Bases de données relationnelles

MP2I - Informatique

Anthony Lick
Lycée Janson de Sailly

Introduction

Exemple

On souhaite représenter l'ensemble des élèves de SUP de Janson, sachant qu'ils sont regroupés par classes.

Les élèves ont certains attributs :

- leur nom
- leur lycée d'origine en terminale
- la filière suivie (MP2I, MPSI ou PCSI)
- le numéro de la classe

Voici une partie de ces données :

Prénom	Nom	Filière	Numéro	Lycée d'origine
Mathilde	Dufour	MP2I	1	Jules Renard
Léa	Dupond	MPSI	2	Janson de Sailly
Paul	Dugommier	MP2I	1	Janson de Sailly
Mathilde	Dugommier	MPSI	3	Blaise Pascal
Clément	Durand	PCSI	3	Carnot

Implémentation

- On pourrait représenter ces données en C ou OCaml selon plusieurs implémentations.
- La représentation la plus naïve consisterait à stocker tous les 5-uplets dans un tableau, mais cela prendrait énormément de place si on le faisait avec tous les élèves...
- D'autres implémentations pourraient permettre d'accéder à certaines données plus rapidement, mais extraire un autre partie des données pourrait devenir moins efficace.

Présentation des bases de données

Rôle des bases de données

Bases de données

Le rôle des **bases de données** est de simplifier ce genre de considérations, en permettant :

- d'avoir une structure de données efficace
- une rapidité d'accès
- d'éviter à l'utilisateur d'avoir à s'interroger sur la manière dont sont stockées les données
- une sauvegarde des modifications
- une gestion des pannes
- une gestion des conflits si plusieurs utilisateurs modifient la base en même temps
- ...

Exemple avec quelques requêtes

Exemple

Pour les élèves des différentes classes, on pourrait imaginer un système avec trois tables.

Table eleve			
prenom	nom	id_classe	id_lycee
Mathilde	Dufour	833	2
Léa	Dupond	832	1
Paul	Dugommier	833	1
Mathilde	Dugommier	834	3
Clément	Durand	837	4

Exemple avec quelques requêtes

Exemple

Pour les élèves des différentes classes, on pourrait imaginer un système avec trois tables.

Table classe		
id_classe	filiere	numero
831	MPSI	1
832	MPSI	2
833	MP2I	1
834	MPSI	3
835	PCSI	1
836	PCSI	2
837	PCSI	3

Table lycee	
id_lycee	nom
1	Janson de Sailly
2	Jules Renard
3	Blaise Pascal
4	Carnot

Exemple avec quelques requêtes

Exemple

- Sur cet exemple, le découpage d'informations permet de limiter la redondance : on n'indique pour chaque élève que l'identifiant de la classe, pas sa filière et son numéro.
- On peut retrouver ces informations facilement à partir de l'identifiant.
- Si dans une version précédente de la base de données, la **MP2I** s'appelait la **MPSI3**, et la **MPSI3** s'appelait la **MPSI4** effectuer le renommage ne nécessiterait pas de modifier toute la base, mais seulement une entrée de la table `classe`.

Exemple avec quelques requêtes

Voici quelques requêtes qu'on peut effectuer sur ces tables :

sélectionner les élèves de MP2I et les classer par ordre alphabétique

```sql
SELECT nom,prenom
FROM eleves
WHERE id_classe = 833
ORDER BY nom
```

Si on ne connait pas l'identifiant de la MP2I, il faut croiser les tables :

sélectionner les élèves de MP2I et les classer par ordre alphabétique

```sql
SELECT nom, prenom
FROM eleves
JOIN classe ON eleves.id_classe=classe.id_classe
WHERE filiere = "MP2I" AND numero = 1
ORDER BY nom
```

Exemple avec quelques requêtes

De même, pour sélectionner les élèves venant de Janson :

```
SELECT nom, prenom
FROM eleve
JOIN lycee ON eleve.id_lycee=lycee.id_lycee
WHERE lycee.nom="Janson de Sailly"
```

Pour connaître le nombre d'élèves en MPSI :

```
SELECT COUNT (*)
FROM eleve
JOIN classe ON eleve.id_classe=classes.id_classe
WHERE filiere="MPSI"
```

Seules les requêtes de recherche dans une base de données sont officiellement au programme, mais voici un exemple d'insertion dans une base de données :

```
INSERT INTO eleve (nom, prenom, id_classe)
VALUES (Ducourneau, Guillaume, 833)
```

Architecture client-serveur

Architecture

Les bases de données sont articulées sous la forme **client-serveur** :

- l'utilisateur travaille sur un poste (**client**)
- qui peut être éloigné de l'ordinateur qui gère les données (**serveur**).
- Il faut alors permettre un dialogue entre ces deux parties.

Une normalisation a permis d'unifier le langage utilisé dans les bases de données : **SQL**.

Architecture client-serveur

SQL

- Du point de vue des utilisateurs, la syntaxe est la même (en tout cas pour les fonctionnalités de base) ;
- Par contre, la programmation en interne de ces logiciels dépend de l'éditeur (Oracle, SAP, IBM, Microsoft, . . .).

Programme de Prépa

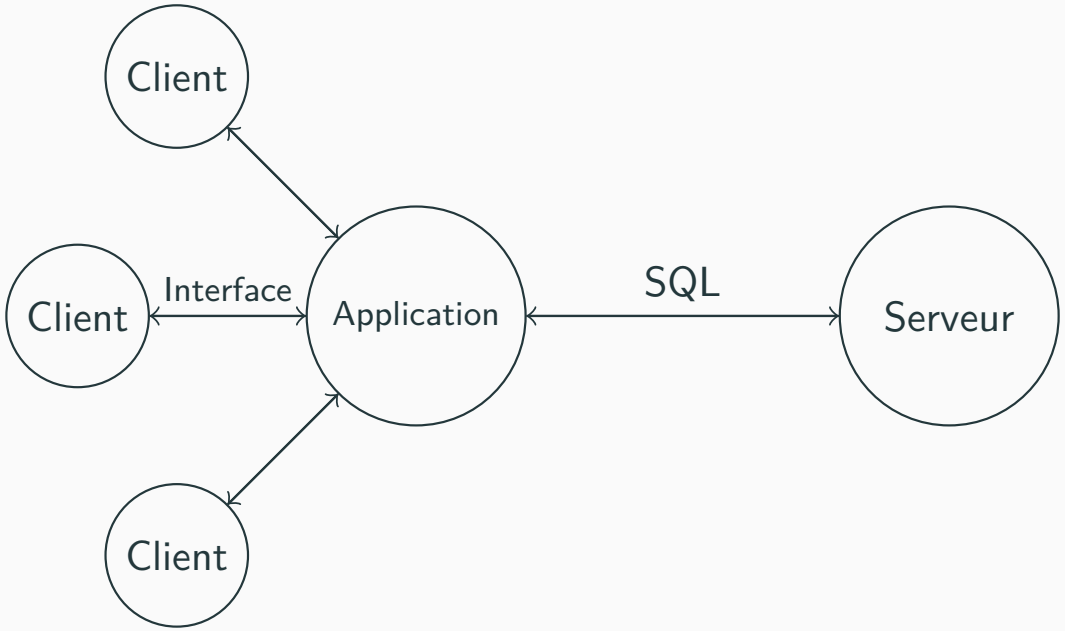
On va voir comment interroger une base de données, sans en créer ni en modifier.

Architecture client-serveur

En pratique

En pratique, le langage SQL n'est en général pas visible pour les usagers.

Entre l'usager et la base de données se trouve en général un serveur applicatif qui traduit les demandes de l'utilisateur (en général via une interface graphique).



Abstraction des bases de données

Algèbre relationnelle

L'**algèbre relationnelle**, théorie inventée en 1970, est une théorie mathématique, proche de la théorie des ensembles, qui définit les opérations pouvant être effectuées sur des relations (ensemble de n-uplets).

C'est cette théorie qui est le cœur des logiciels de base de données, bien qu'elle n'en soit qu'une abstraction.

Dans ce cours, nous verrons les requêtes dans des bases de données à la fois d'un point de vue pratique (SQL) et d'un point de vue théorique (opérations en algèbre relationnelle).

Vocabulaire des bases de données

Modélisation en tableau

Modèle

Le modèle utilisé est très simple : c'est celui d'un tableau à deux dimensions.

Celui-ci se rencontre souvent lorsqu'on traite des données.

Exemple

- un répertoire (nom, téléphone, adresse, ...)
- une fiche de bibliothèque (auteur, titre, année, ...)
- un carnet de commande (client, article, date, prix, ...)

Modélisation en tableau

Représentation

En général, on représente ces données dans un tableau, avec les données d'un élément (une fiche de bibliothèque, une commande, un individu du répertoire) sur une même ligne, les colonnes donnant les différents **attributs** (auteur, titre, année).

Table eleve

prenom	nom	filiere	numero	lycee_origine	note_bac
Mathilde	Dufour	MP2I	1	Jules Renard	18
Léa	Dupond	MPSI	2	Janson de Sailly	14
Paul	Dugommier	MP2I	1	Janson de Sailly	12
Mathilde	Dugommier	MPSI	4	Blaise Pascal	15
Clément	Durand	PCSI	3	Carnot	14

Vocabulaire

Attributs

Les différents titres de colonnes sont appelés **attributs**.

On notera formellement $A_1, A_2, \ldots, A_p$ les attributs.

Les attributs forment un ensemble : il n'y a pas d'attribut en double.

L'ordre des attributs n'est pas fixé : on ne parle pas du premier attribut mais de l'attribut nom (par exemple).

Exemple

La table précédente présente 6 attributs.

Vocabulaire

Domaine

L'ensemble des valeurs que peut prendre un attribut A est son **domaine** $\mathrm{Dom}(A)$.

Exemple

Dans la table précédente :

- 4 attributs ont pour domaine des chaînes de caractères ;
- mais on peut imaginer que le domaine des classes est réduit à l'ensemble des sigles dénotant une filière de classe préparatoire (MPSI,PCSI, BCPST, PSI...) ;
- le numéro est un entier ;
- et la note du bac un flottant (qu'on peut imposer dans l'intervalle $[0, 20]$ avec éventuellement un certain nombre de chiffres significatifs. . .).

Vocabulaire

Schéma relationnel

On appelle **schéma relationnel** un p-uplet d'attributs, vérifiant toujours la contrainte que les attributs sont distincts deux à deux.

Par la suite, on notera $\mathcal{S}$ un schéma relationnel.

Vocabulaire

Tuple

Chaque ligne s'appelle un p-**uplet** (ou **tuple** en anglais).

C'est donc un élément de $\mathrm{Dom}(A_1) \times \mathrm{Dom}(A_2) \times \cdots \times \mathrm{Dom}(A_p)$.

Remarque

En toute rigueur, puisque l'ordre des attributs n'est pas fixé, une ligne est plutôt une fonction

$$\ell : \{A_1, \ldots, A_p\} \to \mathrm{Dom}(A_1) \cup \mathrm{Dom}(A_2) \cup \cdots \cup \mathrm{Dom}(A_p)$$

avec la contrainte que $\ell(A_i) \in \mathrm{Dom}(A_i)$.

Mais il est plus simple de parler du tuple (Mathilde, Dufour, PCSI, 2, Jules Renard, 18) que de l'application qui a chacun des attributs associe sa valeur.

Vocabulaire

Relation

On appelle **relation** (ou **table**) un ensemble fini de p-uplets de $\mathrm{Dom}(A_1) \times \mathrm{Dom}(A_2) \times \cdots \times \mathrm{Dom}(A_p)$.

Une telle relation est souvent notée $\mathcal{R}$, ou encore $\mathcal{R}(\mathcal{S})$ pour préciser que la relation est associée au schéma relationnel $\mathcal{S}$.

Les éléments de $\mathcal{R}$ sont appelés **valeurs** ou **enregistrements**.

Contraintes

Notations

On a déjà vu qu'il n'y avait pas d'attribut en double.

$\hookrightarrow$ Sinon, ils risqueraient d'être affectés avec des valeurs différentes dans des tuples !

Pour t un tuple de $\mathcal{R}(\mathcal{S})$, on note $t[A_i]$ la composante du couple associée à l'attribut A_i.

Par extension, si $X = (B_1, \ldots, B_n) \subseteq \mathcal{S}$, on note $t[X]$ le n-uplet $(t[B_1], \ldots, t[B_n])$.

Contraintes

Définition

Dans une relation $\mathcal{R}(\mathcal{S})$, les valeurs forment un ensemble.

Remarque

Cela a deux implications :

- visuellement, on ne peut pas avoir deux lignes égales dans le tableau (il n'y a donc pas de redondance de données). Deux valeurs doivent donc différer d'au moins un attribut.
- L'ordre des lignes n'est pas fixé (même si, lorsqu'on représente des données dans un tableau, on choisi un ordre pour les attributs et les tuples).

Clés

Clés

Pour garantir la non-répétition des enregistrements, les bases de données réelles contiennent un concept de **clé** qui doit être pensé dès la conception des bases de données, et indiqué à la création.

Ce concept a également son importance lors de l'utilisation de relations multiples.

Super Clés

Définition

Une super-clé d'une relation $\mathcal{R}$ est un ensemble X d'attributs tel que :

$$\forall (t, t') \in \mathcal{R}^2 \quad t[X] = t'[X] \Rightarrow (t = t')$$

Exemple

- Puisque deux enregistrements d'une relation ne peuvent pas être égaux, l'ensemble des attributs est **toujours** une super-clé.
- Dans la table `eleve`, {prenom} et {nom} ne sont pas des super-clés.
- En revanche, {nom, prenom} est une super-clé.

Clés candidates

Définition

Une **clé candidate** d'une relation $\mathcal{R}$ est une super-clé minimale (pour l'inclusion).

K est donc une clé candidate si :

- K est une super-clé ;
- $\forall K' \subsetneq K$, K' n'est pas une super-clé.

Exemple

Dans la table `eleve`, $\{$nom, prenom$\}$ est une clé candidate.

Clés candidates

Proposition

Une super-clé qui ne contient qu'un seul attribut est toujours une clé candidate.

Exemple

Une telle super-clé n'existe pas toujours : par exemple, il n'y en a pas dans la table `eleve`.

En revanche, dans la table `classe`, id_classe en est une.

Clés primaires

Définition

La **clé primaire** est une clé qu'on a choisi parmi les clés candidates.

Table classe		
id_classe	filiere	numero
831	MPSI	1
832	MPSI	2
833	MP2I	1
834	MPSI	3
835	PCSI	1
836	PCSI	2
837	PCSI	3

Clés primaires

En pratique

Pour indiquer la clé primaire, on souligne les attributs correspondants dans la table.

Indiquer au système une clé primaire pour chaque table permet une indexation des données à l'aide de cette clé, ce qui rend les requêtes sur ces tables plus efficaces.

$\hookrightarrow$ Cette indication doit se faire à la création de la table.

Clés primaires

Remarque

On évitera les clés primaires qui ne seraient primaires que “par accident”, car on pourrait insérer des données supplémentaires qui feraient perdre le caractère primaire des clés.

$\hookrightarrow$ En pratique, le serveur interdira l'ajout de ces données.

Par exemple, dans la table `eleve`, le couple $\{$nom, filiere$\}$ est une clé candidate, mais rien ne garantit qu'elle puisse le rester.

Clés primaires

id

On introduit souvent un attribut supplémentaire (commençant en général par "**id**") dans la relation, qui sera un entier qu'on incrémentera à chaque ajout d'un tuple. Ce sera la clé primaire.

On ne le fera pas systématiquement, notamment lorsque la relation contient des **clés étrangères** (voir plus loin).

Table eleve'

id_eleve	prenom	nom	filiere	numero	lycee_origine	note_bac
1	Mathilde	Dufour	MP2I	1	Jules Renard	18
2	Léa	Dupond	MPSI	2	Janson de Sailly	14
3	Paul	Dugommier	MP2I	1	Janson de Sailly	12
4	Mathilde	Dugommier	MPSI	4	Blaise Pascal	15
5	Clément	Durand	PCSI	3	Carnot	14

Algèbre relationnelle

Algèbre relationnelle

L'**algèbre relationnelle** sert à formaliser les questions que l'on peut poser à une relation ou à un ensemble de relations.

Le langage SQL est l'implémentation concrète des opérations de l'algèbre relationnelle.

Définition

L'algèbre relationnelle est un ensemble d'opérateurs qu'on peut appliquer à des relations, et dont le résultat est une relation.

Comme le résultat est toujours une relation on pourra combiner les opérateurs : on forme ainsi, à partir d'opérateurs élémentaires, des requêtes élaborées.

L'objectif est de pouvoir exprimer toute manipulation raisonnable des données par une expression algébrique.

Algèbre relationnelle

Exemple

Voici des exemples de questions qu'on peut demander au serveur de bases de données.

- Quels sont les élèves venant du lycée Blaise Pascal ?
- Ceux de MPSI 2 ?
- Quelle est la moyenne au bac des élèves de Janson ?
- Qui sont ceux qui ont obtenu une mention ?
- Étant données deux relations pour les élèves de Janson en 2018-2019 et 2019-2020, qui sont les 5/2 ?

La dernière requête demande d'avoir deux relations, mais on va quand même y répondre ici.

Opérateurs ensemblistes

Opérateurs ensemblistes

Un premier type d'opérateur combine 2 relations qui ont le même schéma. $\mathcal{R}$ et $\mathcal{R}'$ sont deux relations ayant le même schéma (c'est-à-dire les mêmes attributs).

- $\mathcal{R} \cup \mathcal{R}'$ est la relation de même schéma dont les tuples sont ceux qui appartiennent à R ou à $\mathcal{R}'$.
- $\mathcal{R} \cap \mathcal{R}'$ est la relation de même schéma dont les tuples sont ceux qui appartiennent à R et à $\mathcal{R}'$
- $\mathcal{R} \setminus \mathcal{R}'$ est la relation de même schéma dont les tuples sont ceux qui appartiennent à R mais pas à $\mathcal{R}'$.

Exemple

Il suffit donc deux faire l'intersection de deux relations pour avoir les élèves ayant fait 5/2 en 2019-2020.

Opérateurs spécifiques

On reprend notre exemple de la table `eleve`.

Table eleve					
prenom	nom	filiere	numero	lycee_origine	note_bac
Mathilde	Dufour	MP2I	1	Jules Renard	18
Léa	Dupond	MPSI	2	Janson de Sailly	14
Paul	Dugommier	MP2I	1	Janson de Sailly	12
Mathilde	Dugommier	MPSI	4	Blaise Pascal	15
Clément	Durand	PCSI	3	Carnot	14

Projection

Projection

La projection consiste à ne garder qu'une partie des attributs.

Pour $X \subseteq S$, on note $\pi_X(\mathcal{R})$ la relation extraite de $\mathcal{R}$ avec les mêmes tuples restreints à l'ensemble X.

$$\pi_X(\mathcal{R}) = \{t[X] \mid t \in \mathcal{R}\}$$

Exemple

Par exemple, $\pi_{\text{prenom,nom}}(\text{eleve})$ est la table suivante :

$\pi_{\text{prenom,nom}}(\text{eleve})$	
prenom	nom
Mathilde	Dufour
Léa	Dupond
Paul	Dugommier
Mathilde	Dugommier
Clément	Durand

Projection

Remarque

Dans une relation, les tuples forment un ensemble. Il se peut donc que $\pi_X(\mathcal{R})$ ait moins d'éléments que la table initiale.

En fait, le nombre d'éléments est conservé si et seulement si l'ensemble d'attributs X est une super-clé de la relation.

$\pi_{\text{prenom}}(\text{eleve})$
prenom
Mathilde
Léa
Paul
Clément

Sélection

Sélection

La **sélection** consiste à ne garder que les tuples qui vérifient une propriété.

On note $\sigma_{\mathcal{P}}(\mathcal{R})$ la relation extraite de $\mathcal{R}$ avec les mêmes attributs dont les tuples vérifient la propriété $\mathcal{P}$.

$$\sigma_{\mathcal{P}} = \{t \in \mathcal{R} \mid \mathcal{P}(t)\}$$

Sélection

Propriétés

Les propriétés élémentaires sont de la forme $E \mathrm{op} E'$, où op est un opérateur de comparaison $(=, <, \leq, >, \geq)$, et E et E' sont des expressions construites à partir des attributs et des constantes avec des fonctions usuelles.

Une propriété composée avec des connecteurs logiques (ET, OU et NON, aussi notés $\wedge$, $\vee$ et $\neg$) correspond à des unions et intersections.

Par exemple, $\sigma_{\mathcal{P} \vee \mathcal{P}'}(\mathcal{R}) = \sigma_{\mathcal{P}}(R) \cup \sigma_{\mathcal{P}'}(R)$.

Sélection : exemples

$\sigma_{\text{filiere}=\text{“MPSI”} \vee \text{prenom}=\text{“Mathilde”}}(\text{eleve})$					
prenom	nom	filiere	numero	lycee_origine	note_bac
Mathilde	Dufour	MP2I	1	Jules Renard	18
Léa	Dupond	MPSI	2	Janson de Sailly	14
Mathilde	Dugommier	MPSI	4	Blaise Pascal	15

Remarque

Sélection et projection vont souvent ensemble, et on peut bien sûr composer ces opérations.

$\pi_{\text{prenom,nom}}(\sigma_{\text{filiere}=\text{“MPSI”}}(\text{eleve}))$	
prenom	nom
Léa	Dupond
Mathilde	Dugommier

Renommage

Le **renommage** consiste à renommer un attribut.

Ce sera utile lors de produits de tables lorsque deux tables ont des attributs portant le même nom mais correspondent à des données différentes.

Si $A \in \mathcal{S}$ et $B \notin \mathcal{S}$, on peut renommer l'attribut A en B pour convertir un élément de $\mathcal{R}(\mathcal{S})$ en un élément de $\mathcal{R}(\mathcal{S}')$ où $\mathcal{S}' = (S \setminus \{A\}) \cup \{B\}$:

$$\rho_{A \to B}(\mathcal{R}) = \{t \mid \exists r \in \mathcal{R},\ t[B] = r[A] \\ \text{et } \forall C \in S \setminus \{A\},\ t[C] = s[C]\}$$

Par extension, si les A_i et B_j sont tous distincts, on notera $\rho_{A_1 \to B_1, \ldots, A_p \to B_p}$ pour $\rho_{A_1 \to B_1} \circ \cdots \circ \rho_{A_p \to B_p}$.

Renommage

Table classe		
id_classe	filiere	numero
831	MPSI	1
832	MPSI	2
833	MP2I	1
834	MPSI	4
835	PCSI	1
836	PCSI	2
837	PCSI	3

$\rho_{\text{filiere}\to\text{sec},\text{numero}\to\text{nb}}(\text{classe})$

id_classe	sec	nb
831	MPSI	1
832	MPSI	2
833	MP2I	1
834	MPSI	4
835	PCSI	1
836	PCSI	2
837	PCSI	3

SQL

Le langage SQL

SQL

Pour implémenter l'algèbre relationnelle et faire des requêtes à des bases de données, une norme universelle a été établie : qui permet d'écrire des requêtes de la même façon quelque soit le logiciel.

Le langage utilisé est **SQL** (*Structured Query Langage*). Il reprend la structure de l'algèbre relationnelle en y ajoutant des moyens de calculs (ordonnancement des résultats, par exemple).

Ce langage est très proche du langage humain (anglais).

Vocabulaire

Vocabulaire

Les représentations des relations se font avec le modèle du tableau. En SQL on parlera de :

- tables à la place de relations ;
- colonnes à la place d'attributs ;
- lignes à la place de tuples.

Syntaxe

Syntaxe

La forme générale d'une requête en SQL est :

```
SELECT ... FROM table ... ;
```

Les mots clés de SQL sont en général écrits en majuscules, mais ce n'est pas obligatoire.

Les requêtes se terminent par un point-virgule.

SELECT

Le mot-clé principal d'une requête dans une base de données est **SELECT**, qu'on retrouvera en tête de toutes nos requêtes SQL.

Projection

SELECT

Basiquement, **SELECT** permet de faire une **projection** (et non pas une sélection !). Il suffit d'indiquer les attributs que l'on veut garder juste après **SELECT** :

```
SELECT A1,...,Ap FROM table ... ;
```

Les attributs (colonnes) à garder sont énumérés et séparés par une virgule.

Si on ne veut pas effectuer de projection (c'est-à-dire garder tous les attributs), on peut utiliser le joker * au lieu d'énumérer tous les attributs un à un.

```
SELECT * FROM table ... ;
```

Projection

FROM

Le mot-clé **FROM** permet d'indiquer le nom de la table à utiliser.

Attention

EN SQL, les résultats d'une requête ne forment pas une table car les doublons résultant de projections ne sont pas supprimés.

Il faut utiliser le mot-clé **DISTINCT** pour les supprimer.

```
>>> SELECT prenom FROM eleve ;
"Mathilde"
"Léa"
"Paul"
"Mathilde"
"Clément"
>>> SELECT DISTINCT prenom FROM eleve ;
"Mathilde"
"Léa"
"Paul"
"Clément"
```

Sélection

WHERE

- La **sélection** se fait avec le mot-clé **`WHERE`** ; placé après le nom de la table.
- On utilise `=` et `!=` pour tester l'égalité et la différence.
- Si le domaine de l'attribut le permet, on peut utiliser d'autres comparateurs (`>`, `<`, `>=`, `<=`) et même des fonctions arithmétiques.
- Une condition complexe peut être exprimée à l'aide de conditions plus simples et des connecteurs logiques **`AND`**, **`OR`** et **`NOT`**.

Renommage

On peut renommer un ou plusieurs attributs avec **AS**, ou même en juxtaposant le nouveau nom à droite de l'ancien.

Ceci sera particulièrement utile lorsqu'on aura plusieurs tables dont les noms d'attributs sont les mêmes, ou lorsqu'on fera des sous-requêtes.

La syntaxe générale est :

```
SELECT A1 AS B1,..,Ai AS Bi, C1,..,Cj FROM table ;
```

Le mot-clé **AS** est facultatif, on peut aussi simplement séparer les `Ai` et `Bi` par un espace.

Renommage : exemple

```sql
SELECT prenom p, nom n, notebac/2 note_sur_10 FROM eleve ;
```

Exemple

Par exemple, la requête ci-dessus produit la table suivante.

$\rho_{\text{note\_bac}\to\text{note\_sur\_10}}(\pi_{\text{prenom,nom,note\_bac}/2}(\text{eleve}))$

p	n	note_sur_10
Mathilde	Dufour	9
Léa	Dupond	7
Paul	Dugommier	6
Mathilde	Dugommier	7
Clément	Durand	7

Opérations ensemblistes

Opérations ensemblistes

Si on veut combiner plusieurs requêtes on peut les assembler avec **UNION**, **INTERSECT** ou **EXCEPT**, qui réalise l'union, l'intersection et la différence.

Ceci dit, lorsqu'on a qu'une seule table il est plus simple (et préférable) de combiner les conditions.

Exemple

Si l'on a deux tables `eleves_19_20` et `eleves_20_21` qui donnent les élèves du lycée de deux années successives, alors ceux ayant fait 5/2 en 2019-2020 sont :

```
1  SELECT * FROM eleves_19_20
2  INTERSECT
3  SELECT * FROM eleves_20_21
```

Opérations entre attributs

Opérations entre attributs

Il est possible de réaliser des opérations entre attributs.

Exemple

La requête suivante fait la somme de trois attributs a, b et c d'une table.

```
1  SELECT a+b+c FROM ...
```

Agrégats et fonctions d'agrégation

Agrégats

Définition

Soit $f : E^n \to F$. On dit que f est symétrique si $\forall (x_1, \ldots, x_n) \in E^n$, $\forall i \neq j$, on a :

$$\begin{aligned} & f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_n) \\ =& f(x_1, \ldots, x_{i-1}, x_j, x_{i+1}, \ldots, x_{j-1}, x_i, x_{j+1}, \ldots, x_n) \end{aligned}$$

Définition

On appelle **fonction d'agrégation** $E \to F$ une suite $(f_n)_{n \geq 1}$ de fonctions symétriques de $E^n \to F$.

Fonctions d'agrégation

Exemple

- La somme est une fonction d'agrégation :

$$f_n(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$$

- le produit ;
- le min, le max ;
- le cardinal ;
- la moyenne ;
- ...

Utilisation des fonctions d'agrégation

Utilisation

On peut utiliser de telles fonctions en regroupant les données par paquets pour obtenir une nouvelle table.

Formellement, si

- f est une fonction d'agrégation,
- X un ensemble d'attributs d'une relation $\mathcal{R}(S)$,
- et $A \in \mathcal{S} \setminus X$,

pour chaque tuple t_0 de valeurs de X, on note $f(t_0, A)$ la valeur de f appliquée à l'attribut A des tuples de $\sigma_{t[X]=t_0}(\mathcal{R})$.

Opération d'agrégation

Agrégation

On introduit une nouvelle opération, l'**agrégation** notée ${}_X\gamma_{f(A)}$.

L'image de $\mathcal{R}$ par cette opération est une relation de schéma $X \cup \{f(A)\}$.

$${}_X\gamma_{f(A)} = \{t \mid \exists s \in \mathcal{R},\ t[X] = s[X] \text{ et } t[f(A)] = f(s[X], A)\}$$

En pratique

En pratique, on regroupe la relation selon les valeurs des attributs de X et on calcule $f(A)$ pour chacun des ensembles de lignes définies par ces regroupements.

Opération d'agrégation

Exemple

- $\text{filiere}\gamma_{\text{MOYENNE(notebac)}}(\text{eleve})$ est une relation à deux attributs, donnant la moyenne des notes au bac suivant la filière.
- $\emptyset\gamma_{\text{MOYENNE(notebac)}}(\text{eleve})$ est une relation avec un unique tuple de taille 1, donnant la moyenne des notes au bac de tous les élèves de la table.

Fonctions d'agrégation

Les résultats d'une requête seront souvent utilisés ensuite, par exemple à des fins statistiques.

SQL contient la possibilité de faire quelques uns de ces calculs.

Les fonctions disponibles (de base) sont, parmi d'autres :

- le comptage, c'est-à-dire le nombre de lignes : **COUNT**
- le maximum des éléments dans une colonne : **MAX**
- le minimum des éléments dans une colonne : **MIN**
- la somme des éléments d'une colonne : **SUM**
- la moyenne des éléments d'une colonne (sum/count) : **AVG**

Le résultat est une table avec une unique ligne et une unique colonne, que l'on peut utiliser comme valeur.

SQL

Syntaxe

Si f est une de ces fonctions, on l'emploie sous la forme :

```
SELECT f(attribut) FROM table WHERE condition ;
```

Exemple

Avec la relation `eleves`, `SELECT AVG(notebac) FROM eleves` produit 14.6.

Groupement

Le mot-clé **GROUP BY** sert à indiquer sur quels attributs sont effectuées les regroupements.

Exemple

La requête ci-dessous produit une table ayant une unique colonne :

AVG(notebac)
15
14.5
14

```
1  SELECT AVG(notebac) FROM eleves GROUP BY filiere ;
```

SQL

Exemple

En effectuant une sélection supplémentaire, il est a priori plus intéressant d'afficher aussi la filière :

filiere	AVG(notebac)
MP2I	15
MPSI	14.5
PCSI	14

```
1  SELECT filiere,AVG(notebac) FROM eleves GROUP BY filiere ;
```

Exemple

Pour la moyenne sur chaque classe, on écrirait `GROUP BY filiere,numero`.

Sélection après agrégation

Sélection

Pour faire une sélection après une opération d'agrégation, on utilise **HAVING**.

Exemple

À la suite de la requête précédente, on pourrait vouloir garder uniquement les filières ayant une moyenne supérieure à 14.4.

$$\sigma_{\text{MOYENNE(notebac)}>14.4} \circ \ {}_{\text{filiere}}\gamma_{\text{MOYENNE(notebac)}}(\text{eleve})$$

filiere	AVG(notebac)
MP2I	15
MPSI	14.5

```
1  SELECT filiere,AVG(notebac) FROM eleves GROUP BY filiere HAVING AVG(notebac)>14.4 ;
```

Renommage

Remarque

Il est ici très commode de procéder à un renommage : garder un attribut qui s'appelle AVG(notebac) n'est pas très pratique.

```
1  SELECT filiere,AVG(notebac) AS moy FROM eleves GROUP BY filiere HAVING moy>14.4 ;
```

filiere	moy
MP2I	15
MPSI	14.5

Sélection avec WHERE et HAVING

WHERE vs HAVING

Pour faire des sélections, on a donc deux outils à notre disposition : `WHERE` et `HAVING`.

- `WHERE` sélectionne avant une agrégation (on dit en **amont**),
- et `HAVING` en **aval**.

Lorsqu'on a le choix, il vaut mieux sélectionner en amont, pour n'effectuer l'agrégation que sur un nombre minimal de lignes.

Évidemment dans l'exemple ci-dessus, la sélection en aval est tout à fait légitime car on ne pouvait pas faire de sélection en amont.

Affichage des résultats

Ordonner les résultats

Ordre

Puisqu'une requête SQL affiche le résultats sous un certain ordre, on peut voir ce résultat non pas comme un simple ensemble mais comme un ensemble **ordonné**.

Syntaxe

La syntaxe en SQL pour ordonner les résultats se fait à l'aide du mot-clé `ORDER BY`, couplé à une expression arithmétique des attributs.

L'expression sera souvent un attribut lui-même.

ORDER BY : exemple

```sql
SELECT * FROM eleve ORDER BY notebac ;
```

Table eleve

prenom	nom	filiere	numero	lycee_origine	note_bac
Paul	Dugommier	MP2I	1	Janson de Sailly	12
Clément	Durand	PCSI	3	Carnot	14
Léa	Dupond	MPSI	2	Janson de Sailly	14
Mathilde	Dugommier	MPSI	4	Blaise Pascal	15
Mathilde	Dufour	MP2I	1	Jules Renard	18

ORDER BY

Remarques

- On peut spécifier si on veut le résultat dans l'ordre croissant ou décroissant à l'aide des mots-clés **`ASC`** et **`DESC`** juste après l'expression (pour ascendant et descendant).
  Le comportement par défaut est **`ASC`**.
- On peut mettre plusieurs expressions après **`ORDER BY`**, séparées par des virgules. Dans ce cas, la relation est triée pour l'ordre **lexicographique** selon ces expressions.

Exemple

Exemple

On trie des points par distance à l'origine décroissante.

En cas d'égalité, ils sont triés par valeur absolue d'abscisse croissante.

```
1  SELECT nom,abscisse AS a, ordonnee AS o FROM point ORDER BY a*a+o*o DESC,ABS(a) ASC ;
```

Table point

nom	abscisse	ordonnee
A	0	0
B	1	0
C	0	1
D	2	-3
E	-3	-2

nom	a	o
D	2	-3
E	-3	-2
C	0	1
B	1	0
A	0	0

Limiter l'affichage avec LIMIT et OFFSET

LIMIT et OFFSET

Parfois, on voudra ne garder que quelques lignes du résultat :

- on peut limiter le nombre de résultats d'une requête SQL en utilisant **`LIMIT`** : en ajoutant **`LIMIT`** `n` avec n un entier, on limite le nombre de résultats à n ;
- il est aussi possible de préciser un **`OFFSET`** : avec **`OFFSET`** `m` on ignore les m premiers résultats.

Limiter l'affichage avec LIMIT et OFFSET

Exemple

```sql
SELECT nom,abscisse AS a,ordonnee AS o FROM point
ORDER BY a*a+o*o DESC, ABS(a) ASC LIMIT 2 OFFSET 1 ;
```

Table point

nom	abscisse	ordonnee
A	0	0
B	1	0
C	0	1
D	2	-3
E	-3	-2

nom	a	o
E	-3	-2
C	0	1

Composition de requêtes

Composition de requêtes

```
1  SELECT filiere,AVG(notebac) moy FROM eleve GROUP BY filiere HAVING moy>14.4 ;
```

Exemple

On a déjà vu **HAVING**, qui effectue une sélection en aval d'une agrégation (requête ci-dessus).

Voici un exemple équivalent mais sans **HAVING** : on effectue une première requête (ligne 2) produisant des lignes de la forme (filière, moyenne) qu'on réutilise immédiatement dans une nouvelle requête.

```
1  SELECT filiere,moy FROM
2      (SELECT filiere,AVG(notebac) moy FROM eleve GROUP BY filiere)
3  WHERE moy>14.4 ;
```

Composition de requêtes

Composition

Une requête SQL sur des tables produit une table.

Tout comme en algèbre relationnelle où on peut composer des opérateurs, on peut composer des requêtes SQL.

Exemple

Exemple

Quels sont les élèves ayant eu la plus haute note au bac ?

```sql
SELECT * FROM eleve WHERE notebac=(SELECT MAX(notebac) FROM eleve) ;
```

Remarque

La requête suivante n'a pas vraiment de sens en algèbre relationnelle, mais fonctionne en général en SQL (cela dépend du logiciel utilisé).

En revanche, elle ne reverra toujours qu'une seule ligne, même si plusieurs élèves ont eu la même note maximale.

```sql
SELECT nom,prenom,MAX(notebac) FROM eleve ;
```

IN et WITH (hors programme)

IN

On peut utiliser la composition avec le mot-clé **IN** qui teste si un tuple appartient à une table, en utilisant une requête à droite du **IN**.

élèves de MP2I avec le même prénom qu'un élève de MPSI

```sql
SELECT * FROM eleve WHERE filiere="MP2I" AND prenom IN
(SELECT prenom FROM eleve WHERE filiere="MPSI") ;
```

WITH

Le mot-clé **WITH** permet de réutiliser facilement le résultat d'une requête comme une table :

```sql
WITH r AS (SELECT ...) SELECT ... FROM r ... ;
```

Produit cartésien et jointure

Introduction

Tables multiples

Pour éviter la redondance des informations il est en général préférable d'organiser nos données en différentes tables, plutôt qu'en une unique base de données (plate).

Table eleve			
prenom	nom	id_classe	id_lycee
Mathilde	Dufour	833	2
Léa	Dupond	832	1
Paul	Dugommier	833	1
Mathilde	Dugommier	834	3
Clément	Durand	837	4

Introduction

Tables multiples

Pour éviter la redondance des informations il est en général préférable d'organiser nos données en différentes tables, plutôt qu'en une unique base de données (plate).

Table classe

id_classe	filiere	numero
831	MPSI	1
832	MPSI	2
833	MP2I	1
834	MPSI	3
835	PCSI	1
836	PCSI	2
837	PCSI	3

Table lycee

id_lycee	nom
1	Janson de Sailly
2	Jules Renard
3	Blaise Pascal
4	Carnot

Notion de clé étrangère

Définition

Une **clé étrangère** d'une relation $\mathcal{R}$ est un attribut qui est lié à une clé primaire d'une autre relation $\mathcal{R}'$.

Notamment, les valeurs que peut prendre cet attribut dans $\mathcal{R}$ est limité aux valeurs de la clé primaire dans $\mathcal{R}'$.

Exemple

Dans les tables précédentes, les deux attributs `id_classe` et `id_lycee` de la table `eleve` peuvent naturellement être considérées comme des clés étrangères vers les attributs du même nom dans les tables `classe` et `lycee`.

Remarque

Il n'y a aucune obligation que les noms soient les mêmes.

Notion de clé étrangère

En pratique

Tout comme la déclaration d'une clé primaire, la déclaration d'une clé étrangère se fait à la création de la base.

L'utilité d'une telle contrainte n'apparaît pas vraiment lorsqu'on fait des recherches dans une base (la seule chose au programme), mais néanmoins il est facile d'en percevoir l'intérêt : si on voulait rajouter un élève dans la classe 8340, le serveur nous dirait que cette classe n'existe pas.

Ceci dit, les clés étrangères sont très utiles pour faire des jointures sur les tables, ce qu'on va voir bientôt.

Produit

Définition (produit)

Si $\mathcal{R}$ est une relation de shéma $\mathcal{S}$ et $\mathcal{R}'$ une relation de schéma $\mathcal{S}'$ avec $\mathcal{S} \cap \mathcal{S}' = \emptyset$, alors le **produit** de $\mathcal{R}$ et $\mathcal{R}'$ est la relation $\mathcal{R} \times \mathcal{R}'$ de schéma $\mathcal{S} \cup \mathcal{S}'$ définie par :

$$\mathcal{R} \times \mathcal{R}' = \{u \mid u[\mathcal{S}] \in \mathcal{R} \text{ et } u[\mathcal{S}'] \in \mathcal{R}'\}$$

Remarques

- La condition $\mathcal{S} \cap \mathcal{S}' = \emptyset$ n'est en fait pas contraignante : on peut procéder par renommage pour l'assurer.
- Si $\mathcal{R}$ et $\mathcal{R}'$ possèdent le même attribut A, on note $\mathcal{R}.A$ et $\mathcal{R}'.A$ pour les différentier.

Produit

Cardinal

Une telle table a donc pour cardinal $|\mathcal{R}| \times |\mathcal{R}'|$, ce qui peut facilement devenir très lourd avec des grosses tables.

Exemple

Voici le début de la table eleve $\times$ lycee.

Table eleve $\times$ lycee					
prenom	nom	id_classe	id_lycee	id_lycee	nom_lycee
Mathilde	Dufour	833	2	1	Janson de Sailly
Mathilde	Dufour	833	2	2	Jules Renard
Mathilde	Dufour	833	2	3	Blaise Pascal
Léa	Dupond	832	1	1	Janson de Sailly
Léa	Dupond	832	1	2	Jules Renard

Produit

Remarque

Il semble y avoir deux attributs de même nom (ce qui est impossible) : en fait les noms de ces attributs sont `eleve.id_lycee` et `lycee.id_lycee` (la présentation sous cette forme suit ce qu'il se passe en SQL).

Table eleve × lycee

prenom	nom	id_classe	id_lycee	id_lycee	lycee.nom
Mathilde	Dufour	833	2	1	Janson de Sailly
Mathilde	Dufour	833	2	2	Jules Renard
Mathilde	Dufour	833	2	3	Blaise Pascal
Léa	Dupond	832	1	1	Janson de Sailly
Léa	Dupond	832	1	2	Jules Renard

Jointure naturelle

Jointure naturelle

Dans l'exemple précédent sur le produit cartésien, beaucoup de lignes n'avaient aucun intérêt : celles telles que eleve.id_lycee $\neq$ lycee.id_lycee.

Il est plutôt naturel, lors du produit, d'identifier les colonnes qui portent le même nom : on parle de **jointure naturelle**.

Définition (jointure naturelle)

Pour $\mathcal{R}$ et $\mathcal{R}'$ deux relations de schémas $\mathcal{S}$ et $\mathcal{S}'$ avec $S \cap \mathcal{S}' = \mathcal{S}''$, on définit la **jointure naturelle** de $\mathcal{R}$ et $\mathcal{R}'$ comme la relation $\mathcal{R} \bowtie \mathcal{R}'$ de schéma $\mathcal{S} \cup \mathcal{S}'$ définie par :

$$\mathcal{R} \bowtie \mathcal{R}' = \{u \mid u[\mathcal{S}] \in \mathcal{R} \text{ et } u[\mathcal{S}'] \in \mathcal{R}'\}$$

Jointure naturelle

Cas particuliers

On a déjà vu deux cas particuliers de cette notion :

- Si $\mathcal{S} \cap \mathcal{S}' = \emptyset$, c'est le produit cartésien.
- Si $\mathcal{S} = \mathcal{S}'$, c'est l'intersection.

Jointure naturelle

Attention

Si les deux tables ont plusieurs attributs en commun, la **jointure naturelle** va forcer chacun de ces attributs à coïncider.

Ainsi, la jointure naturelle des tables `eleve` et `lycee` est vide, car on force à la fois `eleve.id_lycee = lycee.id_lycee` **et** `eleve.nom = lycee.nom`.

Jointure

Remarque

La **jointure naturelle** pouvant s'avérer piégieuse, elle n'est pas au programme : la seule est la jointure symétrique qu'on va définir maintenant.

D'un point de vue algébrique, cette jointure est une abréviation : elle consiste à faire une sélection dans le produit cartésien.

Définition (jointure)

Si $\mathcal{R}$ et $\mathcal{R}'$ sont deux relations de schémas $\mathcal{S}$ et $\mathcal{S}'$ avec $\mathcal{S} \cap \mathcal{S}' = \emptyset$, et $\mathcal{C}$ une condition booléenne sur $\mathcal{S} \cup \mathcal{S}'$, alors la **jointure** de $\mathcal{R}$ et $\mathcal{R}'$ selon $\mathcal{C}$ est la relation $\mathcal{R} \bowtie_{\mathcal{C}} \mathcal{R}'$ de schéma $\mathcal{S} \cup \mathcal{S}'$ définie par :

$$\mathcal{R} \bowtie_{\mathcal{C}} \mathcal{R}' = \sigma_{\mathcal{C}}(\mathcal{R} \times \mathcal{R}')$$

Jointure

Cas particuliers

- Si $\mathcal{C}$ est la condition triviale (toujours vraie), on retrouve le produit cartésien.
- La jointure naturelle ressemble à une jointure avec comme condition $\mathcal{R}.A = \mathcal{R}'.A$ pour tous les attributs $A \in S \cap \mathcal{S}'$. La différence est que les colonnes égales sont répétées.

Jointure

Exemple

Voici la jointure des tables `eleve` et `lycee` selon `id_lycee`.

Table eleve $\bowtie_{\texttt{eleve.id\_lycee}=\texttt{lycee.id\_lycee}}$ **lycee**					
prenom	nom	id_classe	id_lycee	id_lycee	lycee.nom
Mathilde	Dufour	833	2	2	Jules Renard
Léa	Dupond	832	1	1	Janson de Sailly
Paul	Dugommier	833	1	1	Janson de Sailly
Mathilde	Dugommier	834	3	3	Blaise Pascal
Clément	Durand	837	4	4	Carnot

Jointure symétrique

Jointure symétrique

On parle de **jointure symétrique** lorsque la condition de jointure est l'égalité de deux attributs (comme dans l'exemple précédent).

C'est le seul type de jointure au programme.

Tables multiples dans SQL

Produit cartésien

- La traduction en SQL du **produit cartésien** est très simple : il suffit d'énumérer les différentes tables dans la clause **FROM**.
- SQL considère les noms des colonnes de la table **T** sous la forme **T.nom**, il n'y a pas de noms identiques.
- Cependant, lors de l'affichage, le titre de la colonne ne sera que le nom de l'attribut et on peut croire que la contrainte d'ensemble pour les attributs n'est pas respectée.
- Lorsqu'on aura des noms d'attributs égaux, il faudra donc spécifier la table lors de l'écriture des requêtes pour lever les ambiguïtés, ce dont on peut se passer si les noms d'attributs sont distincts.
- Il est courant de procéder à un renommage de la table elle-même pour écourter l'écriture des requêtes.

Produit cartésien

```sql
SELECT * FROM eleve,classe,lycee ;
SELECT e.nom,e.prenom FROM eleve e, lycee L
WHERE e.id_lycee=L.id_lycee AND L.nom="Carnot" ;
SELECT e.nom,e.prenom FROM eleve e, lycee L, classe c WHERE e.id_lycee=L.id_lycee AND
c.id_classe=e.id_classe AND L.nom="Carnot" AND c.filiere="MPSI" AND c.numero="2" ;
```

Exemple

Voici trois exemples de requêtes avec des produits cartésiens.

1. La première fait simplement le produit des trois tables (ce qui donne une grosse table !).
2. La deuxième fait une sélection après un produit cartésien : en fait on fait d'abord une jointure sans le dire, pour sélectionner les élèves venant du lycée Carnot.
3. La troisième sélectionne également ceux qui sont en MPSI 2, avec une jointure supplémentaire, toujours sans le dire.

Jointure naturelle

SQL

Il suffit d'employer `NATURAL JOIN` pour effectuer une jointure naturelle entre deux tables.

Mais attention, la jointure naturelle va forcer **toutes** les colonnes portant le même nom à être égales.

Jointure naturelle

Attention

Attention : l'utilisation de **NATURAL JOIN** peut être piégeux !
Si on essaie de l'utiliser pour l'exemple précédent, on pourrait écrire la requête ci-dessous mais elle **ne fonctionne pas**.

```
1  SELECT eleve.nom,eleve.prenom FROM eleve NATURAL JOIN lycee NATURAL JOIN classe
2  WHERE lycee.nom="Carnot" AND filiere="MPSI" AND numero="2" ;
```

Exemple

Voici le piège : les tables **eleve** et **lycee** ont toutes les deux une colonne **nom**.
La jointure naturelle va alors forcer `eleve.nom = lycee.nom`, et ne chercher que les élèves du lycée Carnot qui s'appellent Carnot...

Jointure

SQL

En SQL, la jointure s'effectue avec le mot-clé **JOIN**.

On spécifie la condition de jointure avec le mot-clé **ON**.

↪ C'est la syntaxe que je vous conseille d'utiliser aux concours.

```
table1 JOIN table2 ON condition
```

Exemple

Voici une version de la requête précédente (qui fonctionne correctement cette fois).

```
SELECT nom,prenom FROM eleve e JOIN lycee L ON e.id_lycee=L.id_lycee
JOIN classe c ON e.id_classe=c.id_classe
WHERE L.nom="Carnot" AND c.filiere="MPSI" AND c.numero="2" ;
```

Jointure

Remarque

Il n'y a pas d'ordre dans les attributs en SQL.

Rien n'empêche de donner d'abord les tables que l'on va utiliser (séparées par **JOIN**) pour donner ensuite les conditions de jointures dans un seul **ON** (séparées par **AND**).

Jointure

En pratique

- Bien que cela soit possible il n'est pas recommandé de placer toutes les sélections dans la clause **ON** à la place de la clause **WHERE**.
- Typiquement pour une équijointure (jointure avec condition d'égalité entre deux attributs), on placera la condition d'égalité de deux colonnes dans la clause **ON**, et les autres conditions de sélection dans la clause **WHERE**.
- L'intérêt d'une jointure est de structurer plus clairement les requêtes, bien qu'on puisse s'en passer en faisant des produits cartésiens et en sélectionnant, comme on l'a fait précédemment.

Jointure externe à gauche

Lignes sans correspondance

Lorsqu'on fait une jointure entre deux tables T_1 et T_2, il se peut que certaines lignes de T_1 n'apparaissent jamais dans le résultat, car aucune ligne de T_2 ne lui correspond.

(et inversement, il se peut que certaines lignes de T_2 n'apparaissent jamais non plus dans le résultat).

Exemple

La table `classe` contient les classes de **MPSI1**, **PCSI1**, et **PCSI2**, mais la table `eleve` ne contient aucun élève de ces classes.

Jointure externe à gauche

Table classe		
id_classe	filiere	numero
831	MPSI	1
832	MPSI	2
833	MP2I	1
834	MPSI	3
835	PCSI	1
836	PCSI	2
837	PCSI	3

Table eleve			
prenom	nom	id_classe	id_lycee
Mathilde	Dufour	833	2
Léa	Dupond	832	1
Paul	Dugommier	833	1
Mathilde	Dugommier	834	3
Clément	Durand	837	4

```
1  SELECT filiere, numero, prenom, nom
2  FROM classe JOIN eleve
3  ON classe.id_classe = eleve.id_classe ;
```

filiere	numero	prenom	nom
MP2I	1	Mathilde	Dufour
MPSI	2	Léa	Dupond
MP2I	1	Paul	Dugommier
MPSI	3	Mathilde	Dugommier
PCSI	3	Clément	Durand

Jointure externe à gauche

Jointure externe à gauche

Il est possible de faire une jointure, et de garder **toutes** les lignes de la première table de la jointure : cela s'appelle la **jointure externe à gauche**.

Dans ce cas, si certaines lignes n'ont pas de correspondance, la valeur `NULL` sera utilisée pour remplir les colonnes de la deuxième table.

En SQL, il suffit d'utiliser le mot-clé `LEFT JOIN` au lieu de `JOIN`.

Jointure externe à gauche

Table classe

id_classe	filiere	numero
831	MPSI	1
832	MPSI	2
833	MP2I	1
834	MPSI	3
835	PCSI	1
836	PCSI	2
837	PCSI	3

Table eleve

prenom	nom	id_classe	id_lycee
Mathilde	Dufour	833	2
Léa	Dupond	832	1
Paul	Dugommier	833	1
Mathilde	Dugommier	834	3
Clément	Durand	837	4

```
1  SELECT filiere, numero, prenom, nom
2  FROM classe LEFT JOIN eleve
3  ON classe.id_classe = eleve.id_classe ;
```

filiere	numero	prenom	nom
MP2I	1	Mathilde	Dufour
MPSI	2	Léa	Dupond
MP2I	1	Paul	Dugommier
MPSI	3	Mathilde	Dugommier
PCSI	3	Clément	Durand
MPSI	1	**NULL**	**NULL**
PCSI	1	**NULL**	**NULL**
PCSI	2	**NULL**	**NULL**

Valeur NULL

NULL

Pour détecter quelles lignes n'ont eu aucune correspondance, on pourrait donc sélectionner les lignes contenant `NULL`.

`NULL` est une valeur particulière en SQL, et on ne peut pas tester si une case vaut `NULL` comme on le ferait d'habitude.

Attention

Dans l'exemple précédent :

- le test `WHERE nom = "NULL";` ne fonctionne pas, car on teste si l'élève s'appelle "NULL" ;
- le test `WHERE nom = NULL;` ne fonctionne pas, car on teste si la colonne `nom` contient la même valeur que la colonne `NULL` (colonne qui n'existe pas).

Valeur NULL

NULL

Pour tester si la colonne vaut **NULL**, il faut utiliser le test : `nom_colonne` **IS NULL**.

Il existe également le test `nom_colonne` **IS NOT NULL** pour tester le contraire.

Valeur NULL

Table classe		
id_classe	filiere	numero
831	MPSI	1
832	MPSI	2
833	MP2I	1
834	MPSI	3
835	PCSI	1
836	PCSI	2
837	PCSI	3

Table eleve			
prenom	nom	id_classe	id_lycee
Mathilde	Dufour	833	2
Léa	Dupond	832	1
Paul	Dugommier	833	1
Mathilde	Dugommier	834	3
Clément	Durand	837	4

```sql
SELECT filiere, numero
FROM classe LEFT JOIN eleve
ON classe.id_classe = eleve.id_classe
WHERE nom IS NULL;
```

filiere	numero
MPSI	1
PCSI	1
PCSI	2

Conclusion

Pour conclure

Structure générale d'une requête

L'ordre des mots-clés dans une requête de recherche dans une table SQL est toujours le même.

Voici à quoi ressemble une requête complète, avec jointures.

```
1  SELECT attributs
2  FROM table1 JOIN table2 JOIN ... ON conditions de jointure AND ...
3  WHERE conditions de sélection
4  GROUP BY attributs de regroupement
5  HAVING conditions de sélection après agrégation
6  ORDER BY quantités pour l'ordonnancement
7  LIMIT n OFFSET p
```