Bases de données

Tronc commun informatique

Vincent Puyhaubert

PC* Joffre

14 mars 2023

Table des matières

Principe

Les bases de données sont des outils qui regroupent des informations relatives à un domaine précis de manière très ordonnée (mais invisible pour l'utilisateur moyen) :

- gestion de stocks (magasins, bibliothèques)
- réservations (transports, spectacles)
- données personnelles (pronote)

Plusieurs personnes peuvent intervenir sur la base de données, avec des droits plus ou moins restreints :

- consultation simple : voir les horaires des trains, récupérer une note, lire une appréciation.
- ajout/suppression : réserver/annuler un billet, ajouter des notes de DS, ajouter/annuler un train.

Principe

Difficultés essentielles

- Mettre en place une méthode de stockage intelligente pour garantir l'efficacité des algorithmes de consultation.
- Gérer les problèmes de modifications :
  - Deux utilisateurs se connectent en même temps pour réserver un train. Comment garantir de ne pas leur attribuer la même place ?
  - On supprime un train. Comment anticiper et gérer toutes les répercussions (remboursement, garantie de la poursuite du trajet en cas de correspondance).

Exemple d'implémentation naïve (Python)

On cherche à stocker informatiquement l'ensemble des élèves du lycée Joffre. Chaque élève dispose des caractéristiques suivantes :

- nom, prénom
- filière, classe
- lycée d'origine

Voici un exemple d'extrait de la base de donnée

prénom	nom	filière	classe	lycée
Jean	Bonnot	PCSI	3	Nevers
Bernard	Tichaud	PC	10	Jules Guesde
Robert	Lingot	PC	10	Jean Monnet
Elodie	Nosaure	PSI	1	Jules Guesde

On pourrait bien entendu prévoir d'ajouter bien plus de caractéristiques (options suivies, notes, nom des enseignants).

Exemple d'implémentation naïve (Python)

Divers choix naïfs possibles :

- une liste de tuples (ou de listes).
  - Très facile de rajouter un élève
  - Tout le reste des opérations est couteux : chercher un élève, chercher tous les élèves d'une classe, chercher tous les élèves qui suivent une certaine option.
- Un dictionnaire. QUID de la clé ?
  - Une entrée par élève (nécessite une clé unique par élève). On ajoute ou cherche un élève en temps constant. Les autres opérations restent couteuses.
  - Un entrée par classe. On ajoute ou cherche un élève en temps proportionnel à la taille de sa classe. On récupère l'ensemble d'une classe en temps constant. D'autres opérations restent couteuses (trouver tous élèves en provenance de Jules Guesde par exemple)..

Tables multiples

On reprend l'exemple précédent en se focalisant sur les enseignants.

prénom	nom	filière	classe	maths	physique
Jean	Bonnot	PCSI	3	Pechaud	Ponsolle
Bernard	Tichaud	PC	10	Puyhaubert	Ringot
Robert	Lingot	PC	10	Puyhaubert	Ringot
Elodie	Nosaure	PSI	1	Marino	Berger

- Redondance d'information.
- Nécessité en cas de modification d'une seule information (nom du professeur associé à un cours) d'avoir à le faire pour de nombreuses entrées.

Tables multiples

Il est plus judicieux d'avoir plusieurs tables.

prénom	nom	noclasse
Jean	Bonnot	231
Bernard	Tichaud	450
Robert	Lingot	450
Elodie	Nosaure	189

noclasse	filière	classe	maths	physique
231	PCSI	3	Pechaud	Ponsolle
450	PC	10	Puyhaubert	Ringot
189	PSI	1	Marino	Berger

à condition de garantir des méthodes efficaces pour croiser les données de plusieurs tables !

Vocabulaire

- **attribut :** correspond au nom d'une colonne.

prénom	nom	noclasse
...	...	...
...	...	...

  - correspond à une donnée élémentaire pour chaque ligne de la table.
  - nécessairement deux à deux distincts
  - en général, donnés sans ordre particulier : on parle par exemple ci-dessus de l'attribut nom, et pas du deuxième attribut.

- **domaine :** pour un attribut x, type de données apparaissant dans la colonne correspondante.
  - le programme précise s'en tenir à des types élémentaires : entiers, flottants, chaînes de caractères.
  - les dates sont représentés à priori par des chaînes de caractères.
  - La constante NULL est utilisée lorsque la valeur d'un attribut n'est pas renseignée pour une certaine ligne. Explicitement hors-programme.

Vocabulaire

- **enregistrement :** correspond à une ligne de la base de donnée.

prénom	nom	filière	classe	maths	physique
Jean	Bonnot	PCSI	3	Pechaud	Ponsolle
Bernard	Tichaud	PC	10	Puyhaubert	Ringot
Robert	Lingot	PC	10	Puyhaubert	Ringot
Elodie	Nosaure	PSI	1	Marino	Berger

- **table ou relation :** ensemble d'enregistrements
  - Dans une même table, les enregistrements peuvent avoir quelques attributs identiques, mais sont deux à deux distincts.
- **schéma de tables / schéma relationnel :** liste des couples attribut/domaine correspondants à une table.
  - Dans la plupart des exercices de BDD (à l'écrit uniquement jusqu'à présent), les bases de données sont décrites uniquement par leur schéma relationnel.

Vocabulaire

- **clé primaire :** attribut, ou sous-ensemble d'attributs qui identifient chaque enregistrement de manière unique. Si deux enregistrements ont la même clé primaire, alors ils sont égaux.
  - Le cas le plus fréquent est celui d'une clé réduite à un seul attribut
  - Dans tous les cas, l'ensemble des attributs d'une table forme une clé primaire.
  - Dans le schéma relationnel, on souligne la ou les clés primaires

noclasse	filière	classe	maths	physique
231	PCSI	3	Pechaud	Ponsolle
450	PC	10	Puyhaubert	Ringot
189	PSI	1	Marino	Berger

  - Dans la table des élèves, l'attribut noclasse n'est pas une clé primaire. On parle alors de **clé étrangère** lorsqu'un attribut d'une table coïncide avec la clé primaire d'une autre table.

Programme officiel

- Le programme se limite à la notion de requête, c'est-à-dire à interroger une BDD pour en retirer des informations.
  - Aucune connaissance exigible sur la façon dont sont stockées les données, ou les algorithmes de recherche/modification de la base.
  - Aucune connaissance sur l'algèbre relationnelle et le calcul relationnel (théorie mathématique fondée en 1970, proche de la théorie des ensembles, qui définit rigoureusement la manipulation des relations).
  - Les connaissances relatives à la création ou la modification d'une BDD sont également hors-programme.
  - Aucun logiciel spécifique n'est au programme pour l'utilisation des BDDs (ils sont tous à peu près identiques).
- Les sujets de concours se contentent généralement de 3 à 4 questions, de difficultés progressive, demandant de formuler des requêtes sur une ou plusieurs BDD.

Pour s'entraîner

- Télécharger un système de gestion de bases de données. Par exemple,

  https ://sqlitebrowser.org

  installable sur toutes plates formes (Windows, MacOs, Linux).
- La suite du cours et le premier TD/TP utilisera la base de données *geographie*. Le fichier est en ligne.

Selection, projection, renommage

- La syntaxe générale d'une requête est de la forme

  SELECT exp **FROM** bdd ;

  où bdd est une table et exp est un ensemble d'attributs de bdd ou de nouveaux attributs calculés à partir des données existantes.

  - Les majuscules sont usuelles, mais facultatives.
  - Le résultat d'une sélection est une nouvelle table (parfois réduite à une seule entrée).

- Elle s'accompagne généralement

  - d'une projection, qui consiste à ne pas retenir l'ensemble des attributs de la table mais uniquement une partie. Dans le cas contraire, on utilise * à la place de exp pour récupérer l'ensemble des attributs.
  - d'une sélection, qui consiste à ne retenir que les enregistrements qui satisfont une certaine condition. La syntaxe devient alors

    SELECT exp **FROM** bdd **WHERE** condition ;

Selection, projection, renommage

- Contrairement à la table initiale, le résultat d'une requête peut contenir des doublons. On les supprime avec le mot clé **DISTINCT**.

  SELECT DISTINCT exp **FROM** bdd **WHERE** condition ;

- Les opérateurs au programme (pour la rédaction des conditions ou le calcul de nouveaux attributs) sont exclusivement les suivants :
  - Opérateurs arithmétiques : +, -, *, /
  - Opérateurs de comparaison : =, <, <=, <>, >, >=
  - Opérateurs booléens : **AND, OR, NOT**
- Sont notamment exclus du programme les éléments suivants (mentionnés à titre culturel) :
  - **IN** qui permet de vérifier si un attribut appartient à un ensemble (qui peut être le résultat d'une requête avec une unique colonne)
  - **LIKE** qui permet de vérifier si des attributs de type chaîne de caractère suivent un certain modèle.

Selection, projection, renommage

- Le renommage sert essentiellement à écourter les requêtes. On peut renommer aussi bien une table qu'un attribut.
  - Renommage d'attribut

    SELECT attribut **AS** nouveau_nom **FROM** bdd ;

    ou simplement, le **AS** étant facultatif,

    SELECT attribut nouveau_nom **FROM** bdd ;
  - Renommage de table

    SELECT attribut **FROM** bdd **AS** nouveau_nom ;

    ou à nouveau

    SELECT attribut **FROM** bdd nouveau_nom ;

Selection, projection, renommage

Exemple : (base de donnée *géographie*)

- Ecrire une requête donnant les noms de communes françaises de plus de 100 000 habitants.
- Ecrire une requête donnant les densités de population des communes de l'Hérault.

Agrégats

On veut maintenant répondre à des questions de la nature suivante :

- Combien y a-t-il de communes dans chaque département ?
- Quel est le département de population maximale ?
- Quelles régions comportent le plus grand nombre de département ?
- Quelle est la surface moyenne des départements de l'Hérault ?

On a besoin de fonctions qui réalisent des calculs sur des sous-ensembles de lignes/enregistrements. Ces fonctions s'appelent des fonctions d'agrégation. Celles du programme officiel sont les suivantes :

- **MIN, MAX** : déterminent le minimum et le maximum d'un attribut.
- **SUM, AVG** déterminent la somme et la moyenne d'un attribut.
- **COUNT** compte le nombre de lignes.

Agrégats

- Agrégation sans regroupement : f est une fonction d'agrégation, a est un attribut.

  SELECT f(a) **FROM** table ;

  calcule une table avec une seule ligne et une seule colonne. La valeur de f est calculée à l'aide de l'attribut a sur l'ensemble des lignes.

- Agrégation avec regroupement :

  SELECT a1, a2, . . . , ap, f(a) **FROM** table
  GROUP BY a1, a2, . . . , ap ;

  Cette requête regroupe les lignes qui partagent des attributs $a_1, \ldots, a_p$ égaux, puis calcule f sur l'ensemble des attributs a de chaque regroupement.

Agrégats

On peut bien entendu composer une sélection et une agrégation.

- Selection en amont **WHERE** :

  SELECT a1, a2, . . . , ap, f(a) **FROM** table
  WHERE condition
  GROUP BY a1, a2, . . . , ap ;

  La requête commence par éliminer les enregistrements qui ne vérifient pas la condition, puis applique la fonction d'agrégation au reste.

- Selection en aval **HAVING** :

  SELECT a1, a2, . . . , ap, f(a) **FROM** table
  GROUP BY a1, a2, . . . , ap
  HAVING condition

  La requête élimine **après** le calcul de la fonction d'agrégation les (nouveaux) enregistrements qui ne vérifient pas la condition.

Tri, limites

Pour organiser la présentation des résultats, on peut utiliser

- **ORDER BY** exp **DESC** / **ASC**
  Permet de trier les enregistrements de la séléction selon la valeur de exp, qui peut être un attribut ou plus généralement le résultat d'un calcul sur des attributs.
- **LIMIT** n
  Permet de limiter le nombre de résultats. En l'ajoutant à une requête, on se limite aux n premiers enregistrements.
- **OFFSET** m
  Permet d'ignorer une partie des résultats. En l'ajoutant à une requête, on élimine les m premiers résultats.

LIMIT et **OFFSET** peuvent s'utiliser sans **ORDER BY**, mais c'est rarement pertinent.

Pour résumer (requêtes sur une seule table) :

Attention à l'ordre !

SELECT attributs
FROM table
WHERE condition
GROUP BY attributs
HAVING condition
ORDER BY quantité
LIMIT entier
OFFSET entier

- Pas de fonction d'agrégation sans **GROUP BY**.
- Pas de **HAVING** sans **GROUP BY**.
- A priori, pas de **LIMIT** ou **OFFSET** sans **ORDER BY**.

Composition de requêtes

Une requête produit une table, qu'il est ensuite possible de réutiliser dans une autre requête.

- *a* est un attribut. On cherche les enregistrements pour lesquels *a* est maximal.

```
SELECT * FROM table
WHERE a = (SELECT MAX(a) FROM table)
```

  Rappel : une table à une seule ligne et une seule colonne est identifiée à cette valeur.

- **HAVING** est équivalent à la syntaxe suivante utilisant **WHERE** et une sous-requête :

```
SELECT * FROM
  (SELECT a1, a2, ..., ap, f(a) FROM table
  GROUP BY a1, a2, ..., ap)
WHERE condition
```

Modèles entités-associations

Intérêt principal des bases de données : découper les informations sur **plusieurs** tables.

- évite les redondances d'information (deux élèves d'une même classe partagent la majorité des enseignants).
- garantit les contraintes d'intégrité (plus facile lorsqu'on ajoute un élève à une Bdd de garantir que sa classe existe réelleement).

Conception d'une base de donnée :

- Il y a à priori plusieurs manières de séparer intelligemment les données.
- Le modèle doit être bien pensé avant la création (impossible à modifier par la suite, sauf à tout reprendre).
- Les principes généraux qui guident la conception sont hors-programme (c'est très volumineux). Quelques sources :
  - https ://laurent-audibert. developpez.com/Cours-BD/
  - manga-guide des bases de données

Modèles entités-associations

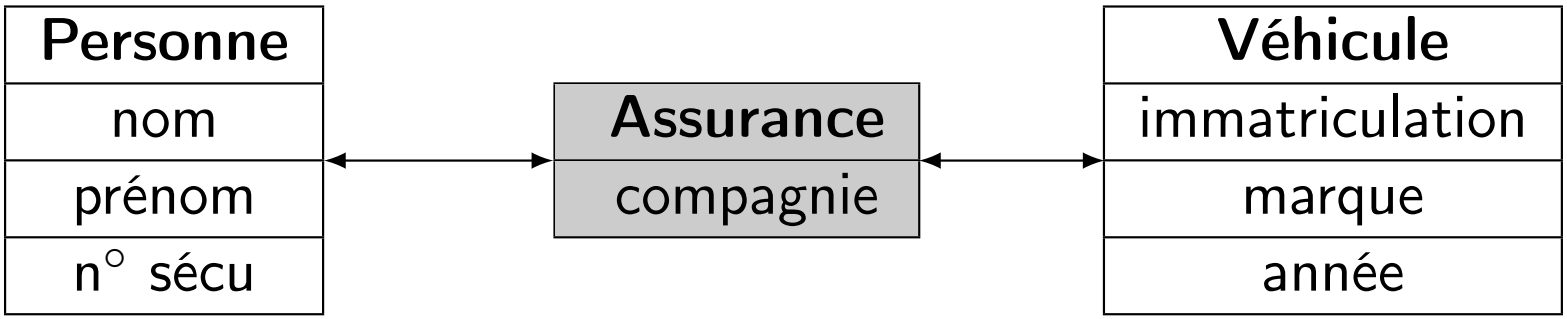
- Une **entité** est un objet, une chose, concrète ou abstraite, qui peut être reconnue distinctement et qui est caractérisée par son unicité.
- Un type-entité est un ensemble d'entité qui possèdent des caractéristiques communes.
- Exemples :
  - Une personne, caractérisée par nom, prénom, numéro de sécurité sociale.
  - Un véhicule, caractérisé par son immatriculation, sa marque, année de fabrication, kilométrage, etc ...
- Un type entité est représenté en colonne avec ses caractéristiques comme suit :

Personne
nom
prénom
n° sécu

Véhicule
immatriculation
marque
année

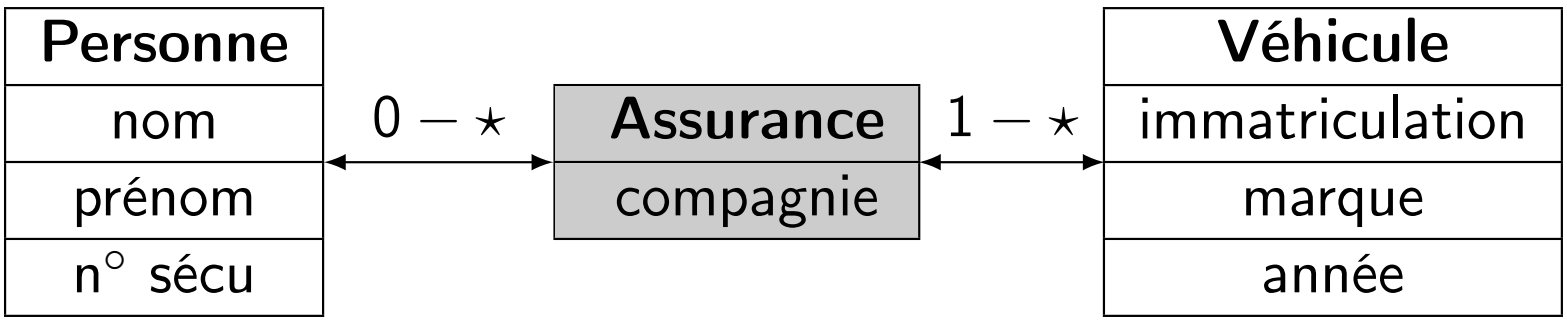
Modèles entités-associations

- Une association consiste simple en un lien entre deux ou plusieurs entités.
- Un type-association est à nouveau un ensemble d'associations entre les mêmes types-entité.
- Exemple :
  - Vincent Puyhaubert est assurée pour conduire sa Saxo.
  - Karim Benzema est assuré pour conduire ses 2 Buggatis.
- Une association peut également posséder (ou pas) des caractéristiques (nom de la compagnie d'assurance).
- Graphiquement, on note



Modèles entités-associations

- Certaines entités peuvent apparaître plusieurs fois dans une association :
  - Une personne peut être assuré pour conduire plusieurs véhicules.
  - Une véhicule peut être assuré pour plusieurs conducteurs.
- La cardinalité d'une association précise le nombre de fois minimal et maximal qu'une entité peut apparaitre dans une association.
- En pratique,
  - le nombre minimal vaut 0 ou 1
  - le nombre maximal vaut 1 ou n'est pas majoré (ce que l'on note $\star$).
- Exemple :



Modèles entités-associations

- Lorsqu'une association comporte deux branches de cardinalité maximales $\star$, cela traduit la nécessité d'introduire une entité intermédiaire et de séparer l'association en deux.
- Exemple :

Personne
nom
prénom
<u>n° sécu</u>

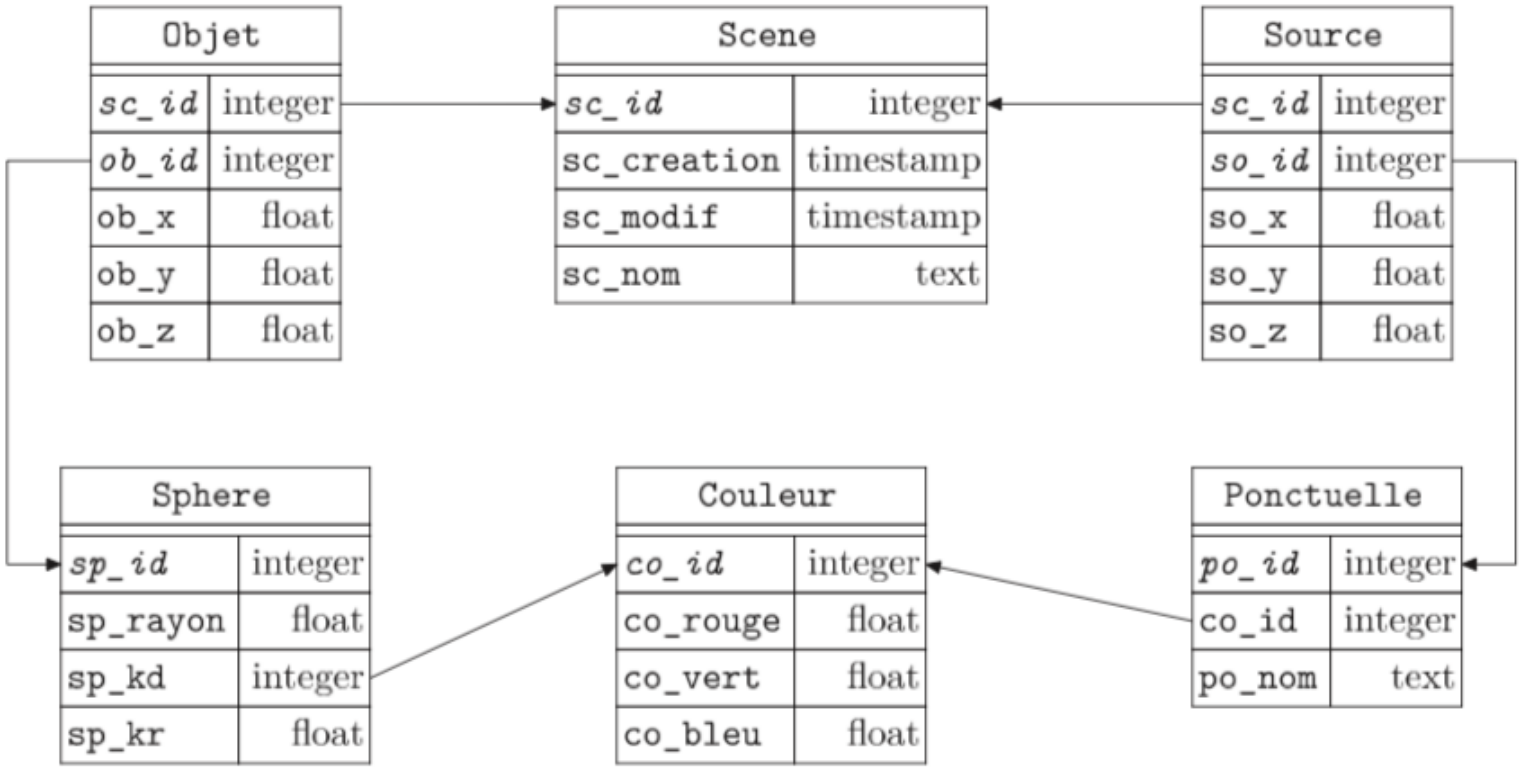
Contrat
compagnie
<u>n° contrat</u>
n° sécu
immatriculation

Véhicule
<u>immatriculation</u>
marque
année

- On a maintenant deux associations : entre les personnes et les contrats, ainsi qu'entre les véhicules et les contrats. Plus d'association entre personnes et véhicules.
- Les attributs n° sécu et immatriculation de l'entité contrat sont des clés étrangères : elles font références à des clés primaires des entités personnes et véhicules.

Modèles entités-associations

Exemple de modèle complet (Centrale 2021) :



Jointures

Objectif : croiser les informations de deux tables.

- Option naïve : produit cartésien.

SELECT * FROM table1, table2

- Sur l'exemple initial des deux tables élèves et classes de Joffre, on obtient

prénom	nom	noclasse	noclasse	filière	classe	maths	physique
Jean	Bonnot	231	231	PCSI	3	Pechaud	Ponsolle
Jean	Bonnot	231	450	PC	10	Puyhaubert	Ringot
Jean	Bonnot	231	189	PSI	1	Marino	Berger
Bernard	Tichaud	450	231	PCSI	3	Pechaud	Ponsolle
Bernard	Tichaud	450	450	PC	10	Puyhaubert	Ringot
Bernard	Tichaud	450	189	PSI	1	Marino	Berger
Robert	Lingot	450	231	PCSI	3	Pechaud	Ponsolle
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

  - Crée une table contenant l'ensemble des enregistrements (a, b) avec a (resp. b) enregistrement de la première (resp. deuxième) table.
  - Une majeure partie des lignes n'a aucune pertinence.

Jointures

Objectif : croiser les informations de deux tables.

- Option efficace : jointure

  SELECT * **FROM** table1 **JOIN** table2 **ON** condition

- La plupart du temps, la condition sera une condition d'égalité entre la clé primaire d'une table, et une clé externe d'une autre (équijointure).
- On utilise la syntaxe table.attribut en cas d'ambiguité (attributs du même nom dans deux tables différentes).
- Sur l'exemple initial des deux tables élèves et classes de Joffre, la requête

  SELECT prénom,nom,noclasse,filière,. . ., physique
  FROM eleves **JOIN** classes **ON** eleves.noclasse = classes.noclasse

  donne le résultat pertinent (en évitant la redondance de noclasse) :

prénom	nom	noclasse	filière	classe	maths	physique
Jean	Bonnot	231	PCSI	3	Pechaud	Ponsolle
Bernard	Tichaud	450	PC	10	Puyhaubert	Ringot
Robert	Lingot	450	PC	10	Puyhaubert	Ringot
Elodie	Nosaure	189	PSI	1	Marino	Berger

Jointures

- On peut effectuer plusieurs jointures simultanées

```
SELECT *
FROM table1 JOIN table2 JOIN table3
ON condition
```

- On peut également joindre une table à elle-même (auto-jointure), mais un renommage est obligatoire.

```
SELECT *
FROM table AS t1 JOIN table AS t2
ON t1.attribut1 ... t2.attribut2
```

Pour résumer (requêtes sur plusieurs tables) :

Attention à l'ordre !

SELECT attributs
FROM table1 **JOIN** table2 **JOIN** ...
ON conditions de jointure
WHERE conditions de selection
GROUP BY attributs de regroupement
HAVING conditions de selection après agrégation
ORDER BY quantité
LIMIT entier
OFFSET entier

Opérations ensemblistes

Les opérateurs suivants permet d'appliquer les opérations ensemblistes usuelles sur plusieurs tables ayant le **même** schéma relationnel.

- **UNION** renvoie l'union (table1 $\cup$ table2), soit tous les enregistrement qui apparaîssent dans l'une ou l'autre des tables (sans créer d'occurence multiple).
- **INTERSECT** renvoie l'intersection (table1 $\cap$ table2), soit tous les enregistrement qui apparaîssent dans les deux tables
- **EXCEPT** renvoie (table1 $\setminus$ table2), soit les enregistrements de table1 qui n'apparaissent pas dans table2.

Opérations ensemblistes

Exemple de syntaxe :

SELECT * **FROM** table1 **UNION** table2

SELECT attributs **FROM** table1
UNION
SELECT attributs **FROM** table2

On peut souvent s'en tirer en utilisant **AND**, **OR** et **NOT** et des conditions de selection.