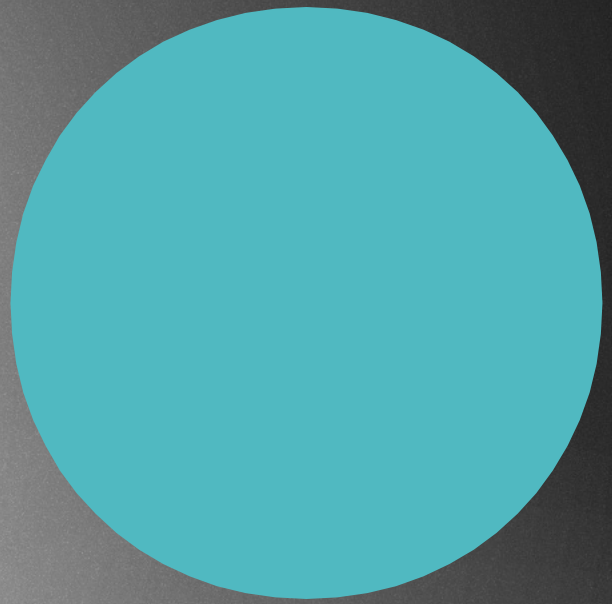
Base de données

MySQL



Qu’est ce qu’une base de données

- Il s’agit d’un ensemble de données stockées sur un support informatique
  - Exemple : Sur un serveur ou localement
- Une base de données permet à une application d’enregistrer diverses informations et d’y accéder.
  - Exemple : N’importe quel site internet ou application qui demanderait à l’utilisateur la saisie de données

Les SGDB

- Système de Gestion de Base de Données
  - C'est un système qui permet de manipuler des données dans une base de données.
    Groupe d'opérations CRUD : Create – Read – Update – Delete
  - Les données sont stockées dans des tables structurées
  - MySQL est un SGDB
- Basé sur un modèle Client-Serveur
  - La base de données se trouve sur un **serveur**
  - Pour interagir avec la base de données, un **client** est nécessaire
  - Les requêtes sont faites pas l'intermédiaire du **client**
  - Un langage est utilisé pour communiquer avec le serveur (SQL par exemple)
- Les requêtes sont les instructions envoyées afin d'interagir avec la base de données

Les SGDBR

- Système de Gestion de Base de Données Relationnel
  - Il s'agit tout simplement d'un SGBD gérant, en plus, les **relations**
  - Il est possible de définir des contraintes afin de garantir l'intégrité référentielle et fonctionnelle des données.

Table Utilisateur

id	nom	prenom
1	Bob	Eponge
2	Krabs	Capitaine
3	Etoile de mer	Patrick

Table Message

id	user_id_from	user_id_to	message
1	1	2	Hello Krabs
2	1	2	☺
3	3	1	Hey Bob !!!!!

- MySQL est donc un SGDBR

Présentation de MySQL

- Il s'agit d'un Système de Gestion de Base de Données Relationnel (SGDBR)
  - Le développement commence en 1994 par David Axmark et Michael Widenius
  - C'est un des SGDBR les plus utilisés
  - OpenSource, donc gratuit
  - Il existe des dizaines d'autres SGDBR (PostgreSQL, SQLite,…)

Mise en place d'un environnement de développement

Deux possibilités (nous travaillerons avec la seconde) :

- Télécharger et installer MySQL depuis le site officiel puis travailler en ligne de commande
- Télécharger et installer un serveur virtuel (Wamp, Xampp, Mamp,…) puis travailler depuis l'application Web de gestion pour les systèmes de gestion de base de données MySQL (PHPMyAdmin)
  - **Attention, dans tout les cas, nous allons écrire nous-mêmes nos requêtes SQL.**

Conventions de nommage

- **JAMAIS** d'espace ou d'accents pour les noms de bases de données, de tables ou de colonnes
  - Exemple : numéro de téléphone => numero_de_telephone
- Eviter les mots clés réservés au langage SQL
  - Exemple : date, text, type, format,…
- La **cohérence** est importante
  - Toutes les tables au singulier ou au pluriel
  - Mots séparés par un underscore (« _ ») ou une majuscule
    - Exemple: « mot_de_passe » ou « motDePasse »
- Mots clés en **MAJUSCULE** (SELECT, UPDATE, INSERT, FROM,…)
  - Donc les nom de tables, bases et colonnes en minuscule

Création, suppression et sélection d'une base de données

```sql
/*----- Création -----*/

CREATE DATABASE my_database;

CREATE DATABASE my_database CHARACTER SET 'utf8';


/*----- Suppression -----*/

DROP DATABASE my_database;

DROP DATABASE IF EXISTS my_database;

/*----- Utilisation (sélection) -----*/

USE my_database;
```

- En créant une base de données avec un certain encodage, les tables créées dans cette base auront le même encodage.
- L'utilisation de **IF EXISTS** est à privilégier pour éviter tout message d'erreur. Notamment si nous ne sommes plus sûr de l'existence de la base.
- Après l'utilisation de USE, toutes les actions seront effectuées sur la table sélectionnée.

Création de tables

1. **Définition des colonnes**

Liste non exhaustive des différents typages pour les colonnes :

Type	Description
VARCHAR(18)	Chaine de caractère de longueur 18
INT(10)	Entier de longueur 10
ENUM	Enumération de valeurs
TEXT	Champ de texte
DATETIME	Date et heure
DATE	Date uniquement
BOOLEAN	Booléen (vrai ou faux)
DECIMAL (10,2)	Nombre décimal ayant de longueur 10 ayant 2 chiffres après la virgule

Création de tables

2. **Valeur NULL et valeur par défaut (DEFAULT)**

- Il est possible de spécifier si la valeur d'une colonne peut être NULL ou non.
  - Si la valeur NULL n'est pas autorisée et qu'aucune valeur n'est spécifiée à l'ajout, une erreur sera retournée

- La valeur par défaut sera attribuée si aucune valeur n'est spécifié à l'ajout

Création de tables

3. **Moteurs de tables**

- Ce sont des moteurs de stockage, ils permettent de gérer différemment les tables suivant l'utilisation que l'on en fait.

- Les deux principaux moteurs sont :

  - **MyISAM**

    Moteur par défaut, les commandes d'insertions et de sélections sont plus rapides. Cependant, il ne gère pas les clés étrangères qui permettent de vérifier l'intégrité d'une référence d'une table à l'autre (relation entre 2 tables).

  - **InnoDB**

    Ce moteur gère les clés étrangères. Il est plus lent que MyISAM et plus gourmand.

Création de tables

4. **Clé primaire et auto-incrémentation**

- La **clé primaire** est une contrainte d'unicité. La clé primaire permet d'identifier de manière unique une ligne donnée dans une table.

- L'**auto-incrémentation** est couplée avec la clé primaire, elle débute à la valeur de notre choix et s'incrémente à chaque ajout d'une ligne. MySQL va donc prendre l'identifiant de la dernière valeur insérée puis l'ajouter de 1.

Syntaxe de création et suppression d'une table

```sql
----- Création -----

CREATE TABLE film (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    titre VARCHAR(40) NOT NULL,
    genre VARCHAR(40) NOT NULL,
    date_sortie DATE NOT NULL,
    commentaire TEXT,
    PRIMARY KEY (id)
)
CHARACTER SET 'utf8'
ENGINE = INNODB;

----- Suppression -----

DROP TABLE film;

DROP TABLE IF EXISTS film;

----- Afficher toutes les tables -----

SHOW TABLES;

----- Lister les colonnes d'une table -----

DESCRIBE film;
```

- **UNSIGNED** signifie que la variable ne pourra pas être négative.
- Il est également possible de créer la table comme vu précédemment puis de la modifier pour y ajouter des colonnes.

Modification d’une table

1. **Précision générale**

- Il est tout à fait possible de modifier une table après sa création.
  Il est tout de même préférable de **réfléchir avant** à la structure de la table afin de ne pas avoir à la modifier par la suite.
- La modification d’une table inclue la modification d’une colonne, d’un index ou même d’une contrainte.

Insertion de données

```sql
----- Création -----

CREATE TABLE film (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    titre VARCHAR(40) NOT NULL,
    genre VARCHAR(40) NOT NULL,
    date_sortie DATE NOT NULL,
    commentaire TEXT,
    PRIMARY KEY (id)
)
CHARACTER SET 'utf8'
ENGINE = INNODB;

----- Ajout sans spécifier de colonnes -----

INSERT INTO film
VALUES (1, 'Mission Impossible', 'Action', '2012-06-07', 'Super film !');

----- Ajout en spécifiant les colonnes -----

INSERT INTO film (id, titre, genre, date_sortie, commentaire)
VALUES (1, 'Mission Impossible 2', 'Action', '2013-06-07', NULL);

INSERT INTO film (titre, genre, date_sortie, commentaire)
VALUES ('Mission Impossible 3', 'Action', '2014-06-07', 'Trop bien !')

----- Ajout multiple -----

INSERT INTO film (titre, genre, date_sortie, commentaire)
VALUES  ('Mission Impossible 4', 'Action', '2016-06-07', NULL),
        ('Mission Impossible 5', 'Action', '2018-06-07', 'Bof !');


----- Syntaxe alternative (non recommandé) -----

INSERT INTO film
SET titre = 'Mission Impossible 6', genre = 'Action', date_sortie = '2019-06-07', commentaire = NULL;
```

- Il est possible de spécifier les colonnes dans lesquelles ajouter des donner ou alors ajouter directement dans toutes les colonnes.

  Attention : Si des colonnes ne sont pas spécifiées et qu'aucune valeur pas défaut n'a été saisie, une erreur sera renvoyée.

- La syntaxe alternative n'est pas **recommandé** :
  - Pas d'ajout multiple
  - Propre à MySQL (problème de compatibilité)

Sélection de données - SELECT

1. **Sélection simple**

- La requête SELECT permet d'afficher directement des données ou de sélectionner des données à partir d'une table

```
3  ----- Afficher des données -----
4
5  SELECT "Hello World !";
6  SELECT 6+2;
7  SELECT 3*8;
8  SELECT (6+3) * (3*2)/5;
9
10
11 ----- Afficher des données à partir d'une table -----
12
13
14 -- Sélectionner des colonnes --
15
16 SELECT titre, date_sortie
17 FROM film;
18
19 -- Afficher toutes les colonnes --
20
21 SELECT *
22 FROM film;
```

Sélection de données - SELECT

2. **La clause WHERE**

- Permet de spécifier des critères de recherche

```
2  ----- Sélectionner en fonction du titre -----
3
4  SELECT *
5  FROM film
6  WHERE titre = "Mission Impossible";
7
8  SELECT *
9  FROM film
10 WHERE titre != "Mission Impossible";
11
12
13 ----- Sélectionner en fonction d'une date -----
14
15 SELECT *
16 FROM film
17 WHERE date_sortie < '2012-06-01';
18
19 SELECT *
20 FROM film
21 WHERE date_sortie >= '2013-04-02';
```

Opérateur	Signification
=	Egal
>	Strictement supérieur
<	Strictement inférieur
<=	Inférieur ou égal
>=	Supérieur ou égal
!= ou <>	Différent
<=>	Egal (est valable pour NULL)

Sélection de données - SELECT

3. **Combinaison de critères (avec des opérateurs logiques)**

```
----- Opérateur logique AND (ou &&) -----

SELECT *
FROM film
WHERE titre = "Mission Impossible"
AND titre "Mission Impossible 2";

SELECT *
FROM film
WHERE titre != "Mission Impossible"
AND date_sortie >= '2013-04-02';


----- Opérateur logique OR (ou ||) -----

SELECT *
FROM film
WHERE date_sortie < '2012-06-01'
OR titre = "Mission Impossible";

SELECT *
FROM film
WHERE date_sortie >= '2013-04-02';

----- Opérateur logique NOT (ou !) -----

SELECT *
FROM film
WHERE NOT titre = "Mission Impossible";

----- Opérateur logique XOR -----

SELECT *
FROM film
WHERE titre = "Mission Impossible"
XOR date_sortie > '2013-06-06';
```

Opérateur	Symbol	Signification
AND	&&	ET Exemple : « Sélectionner tout quand telle valeur vaut 3 ET telle valeur vaut 4 »
OR	||	OU Exemple : « Sélectionner tout quand telle valeur vaut 18 OU telle valeur vaut 12 »
XOR		OU EXCLUSIF Exemple : « Sélectionner tout quand telle valeur vaut 15 OU telle valeur vaut 12 MAIS PAS LES DEUX EN MÊME TEMPS »
NOT	!	NON Exemple : « Sélectionner tout quand telle valeur NE VAUT PAS 15 »

Sélection de données - SELECT

4. **Combinaison de plusieurs critères et valeur NULL**

```sql
----- Plusieurs critères (1) -----

SELECT *
FROM film
WHERE date_sortie > '2010-01-01'
AND (
    genre = "Action"
    OR
    commentaire IS NOT NULL)


----- Plusieurs critères (2) -----

SELECT *
FROM film
WHERE date_sortie > '2010-01-01'
AND (
    genre = "Comédie"
    OR
    commentaire IS NULL)

----- Plusieurs critères (3) -----

SELECT *
FROM film
WHERE date_sortie > '2010-01-01'
AND (
    genre = "Comédie"
    OR
    commentaire <=> NULL)
```

1) Tous les films sorties après le 1er Janvier 2010 et ayant comme titre « Mission Impossible » OU ayant reçu un commentaire.
2) Tous les films sorties après le 1er Janvier 2010 et ayant comme titre « Mission Impossible » OU n'ayant reçu aucun commentaire.
3) Tous les films sorties après le 1er Janvier 2010 et ayant comme titre « Mission Impossible » OU n'ayant reçu aucun commentaire.

- **IS NULL** et **IS NOT NULL** permettent de cibler les valeurs possiblement NULL.

  IS NULL et **<=>** sont équivalent.

Sélection de données - SELECT

5. **Trier les données – ORDER BY (ASC et DESC)**

```sql
----- Tri ASCENDANT sur la colone genre (1) -----

SELECT *
FROM film
WHERE date_sortie > '2010-01-01'
AND genre = "Comédie"
ORDER BY genre ASC;

----- Tri DESCENDANT sur la colone date_sortie (2) -----

SELECT *
FROM film
WHERE date_sortie > '2010-01-01'
AND genre = "Comédie"
ORDER BY date_sortie DESC;

----- Tri sur plusieurs colonnes (3) -----

SELECT *
FROM film
WHERE date_sortie > '2010-01-01'
AND genre = "Comédie"
ORDER BY genre, date_sortie ASC;
```

- Le tri **ASCENDANT** permet d'organiser par ordre alphabétique, du nombre plus petit au plus grand ou bien de la date la plus ancienne à la plus proche. Il s'agit du **tri par défaut**.
- Le tri **DESCENDANT** est donc l'inverse. Ordre anti-alphabétique, du plus grand nombre au plus petit et de la date la plus proche à la plus éloignée.
- **Le tri sur plusieurs colonnes se fait dans l'ordre des colonnes spécifiées**. Dans notre cas, nous allons obtenir les films classés par genre puis par date de sortie.

Sélection de données - SELECT

5. **Limiter les résultats et éliminer les doublons**

```sql
----- Limiter les résultats -----

SELECT *
FROM film
WHERE date_sortie > '2010-01-01'
AND genre = "Comédie"
ORDER BY genre ASC;
LIMIT 10 OFFSET 2;

SELECT *
FROM film
WHERE date_sortie > '2010-01-01'
AND genre = "Comédie"
ORDER BY genre ASC;
LIMIT 1;

----- Eliminer les doublons -----

SELECT genre
FROM film;

/*
Retournera tous les genres donc
plusieurs fois "Action", "Comédie", etc...
*/

SELECT DISTINCT genre
FROM film;
```

- Pour limiter les résultats, on utilise **LIMIT** suivit de la limite à imposer. Cela permet de retourner un nombre défini de résultats, quand bien même il y en aurait 100 fois plus.
  Par défaut, **OFFSET** est à 0, c'est un paramètre facultatif.

- **SELECT DISTINCT** ne retournera que des résultats uniques pour les colonne concernées. Dans notre cas, **DISTINCT** ne s'appliquera que sur la colonne « genre ».

Sélection de données - SELECT

5. **Autres possibilités de la clause WHERE**

- Rechercher approximativement avec **LIKE**
  - Avec **LIKE**, il est possible d'effectue une recherche avec des caractères représentant d'autres caractères.
  - LIKE est **insensible à la casse**.
    Si besoin, on utilisera **LIKE BINARY**.
  - Ce sont des **jokers**.
    Il en existe deux :
    - '%' qui représente n'importe quelle chaîne de caractère
    - '_' qui représente un seul caractère

Exemple	Signification
'u_'	Rechercher toutes les chaînes ayant 2 caractères dont le premier commence par U
'A%'	Rechercher toutes les chaînes de caractères commençant par « A »
'%hey%'	Rechercher toutes les chaînes de caractères contenant « hey »
'B_n_'	Rechercher toutes les chaînes de caractères commençant par « B » suivit d'un caractère puis de « n » puis d'un autre caractère
'%sa%de'	Rechercher toutes les chaînes de caractères contenant « sa » et terminant par « de »

Sélection de données - SELECT

5. **Autres possibilités de la clause WHERE**

- Rechercher approximativement avec **LIKE** – Exemples :

```
2  ----- Recherches approximatives -----
3
4  SELECT *
5  FROM film
6  WHERE date_sortie > '2010-01-01'
7  AND genre LIKE '%me%';
8
9
10 SELECT *
11 FROM film
12 WHERE date_sortie > '2010-01-01'
13 AND genre LIKE BINARY '_H';
14
15
16 SELECT *
17 FROM film
18 WHERE date_sortie > '2010-01-01'
19 AND genre LIKE '%he%co_';
```

Sélection de données - SELECT

5. **Autres possibilités de la clause WHERE**

- Rechercher dans un **intervalle**
  - Pour effectuer une recherche dans un intervalle, on utilisera :
    BETWEEN minimum AND maximum
  - Il est également possible d'utiliser les opérateurs <= et >=. Néanmoins, la requête sera plus performante et plus facile à lire avec BETWEEN
  - Fonctionne avec les dates, les nombres et les chaînes de caractères

```
2  ----- Recherches dans un intervalle -----
3
4  SELECT *
5  FROM film
6  WHERE date_sortie >= '2010-01-01'
7  AND date_sortie <= '2012-01-01'
8
9  SELECT *
10 FROM film
11 WHERE date_sortie BETWEEN '2010-01-01' AND '2012-01-01';
12
13 ----------
14
15 SELECT *
16 FROM film
17 WHERE id >= 5
18 AND id <= 10
19
20 SELECT *
21 FROM film
22 WHERE id BETWEEN 5 AND 10;
```

Modification de données - UPDATE

- Pour mettre à jour/modifier des données on utilise la commande **UPDATE**.
  L'action est **irréversible.**
- Pour spécifier correctement les données à modifier, on utilise la clause **WHERE.**
  Sans la clause WHERE, l'intégralité de la table sera modifiée.

```
2  ----- Modifier un seul élément avec son id -----
3
4  UPDATE film
5  SET genre = "Comédie", commentaire = "Pas si bien que ça finalement"
6  WHERE id = 2;
7
8  ----- Modifier plusieurs éléments suivant un critère -----
9
10 UPDATE film
11 SET genre = "Action/Aventure"
12 WHERE genre = "Action";
13
14 ----- Modifier toutes les données -----
15
16 UPDATE film
17 SET genre = "Aucun genre";
```

Suppression de données - DELETE

- Pour supprimer des données on utilise la commande **DELETE.**
  L'action est **irréversible.**
- Pour spécifier correctement les données à supprimer, on utilise la clause **WHERE.**
  Sans la clause WHERE, l'intégralité de la table sera supprimée.

```
2   ----- Supprimer un seul élément avec son id -----
3
4   DELETE FROM film
5   WHERE id = 2;
6
7   ----- Supprimer plusieurs éléments suivant un critère -----
8
9   DELETE FROM film
10  WHERE genre = "action";
11
12  ----- Supprimer toutes les données -----
13
14  DELETE FROM film;
```

Les index

- Un index est une structure entretenue automatiquement, qui permet de localiser facilement des enregistrements.

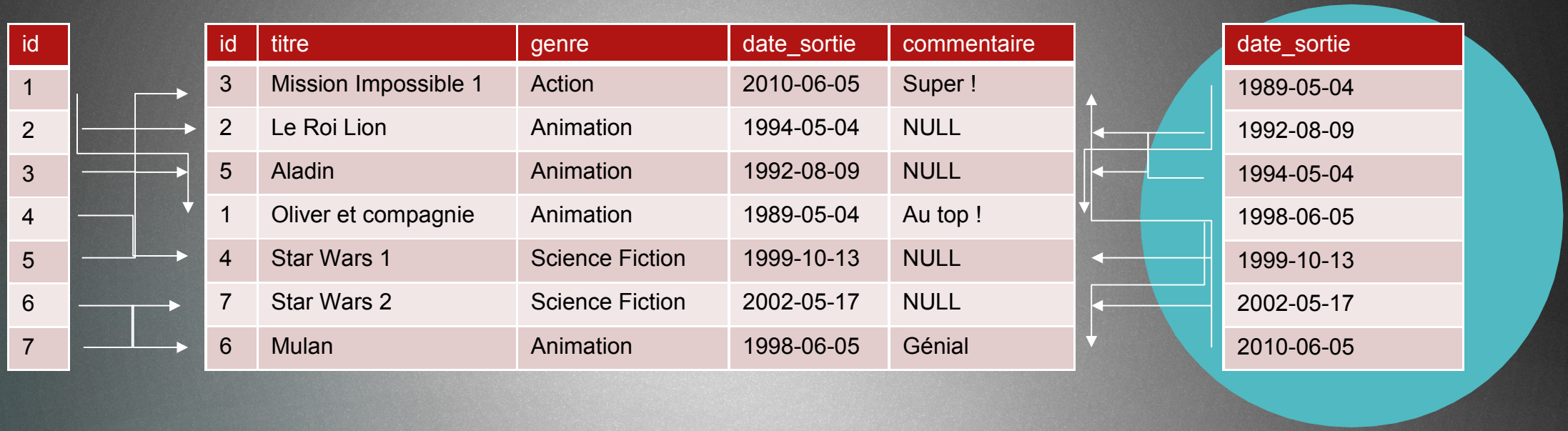
  Exemple: A titre de comparaison, c'est comme si nous allions rechercher nos données dans la tables des matières plutôt que de les chercher une à une.

- Les index sont utilisés par les SGBD pour de nombreuses opérations. Les index facilitent les opérations de tri, de recherche, de regroupement et de jointure.

- Les index sont **indispensables** pour la création de clés **primaires** et **étrangères**.

- Les index peuvent se **cumuler** sur plusieurs colonnes

Les index



id
1
2
3
4
5
6
7

id	titre	genre	date_sortie	commentaire
3	Mission Impossible 1	Action	2010-06-05	Super !
2	Le Roi Lion	Animation	1994-05-04	NULL
5	Aladin	Animation	1992-08-09	NULL
1	Oliver et compagnie	Animation	1989-05-04	Au top !
4	Star Wars 1	Science Fiction	1999-10-13	NULL
7	Star Wars 2	Science Fiction	2002-05-17	NULL
6	Mulan	Animation	1998-06-05	Génial

date_sortie
1989-05-04
1992-08-09
1994-05-04
1998-06-05
1999-10-13
2002-05-17
2010-06-05

- L’index sur l’ID est trié par ordre croissant ce qui permet d’améliorer grandement l’efficacité des recherches.
- Il est tout à fait possible d’avoir plusieurs index. Ici le second index sur la colonne « date_sortie ».
- A la création d’un index, MySQL stocke ce dernier sous forme d'une structure particulière contenant les valeurs de la ou les colonnes impliquées dans l’index. Cette structure stocke les valeurs **triées** et permet d'accéder à chacune de manière efficace et rap*ide*.

Les index – Intérêts et inconvénients

- Accélérer les requêtes qui utilisent des colonnes indexées comme critères de recherche.
  - **Exemple :** Si dans une application, une colonne est sujette à de nombreuses recherches. Il est préférable d'ajouter un index sur cette colonne afin d'optimiser les recherches.
- Les index permettent de garantir l'intégrité des données.
  - Il existe plusieurs types d'index : les simples, UNIQUE, FULLTEXT et SPATIAL
  - Il existe également 2 clés : **primaire** et **étrangère**.
- Les index ralentissent les requêtes d'insertion, de modification et de suppression.
  - Il faut remettre l'index à jour à chaque fois
- Ils prennent également de la place en mémoire

Les index – Exemple

- Reprenons notre table « film » :

id | titre | genre | date_sortie | commentaire

- Dans notre exemple, imaginons que nous avons besoin de faire de nombreuses recherches sur les colonnes suivantes :
  - Titre
  - Titre et genre
  - date_sortie et genre

Nous allons donc avoir besoin de **3 index**

titre
Aladin
Le Roi Lion
Mission Impossible 1
Mulan
Oliver et compagnie
Star Wars 1
Star Wars 2

titre	genre
Aladin	Animation
Le Roi Lion	Animation
Mission Impossible 1	Action
Mulan	Animation
Oliver et compagnie	Animation
Star Wars 1	Science Fiction
Star Wars 2	Science Fiction

date_sortie	genre
1989-05-04	Animation
1992-08-09	Animation
1994-05-04	Animation
1998-06-05	Animation
1999-10-13	Science Fiction
2002-05-17	Science Fiction
2010-06-05	Action

Les index – UNIQUE, FULLTEXT et SPATIAL

- L'index UNIQUE permet de s'assurer que la donnée saisie ne sera pas en doublon.
  Exemple : On peut mettre un index UNIQUE sur la colonne email d'un utilisateur. En effet il ne doit surtout pas y avoir 2 emails identiques dans notre table utilisateur.

- L'index FULLTEXT est utilisé pour faire des recherches de manière puissante et rapide sur un texte. On l'utilise sur les colonnes de type VAR, VARCHAR et TEXT.
  - Attention toutefois, il n'y a pas de « d'index par la gauche » avec FULLTEXT. Il faudra donc créer autant d'INDEX que nécessaire si nous avons des recherches à faire sur des colonnes spécifiques.

- **Index par la gauche :** Lors de la création d'un index sur 2 colonnes ou plus, la recherche se fera par la gauche suivant les index existants. Si l'on commence par la droite ou que l'on saute une colonne, un index supplémentaire aura besoin d'être créé.

Créer et supprimer un index

- La création d'un index peut se faire de plusieurs manières :
  - A la création de la table
  - En les ajoutant après la création
- Lors de la création d'un index, le nom n'est pas obligatoire. S'il n'est pas précisé, MYSQL va en créer un.
- Il est également possible de créer un index sur **un certain nombre de caractères**. En effet, parfois les 20 premiers caractères peuvent suffire à faire le tri. Autrement MySQL va indexer tous les caractères.

```sql
----- Créer un index lors de la création de la table -----

CREATE TABLE film (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    titre VARCHAR(40) UNIQUE NOT NULL,
    genre VARCHAR(40) NOT NULL,
    date_sortie DATE NOT NULL,
    commentaire TEXT,
    PRIMARY KEY (id)
)
CHARACTER SET 'utf8'
ENGINE = INNODB;

-- Liste d'index avec un index sur les 10 premiers caractères --

CREATE TABLE film (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    titre VARCHAR(40) UNIQUE NOT NULL,
    genre VARCHAR(40) NOT NULL,
    date_sortie DATE NOT NULL,
    commentaire TEXT,
    PRIMARY KEY (id),
    INDEX index_titre_genre (titre,genre),
    INDEX index_titre (titre(10))
)
CHARACTER SET 'utf8'
ENGINE = INNODB;

----- Créer un index après la création de la table -----

ALTER TABLE film
ADD INDEX index_titre_genre (titre, genre);

ALTER TABLE film
ADD FULLTEXT index_commentaire (commentaire);

CREATE INDEX index_titre_genre
ON film (titre, genre);

CREATE UNIQUE INDEX index_titre
ON film (titre);

----- Créer un index après la création de la table -----

ALTER TABLE film
DROP INDEX index_titre;
```

Les clés

- Il existe deux types de clés :
  - Les clés **primaires**
    - Il est important de définir une clé primaire **sur chaque table**.
      C'est pour cela que l'on trouve souvent la colonne « id » qui est une **clé primaire incrémentée**.
    - Il ne peut y avoir qu'**une seule clé primaire** par table.
    - La clé primaire permet d'identifier chaque ligne de la table de façon **unique**. Ainsi, la clé primaire ne pourra jamais être NULL.

```
2  ----- Créer et supprimer une clé primaire -----
3
4  CREATE TABLE film (
5      id INT UNSIGNED NOT NULL AUTO_INCREMENT,
6      titre VARCHAR(40) UNIQUE NOT NULL,
7      genre VARCHAR(40) NOT NULL,
8      date_sortie DATE NOT NULL,
9      commentaire TEXT,
10     PRIMARY KEY (id)
11 )
12 CHARACTER SET 'utf8'
13 ENGINE = INNODB;
14
15 ALTER TABLE film
16 ADD PRIMARY KEY (id);
17
18 ALTER TABLE nom_table
19 DROP PRIMARY KEY
```

Les clés

- Il existe deux types de clés :
  - Les clés **étrangères**
    - Elles permettent principalement de **vérifier l'intégrité des données insérées.**
    - Elles font le lien entre deux tables.
    - Le type de la colonne doit être le même sur les deux tables.
    - Un index est automatiquement créé lors de la création d'une clé étrangère.
    - Il est possible d'ajouter des options aux clés étrangères lors de la suppression d'une donnée :
      - **RESTRICT – CASCADE - NO ACTION - SET NULL**

```
21  ----- Créer et supprimer une clé étrangère -----
22
23  CREATE TABLE genre (
24      id INT UNSIGNED NOT NULL AUTO_INCREMENT,
25      nom VARCHAR(40) UNIQUE NOT NULL,
26      PRIMARY KEY (id)
27  )
28  CHARACTER SET 'utf8'
29  ENGINE = INNODB;
30
31  CREATE TABLE film (
32      id INT UNSIGNED NOT NULL AUTO_INCREMENT,
33      titre VARCHAR(40) UNIQUE NOT NULL,
34      genre_id INT UNSIGNED NOT NULL,
35      date_sortie DATE NOT NULL,
36      commentaire TEXT,
37      PRIMARY KEY (id),
38      CONSTRAINT fk_film_genre FOREIGN KEY (genre_id) REFERENCES genre(id) ON DELETE CASCADE
39  )
40  CHARACTER SET 'utf8'
41  ENGINE = INNODB;
42
43  ALTER TABLE film
44  ADD CONSTRAINT fk_film_genre FOREIGN KEY (genre_id) REFERENCES genre(id);
45
46  ALTER TABLE film
47  ADD FOREIGN KEY (genre_id) REFERENCES genre(id);
48
49  ALTER TABLE film
50  DROP FOREIGN KEY fk_film_genre
```

Les jointures

- Les jointures **permettent d'interagir avec plusieurs tables** dans la même requête.
  - Elles permettent donc de joindre plusieurs requêtes.
- Il existe deux types de jointures
  - **Interne :** Lors d'une jointure interne, on exige qu'il y ait des données dans les deux (ou plus) tables sur lesquelles on effectue une jointure.
  - **Externe** : La jointure externe comprend la jointure par la **gauche** & jointure par la **droite**
    - **La jointure par la gauche** signifie que l'on veut toutes les lignes de la table de gauche. Sachant que nous lisons de gauche à droite, la table de gauche est la première table mentionnée dans la clause **FROM**.
    - **La jointure par la droite** est exactement l'inverse de la jointure par la gauche

Les jointures – Jointure interne

- Prenons comme exemple une table **utilisateur** ainsi qu'une table **message**
  - Dans cet exemple, les **alias** sont utilisés grâce au mot clé **AS**. Ils permettent simplement **de renommer ponctuellement la table** afin de rendre la requête plus compacte et visuelle.

Table Utilisateur

id	nom	prenom
1	Bob	Eponge
2	Krabs	Capitaine
3	Etoile de mer	Patrick

Table Message

id	user_id_from	user_id_to	message
1	1	2	Hello Krabs
2	1	2	☺
3	3	1	Hey Bob !!!!!

```sql
----- Récupérer les messages envoyés d'un utilisateur -----

SELECT U.nom, U.prenom, U.message
FROM utilisateur as U
INNER JOIN message as M
    ON U.id = M.user_id_from
WHERE U.id = 1;

----- Equivalent -----

SELECT U.nom, U.prenom, U.message
FROM utilisateur as U, message as M
WHERE U.id = M.user_id_from
AND U.id = 1

----- USING (colonnes de même nom SEULEMENT) -----

SELECT U.nom, U.prenom, U.message
FROM utilisateur U
INNER JOIN message USING(colonne_commune)

----- Jointure naturelle (colonnes de même nom SEULEMENT)  -----

SELECT U.nom, U.prenom, U.message
FROM utilisateur as U
NATURAL JOIN message


----- Récupérer les messages de tous utilisateur -----

SELECT U.nom, U.prenom, U.message
FROM utilisateur as U
INNER JOIN message as M
    ON U.id = M.user_id_from
ORDER BY U.nom ASC;
```

Les jointures – Jointure externe

- Prenons comme exemple trois tables : **utilisateur**, **sport** et **sport_utilisateur**
  - Imaginons que nous souhaitions récupérer toutes les informations des utilisateurs AINSI QUE le sport qu'ils peuvent éventuellement pratiquer.

```
26  SELECT U.*, S.nom as nom_sport
27  FROM utilisateur as U
28  LEFT JOIN sport_utilisateur as SU
29      ON SU.utilisateur_id = U.id
30  LEFT JOIN sport as S
31      ON SU.sport_id = S.id
```

Table Utilisateur

id	nom	prenom
1	Bob	Eponge
2	Krabs	Capitaine
3	Etoile de mer	Patrick

Table Sport

id	nom
1	Basket
2	Tennis
3	Ping-Pong

Table sport_utilisateur

id	utilisateur_id	sport_id
1	3	2
2	2	1

- Si aucun sport n'est pratiqué par l'utilisateur, nom_sport sera **NULL**.

La même requête faite avec une jointure interne n'aurait pas renvoyé les données des utilisateurs sans sports.

Les sous-requêtes

- Une sous-requête est une requête **à l'intérieur** d'une autre requête
- Avec le SQL, il est possible de construire des requêtes imbriquées sur autant de niveaux que l'on souhaite.
- Il est également possible de mélanger jointures et sous-requêtes. Il suffit simplement que la requête soit structurée correctement.

  En SQL, il est possible de construire des requêtes imbriquées sur autant de niveaux que l'on souhaite. Vous pouvez également mélanger jointures et sous-requêtes. Tant que votre requête est correctement structurée, elle peut être aussi complexe que vous le voulez.

- **Les opérateurs de comparaisons** peuvent également être utilisés ainsi que d'autres opérateurs tel que :
  - **IN / NOT IN**
  - **ANY** / **SOME** qui signifient « au moins une valeur »
  - **ALL** qui signifie « toutes les valeurs »
  - **EXISTS / NOT EXISTS** qui renvoient vrai ou faux si la recherche a retourné un résultat au minimum.

Les sous-requêtes

```sql

SELECT *
FROM sport_utilisateur
WHERE sport_id =
    (SELECT id FROM sport WHERE nom = "volley")
ORDER BY id;

/* Retournera les lignes du sport "volley" */

SELECT *
FROM sport_utilisateur
WHERE sport_id <
    (SELECT id FROM sport WHERE nom = "volley")
ORDER BY id;

/*
    Retournera les lignes dont l'id du sport est
    inférieur à celui du volley
*/

SELECT U.*, S.nom as nom_sport
FROM utilisateur as U
LEFT JOIN sport_utilisateur as SU
    ON SU.utilisateur_id = U.id
LEFT JOIN sport as S
    ON SU.sport_id = S.id
WHERE S.nom IN ("Volley");

/*
    Retournera les lignes dont dont le sport est
    "volley"
*/

```

Les sous-requêtes

```sql

SELECT *
FROM sport_utilisateur
WHERE sport_id < ANY
    (SELECT id FROM sport WHERE nom IN ("volley", "Basket")
ORDER BY id;

/*
    Retournera les lignes dont l'id est inférieur à au moins
    des ids retournés.
*/

SELECT *
FROM sport_utilisateur
WHERE sport_id < ALL
    (SELECT id FROM sport WHERE nom IN ("volley", "Basket")
ORDER BY id;
/*
    Retournera les lignes dont l'id est inférieur à tous
    les ids retournés.
*/

SELECT * FROM sport
WHERE NOT EXISTS
    (
        SElECT *
        FROM sport_utilisateur
        WHERE sport_utilisateur.sport_id = sport.id
    )

/*
    Retournera les sports non pratiqués par un utilisateur
*/


----- Requête corrélées -----

SELECT nom
FROM sport
WHERE id IN
    (SELECT SU.sport_id FROM sport_utilisateur as SU WHERE SU.sport_id = sport.id)
ORDER BY id;
```

- Une requête corrélée est une requête qui fait référence à une colonne qui n'est pas définie dans SA CLAUSE FROM.
  - Dans notre cas, si on prends uniquement la sous-requête, elle ne fonctionnera pas car la table « sport » ne sera pas définie.

Les sous-requêtes – INSERT, UPDATE & DELETE

- Les sous requêtes permettent également d'insérer, de modifier ou de supprimer des données :
  - **Insertion :** les sous-requêtes vont permettre d'insérer des données dans une table à partir de données venant d'autres tables
  - **La modification :** les sous requêtes vont permettre ici de sélectionner de façon plus complexe les données à modifier.
  - **La suppression de données :** les sous requêtes vont permettre ici de sélectionner de façon plus complexe les données à supprimer.

Les sous-requêtes – INSERT, UPDATE & DELETE

- Pour la modification de données, **il est impossible** de modifier les données d'une colonne que l'on utilise dans la sous requête, c'est la seule limitation.
- Il y a la même limitation pour la suppression de données.
- Dans le cas d'une suppression avec jointure, il est indispensable de préciser de quelle(s) table(s) les données doivent être supprimées.

```
2
3   ----- Insertion de données -----
4
5   INSERT INTO sport_utilisateur (utilisateur_id, sport_id)
6   SELECT 3, id FROM sport WHERE nom = "Volley";
7
8   INSERT INTO sport_utilisateur (utilisateur_id, sport_id)
9   SELECT U.id, S.id
10  FROM utilisateur as U, sport as S
11  WHERE U.nom = "Bob"
12      AND U.prenom = "Eponge"
13      AND S.nom = "Volley";
14
15  ----- Modification de données -----
16
17  UPDATE sport_utilisateur SET sport_id = (SELECT id FROM sport WHERE nom = "Basket")
18  WHERE utilisateur_id = (SELECT id FROM utilisateur WHERE nom = "Bob" AND prenom = "Eponge");
19
20  UPDATE sport_utilisateur
21  LEFT JOIN utilisateur
22      ON utilisateur.id = sport_utilisateur.utilisateur_id
23  SET sport_utilisateur.sport_id = (SELECT id FROM sport WHERE nom = "Volley")
24  WHERE utilisateur.nom = "Bob";
25
26  /*
27      Modifie l'id sport par celui du Volley pour l'utilisateur Bob
28  */
29
30  ----- Suppression de données -----
31
32  DELETE sport_utilisateur
33  FROM sport_utilisateur
34  LEFT JOIN utilisateur
35      ON utilisateur.id = sport_utilisateur.utilisateur_id
36  WHERE utilisateur.nom = 'Bob';
37
38  /*
39      Supprime le lien entre l'utilisateur Bob et son/ses sport(s)
40  */
41
```

UNION de plusieurs requêtes

- L'union signifie simplement que les **résultats** de **la première requête** et ceux de la **deuxième** seront **réunis**.
  On utilise le mot clé **UNION**.

- On peut unir autant de requêtes que l'on souhaite.

- Il est indispensable qu'il y ait **le même nombre de colonnes** dans chacune des requêtes.

- Lors de la réunion le type n'a pas d'importance, MySQL est très permissif. Néanmoins les résultats obtenus ne seront pas cohérents.

- Avec l'utilisation **d'UNION**, les doublons sont automatiquement supprimés.
  Si l'on souhaite les conserver, on utilisera **UNION ALL**.

- La clause **ORDER BY** n'aura d'influence que sur la totalité de la requête.

UNION de plusieurs requêtes

- La complexité des requêtes n'a pas d'importance tant que l'on sélectionne le même nombre de colonnes dans chaque requêtes.

```sql

----- Union de plusieurs requêtes -----

SELECT nom FROM sport
UNION
SELECT nom FROM utilisateur

/*
    Affiche le nom des sports et ceux des utilisateurs
*/


SELECT nom FROM sport
UNION
SELECT nom FROM utilisateur WHERE date_naissance > '2000-01-01' LIMIT 5

/*
    Affiche les sports et le nom des utilisateur nés après Janvier 2000
*/

SELECT * FROM sport
UNION ALL
SELECT * FROM sport

/*
    Retournera les résultats de sport en doublons
*/

SELECT nom FROM utilisateur
INNER JOIN sport_utilisateur ON
    utilisateur.id = sport_utilisateur.utilisateur_id
UNION
SELECT nom FROM sport

/*
    Affiche les utilisateurs ayant un sport et tous les sports
*/

```

Les différentes fonctions

- **Les fonctions scalaires 1/4**
  - **FLOOR (x)**
    - Arrondit au nombre entier inférieur.
  - **CEIL (x) / CEILING (x)**
    - Arrondit au nombre entier supérieur.
  - **ROUND (x, y)**
    - Arrondit le nombre x à y décimales la plus proche.
  - **TRUNCATE (x, y)**
    - Arrondit en enlevant y décimales.
  - **POWER (x, y) / POW (x, y)**
    - Retourne le nombre x exposant y.
  - **SQRT (x)**
    - Retourne la racine du nombre x.

```sql
CREATE TABLE produit (
    id INT UNSIGNED AUTO_INCREMENT,
    prix DECIMAL(10, 2),
    nom VARCHAR(40),
    PRIMARY KEY(id)
)
CHARACTER SET = 'utf8'
ENGINE = INNODB;

INSERT INTO produit(id, prix, nom) VALUES
        (1, 15, "Chaussure noir"),
        (2, 550, "TV SONY"),
        (3, 15000, "Peugeot 508"),
        (4, 250, "Montre"),
        (5, 50, "Panier en bois"),
        (6, 35.45, "Blue Ray Le Roi Lion"),
        (7, 55, "Sacoche en cuir"),
        (8, 55.10, "Escarpin marron");


----- Exemples -----

SELECT FLOOR(prix)
FROM produit
WHERE id = 6;

/* Retourne 35 au lieu de 35,45 */


SELECT CEIL(prix)
FROM produit
WHERE id = 6;

/* Retourne 36 au lieu de 35,45 */

SELECT ROUND(prix, 2)
FROM produit
WHERE id = 6;

/* Retourne 35.5 au lieu de 35,45 */

SELECT TRUNCATE(prix, 1)
FROM produit
WHERE id = 6;
```

Les différentes fonctions

- **Les fonctions scalaires 2/4**
  - **RAND ()**
    - Génère un nombre aléatoire entre 0 et 1.
  - **SIGN (x)**
    - Retourne 1 si positif, -1 si négatif.
  - **ABS (x)**
    - Retourne la valeur absolue.
  - **MOD (x, y)**
    - Retourne le reste de la division de x par y (modulo)
  - **STRCOMP (x, y)**
    - Compare deux chaines de caractères. Retourne 0 si elles sont identiques, -1 si la première chaîne est classée avant dans l'ordre alphabétique et 1 dans le cas contraire.
  - **REPEAT (x, y)**
    - Retourne le texte x, y fois.

```
2
3   SELECT *
4   FROM produit
5   ORDER BY RAND();
6
7   /*
8       Retourne la liste des produit
9       dans un ordre aléatoire
10  */
11
12  SELECT *, RAND() as aleatoire
13  FROM produit
14  ORDER BY RAND();
15
16  /*
17      Retourne la liste des produit
18      dans un ordre aléatoire ansi
19      qu'un nombre aleatoire compris
20      entre 0 et 1
21  */
22
23  SELECT STRCMP('chaine', 'chaine') AS 'egal',
24         STRCMP('chaine','chaine2') AS 'texte<texte2',
25         STRCMP('chaine2','chaine') AS 'chaine2<chaine';
26
27  /* Retourne 0, -1 et 1 */
28
29  SELECT REPEAT(nom, 2)
30  FROM produit
31  WHERE id = 4;
32
33  /* Retourne "MontreMontre" */
```

Les différentes fonctions

- **Les fonctions scalaires 3/4**

  - **LPAD (x, y, z) / RPAD (x, y, z)**
    - Permet de compléter ou réduire une chaîne. Retourne y caractères de la chaîne x (en lisant de gauche à droit ou droite à gauche. Puis ajoute z pour compléter la longueur si trop court.
  - **SUBSTRING(x, y, z)**
    - Retourne la chaîne x à partir de la position y et de longueur z.
  - **LOWER (x) / UPPER (x)**
    - Retourne la chaîne de caractères x en minuscule ou majuscule.

```
2
3   SELECT  LPAD("Bonjour les copains !", 10, "#"),
4           LPAD("Bonjour", 10, "#"),
5           RPAD("Bonjour", 10, "#");
6
7   /*
8       Retourne "Bonjour le", "###Bonjour", "Bonjour###"
9   */
10
11  SELECT SUBSTRING("Bonjour", 4, 4);
12
13  /* Retourne "jour" */
14
15  SELECT  LOWER("Hello")
16          UPPER("Hello");
17
18  /* Retourne "hello" et "HELLO" */
19
20
```

Les différentes fonctions

- **Les fonctions scalaires 4/4**

  - **LEFT (x, y) / RIGHT (x, y)**
    - Retourne les y premiers caractères de la chaîne x en partant de la droite ou de la gauche.
  - **REVERSE (x)**
    - Retourne l'inverse de la chaîne de caractères donnée.
  - **CONCAT (chaine1, chaine2, …)**
    - Retourne la concaténation des chaînes données.
  - **CONCAT_WS (séparateur, chaine1, chaine2, …)**
    - Retourne la concaténation des chaînes données avec, entre chaque chaînes, le séparateur saisi.

```
2
3   SELECT  LEFT("Bonjour", 4),
4           RIGHT("Bonjour", 4);
5
6   /* Retourne "Bonj" et "jour" */
7
8   SELECT SUBSTRING("Bonjour", 4, 4);
9
10  /* Retourne "jour" */
11
12  SELECT REVERSE("Bonjour les copains !! :)") as "reverse";
13
14  /* Retourne "): !! sniapoc sel ruojnoB" */
15
16  SELECT CONCAT("Bonjour ", "les copains !! :)") as "concat";
17
18  /* Retourne "Bonjour les copains !! :)" */
19
20  SELECT CONCAT_WS("-", "Bonjour", "les", "copains") as "concat";
21
22  /* Retourne "Bonjour-les-copains" */
23
```

Les différentes fonctions

- **Les fonctions d'agrégation**

  - Ces fonctions vont **regrouper les lignes**.

  - **COUNT (x) / COUNT(DISTINCT x)**
    - Retourne le nombre de lignes sélectionnées par la requête.
  - **MIN (x) / MAX (x)**
    - Retourne le minimum ou le maximum de la colonne sélectionnée.
  - **SUM (x)**
    - Retourne la somme de toutes les lignes sur la colonne sélectionnée.
  - **AVG (x)**
    - Retourne la moyenne de toutes les lignes sur la colonne sélectionnée.
  - **GROUP_CONCAT(x)**
    - Retourne la concaténation de chaque valeur de la colonne sélectionnée

```
2
3   SELECT COUNT(nom) as nombre
4   FROM produit;
5
6   /* Retourne 8 */
7
8   SELECT SUM(prix) as prix
9   FROM produit
10
11  /* Retourne 16 010.5 */
12
13  SELECT MIN(prix) as minimum, MAX(prix) as maximum
14  FROM produit
15
16  /* Retourne 15 et 15 000 */
17
18  SELECT AVG(prix) as moyenne
19  FROM produit
20
21  /* Retourne 2001.318750 */
22
23  SELECT GROUP_CONCAT(nom) as concatenation
24  FROM produit
25  WHERE id = 5 OR id = 6 OR id = 8;
26
27  /* Retourne "Panier en bois,
28      Blue Ray Le Roi Lion,Escarpin marron" */
29
```

Regroupement

- Le regroupement permet de regrouper des lignes en **fonction d'un critère**.
- Il est possible de faire un groupement sur :
  - Un seul critère
  - Plusieurs critères
  - Par sécurité, la sélection de colonnes n'étant pas dans les critères de groupement sont interdite.
  - Il est également possible d'avoir des conditions sur le regroupement en utilisant la clause **HAVING**.
    En effet, il est impossible de faire des conditions sur une fonction d'agrégation.

```sql

CREATE TABLE film (
    id INT UNSIGNED AUTO_INCREMENT,
    nom VARCHAR(25),
    code VARCHAR(5),
    categorie VARCHAR(25),
    PRIMARY KEY(id)
)
CHARACTER SET 'utf8'
ENGINE = INNODB;

INSERT INTO film (nom, code, categorie)
    VALUES  ("Le Roi Lion", "LRL", "Animation"),
            ("Le Roi Lion 2", "LRL", "Animation"),
            ("Aladin", "AL", "Animation"),
            ("Aladin 2", "AL", "Animation"),
            ("Star Wars", "SW", "Science Fiction"),
            ("Star Wars 2", "SW", "Science Fiction"),
            ("Star Wars 3", "SW", "Science Fiction"),
            ("Star Wars 4", "SW", "Science Fiction"),
            ("Star Wars 5", "SW", "Science Fiction"),
            ("Star Wars 6", "SW", "Science Fiction"),
            ("Benjamin Gates", "BEN", "Aventure"),
            ("Ca", "IT", "Horreur");


SELECT categorie, COUNT(*) as nombre
FROM film
GROUP BY categorie;
HAVING nombre > 2

/*
    Retourne Animation: 4, Aventure: 1,
    Horreur: 1, Science Fiction: 6

*/

SELECT categorie, COUNT(*) as nombre
FROM film
GROUP BY categorie
HAVING nombre > 2;

/*
    Retourne Animation: 4, Science Fiction: 6

*/
```

Regroupement

- Le regroupement permet de regrouper des lignes en **fonction d'un critère**.

- Il est possible de faire un groupement sur :

  - Un seul critère

  - Plusieurs critères

  - Par sécurité, la sélection de colonnes n'étant pas dans les critères de groupement sont interdite.

  - Il est également possible d'avoir des conditions sur le regroupement en utilisant la clause **HAVING**.
    En effet, il est impossible de faire des conditions sur une fonction d'agrégation.

```sql

SELECT nom, COUNT(*) as utilisateurs
FROM utilisateur
INNER JOIN sport_utilisateur
    ON sport_utilisateur.utilisateur_id = utilisateur.id
GROUP BY utilisateur.nom;

/* Retourne le nombre d'utilisateur pratiquant un sport */

SELECT utilisateur.nom, prenom, COUNT(*) as utilisateurs, sport.nom as sport
FROM utilisateur
INNER JOIN sport_utilisateur
    ON sport_utilisateur.utilisateur_id = utilisateur.id
INNER JOIN sport
    ON sport_utilisateur.sport_id = sport.id
GROUP BY utilisateur.nom, sport.nom;

/*
    Retourne le nombre d'utilisateur pratiquant un sport
    ainsi que le nom du sport. Regroupé d'abord par nom puis par sport.
*/



SELECT code, COUNT(*) as nb_film
FROM film
WHERE code = "LRL";

/* Retourne nb_film: 2 */

SELECT categorie, COUNT(*) as nb_film
FROM film
WHERE categorie = "Animation";

/* Retourne nb_film: 4 */

SELECT code, categorie, COUNT(*) as nb_film
FROM film
GROUP BY code, categorie;

/*
    Retourne les films groupé par code puis par catégorie.
    Avec le nombre associé.
*/
```

Regroupement – Le super-agrégat

- Le super-agrégat s'utilise avec la commande **WITH ROLLUP**

  - Cette commande permet d'ajouter une ligne supplémentaire. Ces lignes représenteront des "super-groupes" (ou super-agrégats), donc des "groupes de groupes«

  - C'est aussi pendant l'utilisation de WITH ROLLUP que l'on constate l'importance de **l'ordre des colonnes** dans le GROUP BY

```
2  ----- Exemple 1 -----
3
4  SELECT code, categorie, COUNT(*) as nb_film
5  FROM film
6  GROUP BY code, categorie WITH ROLLUP;
7
8
9  ----- Exemple 2 -----
10
11
12 SELECT code, categorie, COUNT(*) as nb_film
13 FROM film
14 GROUP BY categorie, code WITH ROLLUP;
15
```

1

code	categorie	nb_film
AL	Animation	2
AL	*NULL*	2
BEN	Aventure	1
BEN	*NULL*	1
IT	Horreur	1
IT	*NULL*	1
LRL	Animation	2
LRL	*NULL*	2
SW	Science Fiction	6
SW	*NULL*	6
NULL	*NULL*	12

2

code	categorie	nb_film
AL	Animation	2
LRL	Animation	2
NULL	Animation	4
BEN	Aventure	1
NULL	Aventure	1
IT	Horreur	1
NULL	Horreur	1
SW	Science Fiction	6
NULL	Science Fiction	6
NULL	*NULL*	12

Gestion des utilisateurs

- SHOW DATABASES
  - Affiche toutes les bases de données de l'utilisateur
    - ***information_schema*** contient les informations sur toutes les bases de données.
    - **performance_schema** contient les informations sur les actions effectuées sur le serveur.
    - ***Mysql*** contient plusieurs informations sur le serveurs. Notamment les **utilisateurs** et leurs **privilèges**.
- Pour modifier le mot de passe, on utilise la commande **PASSWORD()**. En effet, le mot de passe ne sera pas hashé dans le cas d'une modification sans l'utilisation de cette commande.
- « localhost » correspond ici à l'hôte sur laquelle l'utilisateur se connecte. Il s'agit généralement d'une adresse IP.
  % accorde la connexion depuis n'importe quel hôte.

```
2  ----- Afficher toutes les bases de données -----
3
4
5  SHOW DATABASES;
6
7  ----- Créer un utilisateur -----
8
9
10 CREATE USER 'Thibaud'@'localhost' IDENTIFIED BY 'mot_de_passe';
11
12 CREATE USER 'Thibaud'@'%' IDENTIFIED BY 'mot_de_passe';
13
14 ----- Supprimer un utilisateur -----
15
16
17 DROP USER 'Thibaud'@'localhost';
18
19
20 ----- Renommer un utilisateur -----
21
22
23 RENAME USER 'Thibaud'@'localhost' TO 'Thib'@'localhost';
24
25 ----- Modifier le pot de passe d'un utilisateur -----
26
27 SET PASSWORD FOR 'Thibaud'@'%' = PASSWORD('mot_de_passe_2');
```

Gestion des utilisateurs - Privilèges

- Les différents niveaux de privilèges :
  - **« .* »**

    Privilège global : s'applique à toutes les bases de données. Ce privilège sera stocké dans la table *mysql.user.*

  - **« * »**

    Si aucune base de données n'a été sélectionnée, c'est l'équivalent du précédent. Sinon, s'applique à tous les objets de la base de donnée sélectionnée.
    Stocké dans mysql.db

  - « ma_base.* »

    S'applique à tous les objets de la base « ma_base ».
    Stocké dans mysql.db

  - « ma_base.ma_tabl »

    Privilège de table. Stocké dans mysql.tables_priv

  - On peut également aujouter des privilèges à une colonnes.

Les privilèges

Action	Description
CREATE TABLE	Création de tables
ALTER	Modification de tables
DROP	Suppression de tables, vues et bases de données
INDEX	Création et suppression d'index
CREATE USER	Gestion d'utilisateur : CREATE USER, DROP USER, RENAME USER, SET PASSWORD

Ne sont notés que les privilèges en lien avec le contenu de ce cours.
Il en existe d'autres.

Gestion des utilisateurs - Privilèges

```sql
----- Accorder tous les privilèges sur une base de données -----

GRANT ALL PRIVILEGES
ON helloworld.*
TO 'Thibaud'@'localhost';

----- Accorder certains privilèges sur une table -----

CREATE USER 'Thibaud'@'localhost' IDENTIFIED BY 'mot_de_passe';
GRANT SELECT,
      UPDATE (nom, sexe, commentaires),
      DELETE,
      INSERT
ON helloworld.film
TO 'Thibaud'@'localhost';

GRANT SELECT
ON TABLE helloworld.utilisateur
TO 'Thibaud'@'localhost';

----- Accorder le GRANT à un utilisateur  -----

GRANT SELECT, DELETE, GRANT OPTION
ON helloworld.*
TO 'Thibaud'@'localhost' IDENTIFIED BY 'mot_de_passe';

-- Identique à --

GRANT SELECT, DELETE
ON helloworld.*
TO 'Thibaud'@'localhost' IDENTIFIED BY 'mot_de_passe'
WITH GRANT OPTION;


----- Retirer certains privilèges sur une table -----

REVOKE DELETE
ON helloworld.film
FROM 'Thibaud'@'localhost';
```