Informatique pour tous - MP* - PC*

Partie 1 :
Bases de données

Florent Pompigne
pompigne@crans.org

Lycée Buffon

année 2022/2023

Exemple introductif

On souhaite représenter et manipuler informatiquement les élèves de CPGE du lycée Buffon. Chaque élève possède des attributs : son nom, son prénom, sa classe, sa date de naissance, sa moyenne en informatique...

Exemple introductif

On souhaite représenter et manipuler informatiquement les élèves de CPGE du lycée Buffon. Chaque élève possède des attributs : son nom, son prénom, sa classe, sa date de naissance, sa moyenne en informatique...

On voudrait que chaque attribut joue un rôle symétrique, afin qu'il soit aussi simple et rapide de trier les élèves par date de naissance que par classe. On voudrait également pouvoir combiner les informations relatives aux étudiants avec des informations relatives aux classes, aux salles, aux professeurs...

Exemple introductif

On souhaite représenter et manipuler informatiquement les élèves de CPGE du lycée Buffon. Chaque élève possède des attributs : son nom, son prénom, sa classe, sa date de naissance, sa moyenne en informatique...

On voudrait que chaque attribut joue un rôle symétrique, afin qu'il soit aussi simple et rapide de trier les élèves par date de naissance que par classe. On voudrait également pouvoir combiner les informations relatives aux étudiants avec des informations relatives aux classes, aux salles, aux professeurs...

Le langage SQL est un langage dédié à la manipulation de telles bases de données.

Plan

Vocabulaire

Une base de donnée est constituée de **tables** (ou relations). Par exemple, la base du lycée peut contenir une table des élèves, une table des salles, une table des professeurs...

Vocabulaire

Une base de donnée est constituée de **tables** (ou relations). Par exemple, la base du lycée peut contenir une table des élèves, une table des salles, une table des professeurs...

Les lignes d'une table sont aussi appelées **enregistrements**. Chaque ligne correspond à un objet enregistré dans la table. Par exemple, une ligne de la table des élèves sera constituée des informations relatives à un élève précis.

Vocabulaire

Une base de donnée est constituée de **tables** (ou relations).
Par exemple, la base du lycée peut contenir une table des élèves, une table des salles, une table des professeurs...

Les lignes d'une table sont aussi appelées **enregistrements**.
Chaque ligne correspond à un objet enregistré dans la table.
Par exemple, une ligne de la table des élèves sera constituée des informations relatives à un élève précis.

Les colonnes d'une table sont aussi appelées **attributs**. Chaque attribut a un nom et est associé à un domaine, précisé par un type de données.
Par exemple, la table des élèves peut contenir l'attribut `prénom`, ayant pour domaine les chaînes de caractères, avec le type `VARCHAR`.

Vocabulaire

Une base de donnée est constituée de **tables** (ou relations).
Par exemple, la base du lycée peut contenir une table des élèves, une table des salles, une table des professeurs...

Les lignes d'une table sont aussi appelées **enregistrements**.
Chaque ligne correspond à un objet enregistré dans la table.
Par exemple, une ligne de la table des élèves sera constituée des informations relatives à un élève précis.

Les colonnes d'une table sont aussi appelées **attributs**. Chaque attribut a un nom et est associé à un domaine, précisé par un type de données.
Par exemple, la table des élèves peut contenir l'attribut `prénom`, ayant pour domaine les chaînes de caractères, avec le type `VARCHAR`.

le **schéma** d'une table est la liste de ses attributs et de leurs types.

Clés

La **clé primaire** d'une table est un ensemble fixé d'attributs de la table donnant la garantie que deux lignes distinctes ne peuvent jamais avoir les mêmes valeurs sur cet ensemble d'attributs. Par exemple, le numéro d'étudiant forme une clé primaire pour la table des élèves.

Clés

La **clé primaire** d'une table est un ensemble fixé d'attributs de la table donnant la garantie que deux lignes distinctes ne peuvent jamais avoir les mêmes valeurs sur cet ensemble d'attributs.
Par exemple, le numéro d'étudiant forme une clé primaire pour la table des élèves.

Une **clé étrangère** est un attribut d'une table dont les valeurs font référence à un attribut qui est la clé primaire d'une autre table.
Par exemple, la table des élèves peut avoir une clé étrangère faisant référence au nom de la classe à laquelle appartient chaque élève.

Plan

Clauses

Les requêtes SQL se décomposent en **clauses**.

Clauses

Les requêtes SQL se décomposent en **clauses**.

Une requête de recherche commence **toujours** par une clause `SELECT` puis `FROM`, sur le modèle :

`SELECT` $a_1, a_2, \ldots, a_n$ `FROM` R

où R est une table, et $a_1, \ldots a_n$ sont des attributs de cette table. Seules les colonnes des attributs demandés sont alors affichées (on parle de **projection**).

Clauses

Les requêtes SQL se décomposent en **clauses**.

Une requête de recherche commence **toujours** par une clause `SELECT` puis `FROM`, sur le modèle :

```
SELECT a1, a2, ..., an FROM R
```

où R est une table, et $a_1, \ldots a_n$ sont des attributs de cette table. Seules les colonnes des attributs demandés sont alors affichées (on parle de **projection**).

Lorsqu'on veut afficher toutes les colonnes, on écrit

```
SELECT * FROM R
```

Sélection

Il est possible (mais pas obligatoire) d'ajouter ensuite une clause WHERE suivie d'une condition, pour sélectionner les lignes à afficher :

```
SELECT nom, prenom
FROM eleve
WHERE classe = "PCSI"
```

On parle de **sélection**.

Sélection

Il est possible (mais pas obligatoire) d'ajouter ensuite une clause WHERE suivie d'une condition, pour sélectionner les lignes à afficher :

```
SELECT nom, prenom
FROM eleve
WHERE classe = "PCSI"
```

On parle de **sélection**.

Notons que dans cette condition, `classe` ne prend pas de guillemets, car c'est le nom d'un attribut, tandis que `PCSI` est entouré de guillemets, puisque c'est une valeur spécifique de cet attribut, qui prend ses valeurs dans le domaine des chaînes de caractères.

Calculs et renommage

Un SELECT ne permet pas seulement de réaliser une projection, mais peut contenir des calculs. Par exemple :

```
SELECT nom, prenom,
(moyenneMaths+moyennePhysique) / 2.
FROM eleve
WHERE classe = "PCSI"
```

Calculs et renommage

Un SELECT ne permet pas seulement de réaliser une projection, mais peut contenir des calculs. Par exemple :

```
SELECT nom, prenom,
(moyenneMaths+moyennePhysique) / 2.
FROM eleve
WHERE classe = "PCSI"
```

On peut renommer un attribut en le faisant suivre par AS *alias* dans le SELECT :

```
SELECT nom, prenom,
(moyenneMaths + moyennePhysique) / 2.
AS moyenneScience
FROM eleve
WHERE classe = "PCSI"
```

Opérations ensemblistes

On peut utiliser le mot-clé `UNION` pour combiner deux requêtes de recherche pour obtenir une table formée de l'union des lignes des deux résultats :

```
SELECT *
FROM A
UNION
SELECT *
FROM B
```

Opérations ensemblistes

On peut utiliser le mot-clé `UNION` pour combiner deux requêtes de recherche pour obtenir une table formée de l'union des lignes des deux résultats :

```
SELECT *
FROM A
UNION
SELECT *
FROM B
```

De la même façon, le mot-clé `INTERSECTION` (ou `INTERSECT` dans certaines implémentations) permet d'obtenir l'intersection des lignes (celles qui sont communes aux deux résultats), et `EXCEPT` permet d'obtenir la différence ensembliste (les lignes dans le premier résultat mais pas dans le second).
Il est nécessaire que les deux tables combinées aient les mêmes colonnes.

Plan

Jointures

La **jointure** de deux tables est une opération permettant de combiner les informations qu'elles contiennent lors d'une requête. Par défaut, chaque ligne de la première table est mise en relation avec chaque ligne de la seconde (comme un produit cartésien), il faut donc ajouter une condition pour ne garder que les alignements pertinents. On utilise en général une condition d'égalité relative à une clé étrangère.
Par exemple, si la table `eleve` a une clé étrangère `filiere` vers l'attribut `filiere` de la table `classe` :

```
SELECT *
FROM eleve JOIN classe
ON eleve.filiere = classe.filiere
```

Jointures

La **jointure** de deux tables est une opération permettant de combiner les informations qu'elles contiennent lors d'une requête. Par défaut, chaque ligne de la première table est mise en relation avec chaque ligne de la seconde (comme un produit cartésien), il faut donc ajouter une condition pour ne garder que les alignements pertinents. On utilise en général une condition d'égalité relative à une clé étrangère.
Par exemple, si la table `eleve` a une clé étrangère `filiere` vers l'attribut `filiere` de la table `classe` :

```
SELECT *
FROM eleve JOIN classe
ON eleve.filiere = classe.filiere
```

Contrairement aux opérations ensemblistes, ce sont ici les colonnes qui sont combinées, et non les lignes. Comme illustré dans l'exemple, il peut être nécessaire de préfixer le nom d'un attribut par la table dont il vient en cas d'ambigüité.

Jointures

les mots-clés `JOIN` et `ON` font partie de la clause `FROM`, qui peut éventuellement être suivie par une clause `WHERE`. On peut appliquer une projection en remplaçant `*` par la liste d'attributs désirée.

Jointures

les mots-clés `JOIN` et `ON` font partie de la clause `FROM`, qui peut éventuellement être suivie par une clause `WHERE`. On peut appliquer une projection en remplaçant `*` par la liste d'attributs désirée.

Exemple :

```
SELECT filiere
FROM classe JOIN salle
ON numeroSalle = numéro
WHERE effectif > capacite
```

où :

- classe a pour schéma (filiere, effectif, numeroSalle)
- salle a pour schéma (numero, capacite)

Architecture

Les jointures exploitent les associations existant entre les entités d'une table A avec celles d'une autre table B. On peut représenter graphiquement de telles associations entre tables, en précisant leur cardinalité :

Architecture

Les jointures exploitent les associations existant entre les entités d'une table A avec celles d'une autre table B. On peut représenter graphiquement de telles associations entre tables, en précisant leur cardinalité :

- une association 1-1 associe à chaque élément d'une table exactement un élément de l'autre table.

Architecture

Les jointures exploitent les associations existant entre les entités d'une table A avec celles d'une autre table B. On peut représenter graphiquement de telles associations entre tables, en précisant leur cardinalité :

- une association 1-1 associe à chaque élément d'une table exactement un élément de l'autre table.
- Dans une association 1-*, chaque élément de B est associé à exactement un élément de A, mais un élément de A est associé à un nombre quelconque d'éléments de B. Cela signifie que la clé étrangère est dans B.

Architecture

Les jointures exploitent les associations existant entre les entités d'une table A avec celles d'une autre table B. On peut représenter graphiquement de telles associations entre tables, en précisant leur cardinalité :

- une association 1-1 associe à chaque élément d'une table exactement un élément de l'autre table.
- Dans une association 1-*, chaque élément de B est associé à exactement un élément de A, mais un élément de A est associé à un nombre quelconque d'éléments de B. Cela signifie que la clé étrangère est dans B.
- Dans une association *-1, chaque élément de A est associé à exactement un élément de B, mais un élément de B est associé à un nombre quelconque d'éléments de A. Cela signifie que la clé étrangère est dans A.

Architecture

Les jointures exploitent les associations existant entre les entités d'une table A avec celles d'une autre table B. On peut représenter graphiquement de telles associations entre tables, en précisant leur cardinalité :

- une association 1-1 associe à chaque élément d'une table exactement un élément de l'autre table.
- Dans une association 1-*, chaque élément de B est associé à exactement un élément de A, mais un élément de A est associé à un nombre quelconque d'éléments de B. Cela signifie que la clé étrangère est dans B.
- Dans une association *-1, chaque élément de A est associé à exactement un élément de B, mais un élément de B est associé à un nombre quelconque d'éléments de A. Cela signifie que la clé étrangère est dans A.
- une association *-* associe à chaque élément d'une table un nombre quelconque d'éléments de l'autre table.

Séparation d'une association *-*

Une association *-* n'est pas représentable efficacement en utilisant une clé étrangère. Une solution à ce problème est d'introduire une troisième table correspondant à cette association, liée aux tables initiale par des clés étrangères, et donc des associations 1-*.

Séparation d'une association *-*

Une association *-* n'est pas représentable efficacement en utilisant une clé étrangère. Une solution à ce problème est d'introduire une troisième table correspondant à cette association, liée aux tables initiale par des clés étrangères, et donc des associations 1-*.

Par exemple, si on considère des langues et des pays, l'association « est parlé dans » est de cardinalité *-*, puisqu'on peut parler plusieurs langues dans un pays, et qu'on peut parler une même langue dans plusieurs pays. On introduit donc une troisième table `lien_pays_langue` dont les colonnes, `code_pays` et `code_langue`, sont des clés étrangères vers les tables initiales.

Plan

Agrégats

Á la suite des clauses précédentes, on peut écrire une clause `GROUP BY` *nom d'attribut* pour regrouper les lignes en paquets, appelés agrégats, chaque agrégat correspondant à une valeur de l'attribut choisi.

Agrégats

Á la suite des clauses précédentes, on peut écrire une clause `GROUP BY` *nom d'attribut* pour regrouper les lignes en paquets, appelés agrégats, chaque agrégat correspondant à une valeur de l'attribut choisi.

Les lignes de la table renvoyée correspondent alors aux agrégats de lignes de la table de départ.

Agrégats

Á la suite des clauses précédentes, on peut écrire une clause `GROUP BY` *nom d'attribut* pour regrouper les lignes en paquets, appelés agrégats, chaque agrégat correspondant à une valeur de l'attribut choisi.

Les lignes de la table renvoyée correspondent alors aux agrégats de lignes de la table de départ.

En particulier, il faut donc que les attributs apparaissant dans le SELECT soient définis sans ambigüité pour chaque agrégat.

Fonction d'agrégation

Certaines fonctions s'appliquent sur les attributs d'agrégats. Les exemples notables sont :

Fonction d'agrégation

Certaines fonctions s'appliquent sur les attributs d'agrégats. Les exemples notables sont :

- `MIN` : renvoie la valeur minimale de l'attribut pour chaque agrégat ;

Fonction d'agrégation

Certaines fonctions s'appliquent sur les attributs d'agrégats. Les exemples notables sont :

- `MIN` : renvoie la valeur minimale de l'attribut pour chaque agrégat ;
- `MAX` : renvoie la valeur maximale de l'attribut pour chaque agrégat ;

Fonction d'agrégation

Certaines fonctions s'appliquent sur les attributs d'agrégats. Les exemples notables sont :

- `MIN` : renvoie la valeur minimale de l'attribut pour chaque agrégat ;
- `MAX` : renvoie la valeur maximale de l'attribut pour chaque agrégat ;
- `COUNT` : renvoie le cardinal de chaque agrégat avec `COUNT(*)`, ou le nombre de valeurs distinctes de l'attribut *x* de chaque agrégat avec `COUNT(distinct x)` ;

Fonction d'agrégation

Certaines fonctions s'appliquent sur les attributs d'agrégats. Les exemples notables sont :

- `MIN` : renvoie la valeur minimale de l'attribut pour chaque agrégat ;
- `MAX` : renvoie la valeur maximale de l'attribut pour chaque agrégat ;
- `COUNT` : renvoie le cardinal de chaque agrégat avec `COUNT(*)`, ou le nombre de valeurs distinctes de l'attribut *x* de chaque agrégat avec `COUNT(distinct x)` ;
- `SUM` : renvoie la somme des valeurs de l'attribut pour chaque agrégat ;

Fonction d'agrégation

Certaines fonctions s'appliquent sur les attributs d'agrégats. Les exemples notables sont :

- `MIN` : renvoie la valeur minimale de l'attribut pour chaque agrégat ;
- `MAX` : renvoie la valeur maximale de l'attribut pour chaque agrégat ;
- `COUNT` : renvoie le cardinal de chaque agrégat avec `COUNT(*)`, ou le nombre de valeurs distinctes de l'attribut *x* de chaque agrégat avec `COUNT(distinct x)` ;
- `SUM` : renvoie la somme des valeurs de l'attribut pour chaque agrégat ;
- `AVG` : renvoie la moyenne des valeurs de l'attribut pour chaque agrégat ;

Fonction d'agrégation

Certaines fonctions s'appliquent sur les attributs d'agrégats. Les exemples notables sont :

- `MIN` : renvoie la valeur minimale de l'attribut pour chaque agrégat ;
- `MAX` : renvoie la valeur maximale de l'attribut pour chaque agrégat ;
- `COUNT` : renvoie le cardinal de chaque agrégat avec `COUNT(*)`, ou le nombre de valeurs distinctes de l'attribut *x* de chaque agrégat avec `COUNT(distinct x)` ;
- `SUM` : renvoie la somme des valeurs de l'attribut pour chaque agrégat ;
- `AVG` : renvoie la moyenne des valeurs de l'attribut pour chaque agrégat ;

Exemple :

```
SELECT classe, AVG(moyenneMaths)
FROM eleve
GROUP BY classe
```

Agrégation implicite

Si une fonction d'agrégation est utilisée dans une requête ne contenant pas de `GROUP BY`, alors toutes les lignes de la table de départ sont rassemblées en un seul agrégat. La table renvoyée n'a donc qu'une seule ligne.

Agrégation implicite

Si une fonction d'agrégation est utilisée dans une requête ne contenant pas de `GROUP BY`, alors toutes les lignes de la table de départ sont rassemblées en un seul agrégat. La table renvoyée n'a donc qu'une seule ligne.

Exemple :

```
SELECT MAX(moyenneMaths)
FROM eleve
```

Sélection en amont et en aval

Dans une requête contenant un `GROUP BY`, une clause `WHERE` filtre les lignes de la table de départ, avant qu'elles ne soient regroupées en agrégats. Une clause `WHERE` n'utilise donc **pas** de fonctions d'agrégation. On parle de sélection en amont.

Sélection en amont et en aval

Dans une requête contenant un `GROUP BY`, une clause `WHERE` filtre les lignes de la table de départ, avant qu'elles ne soient regroupées en agrégats. Une clause `WHERE` n'utilise donc **pas** de fonctions d'agrégation. On parle de sélection en amont.

Pour opérer une sélection sur les agrégats, nécessitant donc des fonctions d'agrégation, on utilise une clause `HAVING`, s'écrivant après le `GROUP BY`. On parle de sélection en aval.

Sélection en amont et en aval

Dans une requête contenant un `GROUP BY`, une clause `WHERE` filtre les lignes de la table de départ, avant qu'elles ne soient regroupées en agrégats. Une clause `WHERE` n'utilise donc **pas** de fonctions d'agrégation. On parle de sélection en amont.

Pour opérer une sélection sur les agrégats, nécessitant donc des fonctions d'agrégation, on utilise une clause `HAVING`, s'écrivant après le `GROUP BY`. On parle de sélection en aval.

Exemple :

```
SELECT classe
FROM eleve
GROUP BY classe
HAVING COUNT(*)>30
```

Plan

Tri des lignes

À la suite des clauses précédentes, on peut écrire une clause `ORDER BY` pour trier les lignes de la table renvoyée. Le tri peut s'opérer selon les valeurs d'un attribut ou d'une fonction d'agrégation s'il y a eu agrégation.

Tri des lignes

À la suite des clauses précédentes, on peut écrire une clause `ORDER BY` pour trier les lignes de la table renvoyée. Le tri peut s'opérer selon les valeurs d'un attribut ou d'une fonction d'agrégation s'il y a eu agrégation.

Exemples :

```
SELECT prenom, nom
FROM eleve
WHERE classe = "PCSI"
ORDER BY nom
```

```
SELECT classe
FROM eleve
GROUP BY classe
ORDER BY COUNT(*)
```

Tri des lignes

L'ordre par défaut est croissant, ie la première ligne correspond à la valeur minimale de l'attribut. L'ordre décroissant s'obtient avec le mot-clé `DESC` (pour descending).

Tri des lignes

L'ordre par défaut est croissant, ie la première ligne correspond à la valeur minimale de l'attribut. L'ordre décroissant s'obtient avec le mot-clé `DESC` (pour descending).

Le mot-clé `LIMIT` permet de n'afficher que le nombre indiqué de lignes, par défaut les premières, où à partir de l'indice spécifié par le mot-clé `OFFSET`.

Tri des lignes

L'ordre par défaut est croissant, ie la première ligne correspond à la valeur minimale de l'attribut. L'ordre décroissant s'obtient avec le mot-clé `DESC` (pour descending).

Le mot-clé `LIMIT` permet de n'afficher que le nombre indiqué de lignes, par défaut les premières, où à partir de l'indice spécifié par le mot-clé `OFFSET`.

Exemple :

```
SELECT prenom, nom
FROM eleve
WHERE classe = "PCSI"
ORDER BY moyenneMaths DESC
LIMIT 10 OFFSET 20
```

Requêtes composées

Dans une clause `SELECT`, `FROM` ou `WHERE`, on peut utiliser une sous-requête, qui est une requête dont le résultat va être utilisé par la requête principale.

Requêtes composées

Dans une clause `SELECT`, `FROM` ou `WHERE`, on peut utiliser une sous-requête, qui est une requête dont le résultat va être utilisé par la requête principale.

Lorsqu'une sous-requête renvoie une table contenant une seule ligne et une seule colonne, on peut identifier cette table à l'unique valeur qu'elle contient :

```
SELECT COUNT(*) / (SELECT COUNT(*) FROM eleve)
FROM eleve
WHERE moyenneMaths > 10
```

Requêtes composées

Dans une clause `SELECT`, `FROM` ou `WHERE`, on peut utiliser une sous-requête, qui est une requête dont le résultat va être utilisé par la requête principale.

Lorsqu'une sous-requête renvoie une table contenant une seule ligne et une seule colonne, on peut identifier cette table à l'unique valeur qu'elle contient :

```
SELECT COUNT(*) / (SELECT COUNT(*) FROM eleve)
FROM eleve
WHERE moyenneMaths > 10
```

Dans un `WHERE`, on peut utiliser `IN` pour tester l'appartenance d'une ligne à une table :

```
SELECT prenom, nom
FROM eleveBuffon
WHERE filiere IN
    (SELECT filiere from eleveRaspail)
```

Requêtes composées

On peut aussi utiliser une sous-requête dans une clause `FROM`, pour enchaîner plusieurs étapes dans la construction du résultat souhaité.

Requêtes composées

On peut aussi utiliser une sous-requête dans une clause `FROM`, pour enchaîner plusieurs étapes dans la construction du résultat souhaité.

Il peut alors être nécessaire de renommer les colonnes formées dans la sous-requête afin d'y faire référence dans la requête principale.

Requêtes composées

On peut aussi utiliser une sous-requête dans une clause `FROM`, pour enchaîner plusieurs étapes dans la construction du résultat souhaité.

Il peut alors être nécessaire de renommer les colonnes formées dans la sous-requête afin d'y faire référence dans la requête principale.

Exercice :
Écrire une requête affichant la moyenne de la moyenne en Mathématiques de chaque classe.

Suppression des doublons

Lorsque l'on ne souhaite pas afficher de doublons, on ajoute le mot-clé `DISTINCT` à la suite de `SELECT` :

```
SELECT DISTINCT prenom
FROM eleveBuffon
```

Suppression des doublons

Lorsque l'on ne souhaite pas afficher de doublons, on ajoute le mot-clé `DISTINCT` à la suite de `SELECT` :

```
SELECT DISTINCT prenom
FROM eleveBuffon
```

Il est également possible d'utiliser `DISTINCT` devant l'argument de la fonction d'agrégation `COUNT`, pour ne compter que les occurrences distinctes de cet attribut :

```
SELECT classe, COUNT(DISTINCT prenom)
FROM eleveBuffon
GROUP BY classe
```

Exercices

Écrire une requête affichant

- pour chaque client, le nom de l'employé qui en est responsable ;
- pour chaque employé responsable de client, le nombre de clients dont il est responsable ;
- la liste des artistes dans l'ordre décroissant du nombre d'albums qu'ils ont enregistrés ;
- pour chaque album, la longueur de l'album en minutes ;
- le nom et prénom des employés responsables d'autres employés ;
- la liste des albums contenant au moins 10 morceaux de plus de 3 minutes ;
- les deux albums contenant le plus de morceaux de plus de 3 minutes.