

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 concatener “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_réalisateur = id ;
```

Titre	année	id_réalisateur	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 concatener “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 réalisateur = 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 (réalisateur = 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 |  
+-----+-----+  
| Pulp Fiction  | 1995  |  
| Psychose      | 1960  |  
| 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 valeur 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 L2

Un mot sur la modélisation des données

Modéliser plusieurs concepts

Rappel : la table Films

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

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

Modéliser plusieurs concepts

Pourquoi pas tout mettre dans la meme 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

titre	annee	realisateur
--------------	-------	-------------

Realisateurs

id	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			
id	nom	prenom	naissance

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

Films		
titre	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

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

Artistes

id	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	naissan
----	-----	--------	---------

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

- Comment représenter les notes que les utilisateurs donnent aux films?

Solution : ébauche

- 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 implémenté dans le SGBD avec une 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/R 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'
```

Intégrité des données

Contraintes d'intégrité

- Contrainte d'intégrité : propriétés des données que l'on demande au système de garantir
- Exemples :
 - ▶ Un attribut doit toujours avoir une valeur (NOT NULL)
 - ▶ Un (ensemble d') attribut(s) identifie les lignes d'une table
 - ▶ (e.g. id dans la table Artiste)
 - ▶ Un attribut d'une table fait référence à l'identifiant d'une autre table
 - ▶ (e.g id_realisateur dans la table artiste)
 - ▶ Un attribut ne peut prendre qu'une des valeurs prédéfinies d'un ensemble
 - ▶ etc.

Contrainte NOT NULL

Si la valeur d'un attribut n'est pas spécifiée pendant l'insertion, la valeur "vide" NULL lui sera affectée

Exemple

```
INSERT INTO Utilisateurs (prenom, nom) VALUES ('Jean', 'Dupont')
```

Resultat :

Utilisateurs

pseudo	email	nom	prenom	mdp	code_pays
...
NULL	NULL	Dupont	Jean	NULL	NULL

La contrainte de NOT NULL sur un attribut d'une table impose que l'attribut ait une valeur non nulle

(génère une erreur si ce n'est pas le cas)

Contrainte NOT NULL

Spécifier des contraintes de NOT NULL en SQL

```
CREATE TABLE Utilisateur (  
    pseudo VARCHAR(50) NOT NULL,  
    email VARCHAR(50) NOT NULL,  
    nom VARCHAR(20) NOT NULL,  
    prenom VARCHAR(20),  
    mdp VARCHAR(60) NOT NULL,  
    naissance INTEGER,  
    code_pays INTEGER NOT NULL  
);
```

Contrainte NOT NULL et valeur par défaut

Une clause **DEFAULT** peut être spécifiée pour un attribut :

```
CREATE TABLE Notation (  
    titre_film VARCHAR(50) NOT NULL,  
    pseudo VARCHAR(50) NOT NULL,  
    note INTEGER NOT NULL DEFAULT 0  
);
```

Si la valeur de l'attribut n'est pas spécifiée lors d'une insertion, sa valeur sera celle définie par la clause DEFAULT.

INSERT INTO Notation VALUES ('Alien', 'jean87') ne génère pas d'erreur et insère la ligne :

Notation		
titre_film	pseudo	note
...
'Alien'	'jean87'	0

Contraintes de clé

- Une clé d'une table:
 - ▶ Plus petit sous-ensemble d'attributs permettant d'identifier une ligne de manière unique
- Exemples :
 - ▶ `nss` est une clef de la table `Personne(nss, nom, prénom)`
 - ▶ `(ville, rue, numero)` est une clef de la table `Bâtiment (ville, rue, numero, #etages)`
- Une table peut avoir plusieurs clefs
- Exemple :
 - ▶ `pseudo` est une clef de la table `Utilisateur (pseudo, e-mail, nom, prénom, mdp, code_pays)`
 - ▶ comme également `email`

Clés primaires

- Une table a toujours une clé dite

- ▶ clé primaire

(attributs soulignés ci-dessous)

Film (titre, année, id_realisateur)

Artiste (id, nom, prénom, naissance)

Utilisateur (pseudo, e-mail, nom, prénom, mdp, code_pays)

Pays (code, nom, langue)

Cast (id_film, id_acteur)

Notation (id_film, pseudo, note)

- ▶ les autres clés sont appelées **clefs candidates** (ou secondaires)

Spécifier les clés primaires en SQL

- Clé primaire comportant un seul attribut

```
CREATE TABLE Utilisateur (  
pseudo VARCHAR(50) PRIMARY KEY,  
email VARCHAR(50) NOT NULL,  
nom VARCHAR(20) NOT NULL,  
prenom VARCHAR(20),  
mdp VARCHAR(60) NOT NULL,  
naissance INTEGER,  
code_pays INTEGER NOT NULL  
);
```

Spécifier les clés primaires en SQL

- Clé primaire comportant plusieurs attributs

```
CREATE TABLE Notation (  
    titre_film VARCHAR(50),  
    pseudo VARCHAR(50),  
    note INTEGER NOT NULL DEFAULT 0,  
    PRIMARY KEY (titre_film, pseudo)  
);
```

- Remarque : **PRIMARY KEY** implique **NOT NULL**, pas besoin de le spécifier explicitement pour les attributs d'une clé
- Chaque table devrait avoir une clé primaire

Clés candidates

- Les autres clés de la table, pas choisies comme clés primaires, peuvent être spécifiées avec la contrainte UNIQUE

```
CREATE TABLE Artiste (  
    id INTEGER PRIMARY KEY,  
    nom VARCHAR(50) NOT NULL,  
    prenom VARCHAR(50) NOT NULL,  
    naissance INTEGER,  
    UNIQUE (nom, prenom, naissance)  
);
```

- (nom, prenom, naissance) : clé candidate
- Remarque : **UNIQUE** n'implique pas **NOT NULL**

Clés et clés candidates : erreurs

- Tentative d'insertion de clé primaire existante :

```
mysql> SELECT * FROM Artiste;
```

id	nom	prenom	naissance
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899

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

```
mysql> INSERT INTO Artiste (id, nom, prenom, naissance)
```

```
-> VALUES (2, 'Woo', 'John', 1946);
```

```
ERROR 1062 (23000):Duplicata du champ '2' pour la clef 1
```

Clés et clés candidates : erreurs

- Tentative d'insertion duplication clé candidate :

```
mysql> INSERT INTO Artiste (id, nom, prenom, naissance)  
-> VALUES (3, 'Hitchcock', 'Alfred', 1899);
```

```
ERROR 1062 (23000): Duplicata du champ `Hitchcock-  
Alfred-1899` pour la clef 2
```

Clés à incrémentation automatique

- MySQL offre la possibilité de définir des attributs dont la valeur est un entier automatiquement incrémenté à chaque insertion (pas présent dans le standard SQL)
- Très utile pour définir des identifiants “internes”

```
CREATE TABLE Artiste (  
    id INTEGER PRIMARY KEY AUTO_INCREMENT,  
    nom VARCHAR(50) NOT NULL,  
    prenom VARCHAR(50) NOT NULL,  
    naissance INTEGER,  
    UNIQUE (nom, prenom, naissance)  
);
```

- Pas besoin de fournir l'`id` lors de l'insertion d'un Artiste : il sera automatiquement affecté au `dernier id inséré + 1`

Clés à incrémentation automatique : insertion

Exemple. Supposer la table `Artiste` initialement vide.

```
mysql> INSERT INTO Artiste (nom, prenom, naissance)  
      -> VALUES ('Scott', 'Ridley', 1943);
```

Query OK, 1 row affected (0.03 sec)

```
mysql> INSERT INTO Artiste (nom, prenom, naissance)  
      -> VALUES ('Hitchcock', 'Alfred', 1899);
```

Query OK, 1 row affected (0.00 sec)

- Remarque : on ne précise pas la clé "id".

Clés à incrémentation automatique : insertion

```
mysql> SELECT * FROM Artiste;
```

```
+-----+-----+-----+-----+
| id     | nom           | prenom  | naissance |
+-----+-----+-----+-----+
|      1 | Scott        | Ridley  | 1943      |
|      2 | Hitchcock    | Alfred  | 1899      |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

- La clé a été automatiquement générée (séquence croissante)

Clés à incrémentation automatique : insertion

Une clé AUTO_INCREMENT peut également être précisée explicitement

```
mysql> INSERT INTO Artiste (id, nom, prenom, naissance)
```

```
-> VALUES (14, 'Woo', 'John', 1946);
```

Query OK, 1 row affected (0.00 sec)

```
mysql> SELECT * FROM Artiste;
```

id	nom	prenom	naissance
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899
14	Woo	John	1946

3 rows in set (0.00 sec)

Clés à incrémentation automatique : reprise incrémentation

```
mysql> INSERT INTO Artiste (nom, prenom, naissance)  
      -> VALUES ('Kurosawa', 'Akira', 1910);
```

Query OK, 1 row affected (0.00 sec)

```
mysql> SELECT * FROM Artiste;
```

ident	nom	prenom	naissance
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899
14	Woo	John	1946
15	Kurosawa	Akira	1910

Clés à incrémentation automatique : reprise incrémentation

La sequence continue toujours du dernier id inséré, même si sa ligne a été supprimée

```
mysql> DELETE FROM Artiste WHERE id > 2
```

Query OK, 1 row affected (0.00 sec)

```
mysql> INSERT INTO Artiste (nom, prenom, naissance)
-> VALUES ('Kurosawa', 'Akira', 1910);
```

Query OK, 1 row affected (0.00 sec)

```
mysql> SELECT * FROM Artiste;
```

ident	nom	prenom	naissance
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899
16	Kurosawa	Akira	1910

Clés à incrémentation automatique : reinitialisation

Pour réinitialiser le compteur de la séquence AUTO_INCREMENT :

```
mysql> ALTER TABLE Artiste AUTO_INCREMENT = 4;
```

Mais la nouvelle valeur du compteur doit être strictement supérieure à l'id maximum present dans la table

Clés à incrémentation automatique : reinitialisation

```
mysql> SELECT * FROM Artiste;
```

id	nom	prenom	naissance
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899

```
mysql> ALTER TABLE Artiste AUTO_INCREMENT = 4;
```

```
mysql> INSERT INTO Artiste (nom, prenom, naissance)  
-> VALUES ('Kurosawa', 'Akira', 1910);
```

```
mysql> SELECT * FROM Artiste;
```

id	nom	prenom	naissance
1	Scott	Ridley	1943
2	Hitchcock	Alfred	1899
4	Kurosawa	Akira	1910

Clés étrangères

Contrainte entre deux tables : sert à relier des attributs d'une table avec la clef primaire d'une autre table

```
CREATE TABLE Utilisateur (  
    pseudo VARCHAR(50) PRIMARY KEY,  
    email VARCHAR(50) NOT NULL,  
    nom VARCHAR(20) NOT NULL,  
    prenom VARCHAR(20),  
    mdp VARCHAR(60) NOT NULL,  
    naissance INTEGER,  
    code_pays INTEGER NOT NULL  
    FOREIGN KEY (code_pays) REFERENCES Pays(code)  
);
```

Impose que la colonne `code_pays` de la table Utilisateur contienne uniquement des valeurs qui apparaissent dans la colonne `code` de la table Pays

Clés étrangères

Les attributs reliés doivent avoir exactement le même type

L'attribut référencé (celui qui suit la clause REFERENCES) doit être une clef primaire de sa table

```
CREATE TABLE Notation (  
    titre_film VARCHAR(50),  
    pseudo VARCHAR(50),  
    note INTEGER NOT NULL DEFAULT 0,  
    PRIMARY KEY (titre_film, pseudo),  
    FOREIGN KEY (titre_film) REFERENCES Films(titre),  
    FOREIGN KEY (pseudo) REFERENCES Utilisateurs(pseudo),  
);
```

Similaire pour la table `Cast`

Clés étrangères

```
CREATE TABLE Films (  
    titre VARCHAR(50) PRIMARY KEY,  
    annee INTEGER NOT NULL,  
    id_realisateur INTEGER,  
    FOREIGN KEY (id_realisateur) REFERENCES Artiste (id)  
);
```

Clés étrangères

Conséquences de la contrainte de clé étrangère
(sur les tables Film/Artiste par exemple)

- Lors d'une insertion dans la table Films :
 - ▶ vérification : la valeur `id_realisateur` doit être parmi les `id` dans la table Artiste
- Lors de la mise à jour / suppression d'un `id` dans la table Artiste
 - ▶ vérification : aucun Film a cet `id` de réalisateur
- Si ces vérifications n'ont pas de succès
 - ▶ comportement par défaut : erreur
 - ▶ on peut demander explicitement un autre comportement (ON DELETE CASCADE....)
- Remarque : MySQL fait les vérifications de clé étrangère uniquement si les tables ont été créées avec le moteur **InnoDB**

Contraintes génériques

D'autres contraintes sur les données ne peuvent pas être exprimées par les mécanismes vus jusqu'à maintenant

Exemple :

la note qu'un utilisateur donne à un film est entre 0 et 5

le salaire d'un manager est plus élevé que celui de ses subalternes

Contraintes sur une seule table : [clause CHECK](#)

Contraintes sur plusieurs tables : [Assertions SQL](#)

Clause CHECK

- Clause CHECK de SQL
 - Mêmes expressions que la clause WHERE des requêtes SQL

- Exemple : dans la table Films:

```
CREATE TABLE Films (  
    titre VARCHAR(50) PRIMARY KEY,  
    annee INTEGER CHECK (annee BETWEEN 1980 AND 2000) NOT NULL,  
    genre VARCHAR(30)  
        CHECK (genre IN ('Documentaire', 'Western', 'Drame'));  
    id_realisateur INTEGER,  
    FOREIGN KEY (id_realisateur) REFERENCES Artiste (id)  
);
```

- CHECK accepté mais pas traité par MySQL