN classe
  ON pr.num_p = p.num_p AND num_prof = p.num_p AND no = num_classe
WHERE p.num_p != prof_princ;
```

f Opérations ensemblistes

Définition 5 : 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 9

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 10 : Donner le nom de tous les élèves en spé

Exemple 11 : Donner le nom de tous les élèves n'étant pas en PSI*

Exemple 12 : 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 13 : 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 14 : Prix total des articles :

```sql
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`.

```sql
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 15 : Moyenne des anciennetés des professeurs

```sql
SELECT AVG(anciennete) FROM prof;
```

Exemple 16 : Nombre d'élèves dont le nom est Payet

```sql
SELECT COUNT(*)
FROM personne p JOIN eleve e ON e.num_p = p.num_p
WHERE p.nom = 'Payet';
```

Remarque 2

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

```sql
SELECT * FROM personne ORDER BY nom, prenom;
```

Exemple 18 : Enseignants triées par ancienneté décroissante.

```sql
SELECT * FROM prof ORDER BY anciennete DESC;
```

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

```sql
SELECT * FROM prof ORDER BY anciennete DESC LIMIT 1 OFFSET 1;
```

4 Requêtes avec blocs imbriqués

Remarque 3

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

$$\texttt{SELECT ... FROM ... WHERE expr} \left\{ \begin{array}{c} = \\ != \\ < \\ > \\ <= \\ >= \end{array} \right\} \underbrace{\texttt{(SELECT ...)}}_{\text{ne doit renvoyer qu'un résultat.}} ;$$

Exemple 20 : Professeur(s) le(s) plus jeune(s)

```sql
SELECT num_p
FROM prof
WHERE anciennete = (SELECT MIN(anciennete) FROM prof);
```

b Si la requête interne renvoie plusieurs résultats

On utilise alors :

- `IN ...` ($\in$)
- `NOT IN ...` ($\notin$)
- $\left\{ \begin{array}{c} = \\ != \\ < \\ > \\ <= \\ >= \end{array} \right\}$ `ANY ...` ($\exists$)
- $\left\{ \begin{array}{c} = \\ != \\ < \\ > \\ <= \\ >= \end{array} \right\}$ `ALL ...` ($\forall$).

Remarques 4

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

R2 – `IN` $\Longleftrightarrow$ `= ANY`.

R3 – `NOT IN` $\Longleftrightarrow$ `!= ALL`.

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

Exemples 3

E1 – Numéros des professeurs de MPSI

```sql
SELECT num_prof
FROM enseigne
WHERE num_classe IN (SELECT no FROM classe
                     WHERE nom LIKE 'MPSI%');
```

(préférer une jointure dans ce cas.)

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

```sql
SELECT nF
FROM L L1
WHERE nP IN (SELECT nP FROM L
             WHERE nF = 1 AND L1.quantite >= quantite);
```

E3 – Effectif des classes n° 1, 2, 3.

```sql
SELECT effectif FROM classe WHERE no IN (1,2,3);
```

E 4 – Professeurs les plus jeunes

```sql
SELECT num_p
FROM prof
WHERE anciennete <= ALL (SELECT anciennete FROM prof);
```

C Existence

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

Exemple 21 : Nom des élèves n'étant pas seuls issus d'un lycée

```sql
SELECT nom
FROM personne p JOIN eleve e ON e.num_p = p.num_p
WHERE EXISTS (SELECT * FROM eleve
              WHERE lycee_origine = e.lycee_origine
                AND num_p != e.num_p);
```

ou bien

```sql
SELECT nom
FROM personne p JOIN eleve e ON e.num_p = p.num_p
WHERE 1 < (SELECT COUNT(*) FROM eleve
           WHERE lycee_origine = e.lycee_origine);
```

ou encore

```sql
SELECT nom
FROM personne p JOIN eleve e ON e.num_p = p.num_p
WHERE lycee_origine = ANY (SELECT lycee_origine FROM eleve
           WHERE num_p != e.num_p);
```

ou encore avec une jointure (à préférer)

```sql
SELECT nom
FROM personne p JOIN eleve e1 JOIN eleve e2
  ON e1.num_p = p.num_p AND e2.lycee_origine = e1.lycee_origine
WHERE e1.num_p != e2.num_p;
```

Remarque 4

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

Exemple 22 : Nom des professeurs non professeurs principaux

```sql
SELECT nom
FROM personne pp JOIN prof p ON pp.num_p = p.num_p
WHERE NOT EXISTS (SELECT * FROM classe WHERE num_p = prof_princ);
```

ou bien

```sql
SELECT nom
FROM personne pp JOIN prof p ON pp.num_p = p.num_p
WHERE p.num_p NOT IN (SELECT prof_princ FROM classe);
```

ou encore

```sql
SELECT nom
FROM personne pp
JOIN prof p ON pp.num_p = p.num_p
EXCEPT
SELECT nom
FROM personne
JOIN classe ON prof_princ = num_p;
```

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.

```sql
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 4

E 1 – Déterminer le nombre moyen d'élèves de chaque professeur.

```sql
SELECT num_prof, AVG(effectif) AS "Effectif moyen"
FROM classe JOIN enseigne ON num_classe = no
GROUP BY num_prof;
```

Mais attention, les éventuels professeurs n'ayant pas d'élèves (???) n'apparaîtront pas.
Pour les voir, on peut par exemple faire :

```sql
SELECT num_prof, 0 AS "Effectif moyen" FROM prof
WHERE num_prof NOT IN (SELECT num_prof from enseigne)

UNION

SELECT num_prof, AVG(effectif) AS "Effectif moyen"
FROM classe JOIN enseigne ON num_classe = no
GROUP BY num_prof;
```

E 2 – Combien d'élèves au total pour chaque professeur ?

```sql
SELECT nF, COUNT(DISTINCT nP)
FROM L
GROUP BY nF;
```

(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` :

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

```sql
SELECT p.nom, AVG(effectif)
FROM enseigne JOIN classe JOIN personne p
  ON num_classe = no AND num_prof = num_p
GROUP BY num_prof HAVING COUNT(DISTINCT num_classe) > 1;
```

c Agrégats de double niveau

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

Exemple 24 : Moyenne du nombre total d'élèves par professeur (ayant des élèves)

```sql
SELECT AVG(somme)
FROM (SELECT num_prof, SUM(effectif) AS somme
      FROM classe JOIN enseigne ON num_classe = no
      GROUP BY num_prof);
```

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

```sql
SELECT AVG(somme)
FROM
  ( SELECT num_prof, SUM(effectif) AS somme
    FROM
     ( SELECT num_prof, effectif
       FROM enseigne JOIN classe ON num_classe = no
       GROUP BY num_prof, num_classe )
    GROUP BY num_prof );
```

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` :

$$\begin{aligned} x \in T_1 \div T_2 &\iff \forall\, y \in T_2,\ \ (x,y) \in T_1 \\ &\iff \text{non}\left(\exists\, y \in T_2,\ \ (x,y) \notin T_1\right) \\ &\iff \nexists\, y \in T_2,\ \ (x,y) \notin T_1 \end{aligned}$$

- Avec deux `NOT EXISTS` imbriqués :

$$\begin{aligned} x \in T_1 \div T_2 &\iff \forall\, y \in T_2,\ \ (x,y) \in T_1 \\ &\iff \nexists y \in T_2 \mid \nexists z \in T_1 \ ;\ z = (x,y) \end{aligned}$$

- Avec `COUNT` : $x \in T_1 \div T_2$ si et seulement si le nombre de $(x, y) \in T_1$ tel que $y \in T_2$ est le nombre de tuples de T_2.
- Avec `GROUP BY ... HAVING COUNT(...) = ...`

Exemple 25 : Numéros des professeurs enseignant à tous les étudiants

- il n'existe pas d'étudiants qui ne soit pas dans l'une des classes de ce professeur.

SQL
```sql
SELECT num_p FROM prof
WHERE NOT EXISTS (SELECT * FROM eleve e
                  WHERE num_classe NOT IN
                          (SELECT num_classe FROM enseigne
                           WHERE num_prof = num_p)
                  );
```

- il n'existe pas d'étudiants tel qu'il n'existe pas de cours de ce professeur suivi par cet étudiant.

SQL
```sql
SELECT num_p FROM prof
WHERE NOT EXISTS
        (SELECT * FROM eleve e
         WHERE NOT EXISTS (SELECT * FROM enseigne
                           WHERE num_prof = num_p
                             AND num_classe = e.num_classe)
         );
```

- Le nombre d'étudiant qui suivent un cours de ce professeur est le nombre total d'étudiants.

SQL
```sql
SELECT num_p FROM prof
WHERE (SELECT COUNT(*) FROM eleve e)
      = (SELECT COUNT(DISTINCT *) FROM eleve e
        JOIN enseigne en ON e.num_classe = enseigne.num_classe
        WHERE en.num_prof = num_p );
```

SQL
```sql
SELECT num_prof
FROM enseigne JOIN classe c JOIN eleve e
  ON num_classe = no AND e.num_classe = c.num_classe
GROUP BY num_prof
HAVING COUNT(DISTINCT e.num_p) = (SELECT COUNT(*) FROM eleve);
```

III Exercices

Écrire en SQL les requêtes suivantes.

Exercices 1

Ex 1 – Donner le nom et le numéro de téléphone de toutes les personnes enregistrées et qui ont bien un numéro de téléphone.

Ex 2 – Donner le nom et le numéro de téléphone de tous les élèves.

Ex 3 – Donner le nom et le prénom de tous les élèves dont le nom de la classe contient 'SI'.

Ex 4 – Donner d'au moins deux autres façons différentes le nom et le prénom de tous les élèves de sup.

Ex 5 – Donner les classes dont l'effectif est l'ancienneté d'un de ses professeurs, puis le double, puis le nom du professeur.

Ex 6 – Déterminer les couples d'élèves venant d'un même lycée.

Ex 7 – Donner le nom et prénom des élèves et des professeurs de MP2I.

Ex 8 – Donner le nom et le prénom des élèves ayant le même prénom qu'un de leurs professeurs.

Ex 9 – Donner le nom des élèves ayant un professeur avec moins de 15 ans d'ancienneté.

Ex 10 – Donner le nom et la matière des professeurs enseignant dans une classe dont ils ne sont pas professeur principal.

Exercice 1 : correction

SQL

```sql
-- 1. Donner le nom et le numéro de téléphone de toutes les personnes
-- enregistrées ayant un numéro de téléphone.

SELECT nom, tel
FROM personne
WHERE tel IS NOT NULL;

-- 2. Donner le nom et le numéro de téléphone de tous les élèves.

SELECT nom, tel
FROM eleve JOIN personne
  ON eleve.num_p = personne.num_p;

SELECT nom, tel
FROM personne
WHERE num_p IN (SELECT num_p FROM eleve);

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

SELECT p.nom, prenom
FROM personne p JOIN eleve e JOIN classe c
  ON p.num_p = e.num_p AND num_classe = no
WHERE c.nom LIKE '%SI%';

-- 4. Donner d'au moins trois façons différentes le nom et le prénom de tous
-- les élèves de sup.

SELECT p.nom, prenom
FROM personne p JOIN eleve e JOIN classe c
  ON p.num_p = e.num_p AND num_classe = no
WHERE c.nom LIKE 'MPSI%' OR c.nom LIKE 'PCSI%' or c.nom LIKE 'MP2I%';

SELECT p.nom, prenom
FROM personne p JOIN eleve e JOIN classe c
  ON p.num_p = e.num_p AND num_classe = no
WHERE c.nom LIKE 'MPSI%'

UNION

SELECT p.nom, prenom
FROM personne p JOIN eleve e JOIN classe c
  ON p.num_p = e.num_p AND num_classe = no
WHERE c.nom LIKE 'PCSI%'

UNION

SELECT p.nom, prenom
FROM personne p JOIN eleve e JOIN classe c
  ON p.num_p = e.num_p AND num_classe = no
WHERE c.nom LIKE 'MP2I%';
```

SQL

```sql
SELECT p.nom, prenom
FROM personne p JOIN eleve e JOIN classe c
  ON p.num_p = e.num_p AND num_classe = no
WHERE c.nom LIKE '___I%';

-- 5. Y a-t-il une classe dont l'effectif est l'ancienneté du professeur ?
-- Le double ? Si oui, quel est le nom du professeur ?

SELECT pp.nom
FROM classe c JOIN enseigne e JOIN prof p JOIN personne pp
  ON c.no = e.num_classe AND e.num_prof = p.num_p AND pp.num_p = p.num_p
WHERE c.effectif = p.anciennete;

SELECT pp.nom
FROM classe c JOIN enseigne e JOIN prof p JOIN personne pp
  ON c.no = e.num_classe AND e.num_prof = p.num_p ANd pp.num_p = p.num_p
WHERE  c.effectif = p.anciennete * 2
;

-- 6. Déterminer les couples d'élèves venant d'un même lycée.

SELECT e1.num_p, e2.num_p
FROM eleve e1 JOIN eleve e2
  ON e1.lycee_origine = e2.lycee_origine
WHERE e1.num_p < e2.num_p
;

-- 7. Donner le nom et prénom des élèves et des professeurs de MP2I.

SELECT pp.nom, pp.prenom
FROM personne pp JOIN prof p JOIN enseigne e JOIN classe c
  ON pp.num_p = p.num_p AND num_prof = p.num_p AND num_classe = no
WHERE c.nom = 'MP2I'

UNION

SELECT pe.nom, pe.prenom
FROM personne pe JOIN eleve e JOIN classe c
  ON pe.num_p = e.num_p AND num_classe = no
WHERE c.nom = 'MP2I';

Ou bien

SELECT nom, prenom
FROM personne
WHERE num_p IN (SELECT num_p FROM prof p
                  JOIN enseigne e JOIN classe c
                    ON num_prof = p.num_pON num_classe = no
                WHERE c.nom = 'MP2I')
    OR num_p IN (SELECT num_p FROM eleve
                    JOIN classe c ON num_classe = no
                  WHERE c.nom = 'MP2I');
```

SQL

```sql
-- 8. Donner le nom et le prénom des élèves ayant le même prénom qu'un
-- de leurs professeurs.

SELECT pe.nom, pe.prenom
FROM personne pp JOIN prof JOIN personne pe
JOIN eleve e JOIN enseigne
  ON pp.num_p = prof.num_p AND pe.prenom = pp.prenom
 AND pe.num_p = e.num_p AND e.num_classe = enseigne.num_classe
 AND num_prof = prof.num_p;

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

SELECT nom
FROM personne pe JOIN eleve e  JOIN enseigne en JOIN prof p
  ON pe.num_p = e.num_p AND e.num_classe = en.num_classe
 AND p.num_p = num_prof
WHERE anciennete >= 15;

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

SELECT pp.nom, matiere
FROM personne pp JOIN prof p JOIN enseigne JOIN classe
  ON pp.num_p = p.num_p AND num_prof = p.num_p
 AND num_classe = no
WHERE prof_princ != p.num_p;
```