

# Bases de données pour le Web

IO2

## Internet et outils

Matej Stehlik

IRIF, Université Paris Cité

[matej@irif.fr](mailto:matej@irif.fr)

# Bases de données et Web

Toutes les applications Web manipulent des données persistantes

Exemples :

- Site de vente en ligne : descriptions et prix des articles vendus, le transitions de vente, ...
- Site d'une compagnie aérienne : quels vols à quels horaires, ..
- Espace utilisateur de n'importe quel site : informations d'enregistrement des utilisateurs, ...
- Réseaux sociaux : les profils des membres, les posts, qui suit qui, qui aime quoi...

Toutes ces données doivent être stockées de façon durable chez le serveur,

- pas uniquement pendant une session utilisateur
- pas uniquement pendant que le serveur est actif - elles doivent continuer à exister même si le serveur est arrêté

Ces données doivent donc exister ailleurs que dans la mémoire centrale du serveur (et elle ne serait pas suffisante de toute façon!)

# La gestion des données persistantes

Un problème vieux comme l'informatique

Pour gérer une liste (par exemple de films),  
pourrait-on utiliser un fichier texte?

```
> cat films.txt
```

Alien 1979 Scott

Vertigo 1958 Hitchcock

Psychose 1960 Hitchcock

Kagemusha 1980 Kurosawa

Volte-face 1997 Woo

# La gestion des données persistantes

Ou une structure de données sérialisée sur disque?

- Définir une classe d'objets qu'on va mettre dans un tableau.

```
class Film {  
    public $titre,  
        $année,  
        $réalisateur,  
}
```

- Créer un tableau d'objets films le sérialiser dans un fichier sur sur le disque

# Les problèmes des systèmes à base de fichiers

## Accès difficile aux données

Ouvrir le fichier

Parcourir les “lignes”, ou les objets sérialisés

Effectuer les comparaisons

Format complexe (tenir compte des espaces...)

Format de fichiers non standards  $\Rightarrow$  partage des données difficile

Écriture d'un programme spécifique pour chaque interrogation/modification des données : coûts de développement élevés, coûts de maintenance élevés

## Intégrité logique des données :

Comment garantir que les données restent cohérentes

(par exemple éviter :

Vertigo 1958 Hitchcock

Vertigo 1962 Hitchcock)

# Les problèmes des systèmes à base de fichiers

## Concurrence

Que se passe-t-il si plusieurs utilisateurs ajoutent, modifient ou suppriment des lignes simultanément ?

## Gestion des pannes

Comment faire en sorte que les données ne soient pas laissées dans un état incohérent si le serveur va en panne pendant qu'il les manipule?  
À programmer soi même (l'OS ne suffit pas)

## Sécurité

Comment empêcher un programme erroné ou malveillant d'écrire des données autres (e.g. écraser le contenu du fichier) ?

etc.

# SGBD

Les **S**ystèmes de **G**estion de **B**ases de **D**onnées ont été conçus pour apporter des (bonnes) réponses à ces problèmes

- Manipuler informations complexes de façon abstraite
  - sans se préoccuper de comment elle sont physiquement organisées sur disque (indépendance physique)
- Optimiser et rendre efficace l'accès aux données
- Garantir l'intégrité des données et la tolérance aux pannes
- Assurer un résultat cohérent quand plusieurs utilisateurs accèdent simultanément aux données

# SGBD relationnels

Représentation de l'information sous forme de **tables**

La manière dont l'information est réellement stockée sur disque est inconnue de l'application

L'application ne voit que les “tables” présentées par le SGBD

Langage “standard” pour l'interrogation/manipulation de ces tables :  
**SQL** (norme ANSI de 1992) - *Structured Query Language*

Il existe de très nombreux systèmes de gestion de bases de données relationnelles

Oracle, PostgreSQL, SQL Server, DB2, MySQL...



# Un aperçu du modèle relationnel des données

# Modèle relationnel des données

- Une base de données consiste en plusieurs **tables (relations)**
- Chaque table a un nom
- Dans une table chaque colonne a un nom
- les noms des colonnes d'une table sont appelés **attributs (ou champs)**
- Chaque attribut a un **domaine** associé (i.e. l'ensemble des valeurs possibles pour cet attribut)
- Les données dans chaque table sont un ensemble de **lignes (tuples)**
  - **une ligne** fournit à chaque attribut une valeur de son domaine

# Modèle relationnel des données

Nom de la relation

Attributs

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Lignes

## Exemple : les films sous forme de table

### Films

<i><b>titre</b></i>	<i><b>annee</b></i>	<i><b>realisateur</b></i>
Alien	1979	Scott
Vertigo	1958	Hitchcock
Psychose	1960	Hitchcock
Kagemusha	1980	Kurosawa
Volte-face	1997	Woo
Pulp Fiction	1995	Tarantino
Titanic	1997	Cameron
Sacrifice	1986	Tarkovski

## Schéma et instance d'une table

- La structure des données est rigide (toutes les lignes ont les mêmes attributs)
  - ▶ Cette structure est appelé **schéma de la table** :

### **STUDENT**

Name	SSN	Phone	Adress	OfficePhone	Age	GPA
------	-----	-------	--------	-------------	-----	-----

- ▶ L'ensemble des lignes est appelé **instance de la table**
- Le schéma est défini une fois pour toute
- L'instance varie dans le temps selon les données qu'on inséré et on supprime
- Des **contraintes d'intégrité** sont en général associées à chaque table, qui empêchent les instances non valides (valeurs requis, uniques, etc...)

# Schéma et instance d'une base de données

- Une bases de données est formée par plusieurs tables

## STUDENT

Name	SSN	Phone	Adress	OfficePhone	Age	GPA
Bayer	347294	333	Albyn Pl.	367	18	3.24
Ashly	5784673	466	Queen St.	390	20	3.53
Davidson	4357387	589	Princes St.	678	25	3.25

## EXAM

## COURSE

Course-Id	Title
12	CS
34	DB

Student-SSN	Course-Id
347294	12
5784673	12
4357387	34
347294	34

- **Schéma de la base de donnée** : l'ensembles de schémas de toutes ses tables
- **Instance de la base de données** : l'ensemble des instances de toutes ses tables (i.e les données)

# SQL

Langage déclaratif

Permet de manipuler à la fois le schéma et les instances d'une bases de données

**Composante DDL** (Data Definition Language)

- définir le schéma de la bases de données

**Composante DML** (Data Manipulation Language)

- interroger (extraire de l'information de) la BD de manière **déclarative** :

```
SELECT titre          -- l'attribut recherché
FROM Films           -- la table interrogée
WHERE annee > 1980;   -- le critère de sélection
```

# Quelques éléments de SQL DDL



# Création d'une base de données

```
CREATE DATABASE ma_base; -- commande SQL
```

Au sein d'un serveur on peut créer plusieurs **bases de données** différentes

Chaque base à ses propres tables, mais une table appartient à une seule base

On peut associer des droits d'accès aux utilisateurs des bases, et des tables

# Création d'une table (schéma)

Au sein de la base de données courante :

```
CREATE TABLE Film
```

```
(titre      VARCHAR(30),
```

```
  annee      INTEGER,
```

```
  realisateur VARCHAR(30)
```

```
);
```

Les types des attributs  
varient d'un SGBD à l'autre

Cette déclaration contient en général aussi la définition de plusieurs contraintes d'intégrité (donc on discutera plus loin)

À sa création, une table est “vide” (**instance vide, on ne crée que le schéma**)

Elle ne contient aucune donnée, i.e. aucune ligne

# Détruire une table

Détruire la table (schéma et données):

```
DROP TABLE Film;
```

Supprimer toutes les données (mais garder le schéma):

```
TRUNCATE TABLE Film;
```

# Quelques éléments de SQL - DML

# Insérer des données

```
INSERT INTO Film
```

```
VALUES ('Pulp Fiction', 1995, 'Tarantino');
```

**Note** : on peut ne saisir que quelques attributs, et dans un ordre différent de celui défini pour la table.

```
INSERT INTO Film (realisateur, titre)
```

```
VALUES ('Allen', 'Match Point');
```

## Insérer des données

Resultat :

### Films

<i>titre</i>	<i>annee</i>	<i>realisateur</i>
Alien	1979	Scott
Vertigo	1958	Hitchcock
Psychose	1960	Hitchcock
Kagemusha	1980	Kurosawa
Volte-face	1997	Woo
Pulp Fiction	1995	Tarantino
Titanic	1997	Cameron
Sacrifice	1986	Tarkovski
Match Point	NULL	Allen

Attention : génère une erreur si les valeurs pas spécifiées sont requises dans la table

# Supprimer des données

```
DELETE FROM Film WHERE annee <= 1960;
```

Supprime toutes les lignes pour lesquelles l'année est inférieure ou égale à 1960.

# Modifier des données

```
UPDATE Film  
SET réalisateur = 'Wu'  
WHERE réalisateur = 'Woo';
```

Met à jour le champs **réalisateur** de toutes les lignes sélectionnées par la clause **WHERE**.



# SQL - DML : Interrogation des données (aperçu)

# Interrogation de bases de données

Interroger une base de données: “extraire” des données de la base

Requête: produire une nouvelle table à partir des tables dans la base

Exemples de requêtes:

quelles sont tous les films de 1954 ?

quels réalisateurs ont réalisé le plus de films?

SQL permet d'exprimer les requêtes de façon **déclarative**

on exprime **quelles sont les données** qu'on veut extraire et

**NON PAS comment les extraire**

## Interroger une table

```
SELECT * FROM Film;
```

sélectionne toutes les colonnes (\*) de la table Film

Resultat :

titre	annee	realisateur
Alien	1979	Scott
Vertigo	1958	Hitchcock
Psychose	1960	Hitchcock
Kagemusha	1980	Kurosawa
Volte-face	1997	Woo
Pulp Fiction	1995	Tarantino
Titanic	1997	Cameron
Sacrifice	1986	Tarkovski
Match Point	NULL	Allen

# Interroger une table

On peut sélectionner les colonnes de son choix

```
SELECT titre, année FROM Film;
```

Resultat :

titre	annee
Alien	1979
Vertigo	1958
Psychose	1960
Kagemusha	1980
Volte-face	1997
Pulp Fiction	1995
Titanic	1997
Sacrifice	1986
Match Point	NULL

# Interroger une table

La clause **WHERE** permet de définir un ensemble de conditions qui doivent être vérifiées par les lignes retenues

```
SELECT titre, annee
FROM Film
WHERE titre = 'Vertigo'
OR realisateur = 'Hitchcock'
OR (annee >= 1995
    AND annee < 2000);
```

Resultat :

titre	annee
Vertigo	1958
Psychose	1960
Volte-face	1997
Pulp Fiction	1995
Titanic	1997

Sélectionne les titres et les années des films dont le titre est 'Vertigo' ou le réalisateur est 'Hitchcock', ou l'année est entre 1995 et 2000

# Introduction à MySQL

# MySQL

Un système de gestion de bases de données relationnelles

Un des plus utilisés pour les applications Web

Relativement “léger” par rapport à d’autres SGBD

L’installation de MySQL fournit :

Un serveur : `mysqld`

le sgbd - seul capable d'accéder et manipuler les données

Des utilitaires

`mysql` : client permettant de se connecter au serveur de bases de données pour demander la manipuler et interrogation des données

`mysqldump` : client permettant de faire des sauvegardes

# Connexion au serveur MySQL

Lancer le client mysql :

```
> mysql -h nom_serveur -u nom_utilisateur -p
```

```
Enter password: *****
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 13
```

```
Server version: 8.0.11 MySQL Community Server - GPL
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql> help;
```

Pour voir les bases de données disponibles :

```
mysql> show databases;
```



# Connexion à une base de données MySQL

- Pour sélectionner une base de données :

```
mysql> USE ma_base
```

```
Database changed
```

- Pour se connecter et en même temps sélectionner la base `io2021` :

```
> mysql -h nom_serveur -u nom_utilisateur -p io2021
```

- On peut aussi utiliser le fichier de configuration (attention ! force à stocker le mot de passe en clair)

```
$HOME/.my.cnf
```

```
[client]
```

```
host = nom_serveur
```

```
user = nom_utilisateur
```

```
password = motdepasse
```

```
> mysql io2021
```

# MySQL : utilisation du client mysql

Un fois la base de données sélectionnée, on peut taper à l'invite du shell mysql toute commande SQL pour cette base :

```
mysql> CREATE TABLE Film  
-> (titre      VARCHAR(30),  
->  annee      INTEGER,  
->  realisateur VARCHAR(30)  
-> );
```

Query OK, 0 rows affected (0.05 sec)

```
mysql> INSERT INTO Film (realisateur, titre)  
-> VALUES ('Allen', 'Match Point');
```

Query OK, 1 row affected (0.00 sec)

# MySQL : utilisation du client mysql

Un fois la base de données sélectionnée, on peut taper à l'invite du shell mysql toute commande SQL pour cette base :

```
mysql> SELECT *  
      -> FROM Film
```

```
+-----+-----+-----+  
| titre      | annee | realisateur |  
+-----+-----+-----+  
| Match Point |  NULL | Allen      |  
+-----+-----+-----+
```

```
1 row in set (0.00 sec)
```

# MySQL : exécution de scripts SQL

On peut entrer les commandes SQL depuis un fichier préalablement édité.

On utilise conventionnellement l'extension `.sql`

```
> mysql -h nom_serveur -u nom_utilisateur -p < script.sql
```

Ou depuis mysql :

```
mysql> SOURCE script.sql
```

script\_films.sql

```
mysql> source script_films.sql
```

# Quelques commandes MySQL

On peut également utiliser des commandes propres à MySQL par exemple :

décrire le schema d'une table :

```
mysql> DESCRIBE Film;
```

Field	Type	Null	Key	Default	Extra
titre	varchar(30)	YES		NULL	
annee	int(11)	YES		NULL	
realisateur	varchar(30)	YES		NULL	

```
3 rows in set (0.00 sec)
```

# Quelques commandes MySQL

On peut également utiliser des commandes propres à MySQL par exemple :

Lister les bases de données

```
SHOW DATABASES;
```

Lister les tables de la base courante

```
SHOW TABLES;
```

Afficher le nom de la base courante

```
SELECT DATABASE();
```

Afficher le nom de l'utilisateur courant

```
SELECT USER();
```

# Chargement de données

On peut aussi insérer des données depuis un fichier (insertion dite “en masse”)

Exemple :

```
mysql> LOAD DATA LOCAL INFILE  
-> 'films.txt' INTO TABLE Film  
-> FIELDS TERMINATED BY ',';
```

Query OK, 8 rows affected (0.00 sec)

Records: 8 Deleted: 0 Skipped: 0 Warnings: 0

films.txt

mysql> load data...

# Types de données (My)SQL

Principaux types de données SQL ANSI supportés par MySQL :

Nombres :

`SMALLINT` (2 octets)

`INTEGER` (4 octets)

`BIGINT` (8 octets)

`FLOAT` (4 octets)

`REAL` (8 octets)

`DOUBLE PRECISION` (8 octets)

`DATE` (3 octets)

`TIME` (3 octets)



# Types de données (My)SQL

Principaux types de données SQL ANSI supportés par MySQL :

Chaînes de caractères :

`VARCHAR (M)` avec `M < 256` taille variable

`CHAR (M)` avec `M < 256` taille fixe

⚠ MySQL only : `BLOB` / `TEXT`

suite de octets / suite de caractères, taille  $< 2^{16}$

⚠ MySQL only : `MEDIUMBLOB` / `MEDIUMTEXT` taille  $< 2^{24}$

⚠ MySQL only : `LONGBLOB` / `LONGTEXT` taille  $< 2^{32}$

# Types de données (My)SQL

Principaux types de données SQL ANSI supportés par MySQL :

Dates :

`DATE` (3 octets)

`TIME` (3 octets)

`DATETIME` (8 octets)

`TIMESTAMP` (4 octets)

`YEAR` (1 octets)

Ensembles :

⚠ MySQL only : `ENUM ('val1', 'val2', ...)` ( $\leq 2$  octets)

⚠ MySQL only : `SET ('val1', 'val2', ...)` ( $\leq 8$  octets)

# Types de données (My)SQL

INTEGER[(M)] [UNSIGNED] [ZEROFILL]

taille max : **M** chiffres (seulement pour l'affichage)

Non signé si **UNSIGNED**

**ZEROFILL** : 3  $\Rightarrow$  0000000003

VARCHAR(M) [BINARY]

Par défaut, pas de différence entre minuscules et majuscules, i.e. 'aaa'  
identique à 'AAA', sauf si **BINARY** est précisé

# Interaction PHP / base de données

# Accès à MySQL depuis PHP

- L'interface MySQL/PHP permet de récupérer, modifier, créer, toute information d'une base de données MySQL depuis un script PHP.
- L'accès à MySQL se fait via des fonctions PHP prédéfinies (module `mysqli`)
- PHP a un module différent pour l'accès à la plupart des SGBD
- Il existe également le module PHP `PDO`, qui uniformise l'accès aux différents SGBD (légèrement plus complexe à utiliser)

# PHP/mysqli : connexion à la base de données

Accès à MySQL depuis PHP au travers de l'extension `mysqli`

Connexion à une base de données

```
$connex = mysqli_connect($serveur, $login, $mdp, $base);
```

Tester si la connexion a eu lieu

```
if (! $connex) {  
    //afficher page d'erreur  
}
```

En phase de débogage il peut être utile de récupérer l'erreur renvoyé par MySQL

```
echo mysqli_connect_error($connex);
```

# PHP/mysqli : exécution de requêtes

Exécution d'une requête

```
$req = "SELECT ...FROM ..." // ou $req = "INSERT ...INTO ..." etc.  
$resultat = mysqli_query($connex, $req);
```

Tester si la requête a été exécutée sans erreur

```
if (! $resultat) {  
    //afficher page d'erreur  
}
```

Récupérer la dernière erreur de MySQL :

```
echo mysqli_error($connex);
```

# PHP/mysqli : traitement du résultat d'une requête

Si `$req` est une requête `SELECT`

```
$resultat = mysqli_query($connex, $req);
```

`$resultat` est de type `resource`,

on peut voir cela comme une collection de lignes, chaque ligne représentée par un tableau

Récupération d'une ligne du résultat sous forme de tableau associatif

```
$ligne = mysqli_fetch_assoc ($resultat)
```

Récupération d'une ligne du résultat sous forme de tableau indicé

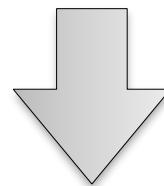
```
$ligne = mysqli_fetch_row ($resultat);
```



# PHP/mysqli : traitement du résultat d'une requête

Une ligne du résultat en tant que tableau associatif :  
entrées du tableau indexées par 'attribut'

```
+-----+-----+-----+  
| titre      | annee | realisateur |  
+-----+-----+-----+  
| Pulp Fiction | 1995 | Tarantino  |  
+-----+-----+-----+
```



\$ligne

"Pulp Fiction"	1995	"Tarantino"
"titre"	"annee"	"realisateur"

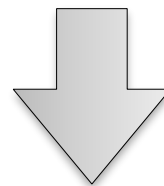
Ex : `$ligne['realisateur']` donne le nom du réalisateur

# PHP/mysqli : traitement du résultat d'une requête

Une ligne du résultat en tant que tableau indicé :

entrées du tableau indexés par position :

```
+-----+-----+-----+  
| titre      | annee | realisateur |  
+-----+-----+-----+  
| Pulp Fiction | 1995 | Tarantino  |  
+-----+-----+-----+
```



`$ligne`

"Pulp Fiction"	1995	"Tarantino"
----------------	------	-------------

0

1

2

Ex : `$ligne[2]` donne le nom du réalisateur

# PHP/mysqli : traitement du résultat d'une requête

`mysqli_fetch_assoc ($resultat)` et `mysqli_fetch_row ($resultat)` renvoient une ligne du résultat et font avancer un pointeur interne de ligne dans `$resultat`

les deux renvoient NULL s'il n'y a plus de ligne à lire

⇒ une suite d'appels à ces fonctions parcourt toutes les lignes du résultat :

```
$ligne = mysqli_fetch_assoc($resultat);  
while ($ligne) {  
    //traiter $ligne  
    $ligne = mysqli_fetch_assoc($resultat); //prochaine ligne  
}
```

# PHP/mysqli : traitement du résultat d'une requête

`mysqli_fetch_assoc ($resultat)` et `mysqli_fetch_row ($resultat)` renvoient une ligne du résultat et font avancer un pointeur interne de ligne dans `$resultat`

les deux renvoient NULL s'il n'y a plus de ligne à lire

⇒ une suite d'appels à ces fonctions parcourt toutes les lignes du résultat :

...ou de façon équivalente (plus compacte)

```
while ($ligne = mysqli_fetch_assoc($resultat)) {  
    //traiter $ligne  
}
```

possible parce qu'une affectation (`$var1 = $var2`) est aussi une expression qui renvoie la valeur affectée

# PHP/mysqli : traitement du résultat d'une requête

Nombre de lignes du résultat

`mysqli_num_rows ($resultat)`

Libération des ressources allouées pour stocker le résultat

`mysqli_free_result ($resultat)`

Il est conseillé de toujours utiliser cette fonction lorsque l'objet `$resultat` n'est plus utile

# PHP/mysqli : fermeture de la connexion

Fermeture de la connexion

```
mysqli_close($connex);
```

À appeler lorsque l'interaction du script PHP avec la base de données est terminée (aucune autre requête à envoyer)

## Exemple de page Web qui interagit avec une BD

- Page Web très simple qui réalise une tâche de base
  - ▶ récupérer la liste des films de la BD et l'afficher sous forme de tableau HTML
- Mais on structure le script PHP de façon très générale, qui peut être facilement adaptée à gérer une interaction plus complexe

05-films /films.php

/connex.php

/affichage.php

localhost/.../films.php

## Affichage des données lues de la BD

- Les données sont en général stockées dans la BD sans protection par rapport aux caractères spéciaux HTML (i.e. sans `htmlspecialchars` préalable)
- Raisons : les données peuvent avoir une autre utilisation que l'affichage HTML
  - ▶  $\Rightarrow$  les données lues de la bases de données devront être protégées avec `htmlspecialchars` avant l'affichage, tout comme les données d'un formulaire
  - ▶ Exemple : pour afficher une ligne du tableau des Films

```
function afficher_ligne(&$ligne) {  
    echo "<tr>",  
        "<td>". htmlspecialchars($ligne["titre"]). "</td>",  
        "<td>". htmlspecialchars($ligne["annee"]). "</td>",  
        "<td>". htmlspecialchars($ligne["realisateur"]). "</td>",  
        "</tr>";  
}
```



# Envoyer des requêtes avec paramètres : une page de recherche

- Dans la plupart des cas les requêtes envoyées à la BD contiennent des paramètres
  - ▶ valeurs récupérés de l'input de l'utilisateur (par exemple à travers un formulaire)
- Exemple :
  - ▶ avec un formulaire l'utilisateur spécifie le nom d'un réalisateur
  - ▶ le serveur récupère ce paramètre et construit une requête qui demande les films de ce réalisateur
  - ▶ la requête est envoyée à la BD, le résultat est récupéré et affiché

05-recherche /recherche.php

/connex.php

/affichage.php

localhost/.../recherche.php

# Envoyer des requêtes avec paramètres : authentication

- Avec une structure similaire : authentication des utilisateurs sur le site
- La base de données contiendra une table utilisateurs

```
CREATE TABLE Users (  
    login varchar(30) primary key,  
    mdp varchar(6)  
    -- etc. par exemple email, ...  
);
```

- L'utilisateur renseigne login et mdp dans un formulaire
- le script qui gère le login récupère ces valeurs dans les variables `$login`, `$mdp` et prépare la requête :

```
$req = "SELECT * FROM Users WHERE login = '$login' AND mdp = '$mdp'"
```

- Si le résultat est vide  $\Rightarrow$  réaffichage du formulaire
- Sinon  $\Rightarrow$  page "Bienvenue" (ainsi que affectation des variables de sessions etc.)

```
mysql> source 05-login/user.sql      05-login/      localhost/.../05-login/login.php
```

## Sécurité de l'interaction avec la BD

- Pour que l'interaction avec la BD n'ait pas de failles de sécurité il faut protéger les données reçues par l'utilisateur (par exemple par un formulaire) avant de les inclure dans une requête
- Raison : un utilisateur malveillant pourrait inclure du code SQL dangereux dans ce paramètres
- Qu'est-ce qui se passe si l'utilisateur renseigne dans le champ login :  
(avec un espace à la fin)

Login

Login :  Mot de passe :

- Rappel:
  - ▶ 1 est une condition booléenne toujours vrai en SQL
  - ▶ -- introduit un commentaire SQL

# Injection de code SQL

Qu'est-ce qui s'est passé ?

- Rappel : on a construit la requête à envoyer comme :

```
SELECT * FROM Users WHERE login = '$login' AND mdp = '$mdp'
```

- Avec l'input donné par l'utilisateur :

- ▶ `$login` a valeur ' **OR 1 --**

- ▶ `$mdp` a valeur vide

- Donc la requête envoyée a valeur :

```
SELECT * FROM Users WHERE login = ' ' or 1 -- ' AND mdp = ''
```

- Ou bien, avec un peu de coloration syntaxique :

```
SELECT * FROM Users WHERE login = ' ' or 1 -- ' AND mdp = ''
```

Cette requête sélectionne tous les utilisateurs dans la table Users : résultat non vide

Accès accordé!!!

## Injection de code SQL : solution

- Chaque module PHP associé à un SGBD offre une fonction qui échappe les caractères spéciaux SQL dans une chaîne de caractères
- pour mysql :

`mysqli_real_escape_string($connex,$str)`

- Utiliser cette fonction sur toutes les chaînes de caractères php destinées à être incluse dans une requête SQL !

```
$login = mysqli_real_escape_string($connex,$login);
```

```
$mdp = mysqli_real_escape_string($connex,$mdp);
```

(Remarque : `stripslashes($str)` fait l'opération inverse si nécessaire )

[05-login/connex.php](#)

[localhost/.../05-login/login.php](#)

## Un autre exemple : une page d'enregistrement

- Effectuer l'enregistrement d'un nouvel utilisateur : plusieurs étapes
  - ▶ le formulaire d'enregistrement est validé comme vu dans le cours sur le traitement des formulaires
  - ▶ Une fois les données validées, elles sont insérées dans la BD par une requête INSERT INTO

05-enreg /05-enreg.php

/valid.php

/affichage.php

/connex.php

localhost/.../05-enreg.php

## Une page d'enregistrement : remarques

- Les données brutes sont insérées dans la BD (htmlspecialchars appelé uniquement avant l'affichage)

```
Login : <input type="text" name="login" size="30"  
        value="<?php echo htmlspecialchars($donnees['login'])?>">
```

...

- Comme toute variable destinée à faire partie d'une requête SQL, les données sont protégées par mysqli\_real\_escape\_string avant l'insertion

```
$login = mysqli_real_escape_string($connex,$donnees['login']);
```

...

## Une page d'enregistrement : hachage du mdp

- Le mot de passe n'est pas stocké dans la BD en clair, on en stocke une version hachée

```
$mdp = password_hash($mdp, PASSWORD_DEFAULT);
```

- La fonction `password_hash` “transforme” le mdp pour qu’il devienne indéchiffrable (i.e. à priori très difficile de dériver le mdp d’origine de son chiffrement)
- Elle utilise un sel aléatoire et un algorithme de chiffrement (le deuxième paramètre)
- Ces deux informations, entre autres, sont incluses dans le chiffrement
- En phase de login utiliser la fonction

```
password_verify($mdp, $mdp_haché);
```

pour vérifier que le `$mdp` fourni par l'utilisateur coïncide avec le `$mdp_haché` lu dans la BD



# Interrogation des données en SQL

# Requêtes SQL : forme générique

- Forme générique d'une requête :

SELECT...

FROM ...

WHERE ...

- Au delà des requêtes de base : plusieurs options pour les clauses **SELECT FROM** et **WHERE**

## Clause WHERE : opérateurs

égalité =

inégalité <>

inférieur <

inférieur ou égal <=

supérieur >

supérieur ou égal >=

intervalle BETWEEN ... AND ...

reconnaissance de motifs LIKE '...'

On peut utiliser les opérateurs booléens NOT, AND et OR

# Clause WHERE : BETWEEN

```
SELECT titre, annee
FROM Films
WHERE titre BETWEEN 'Psychose' AND 'Titanic';
```



Films

<i>titre</i>	<i>annee</i>	<i>realisateu</i>
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



titre	annee
Pulp Fiction	1995
Psychose	1960
Titanic	1997
Sacrifice	1986

# Clause WHERE : opérateurs booléens

SELECT titre, annee

FROM Films

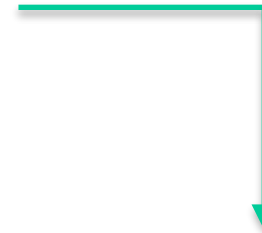
WHERE titre BETWEEN 'Psychose' AND 'Titanic'

OR annee < 1965



**Films**

<i><b>titre</b></i>	<i><b>annee</b></i>	<i><b>realisateu</b></i>
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



+-----+	+-----+
titre	annee
+-----+	+-----+
Vertigo	1958
Pulp Fiction	1995
Psychose	1960
Titanic	1997
Sacrifice	1986
+-----+	+-----+

# Clause WHERE : LIKE

- A LIKE 'motif'  
vrai si la valeur de A est conforme au motif
- 'motif': une chaîne de caractères qui peut inclure les caractères spéciaux "" et "%"
- "" : un seul caractère (n'importe lequel)
- "%" : un nombre quelconque de caractères (y compris zéro caractères)
- le motif est insensible à la casse (ne distingue pas minuscules et majuscules)

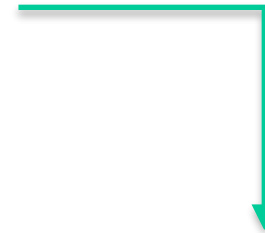
# Clause WHERE : LIKE

```
SELECT titre, annee  
FROM Film  
WHERE titre LIKE '%ti%';
```



**Films**

<i><b>titre</b></i>	<i><b>annee</b></i>	<i><b>realisateu</b></i>
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



```
+-----+-----+  
| titre          | annee |  
+-----+-----+  
| Pulp Fiction   | 1995  |  
| Vertigo        | 1958  |  
| Titanic        | 1997  |  
+-----+-----+
```

# Clause WHERE : LIKE

```
SELECT titre, annee
FROM Film
WHERE titre LIKE '_a%';
```



Films

titre	annee	realisateu
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



titre	annee
Kagemusha	1995
Sacrifice	1958



# Clause SELECT : DISTINCT

SELECT DISTINCT annee  
FROM Film



Films

<i>titre</i>	<i>annee</i>	<i>realisateu</i>
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



+-----+
annee
+-----+
1979
1958
1960
1980
1997
1995
1986
+-----+

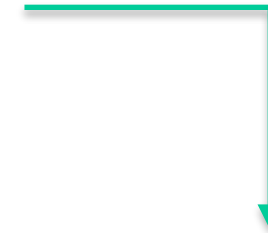
## Clause SELECT : renommage

```
SELECT titre AS titre_film  
FROM Films ;
```



**Films**

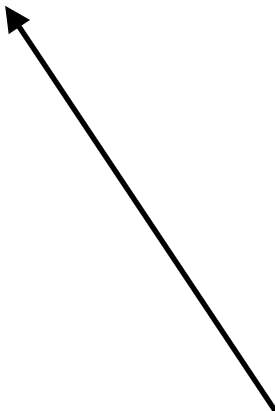
<i><b>titre</b></i>	<i><b>annee</b></i>	<i><b>realisateu</b></i>
Alien	1979	1
Vertigo	1958	2
Psychose	1960	2
Kagemusha	1980	3
Volte-face	1997	4
Pulp Fiction	1995	5
Titanic	1997	6
Sacrifice	1986	7



```
+-----+  
| titre_film |  
+-----+  
| Alien      |  
| Vertigo    |  
| Psychose   |  
| Kagemusha  |  
| Volte-face |  
| Pulp Fiction |  
| Titanic    |  
| Sacrifice  |  
+-----+
```

# Clause SELECT : attributs

```
SELECT Films.titre  
FROM Films
```



Optionnel (sauf en  
cas d'ambiguïté)

## Clause FROM : accéder à plusieurs tables

- Clause FROM : plusieurs tables dans la forme générale
- But : recomposer de l'information distribuée dans plusieurs tables
- Exemple : obtenir titre et réalisateur de tous les films à partir de ce schéma :

### Films

<i>titre</i>	<i>annee</i>	<i>id_realisate</i>
Alien	1979	1
Sacrifice	1986	6

### Artistes

<i>id</i>	<i>nom</i>	<i>prenom</i>	<i>naissan</i>
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899
3	Kurosawa	Akira	1910
4	Woo	John	1946
5	Cameron	James	1954
6	Tarkovski	Andrei	1932

## Clause FROM : accéder à plusieurs tables

- la clause FROM avec deux tables renvoie leur “produit” :
  - chaque ligne de la première table concaténée avec chaque ligne de la deuxième
  - appelé **produit cartésien**

```
SELECT * FROM Films, Artiste;
```

titre	année	lid_réalisateur	lid	nom	prénom	naissance
Alien	1979	1	1	Scott	Ridley	1943
Alien	1979	1	2	Hitchcock	Alfred	1899
Alien	1979	1	3	Kurosawa	Akira	1910
Alien	1979	1	4	Woo	John	1946
Alien	1979	1	5	Tarkovski	Andrei	1932
Alien	1979	1	6	Cameron	James	1954
Sacrifice	1986	6	1	Scott	Ridley	1943
Sacrifice	1986	6	2	Hitchcock	Alfred	1899
Sacrifice	1986	6	3	Woo	John	1946
Sacrifice	1986	6	4	Kurosawa	Akira	1910
Sacrifice	1986	6	5	Cameron	James	1954
Sacrifice	1986	6	6	Tarkovski	Andrei	1932

## Clause FROM : accéder à plusieurs tables

- La plupart du temps le produit cartésien contient des lignes “inutiles”
  - ▶ e.g. : pas significatif de concaténer “Alien” avec “Hitchcock”
- On peut utiliser la condition WHERE pour sélectionner uniquement les lignes du produit qui sont “reliées”

```
SELECT * FROM Films, Artistes
WHERE id_realisateur = id ;
```

Titre	année	id_realisateur	id	nom	prénom	naissance
Alien	1979	1	1	Scott	Ridley	1943
Sacrifice	1986	6	6	Tarkovski	Andrei	1932

## Clause FROM : accéder à plusieurs tables

- La plupart du temps le produit cartésien contient des lignes “inutiles”
  - ▶ e.g. : pas significatif de concaténer “Alien” avec “Hitchcock”
- On peut utiliser la condition WHERE pour sélectionner uniquement les lignes du produit qui sont “reliées”
- Ensuite la clause SELECT pour retenir uniquement les colonnes qui nous intéressent

```
SELECT titre, nom FROM Films, Artistes  
WHERE id_realisateur = id ;
```

+	-----	+	-----	+
	titre		nom	
+	-----	+	-----	+
	Alien		Scott	
	Sacrifice		Tarkovski	
+	-----	+	-----	+

## Clause FROM : accéder à plusieurs tables

- L'opération de produit cartésien (`FROM Table1, Tables2`) suivie d'une condition de sélection (`WHERE condition`) est appelée **JOINTURE** (JOIN)
- Syntaxe alternative pour la même requête

```
SELECT titre, nom  
FROM Films JOIN Artistes ON (id_realisateur = id );
```



## Clause FROM : renommage

```
SELECT F.titre  
FROM Films AS F
```

AS optionnel
--------------

Renommage des tables nécessaire si la même table est présente plusieurs fois dans la partie FROM, pour pouvoir distinguer les attributs

```
SELECT F1.titre  
FROM Film F1, Film F2  
WHERE F2.annee > F1.annee;
```

(les films qui ne sont pas le plus récents)

# Utilisation de fonctions prédéfinies dans les requêtes

On peut utiliser des fonctions dans les requêtes

Exemple : dans les clauses SELECT, WHERE ou dans une expression pour affecter des valeurs à des champs

Pour la plupart ce sont des ajouts de MySQL à la norme SQL

Exemples :

**ABS**(num) : valeur absolue

**CONCAT**(str, [str2,...]) : concaténation des chaînes

**NOW**() : la date et heure courante

...

# Utilisation de fonctions prédéfinies dans les requêtes

```
SELECT CONCAT ('réalisateur : ', nom)
FROM Artistes ;
```



```
+-----+
| concat('réalisateur : ', nom)|
+-----+
|réalisateur : Scott           |
|réalisateur : Hitchcock      |
|réalisateur : Kurosawa       |
+-----+
```

```
SELECT YEAR(NOW()) - naissance AS age
FROM Artistes ;
```



```
+-----+
| age  |
+-----+
| 73   |
| 117  |
| 106  |
+-----+
```

**Artistes**

<i>id</i>	<i>nom</i>	<i>prenom</i>	<i>naissan</i>
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899
3	Kurosawa	Akira	1910

# Quelques fonctions MySQL

**CEILING**(num)

Renvoie l'entier immédiatement supérieur ou égal à num

**FLOOR**(num)

Renvoie l'entier immédiatement inférieur ou égal à num

**CURDATE**()

Renvoie la date courante AAAAMMJJ ou AAAA-MM-JJ

**CURTIME**

Renvoie l'heure courante HHMMSS ou HH:MM:SS

**DATE\_FORMAT**(date, format)

Formate date selon format

# Quelques fonctions MySQL

**IF**(test, val1, val2)

Renvoie val1 si test est vrai, val2 sinon

**INSTR**(str, substr)

Position de substr dans str

**LENGTH**(str)

Renvoie la longueur de str

**STRCMP**(str1, str2)

0 si égalité, -1 si str1 < str2, +1 sinon

# Trier les résultats des requêtes : ORDER BY

```
mysql> SELECT titre, annee
```

```
-> FROM Films
```

```
-> WHERE titre BETWEEN 'Psychose' AND 'Titanic'
```

```
-> ORDER BY titre;
```

```
+-----+-----+
| titre          | annee |
+-----+-----+
| Psychose       | 1960  |
| Pulp Fiction   | 1994  |
| Sacrifice      | 1986  |
| Titanic        | 1997  |
+-----+-----+
```

# Trier les résultats des requêtes : ORDER BY

**Note** : le tri n'est pas lié à BETWEEN

On peut faire un tri sur plus d'une colonne

On peut trier dans l'ordre croissant (ASC) ou décroissant (DESC)

Exemple :

Liste les films par année et, dans une année, par ordre alphabétique inverse

```
SELECT annee, titre  
FROM Films  
ORDER BY annee ASC, titre DESC;
```

# Résultat

```
SELECT annee, titre FROM Films  
ORDER BY annee ASC, titre DESC;
```

annee	titre
1958	Vertigo
1960	Psychose
1979	Alien
1980	Kagemusha
1986	Sacrifice
1995	Pulp Fiction
1997	Volte-face
1997	Titanic



## Une condition de WHERE plus complexe : IN

Rechercher des attributs appartenant à un ensemble :

```
SELECT titre FROM Films
```

```
WHERE nom IN ('Hitchcock', 'Scott', 'Kurosawa');
```

Plus simple qu'une suite de OR

```
SELECT titre FROM film-simple
```

```
WHERE nom = 'Hitchcock' OR nom = 'Scott' OR nom = 'Kurosawa');
```

## Une condition de WHERE plus complexe : IN

Plus intéressant : les valeurs de l'ensemble dans lequel rechercher peuvent être le résultat d'une (sous-) requête

```
SELECT id_realisateur FROM Films  
WHERE titre IN (SELECT titre FROM Notation WHERE note > 5);
```

La condition IN peut être combinée avec d'autres conditions à l'aide des opérateurs booléens AND, OR, NOT

```
SELECT nom FROM Films, Artiste  
WHERE id = id_realisateur  
AND titre NOT IN (SELECT titre FROM Notation);
```

D'autres conditions complexes introduisent des sous-requêtes,...cf. cours BD L3

Un mot sur la modélisation des données

# Modéliser plusieurs concepts

Rappel : la table Films

**Films**

<i><b>titre</b></i>	<i><b>annee</b></i>	<i><b>realisateur</b></i>
Alien	1979	Scott
Vertigo	1958	Hitchcock
Psychose	1960	Hitchcock
Kagemusha	1980	Kurosawa
Volte-face	1997	Woo
Pulp Fiction	1995	Tarantino
Titanic	1997	Cameron
Sacrifice	1986	Tarkovski

Et si on voulait représenter plusieurs informations sur les réalisateurs (nom, prénom, date de naissance, ...) ?

# Modéliser plusieurs concepts

Pourquoi pas tout mettre dans la même table?

## Films

<i>titre</i>	<i>annee</i>	<i>realisateur</i>	<i>prenom</i>	<i>naissance</i>
Alien	1979	Scott	Ridley	1943
Vertigo	1958	Hitchcock	Alfred	1899
Psychose	1960	Hitchcock	Alfred	1899
Kagemusha	1980	Kurosawa	Akira	1910
Volte-face	1997	Woo	John	1946
Pulp Fiction	1995	Tarantino	Quentin	
Titanic	1997	Cameron	James	1954
Sacrifice	1986	Tarkovski	Andrei	1932

# Problèmes avec la table simple

## Redondance

Les informations sur les réalisateurs sont répétées pour chaque film qu'ils ont réalisé.

## Anomalies d'insertion

Possibilité d'insérer des données incohérentes

exemple : le même réalisateur avec deux dates de naissance différentes

## Anomalies de mise à jour

Si on a besoin de rectifier une erreur sur l'année de naissance, il faut penser à le faire pour tous les films. Sinon, la table contient des informations incohérentes...

## Anomalies de suppression

La suppression d'un film de la table, entraîne la suppression des informations associées sur le réalisateur.

Si le réalisateur n'était présent que pour un seul film, la suppression de ce film entraîne la disparition de toutes les informations relatives au réalisateur.

# Solution

Utiliser plusieurs tables pour représenter les films et les réalisateurs indépendamment les uns des autres

insertions, mises-à-jour et destructions indépendantes.

- Identifier les films (et les réalisateurs) pour s'assurer qu'aucun doublon ne figure dans nos tables
  - ▶ Films: 2 films ne peuvent avoir le même titre (supposons-le)
  - ▶ Réalisateurs : 2 réalisateurs peuvent avoir le même nom; on les distingue grâce à un identificateur (*id*)
- Lier les films et les réalisateurs sans introduire de redondance d'information

## Solution

Ajout d'un attribut dans la table film : “*realisateur*”

Il n'y a plus de redondance dans la base de données :

### Films

<i>titre</i>	<i>annee</i>	<i>realisateur</i>
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

### Realisateurs

<i>id</i>	<i>nom</i>	<i>prenom</i>	<i>naissance</i>
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899
3	Kurosawa	Akira	1910
4	Woo	John	1946
5	Tarantino	Quentin	
6	Cameron	James	1954
7	Tarkovski	Andrei	1932



# Solution

## Insertion

Les informations concernant un même réalisateur sont présentes une seule fois dans la base : pas possible de stocker des informations incohérentes (e.g. deux dates de naissance différentes)

ou bien il s'agit d'un réalisateur différent !

## Mise à jour

Il n'y a plus de redondance, donc une mise à jour ne risque pas d'introduire d'incohérence

## Suppression

La suppression d'un film n'affecte pas le réalisateur

# Modélisation

Remarque :

la modélisation ne concerne que le schema de la base de données  
pas les données (lignes)

## Films

<b>titre</b>	annee	realisateur
--------------	-------	-------------

## Realisateurs

<b>id</b>	nom	prenom	naissance
-----------	-----	--------	-----------

# Une schema de base de données plus complexe

- On veut représenter:
  - ▶ Des films,
  - ▶ Les réalisateurs et les acteurs qui jouent,
  - ▶ Les pays où ces films ont été réalisés,
  - ▶ Des utilisateurs du site des films
- Permettre aux utilisateurs de noter les films

## Solution : ébauche

- Les informations concernant les acteurs et les réalisateurs seront vraisemblablement les mêmes (nom, prénom, année de naissance)
  - ▶ on les représente avec une unique table Artistes

Artistes			
<b>id</b>	nom	prenom	naissance

- Avec les films on représente l'id de l'artiste qui en est le réalisateur

Films		
<b>titre</b>	annee	id_realisateur

- Comment représenter les acteurs qui jouent dans un film sans redondance?
  - ▶ Pourquoi la solution adoptée pour le réalisateur (ajouter son id dans la table Films) n'est pas valable?

## Solution : ébauche

- Un film a plusieurs acteurs, un attribut `id_acteur` peut représenter une seule valeur!
- Solution : une nouvelle table qui fait le “lien” entre films et acteurs
  - ▶ Seuls les identifiants dans cette table, pour éviter la redondance

### Cast

titre_film	id_acteur
------------	-----------

### Films

<b>titre</b>	annee	id_realisateur
--------------	-------	----------------

### Artistes

<b>id</b>	nom	prenom	naissance
-----------	-----	--------	-----------

- Et si on voulait représenter également les rôles des acteurs dans les film ?

## Solution : ébauche

- Le rôle n'est pas associé au film ou à l'acteur, mais à la participation de l'acteur dans un film.
  - ▶ attribut de la table Cast

### Cast

titre_film	id_acteur	rôle
------------	-----------	------

### Films

titre	annee	id_realisateur
-------	-------	----------------

### Artistes

id	nom	prenom	naissance
----	-----	--------	-----------

## Solution : ébauche

- Représentation des pays :

Pays		
code	nom	langue

- Et des utilisateurs : chaque utilisateur a un pseudo, nom, prénom, mdp, mais également un pays
  - ▶ comment représenter le pays?

## Solution : ébauche

- Représentation des pays :

Pays		
code	nom	langue

- Et des utilisateurs : chaque utilisateur a un pseudo, nom, prénom, mdp, mais également un pays
  - ▶ comment représenter le pays?

Utilisateurs					
pseudo	email	nom	prenom	mdp	code_pays



## Solution : ébauche

- Comment représenter les notes que les utilisateurs donnent aux films?
- Un utilisateur peut noter plusieurs films et un film peut être noté par plusieurs utilisateurs
  - ▶  $\Rightarrow$  la note ne peut pas être un attribut du film, ni de l'utilisateur

## Solution : ébauche

- Comment représenter les notes que les utilisateurs donnent aux films?
- Un utilisateur peut noter plusieurs films et un film peut être noté par plusieurs utilisateurs
  - ▶  $\Rightarrow$  la note ne peut pas être un attribut du film, ni de l'utilisateur
- Solution : une nouvelle table qui fait le “lien” entre films et utilisateurs
  - ▶ Seuls les identifiants dans cette table, pour éviter la redondance
  - ▶ la note est un attribut additionnel de cette table

### Notation

titre_film	pseudo	note
------------	--------	------

## Solution : schema complet

### Films

titre	annee	id_realisateur
-------	-------	----------------

### Artistes

id	nom	prenom	naissance
----	-----	--------	-----------

### Cast

titre_film	id_acteur
------------	-----------

### Pays

code	nom	langue
------	-----	--------

### Utilisateurs

pseudo	email	nom	prenom	mdp	code_pays
--------	-------	-----	--------	-----	-----------

### Notation

titre_film	pseudo	note
------------	--------	------

Ce schéma sera ensuite implementé dans le SGBD avec un suite de commandes CREATE TABLE, après avoir choisi le type de chaque attribut

# Modèles E/A

Pour simplifier le processus de modélisation en général on ne cherche pas à trouver les bonnes tables directement (comme dans l'exemple précédent)

On s'appuie sur des **modèles** dits “**Entités / Associations**” (ou E/A)

- modèles E/A (1976) à la base de méthodes de conception comme OMT (UML)
- plus haut-niveau que le modèle relationnel
- notions d'entité pour représenter les données d'intérêt et d'association pour représenter comment elles sont reliées

Il existe ensuite des règles qui nous permettent de traduire un schéma E/A en un schéma relationnel, et définir donc les tables de la base

cf. cours BD L3

# Interroger un schema complexe

- L'information repartie sur plusieurs tables sera “reconstruite” au moment de l'interrogation, par jointure

Les noms et les notes des utilisateurs qui ont noté le film “Alien” :

## Utilisateurs

pseudo	email	nom	prenom	mdp	code_pays
--------	-------	-----	--------	-----	-----------

## Notation

titre_film	pseudo	note
------------	--------	------

# Interroger un schema complexe

- L'information repartie sur plusieurs tables sera “reconstruite” au moment de l'interrogation, par jointure

Les noms et les notes des utilisateurs qui ont noté le film “Alien” :

## Utilisateurs

pseudo	email	nom	prenom	mdp	code_pays
--------	-------	-----	--------	-----	-----------

## Notation

titre_film	pseudo	note
------------	--------	------

```
SELECT nom, note
FROM Notation, Utilisateurs
WHERE titre_film = 'Alien'
AND Notation.pseudo = Utilisateurs.pseudo
```

## Alternative

```
SELECT nom, note
FROM Notation JOIN Utilisateurs
      ON (Notation.pseudo = Utilisateurs.pseudo)
WHERE titre_film = 'Alien'
```

# Interroger un schema complexe

- L'information repartie sur plusieurs tables sera “reconstruite” au moment de l'interrogation, par jointure

Les noms et les notes des utilisateurs qui ont noté les films de 1995 :

## Utilisateurs

pseudo	email	nom	prenom	mdp	code_pays
--------	-------	-----	--------	-----	-----------

## Films

titre	annee	id_realisateur
-------	-------	----------------

## Notation

titre_film	pseudo	note
------------	--------	------

# Interroger un schema complexe

- L'information repartie sur plusieurs tables sera “reconstruite” au moment de l'interrogation, par jointure

Les noms et les notes des utilisateurs qui ont noté les films de 1995 :

## Utilisateurs

pseudo	email	nom	prenom	mdp	code_pays
--------	-------	-----	--------	-----	-----------

## Films

titre	annee	id_realisateur
-------	-------	----------------

## Notation

titre_film	pseudo	note
------------	--------	------

```
SELECT nom, note
FROM Notation, Utilisateurs, Films
WHERE Notation.pseudo = Utilisateurs.pseudo
AND titre_film = titre
AND année = 1995
```



# Interroger un schema complexe

- L'information repartie sur plusieurs tables sera “reconstruite” au moment de l'interrogation, par jointure

Les noms et les notes des utilisateurs qui ont noté les films de ‘Tarantino’ :

## Films

titre	annee	id_realisateur
-------	-------	----------------

## Artistes

id	nom	prénom	naissanc
----	-----	--------	----------

## Utilisateurs

pseudo	email	nom	prenom	mdp	code_pays
--------	-------	-----	--------	-----	-----------

## Notation

titre_film	pseudo	note
------------	--------	------

# Interroger un schema complexe

- L'information repartie sur plusieurs tables sera “reconstruite” au moment de l'interrogation, par jointure

Les noms et les notes des utilisateurs qui ont noté les films de 'Tarantino' :

## Films

titre	annee	id_realisateur
-------	-------	----------------

## Artistes

id	nom	prénom	naissanc
----	-----	--------	----------

## Utilisateurs

pseudo	email	nom	prenom	mdp	code_pays
--------	-------	-----	--------	-----	-----------

## Notation

titre_film	pseudo	note
------------	--------	------

```
SELECT Utilisateurs.nom, note
FROM Notation, Utilisateurs, Films, Artistes
WHERE Notation.pseudo = Utilisateurs.pseudo
AND titre_film = titre
AND id_realisateur = id
AND Artistes.nom = 'Tarantino'
```