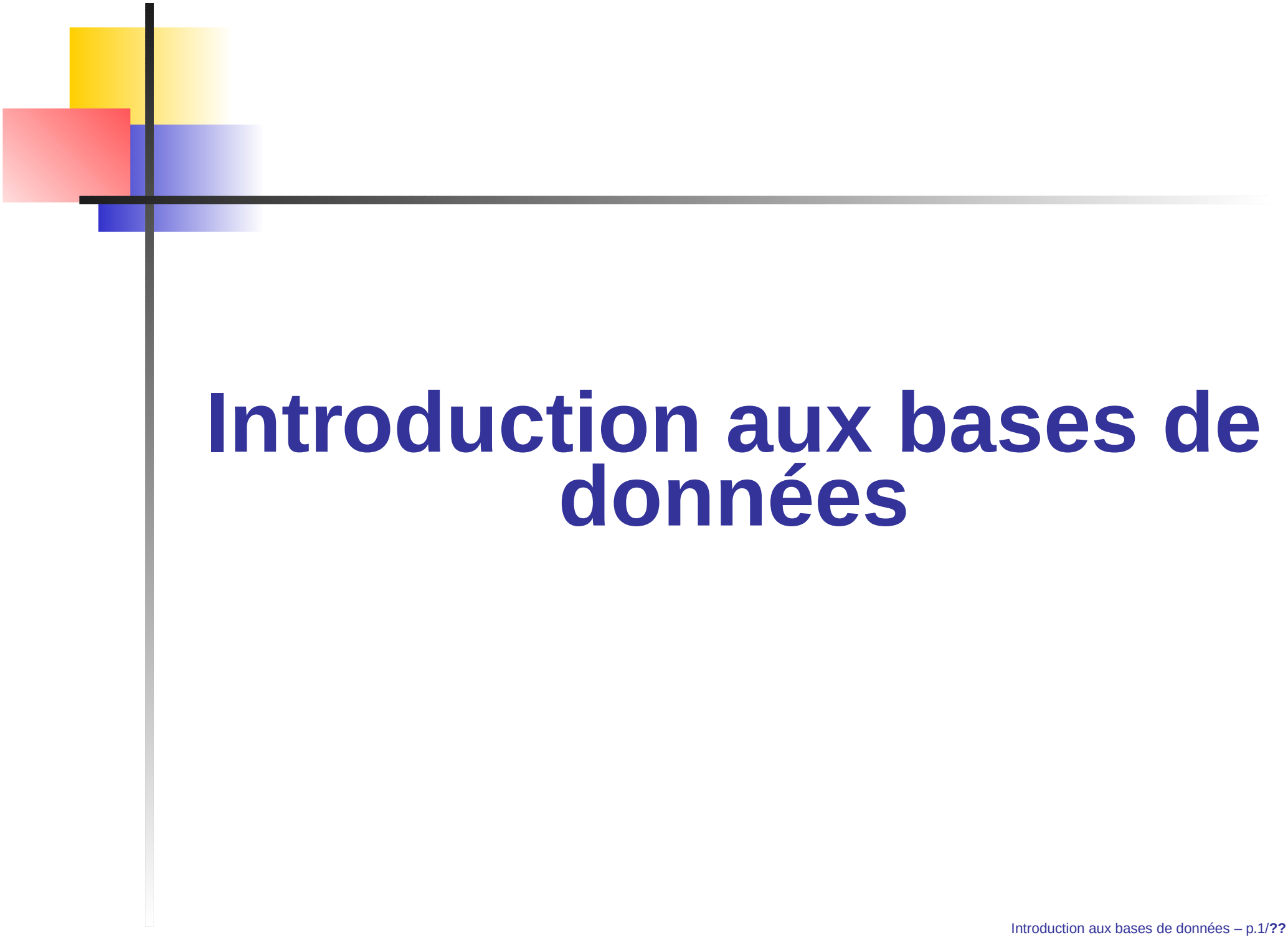


Introduction aux bases de données

Sommaire

Notions de base sur les bases de données relationnelles et SQL:

- qu'est-ce qu'une base de données, un SGBD;
- comment créer une base;
- comment y stocker des données;
- comment interroger une base de données avec SQL,
- une démonstration

Sujet *très* vaste: ceci est juste un aperçu....

Terminologie: bases de données

Nous avons déjà vu comment accéder à des données stockées dans un fichier (les films). C'était une base de données!

- BD = ensemble d'informations placées dans un ou plusieurs fichiers
- fichier = *persistance* = les données survivent à l'arrêt des programmes
- une base de données a une structure précise (on ne met pas tout en vrac)

Terminologie: SGBD

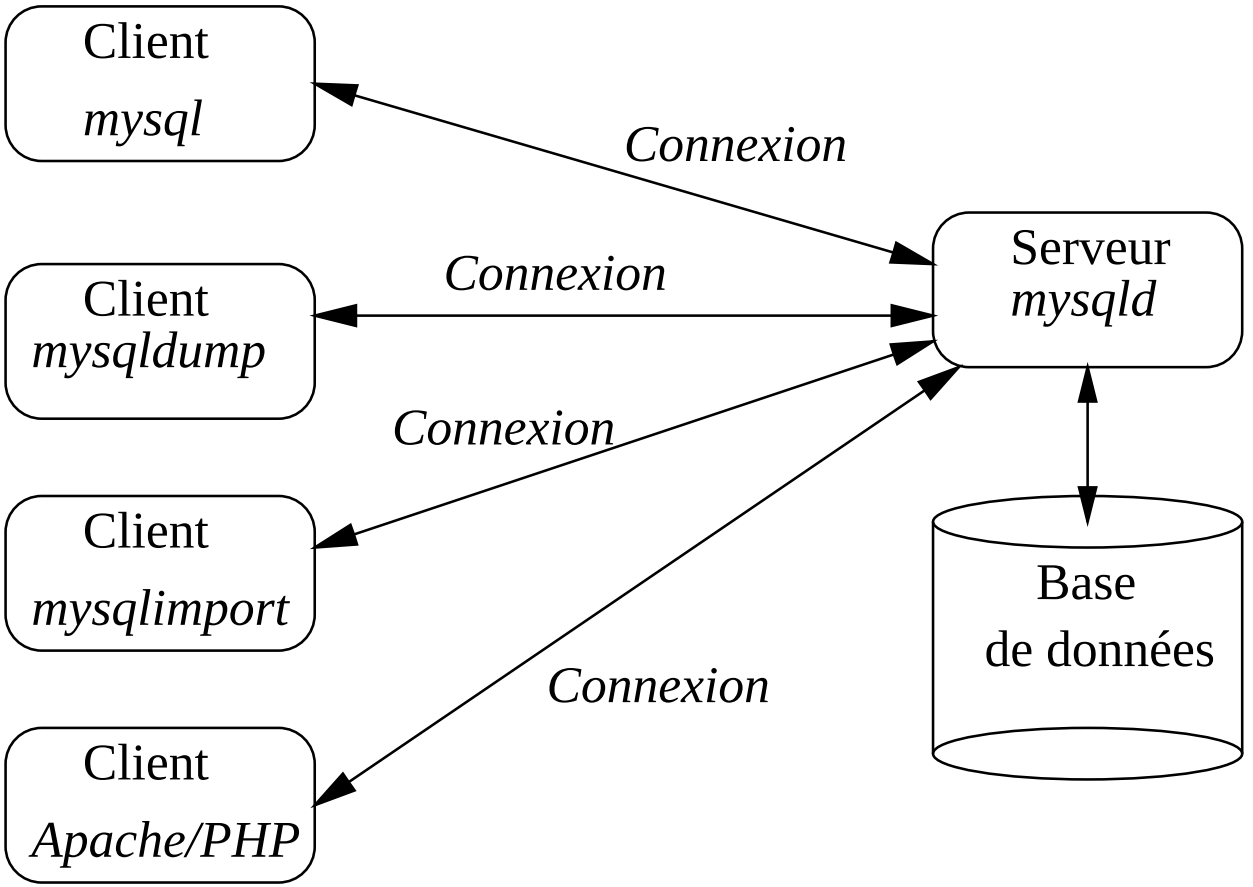
Nous avons déjà vu comment écrire des programmes pour accéder à des fichiers: lourd et peu pratique!

- on fait toujours appel à des logiciels spécialisés pour accéder à une BD: *Les Systèmes de Gestion de Bases de Données (SGBD)*
- ils se chargent de tout: stockage, accès, recherche, sécurité, ...
- on communique avec eux par un langage spécialisé: SQL

Exemples de SGBD: ORACLE, MySQL, PostgreSQL, ...

Exemple: MySQL

Fonctionne en client/serveur: le serveur gère la base, les clients communiquent avec le serveur.



Utilisation d'un SGBD

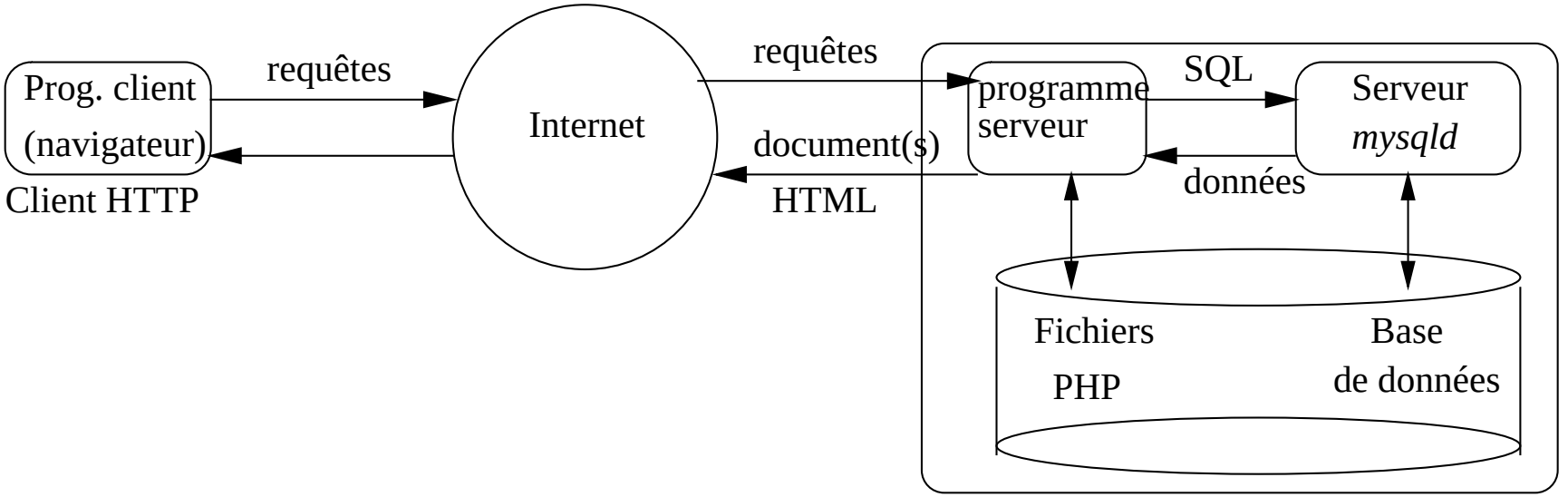
En pratique on ne se soucie pas de ce que fait le serveur. On doit:

- choisir une application cliente qui permet de dialoguer avec un serveur MySQL.
- se connecter en désignant le serveur, et en donnant un login et un mot de passe;
- envoyer des ordres en langage SQL.

Dans notre contexte: les applications clientes seront des scripts PHP.

Notre architecture

En plus du serveur Web, nous avons maintenant un serveur MySQL.



Site web avec scripts PHP et MySQL

Les données

MySQL nous présente les données sous forme de *table*.

titre	année	nom_realisateur	prénom_realisateu
Alien	1979	Scott	Ridley
Vertigo	1958	Hitchcock	Alfred
Psychose	1960	Hitchcock	Alfred
Kagemusha	1980	Kurosawa	Akira
Volte-face	1997	Woo	John
Pulp Fiction	1995	Tarantino	Quentin

Les tables d'une base de données

Base de données = un ensemble de tables.

- chaque table a un nom unique (par exemple *Film*);
- une table comprend une ou plusieurs *colonnes* ayant chacune un nom;
- une table comprend une ou plusieurs *lignes*, chacune constituée d'une valeur pour chaque colonne.
- les noms de table et de colonne constituent le *schéma* de la base; les lignes constituent le *contenu* de la base.

Ce qu'il faut savoir faire

Créer des tables:

```
CREATE TABLE Personne (nom VARCHAR(30),
    prénom VARCHAR(30))
```

Y placer des données:

```
INSERT INTO Personne (nom, prénom)
    VALUES ('Rigaux', 'Philippe')
```

Rechercher des données:

```
SELECT * FROM Personne
    WHERE nom='Rigaux'
```

$\Rightarrow$ c'et le langage SQL.

Création de tables

La commande CREATE TABLE

Voici la syntaxe:

```
CREATE TABLE nom_table (
    nom_colonne1 type_colonne1 [NOT NULL],
    nom_colonne2 type_colonne2 [NOT NULL],
    [...]
    PRIMARY KEY (clé)
  )
```

NB : l'emploi des majuscules et minuscules est indifférent, *sauf pour le nom des tables*.

Ce qu'il faut retenir

- on ne peut pas avoir deux colonnes avec le même nom; choisir des noms simples et significatifs;
- les types SQL sont simples: on peut se contenter de
  1. `INTEGER`: les entiers
  2. `VARCHAR(`*n*`)`: chaînes de longueur inférieure à n
  3. `DECIMAL(`*n*`,`*d*`)`: les numériques avec d décimales
- un des attributs doit identifier une ligne de manière unique: c'est la *clé primaire*.
- l'option `NOT NULL` indique qu'on force un attribut à prendre une valeur (impératif pour la clé primaire).

Exemple de création de table

On veut créer une table pour les artistes (acteurs, réalisateurs):

```
CREATE TABLE Artiste (id INTEGER NOT NULL,
    nom VARCHAR (30) NOT NULL,
    prenom VARCHAR (30) NOT NULL,
    annee_naissance INTEGER,
    PRIMARY KEY (id)
)
```

La colonne `id` est *l'identifiant*: on l'indique avec la commande `PRIMARY KEY`.

Autre exemple: les films

```
CREATE TABLE Film (titre VARCHAR (50) NOT NULL,
annee INTEGER NOT NULL,
id_realisateur INTEGER NOT NULL,
genre VARCHAR(30),
resume TEXT,
code_pays VARCHAR (4),
PRIMARY KEY (titre)
)
```

Important : la colonne `id_realisateur` contient, pour chaque ligne, *l'identifiant* du metteur en scène du film (c'est un lien).

Contenu: la table des artistes

id	nom	prénom	année_naissance
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899
3	Kurosawa	Akira	1910
4	Woo	John	1946
5	Tarantino	Quentin	1963
6	Cameron	James	1954
7	Tarkovski	Andrei	1932

La table des films (3 colonnes)

titre	année	id_realisateur
Alien	1979	1
Vertigo	1958	2
Psychose	1960	2
Kagemusha	1980	3
Volte-face	1997	4
Pulp Fiction	1995	5
Titanic	1997	6
Sacrifice	1986	7

Autres commandes utiles

- Pour détruire une table (bien réfléchir...):

  `DROP TABLE` *`nom_table`*

  Attention: le contenu est perdu, définitivement

- Pour ajouter une colonne à une table:

  `ATER TABLE` *`nom_table`*
  `ADD` *`nom_colonne type_colonne`*

- Pour renommer une table:

  `ATER TABLE` *`nom_table`* `RENAME AS` *`nouvau_nom`*

Insertions, recherches

La commande INSERT

```
INSERT INTO nom_table (nom_col1, nom_cl2.  ....)
    VALUES (val1, val2, ...)
```

- Pour insérer un artiste:

```
INSERT INTO Artiste
    (id, nom, prénom, année_naissance)
    VALUES ('8', 'Woody', 'Allen', '1941')
```

- Pour insérer un film

```
INSERT INTO Film
    (titre, année, id_réalisateur)
    VALUES ('Match Point', '2005', '8')
```

Quelques précautions

Toujours respecter les règles suivantes, sous peine de voir la commande rejetée par MySQL.

- les valeurs sont entre guillemets simples (sinon on ne sait pas les distinguer des noms de colonnes),
- il faut toujours indiquer une valeur pour les colonnes `NOT NULL`
- on ne peut pas mettre deux fois la même valeur dans une colonne qui est clé primaire.

La commande SELECT

```
SELECT nom_col1, nom_col2, [, ...]
   FROM nom_table
   WHERE critères_sélection
```

- on recherche dans toutes les lignes de la table *nom_table* (FROM),
- pour chaque ligne on vérifie les critères
- si oui on place dans le résultat les valeurs de *nom_col1*, *nom_col2*, ...

Important: on peut combiner de recherches sur plusieurs tables – *non présenté dans ce cours*

Exemples: la base *Films*

Des films, des réalisateurs, des acteurs $\Rightarrow$: voir poly.

titre	**année**	**id_realisateur**	**genre**
Impitoyable	1992	20	Western
Van Gogh	1990	29	Drame
Kagemusha	1980	68	Drame
Les pleins pouvoirs	1997	20	Policier

Exemples: la base *Films*

Les artistes: notez une valeur inconnue (ou “NULL”): l’année de naissance de J. Dutronc.

id	nom	prénom	année_naissance
20	Eastwood	Clint	1930
21	Hackman	Gene	1930
29	Pialat	Maurice	1925
30	Dutronc	Jacques	
68	Kurosawa	Akira	1910

Quelques requêtes de base

- `SELECT titre FROM Film WHERE genre='Drame'`
- `SELECT * FROM Film` (le "*" désigne tous les nom de colonnes)
-
```
SELECT titre FROM Film
  WHERE genre='Drame' OR genre='Western'
```
-
```
SELECT titre FROM Film
  WHERE genre='Drame' AND annee < 1985
```
-
```
SELECT titre FROM Film
  WHERE (genre='Drame' AND annee < 1985)
  OR genre='Western'
```

Calculs et renommages

- On peut des calculs arithmétiques, ou appliquer des fonctions aux valeurs ramenées.

```
SELECT nom, 2006 - annee_naissance
FROM Artiste
```

- On peut donner un nom aux valeurs obtenues avec la clause AS

```
SELECT nom, 2006 - annee_naissance AS age
FROM Artiste
```

Elimination des doublons, tri

- Le résultat de certaines requêtes peut contenir plusieurs fois la même ligne.

```
SELECT annee_naissance FROM Artiste;
```

On obtient 2 fois 1930 (Eastwood et Hackman). Avec la clause `DISTINCT` on élimine ces doublons.

```
SELECT DISTINCT annee_naissance FROM Artiste;
```

- La clause `ORDER BY` permet de trier le résultat.

```
SELECT titre, genre FROM Film
ORDER BY genre, annee
```

Valeurs nulles

Certaines valeurs sont absentes (l'année de naissance de J. Dutronc): *on ne peut rien en dire, et rien en tirer*.

- Pas de comparaison possible avec une valeur à `NULL`
- Pas de calcul possible avec une valeur à `NULL`

Pour rechercher les valeurs absentes on ne peut pas faire:

```
SELECT * FROM Artiste
WHERE annee_naissance = NULL
```

Mais on peut utiliser le comparateur spécial `IS NULL`

```
SELECT * FROM Artiste
WHERE annee_naissance IS NULL
```

La clause `WHERE`

On fait une combinaison booléenne de comparaisons.

- Les comparateurs: $=$, $<$, $<=$, $>$, $>=$, $!=$.
- Les connecteurs booléens: `AND`, `OR`, `NOT`
- Pour les chaînes: le comparateur `LIKE` permet de faire des recherches partielles.

```
SELECT * FROM Artiste
WHERE nom LIKE 'S%'
```

- Les lignes sélectionnées = celles pour lesquelles l'évaluation du `WHERE` renvoie `vrai`.