

Bases de données

2. Langage SQL déclaratif

Vincent Zucca

vincent.zucca@univ-perp.fr

Université de Perpignan Via Domitia

S3 Licence 2022-2023



1. Langage de définition de données (LDD)
2. Insertion, modification, suppression de données
3. Requête SELECT sur une seule table
4. Requête SELECT sur plusieurs tables
5. Agrégation
6. Union
7. Requêtes imbriquées

Introduction

- Ce chapitre présente le langage SQL de définition (création des tables), de manipulation de données (insertion, mise à jour, destruction) et d'interrogation (visualisation du contenu des tables).
- La syntaxe est celle de la norme SQL2, implantée plus ou moins complètement dans la plupart des principaux SGBDR et notamment MySQL.
- Lien utile : <https://dev.mysql.com/doc/refman/8.0/en/> (anglais)

1. Langage de définition de données (LDD)
2. Insertion, modification, suppression de données
3. Requête SELECT sur une seule table
4. Requête SELECT sur plusieurs tables
5. Agrégation
6. Union
7. Requêtes imbriquées

Création et utilisation d'une base de données

Une base de données est une ensemble de tables (ou relations).

- Création d'une base :

```
CREATE DATABASE nom_de_la_base;
```

- Suppression d'une base :

```
DROP DATABASE nom_de_la_base;
```

- Listing des bases de données présentes sur le système :

```
SHOW DATABASES;
```

- Chargement d'une base de données :

```
USE nom_de_la_base;
```

après le chargement on peut travailler (modifier ou afficher leurs contenus) sur les tables de la base de données chargée.

Création et utilisation d'une BD : exemple

Voici un exemple de création d'une base de donnée essai

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
+-----+
3 rows in set (0.028 sec)

mysql> CREATE DATABASE essai;
Query OK, 1 row affected (0.022 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| essai |
| information_schema |
| mysql |
| performance_schema |
+-----+
4 rows in set (0.001 sec)
```

Voici un exemple d'une destruction de BD

```
mysql> DROP DATABASE essai;
Query OK, 0 rows affected (0.001 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
+-----+
3 rows in set (0.001 sec)
```

Creation d'une table

- Voici la commande SQL pour créer une table :

```
CREATE TABLE nom_table(col1 type1, col2 type2,...,  
                        ...,colk typek);
```

qui contient k colonnes.

- Quelques types pour les colonnes :

- ▶ INT : nombre entier
- ▶ FLOAT : nombre flottant (i.e., chiffre à virgule 2.35).
- ▶ DECIMAL(M,D) nombre sur M positions, dont D après la virgule
- ▶ CHAR(n) : chaîne de n caractères
- ▶ VARCHAR(n) : chaîne d'au plus n caractères
- ▶ DATE : date au format AAAA-MM-JJ
- ▶ et bien d'autres

Suppression, description, listings de tables

- Pour connaître la définition d'une table :
`DESCRIBE nom_table;`
- Pour supprimer une table :
`DROP TABLE nom_table;`
- Pour lister les tables d'une base de données :
`SHOW TABLES;`

Exemple de création de table

```
mysql> USE Film;
Database changed
mysql> SHOW TABLES;
Empty set (0.000 sec)

mysql> CREATE TABLE Artiste(idArt INT,nom VARCHAR(50), prenom VARCHAR(50), anneeNais INT);
Query OK, 0 rows affected (0.277 sec)

mysql> SHOW TABLES;
+-----+
| Tables_in_Film |
+-----+
| Artiste         |
+-----+
1 row in set (0.000 sec)

mysql> DESCRIBE Artiste;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| idArt      | int(11)       | YES  |     | NULL    |       |
| nom        | varchar(50)   | YES  |     | NULL    |       |
| prenom     | varchar(50)   | YES  |     | NULL    |       |
| anneeNais  | int(11)       | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.001 sec)

mysql> DROP TABLE Artiste;
Query OK, 0 rows affected (0.137 sec)

mysql> SHOW TABLES;
Empty set (0.000 sec)
```

Modification de la définition d'une table

■ Ajout d'une colonne :

```
ALTER TABLE nom_table
```

```
ADD nom_col type_col [FIRST | AFTER col]
```

les éléments optionnels FIRST, AFTER permettent de préciser la place de la nouvelle colonne.

- ▶ En théorie l'ordre des colonnes n'a aucune importance.
- ▶ En pratique ordonner les colonnes peut faciliter la lecture (humaine) de la table.

■ Suppression d'une colonne

```
ALTER TABLE nom_table DROP COLUMN nom_col
```

■ Modification du type d'une colonne :

```
ALTER TABLE nom_table MODIFY nom_col NewType
```

Exemple de modification de table : ajout d'une colonne

```
mysql> CREATE TABLE Artiste(idArt INT,nom VARCHAR(50),  
-> prenom VARCHAR(50), anneeNais INT);  
Query OK, 0 rows affected (0.160 sec)
```

```
mysql> DESCRIBE Artiste;
```

Field	Type	Null	Key	Default	Extra
idArt	int(11)	YES		NULL	
nom	varchar(50)	YES		NULL	
prenom	varchar(50)	YES		NULL	
anneeNais	int(11)	YES		NULL	

4 rows in set (0.001 sec)

```
mysql> ALTER TABLE Artiste  
-> ADD Nationalite VARCHAR(25) AFTER prenom;  
Query OK, 0 rows affected (0.585 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> DESCRIBE Artiste;
```

Field	Type	Null	Key	Default	Extra
idArt	int(11)	YES		NULL	
nom	varchar(50)	YES		NULL	
prenom	varchar(50)	YES		NULL	
Nationalite	varchar(25)	YES		NULL	
anneeNais	int(11)	YES		NULL	

5 rows in set (0.001 sec)

Exemple de modification de table : suppression d'une colonne

```
mysql> DESCRIBE Artiste;
```

Field	Type	Null	Key	Default	Extra
idArt	int(11)	YES		NULL	
nom	varchar(50)	YES		NULL	
prenom	varchar(50)	YES		NULL	
Nationalite	varchar(25)	YES		NULL	
anneeNais	int(11)	YES		NULL	

```
5 rows in set (0.001 sec)
```

```
mysql> ALTER TABLE Artiste DROP COLUMN Nationalite;
```

```
Query OK, 0 rows affected (0.446 sec)
```

```
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> DESCRIBE Artiste;
```

Field	Type	Null	Key	Default	Extra
idArt	int(11)	YES		NULL	
nom	varchar(50)	YES		NULL	
prenom	varchar(50)	YES		NULL	
anneeNais	int(11)	YES		NULL	

```
4 rows in set (0.001 sec)
```

Exemple de modification de table : modification d'une colonne

```
mysql> DESCRIBE Artiste;
```

Field	Type	Null	Key	Default	Extra
idArt	int(11)	YES		NULL	
nom	varchar(50)	YES		NULL	
prenom	varchar(50)	YES		NULL	
anneeNais	int(11)	YES		NULL	

```
4 rows in set (0.001 sec)
```

```
mysql> ALTER TABLE Artiste MODIFY nom CHAR(100);
```

```
Query OK, 0 rows affected (0.487 sec)
```

```
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> DESCRIBE Artiste;
```

Field	Type	Null	Key	Default	Extra
idArt	int(11)	YES		NULL	
nom	char(100)	YES		NULL	
prenom	varchar(50)	YES		NULL	
anneeNais	int(11)	YES		NULL	

```
4 rows in set (0.001 sec)
```

Clef primaire

- Le mot clef `PRIMARY KEY` désigne la colonne qui sert de clef primaire (identifie de manière unique une ligne de la table).
- Si plusieurs colonnes (`col1,..., colk`) forment la clef primaire alors en fin de table on met `PRIMARY KEY(col1,...,colk)`
- Le mot clef `AUTO_INCREMENT` indique au système qu'il met la prochaine valeur non utilisée pour la valeur de la clef primaire.
- Exemple :

```
CREATE TABLE Artiste (  
    idArt INT PRIMARY KEY AUTO_INCREMENT ,  
    nom VARCHAR(60),  
    prenom VARCHAR(40),  
    anneeNais INT);
```

Clefs étrangères

- Lorsqu'un ou plusieurs champs d'une table correspondent à la clé primaire d'une autre table, on peut forcer le SGBD à vérifier la cohérence entre ces deux ensembles de valeur en ajoutant en fin de table :

```
FOREIGN KEY (col1, col2, ...)  
REFERENCES  foreign_tab (col1, col2, ...)
```

- Exemple :

```
CREATE TABLE Film(  
  id INT PRIMARY KEY,  
  titre VARCHAR(1000),  
  genre VARCHAR(20),  
  idArt INT,  
  FOREIGN KEY(idArt) REFERENCES Artiste(idArt));
```

Clefs étrangères (suite)

Au moment de la création d'une table, on peut spécifier (optionnel) comment des DELETE et UPDATE faits sur la table référencée sont répercutés sur la clef étrangère avec

```
ON DELETE [RESTRICT | CASCADE | SET NULL | NO ACTION | SET  
          DEFAULT]
```

```
ON UPDATE [RESTRICT | CASCADE | SET NULL | NO ACTION | SET  
          DEFAULT]
```

- CASCADE : indique que lors d'un DELETE/UPDATE, on détruit/met à jour aussi la ligne qui référence cette table.
- ON UPDATE { NO ACTION|CASCADE|SET NULL|SET DEFAULT}
pour un UPDATE on fait au choix : rien, on fait la même modification, on met à NULL ou on met la valeur par défaut.

Contraintes d'intégrité

Une contrainte d'intégrité est une clause permettant de contraindre les données insérées dans la table à vérifier certaines conditions.

- **DEFAULT** permet de définir une valeur par défaut lorsqu'une colonne de la table n'est pas renseigné.
- **NOT NULL** permet de spécifier qu'une colonne doit être saisie.
- **UNIQUE** impose de vérifier que la valeur saisie pour une colonne n'existe pas déjà dans la table.
- **CHECK** impose de vérifier si une condition plus générale est vérifiée.

Contrainte nommée

- Le mot clef `CONSTRAINT` permet de donner un nom à une contrainte (`PRIMARY` OR `FOREIGN KEY`, `CHECK` ou `UNIQUE`), on place alors toujours la contrainte en fin de table.
- Exemple :

```
CREATE TABLE Personne(  
  id INT PRIMARY KEY,  
  nom VARCHAR(50) NOT NULL,  
  prenom VARCHAR(50),  
  CONSTRAINT Unicite UNIQUE(nom,prenom));
```

1. Langage de définition de données (LDD)
2. Insertion, modification, suppression de données
3. Requête SELECT sur une seule table
4. Requête SELECT sur plusieurs tables
5. Agrégation
6. Union
7. Requêtes imbriquées

Insertion, modification, et destruction de données

Nous allons voir maintenant les commandes SQL qui permettent d'insérer ou mettre à jour des données dans les tables. Il y a trois commandes SQL essentiellement pour mettre à jour des données :

- INSERT pour insérer de nouvelles lignes,
- UPDATE pour mettre à jour des lignes existantes,
- DELETE pour supprimer des lignes.

Insertion avec INSERT

- La commande INSERT permet d'insérer des données dans une table. Sa syntaxe :

```
INSERT INTO nom_table [(colone1,[colone2,[...]])]  
VALUES (constantes1,[constante2,...])
```

- On spécifie les colonnes auxquelles on va affecter une valeur (les autres seront NULL). Et après VALUES on donne la liste de ces valeurs.

```
INSERT INTO Artiste (idArtiste,nom,prenom)  
VALUES (5,Kubrick, Stanley)
```

- Lorsqu'une table a une colonne qui est auto incrémentée **on ne spécifie pas la valeur de cette colonne** lorsque l'on insère une donnée. Par exemple :

```
CREATE TABLE Artiste( idArtiste INT AUTO_INCREMENT,  
                        Nom VARCAHR(20),  
                        Prenom VARACHAR(20),  
                        anneeNais INT,  
                        PRIMARY KEY(IdArtiste))  
INSERT INTO Artiste ("Durand", "Bernard", 30)
```

Modification avec UPDATE

- La commande UPDATE permet de modifier un attribut d'une table. Sa syntaxe est la suivante

```
UPDATE nom_table  
SET colonnes=constante  
[WHERE predicats]
```

- Par exemple si l'on veut modifier l'année de naissance d'Hitchcock

```
UPDATE artiste  
SET anneeNais=1920  
WHERE nom=Hitchcock
```

- La clause UPDATE permet de spécifier la table, la clause SET la ou les colonnes concernées et leur nouvelle valeur. Enfin la clause WHERE permet de désigner les lignes concernées.

Suppression avec DELETE

La commande DELETE permet de supprimer certaines lignes de la table.

```
DELETE FROM nom_table  
WHERE Predicat
```

La prédicat du WHERE permet de spécifier les lignes à supprimer.

1. Langage de définition de données (LDD)
2. Insertion, modification, suppression de données
3. Requête SELECT sur une seule table
4. Requête SELECT sur plusieurs tables
5. Agrégation
6. Union
7. Requêtes imbriquées

Requête SELECT

La requête de selection de SQL suit la syntaxe

```
SELECT A1,...,Ak  
FROM nom_table  
WHERE Predicat1,...,Predicatn
```

- FROM indique la ou les tables dans lesquelles on va sélectionner des lignes et colonnes.
- SELECT indique les colonnes A_i que l'on veut afficher.
- WHERE indique les conditions

Predicat1,...Predicatn

que doivent vérifier les lignes du résultat.

Exemples

- Pour afficher tout le contenu de la table Film on fait :

```
SELECT *  
FROM Film
```

- Par exemple pour extraire les titres des films de genre *horreur* on fait

```
SELECT titre  
FROM Film  
WHERE genre='horreur'
```

La clause SELECT - Suite

La clause SELECT spécifie les attributs qui vont apparaître dans le résultat. On peut modifier ces attributs des façons suivantes :

1. Appliquer des opérateurs numériques (+,-,×,/) ou des fonctions comme COS,EXP, etc.
2. Appliquer des opérateurs sur les chaînes (CONCAT concaténation, SUBSTRING extraction de sous-chaîne, LENGTH longueur de la chaîne, etc.).
3. Renommer ces attributs grâce au mot clef AS
`nom_attribut AS nouveau_nom`

Exemple

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

idArt	nom	anneeNais
1	Spielberg	1946
2	Demme	1944
3	Shyamalan	1970
4	Hitchcock	1899
5	Lawrence	1971

```
5 rows in set (0.000 sec)
```

```
mysql> SELECT nom,2022-anneeNais AS age FROM Artiste;
```

nom	age
Spielberg	76
Demme	78
Shyamalan	52
Hitchcock	123
Lawrence	51

```
5 rows in set (0.000 sec)
```

La clause SELECT - Suite

1. **Doublons.** Le langage SQL autorise l'apparition de doublons dans le résultat de requête SELECT. On peut forcer le résultat à supprimer les doublons avec le mot clef DISTINCT :

```
SELECT DISTINCT nom_colonne  
FROM nom_table
```

2. **Tri du Résultat.** On peut trier le résultat avec la clause ORDER BY suivit de la liste des attributs servant de critère au tri. Par exemple

```
SELECT *  
FROM Artiste  
ORDER BY anneeNais, Nom
```

Le tri s'effectue d'abord sur l'année de naissance, pour la même année alors le tri est effectué sur le nom de l'artiste.

Exemple - DISTINCT

On liste les différents genres de film de la base.

```
mysql> SELECT DISTINCT * FROM Film;
```

id	titre	genre	idArt
1	Le Silence des agneaux	thriller	2
2	Psychose	horreur	4
3	E.T. l extra-terrestre	sciences-fiction	1
4	Red Sparrow	thriller	5
5	Split	drame	3
6	Le sixieme sens	thriller	3
7	Les dents de la mer	horreur	1

```
7 rows in set (0.000 sec)
```



```
mysql> SELECT DISTINCT genre FROM Film;
```

genre
thriller
horreur
sciences-fiction
drame

```
4 rows in set (0.001 sec)
```

Exemple - ORDER BY

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

idArt	nom	anneeNais
1	Spielberg	1946
2	Demme	1944
3	Shyamalan	1970
4	Hitchcock	1899
5	Lawrence	1971

```
5 rows in set (0.000 sec)
```

```
mysql> SELECT * FROM Artiste ORDER BY anneeNais;
```

idArt	nom	anneeNais
4	Hitchcock	1899
2	Demme	1944
1	Spielberg	1946
3	Shyamalan	1970
5	Lawrence	1971

```
5 rows in set (0.000 sec)
```

La clause WHERE

La clause WHERE permet de spécifier des critères que doivent vérifier les attributs des tables définies dans FROM. On forme la condition avec :

- Opérateurs arithmétiques : $+$, $-$, $*$, $/$.
- Opérateurs logiques AND , OR, NOT, IN
- Les prédicats de comparaison :
 - ▶ $=$ (égal),
 - ▶ $<>$ (différent),
 - ▶ $<, >$ (supérieur/inférieur strict),
 - ▶ $<=, >=$ (supérieur/inférieur ou égal).
- Les prédicats d'intervalle BETWEEN min AND max
- L'appartenance à une liste année in ('1995', '1997', '1999')

Exemple

On affiche les Artiste nés entre 1945 t 1970.

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

idArt	nom	anneeNais
1	Spielberg	1946
2	Demme	1944
3	Shyamalan	1970
4	Hitchcock	1899
5	Lawrence	1971

```
5 rows in set (0.000 sec)
```

```
mysql> SELECT nom FROM Artiste  
WHERE AnneeNais >= 1945 AND AnneeNais <= 1970;
```

nom
Spielberg
Shyamalan

```
2 rows in set (0.000 sec)
```

La clause WHERE : prédicat sur les chaînes de caractères

Les prédicats sur les chaînes de caractères : pour s'assurer qu'une chaîne de caractères est bien d'un motif donné

- LIKE (ou NOT LIKE) motif : ou le motif est formé de caractères imprimables et de symboles spéciaux
 - ▶ _ = un caractère quelconque,
 - ▶ % = une chaîne quelconque.
- Par exemple la requête suivante sélectionne les films commençant par A.

```
SELECT titre  
FROM film  
WHERE titre LIKE 'A%'
```

- REGEXP (ou NOT REGEXP) : expressions régulières plus complètes mais non traitées dans ce cours.

Exemple - LIKE

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

idArt	nom	anneeNais
1	Spielberg	1946
2	Demme	1944
3	Shyamalan	1970
4	Hitchcock	1899
5	Lawrence	1971

```
5 rows in set (0.000 sec)
```

```
mysql> SELECT * FROM Artiste WHERE nom LIKE 'S%';
```

idArt	nom	anneeNais
1	Spielberg	1946
3	Shyamalan	1970

```
2 rows in set (0.000 sec)
```

Prédicat sur les dates

- Une date est spécifiée en SQL2 par le mot-clé DATE suivi d'une chaîne de caractères au format 'aaaa-mm-jj'.
- On les compare avec =, <>, >, <, >=, <=.
- Voici par exemple la requête qui donne les identifiants des artistes nés entre le 1998-07-01 et le 1998-07-31

```
SELECT idArtiste  
FROM Artiste  
WHERE dateNaiss BETWEEN DATE '1998-07-01'  
AND DATE '1998-07-31'
```

1. Langage de définition de données (LDD)
2. Insertion, modification, suppression de données
3. Requête SELECT sur une seule table
4. Requête SELECT sur plusieurs tables
5. Agrégation
6. Union
7. Requêtes imbriquées

Produit cartésien

- Ceci consiste à associer toutes les lignes d'une table $T1$ à chacune des lignes d'une table $T2$.
- Pour faire le produit de deux tables $T1$ et $T2$ on a juste à les mettre en argument de la clause `FROM` :

```
SELECT *  
FROM T1,T2
```

- On peut ensuite modifier la clause `SELECT` pour ne garder que certaines colonnes et ajouter une clause `WHERE` pour ne garder que certaines lignes.

Exemple de produit cartésien

On a une table avec 'a','b' et 'c' et une table avec 'un' et 'deux'.

```
mysql> SELECT * From Lettre;
+----+-----+
| id | lettre |
+----+-----+
| 1  | a      |
| 2  | b      |
| 3  | c      |
+----+-----+
3 rows in set (0.000 sec)
mysql> SELECT * From Chiffre;
+----+-----+
| id | chiffre |
+----+-----+
| 1  | un      |
| 2  | deux    |
+----+-----+
2 rows in set (0.000 sec)
```

On fait le produit cartésien.

```
mysql> SELECT * FROM Lettre, Chiffre;
+----+-----+----+-----+
| id | lettre | id | chiffre |
+----+-----+----+-----+
| 1  | a      | 1  | un      |
| 1  | a      | 2  | deux    |
| 2  | b      | 1  | un      |
| 2  | b      | 2  | deux    |
| 3  | c      | 1  | un      |
| 3  | c      | 2  | deux    |
+----+-----+----+-----+
6 rows in set (0.000 sec)
```

- La jointure est une des opérations les plus utiles puisqu'elle permet d'exprimer des requêtes portant sur des données réparties dans plusieurs tables.
- Ceci se fait en résolvant les références des clefs étrangères.
- La syntaxe de la requête de jointure est encore du type
`SELECT .. FROM .. WHERE`
 - ▶ On spécifie dans le `FROM` les tables que l'on joint.
 - ▶ On donne dans la clause `WHERE` la condition de jointure : qui oblige d'avoir une clef étrangère égale à la clef primaire de l'autre table.

Exemple de jointure

Par exemple la jointure sur la table des films et la table des Artistes sur idMES.

```
mysql> SELECT * FROM Film, Artiste WHERE Artiste.idArt = Film.idArt;
```

id	titre	genre	idArt	idArt	nom	anneeNais
1	Le Silence des agneaux	thriller	2	2	Demme	1944
2	Psychose	horreur	4	4	Hitchcock	1899
3	E.T. 1 extra-terrestre	sciences-fiction	1	1	Spielberg	1946
4	Red Sparrow	thriller	5	5	Lawrence	1971
5	Split	drame	3	3	Shyamalan	1970
6	Le sixieme sens	thriller	3	3	Shyamalan	1970
7	Les dents de la mer	horreur	1	1	Spielberg	1946

7 rows in set (0.023 sec)

Comme il y a ambiguïté pour l'attribut idArt dans la clause WHERE on a préfixé les attributs avec le nom de la table dont ils proviennent.

Jointure (suite)

On a la possibilité d'attribuer un nom a une table si celui la est trop long.
On a le contenu suivant dans les tables Film et Artiste.

```
mysql> SELECT * FROM Film;
```

id	titre	genre	idArt
1	Le Silence des agneaux	thriller	2
2	Psychose	horreur	4
3	E.T. l extra-terrestre	sciences-fiction	1
4	Red Sparrow	thriller	5
5	Split	drame	3
6	Le sixieme sens	thriller	3
7	Les dents de la mer	horreur	1

```
7 rows in set (0.000 sec)
```

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

idArt	nom	anneeNais
1	Spielberg	1946
2	Demme	1944
3	Shyamalan	1970
4	Hitchcock	1899
5	Lawrence	1971

```
5 rows in set (0.000 sec)
```

La requête suivante
affiche tous les titres de
film réalisés par
'Shyamalan'

```
mysql> SELECT titre  
FROM Film F, Artiste A  
WHERE F.idArt=A.idArt
```

```
AND A.nom = 'Shyamalan';
```

titre
Split
Le sixieme sens

```
2 rows in set (0.000 sec)
```

Jointure (suite) avec INNER JOIN

Il existe aussi la syntaxe `INNER JOIN` pour faire une jointure. Voici un exemple de jointure entre les tables `Artiste` et `Film`.

```
mysql> SELECT * FROM Film F INNER JOIN Artiste A ON F.idArt = A.idArt;
```

id	titre	genre	idArt	idArt	nom	anneeNais
1	Le Silence des agneaux	thriller	2	2	Demme	1944
2	Psychose	horreur	4	4	Hitchcock	1899
3	E.T. l extra-terrestre	sciences-fiction	1	1	Spielberg	1946
4	Red Sparrow	thriller	5	5	Lawrence	1971
5	Split	drame	3	3	Shyamalan	1970
6	Le sixieme sens	thriller	3	3	Shyamalan	1970
7	Les dents de la mer	horreur	1	1	Spielberg	1946

```
7 rows in set (0.000 sec)
```

1. Langage de définition de données (LDD)
2. Insertion, modification, suppression de données
3. Requête SELECT sur une seule table
4. Requête SELECT sur plusieurs tables
5. Agrégation
6. Union
7. Requêtes imbriquées

Agrégation

Le langage SQL permet d'exprimer des conditions sur des groupes de lignes d'une table, et de constituer le résultat par agrégation de valeurs au sein de chaque groupe :

1. La clause `GROUP BY` sert à partitionner une table en groupes de lignes selon certains critères.
2. La clause `HAVING` permet d'exprimer des conditions sur ces groupes de lignes.
3. Les fonctions d'agrégation permettent d'exprimer les résultats ou les conditions sur les groupes de lignes.

Fonctions d'agrégation

Ces fonctions s'appliquent à un attribut (en général de type numérique) d'un groupe de lignes. Ce sont :

1. COUNT qui compte le nombre de valeurs non NULL.
2. MAX et MIN qui déterminent la valeur maximale et minimale, respectivement.
3. AVG qui calcule la moyenne des valeurs de la colonne.
4. SUM qui effectue la somme.

Exemple d'utilisation de fonctions d'agrégation

Contenu de la table Film.

```
mysql> SELECT * FROM Film;
+-----+-----+-----+
| id | titre | genre | idArt |
+-----+-----+-----+
| 1 | Le Silence des agneaux | thriller | 2 |
| 2 | Psychose | horreur | 4 |
| 3 | E.T. 1 extra-terrestre | sciences-fiction | 1 |
| 4 | Red Sparrow | thriller | 5 |
| 5 | Split | drame | 3 |
| 6 | Le sixieme sens | thriller | 3 |
| 7 | Les dents de la mer | horreur | 1 |
+-----+-----+-----+
7 rows in set (0.000 sec)
```

Contenu de Artiste

```
mysql> SELECT *
FROM Artiste;
+-----+-----+-----+
| idArt | nom | anneeNais |
+-----+-----+-----+
| 1 | Spielberg | 1946 |
| 2 | Demme | 1944 |
| 3 | Shyamalan | 1970 |
| 4 | Hitchcock | 1899 |
| 5 | Lawrence | 1971 |
+-----+-----+-----+
5 rows in set (0.000 sec)
```

La requête pour le nombre de film réalisé par Spielberg

```
mysql> SELECT COUNT(titre) FROM Film F, Artiste A
WHERE F.idArt = A.idArt AND nom = 'Spielberg';
+-----+
| COUNT(titre) |
+-----+
| 2 |
+-----+
1 row in set (0.000 sec)
```

La requête pour l'année de naissance la plus petite.

```
mysql> SELECT MIN(A.anneeNais) FROM Film F, Artiste A WHERE F.idArt = A.idArt;
+-----+
| MIN(A.anneeNais) |
+-----+
| 1899 |
+-----+
1 row in set (0.000 sec)
```

La clause GROUP BY

- La clause GROUP BY sert à partitionner les lignes suivant la valeur d'un ou plusieurs attributs.
- Par exemple l'évaluation de la requête :

```
mysql> SELECT genre, COUNT(*) FROM Film GROUP BY genre;
```

genre	COUNT(*)
drame	1
horreur	2
sciences-fiction	1
thriller	3

4 rows in set (0.001 sec)

groupe les lignes suivant la valeur de `genre`. Sur chacun des groupe on compte avec `COUNT(*)` le nombre de lignes.

La Clause HAVING

- La clause HAVING permet d'exprimer des conditions sur les groupes de lignes formé par GROUP BY que l'on va considérer dans le résultat.
- Par exemple la requête suivante ne conserve que idArt des metteur en scène ayant réalisé plus d'un Film.

```
mysql> SELECT idArt FROM Film GROUP BY idArt HAVING COUNT(*) >= 2;
+-----+
| idArt |
+-----+
|     1 |
|     3 |
+-----+
2 rows in set (0.000 sec)
```

1. Langage de définition de données (LDD)
2. Insertion, modification, suppression de données
3. Requête SELECT sur une seule table
4. Requête SELECT sur plusieurs tables
5. Agrégation
6. Union
7. Requêtes imbriquées

Union et intersection

- Avec le mot clef UNION on peut faire l'union du résultat de deux requêtes SELECT.

```
(SELECT .. FROM .. WHERE .. ) UNION (SELECT .. FROM .. WHERE .. );
```

Le résultat contiendra les lignes des deux requêtes SELECT.

- Point important : il faut que les deux requêtes SELECT retournent le même type de résultat (même nombre de colonnes et mêmes types).

Exemple d'union

Contenu de la table Film.

```
mysql> SELECT * FROM Film;
+-----+-----+-----+-----+
| id | titre                | genre                | idArt |
+-----+-----+-----+-----+
| 1 | Le Silence des agneaux | thriller             | 2     |
| 2 | Psychose              | horreur              | 4     |
| 3 | E.T. l extra-terrestre | sciences-fiction     | 1     |
| 4 | Red Sparrow           | thriller             | 5     |
| 5 | Split                 | drame                | 3     |
| 6 | Le sixieme sens       | thriller             | 3     |
| 7 | Les dents de la mer   | horreur              | 1     |
+-----+-----+-----+-----+
7 rows in set (0.000 sec)
```

Contenu de Artiste

```
mysql> SELECT *
FROM Artiste;
+-----+-----+-----+
| idArt | nom                | anneeNais |
+-----+-----+-----+
| 1     | Spielberg          | 1946     |
| 2     | Demme              | 1944     |
| 3     | Shyamalan          | 1970     |
| 4     | Hitchcock          | 1899     |
| 5     | Lawrence           | 1971     |
+-----+-----+-----+
5 rows in set (0.000 sec)
```

Voici un exemple où l'on liste les films qui sont des films d'horreurs ou de sciences-fiction.

```
mysql> (SELECT titre FROM Film WHERE genre='horreur')
UNION (SELECT titre FROM Film WHERE genre='sciences-fiction');
+-----+
| titre                |
+-----+
| Psychose              |
| Les dents de la mer   |
| E.T. l extra-terrestre |
+-----+
3 rows in set (0.001 sec)
```

1. Langage de définition de données (LDD)
2. Insertion, modification, suppression de données
3. Requête SELECT sur une seule table
4. Requête SELECT sur plusieurs tables
5. Agrégation
6. Union
7. Requêtes imbriquées

Requêtes imbriquées

- La notion de requête imbriquée consiste à exprimer la clause WHERE d'une requête SELECT .. FROM .. WHERE .. en fonction du résultat d'une première requête SELECT .. FROM .. WHERE ...
- Exemple : la requête suivante liste les noms de metteur en scène ayant fait un film d'horreur (sans jointure) :
 1. Dans une première requête on liste les idArt ayant fait un film de genre 'horreur'.
 2. Dans la requête principale on liste les noms des artistes pour lesquels l'idArt appartient au résultat de la première requête.

```
mysql> SELECT nom FROM Artiste
WHERE idArt IN (SELECT idArt FROM Film WHERE genre='horreur');
+-----+
| nom   |
+-----+
| Spielberg |
| Hitchcock |
+-----+
2 rows in set (0.001 sec)
```

Prédicats sur le résultat d'une requête imbriquée

Il y a d'autres prédicats que l'appartenance (IN) que l'on peut appliquer à une requête imbriquée (R) :

1. EXISTS R. Renvoie TRUE si R n'est pas vide, FALSE sinon.
2. $t \text{ IN } R$ où t est un tuple dont le type est celui de R et renvoie TRUE si t appartient à R , FALSE sinon.
3. $v \text{ comp ANY } R$ ou comp est un comparateur SQL ($=, \leq$, etc.). Renvoie TRUE si la comparaison avec au moins une ligne de R est vraie.
4. $v \text{ comp ALL } R$ ou comp est un comparateur SQL ($=, \leq$, etc.). Renvoie TRUE si les comparaisons avec TOUTES les lignes de R sont vérifiées
5. Toutes les expressions précédentes peuvent être préfixées par NOT pour obtenir la négation.

Exemple de requête imbriquée

La requête suivante donne le réalisateur le nom du réalisateur le plus vieux.

```
mysql> SELECT nom FROM Artiste
WHERE anneeNais <= ALL (SELECT A.anneeNais FROM Artiste A, Film F WHERE A.idArt=F.idArt);
+-----+
| nom      |
+-----+
| Hitchcock |
+-----+
1 row in set (0.001 sec)
```

La requête imbriquée liste toutes les années de naissance des réalisateurs, la requête principale ne retient que le nom de l'artiste ayant une année inférieure à toutes les années de naissance.