PARTIE 3

Manipuler les données

Dans ce module, vous allez :

- Vous initier à créer des Bases de Données
- Réaliser les différentes requêtes SQL
- Administrer une BDD

CHAPITRE 1

Créer une Base de Données

Ce que vous allez apprendre dans ce chapitre :

- Création d'une base de données
- Création des tables et des colonnes
- Intégration des contraintes d'intégrité sur les colonnes
- Manipulation des objets d'une Base de données

CHAPITRE 1

Créer une Base de Données

1. **Création des Bases de Données**
2. Choix de moteur
3. Création des tables
4. Définition des colonnes
5. Typage des colonnes
6. Contraintes d’intégrité
7. Manipulation d’objet table

Rappel: Définition d'une base de données

Une **base de données** est une structure permettant de stocker un grand nombre d'informations afin d'en faciliter l'utilisation.

- Le fait de structurer les données a pour but d'assurer les fonctions fondamentales suivantes :
    - La fiabilité du stockage de l'information : La possibilité de restituer l'information stockée dans la base de données ;
    - La massification : Le traitement de grands volumes de données ;
    - L'optimisation : En terme de temps de traitement et espace de stockage ;
    - La sécurisation des accès aux données ;
    - La qualité des données : Vérification des règles de gestion et la conformité avec les modèles de conception ;
    - Le partage des données entre plusieurs acteurs (utilisateurs, applications...) et la gestion des concurrences d'accès.
- Ces fonctions sont assurées au moyen des SGBD : Système de Gestion de Bases de Données

Les objets d'une base de données:

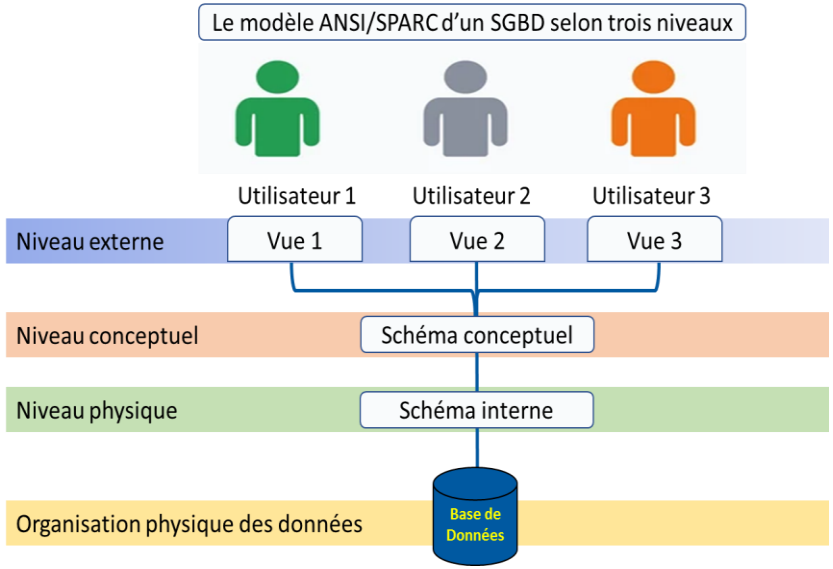
- **Les tables :** contenant des données;
- **Les index :** servant à retrouver, trier, regrouper rapidement les données ;
- **Les déclencheurs (triggers) :** permettant d'exécuter des opérations particulières lors de l'insertion, la modification ou la suppression de données ;
- **Les types de données définis par l'utilisateur (UDDT) :** servant de référentiel à plusieurs tables ;
- **Les valeurs par défaut (Defaults) :** autorisant le système à insérer des valeurs dans les colonnes non renseignées par l'utilisateur ;
- **Les vues, ou pseudo-tables (Views) :** offrant une vue particulière des données aux utilisateurs ;
- **Les fonctions définies par l'utilisateur (UDF) :** permettant de renvoyer soit une valeur, soit une table ;
- **Les procédures stockées :** exécutées par l'utilisateur pour produire un résultat donné ;
- **Les diagrammes (Diagrams) :** qui visualisent les relations entre les tables.

SGBD (Système de Gestion de Bases de Données)

Un SGBD (Système de Gestion de Bases de Données) est un ensemble logiciel qui permet la structuration, le stockage, et la manipulation d'une base de données.

En termes d'architecture, et selon le modèle ANSI/SPARC d'un SGBD comporte 3 niveaux :

- **Niveau conceptuel :**
  - Le niveau central d'un SGBD. Il correspond à une vue générale de toutes les données existant dans l'entreprise.
- **Niveau interne ou physique :**
  - Spécification du stockage physique des données (fichiers, disques, etc.) et des méthodes d'accès (index, etc.).
- **Niveau externe :**
  - Il s'agit d'une vue externe pour chaque groupe d'utilisateurs sur un sous ensemble de la base ;
  - Chaque schéma externe est généralement un sous schéma du schéma conceptuel mais peut contenir parfois des informations supplémentaires.



Types de SGBD :

- **SGBD hiérarchique :**
  - Les données sont représentées dans la base sous la forme d'un arbre dont le parcours se fait du père vers le fils à l'aide de pointeurs.
- **SGBD relationnel :**
  - Les SGBDR Dominent le marché des SGBD. La théorie derrière ce type de systèmes est fondée sur la théorie mathématique des relations. Il s'agit de représentation simple des données sous forme de tables constituées de lignes et de colonnes.
- **SGBD objet :**
  - Les SGBDOO enregistrent les données sous forme d'objets ; les données sont enregistrées avec les procédures et les fonctions qui permettent de les manipuler.

Exemples de SGBD :

- **Oracle** est un SGBD relationnel et relationnel-objet très utilisé pour les applications professionnelles.
- **PostgreSQL** est un SGBD relationnel puissant qui offre une alternative libre (licence BSD) aux solutions commerciales comme Oracle ou IBM.
- **Access** est un SGBD relationnel Microsoft, qui offre une interface graphique permettant de concevoir rapidement des applications de petite envergure ou de réaliser des prototypes.
- **MongoDb** est un SGBD non-relationnel libre (licence Apache) orienté document. Il permet de gérer facilement de très grandes quantités de données - dans un format arborescent JSON - réparties sur de nombreux ordinateurs.
- **MySQL** est un système de gestion de bases de données relationnelles (SGBDR) open source. Ce SGBDR d'Oracle est basé sur le langage SQL (Structured Query Language) et fonctionne sur pratiquement toutes les plates-formes comme Linux, UNIX et Windows.

Remarque : Nous utiliserons pour les TD/TP le SGBD MySQL.

Le langage de requêtes SQL (Structured Query Language) :

- Il s'agit d'un langage d'interrogation des bases de données relationnelles qui permet d'effectuer les tâches suivantes :
  - Définition et modification de la structure de la base de données
  - Interrogation et modification non procédurale (c'est à dire interactive) de la base de données
  - Contrôle de sécurité et d'intégrité de la base
  - Sauvegarde et restauration des bases
- Le langage SQL n'est pas sensible à la casse, cela signifie que l'on peut aussi bien écrire les instructions en minuscules qu'en majuscule.
- Le langage de commandes SQL peut être réparti en quatre catégories :
  - **LDD (Langage de définition des données) :** langage de manipulation des structures de la base.
  - **LMD (Langage de manipulation des données) :** il permet de consulter et de modifier le contenu de la base de données.
  - **LQD (Langage des queries des données) :** langage de requêtes sur les données.
  - **LCD (Langage de contrôle des données) :** il permet de gérer les privilèges et les différents droits d'accès à la base de données.

Création d'une base de données

Les interfaces des SGBD offrent la possibilité de créer des bases de données. Cette opération est aussi possible à partir de la ligne de commande.

- **Syntaxe :**
  - Pour créer une base de données nommée « nom_base », il suffit d'utiliser la requête suivante : 'CREATE DATABASE nom_base'
- **Options :**
  - La syntaxe utilisée pour chacune des options qui peuvent accompagner la commande CREATE DATABASE dépendent du SGBD utilisé. Il convient alors de vérifier la documentation du SGBD pour avoir une idée sur les différentes options possibles : définition des jeux de caractères, le propriétaire de la base, les limites de connexion…
- **Exemple :**
  - Création d'une base de données sur MySQL (Page suivante)

Exemple : Création d'une base de données sur MySQL

Pour créer une nouvelle base de données sur MySQL, nous utilisons la commande CREATE DATABASE avec la syntaxe suivante :

```
CREATE DATABASE [IF NOT EXISTS] database_name
[CHARACTER SET charset_name]
[COLLATE collation_name]
```

Il faut noter que :

- Le nom de la base de données doit être unique. Si une autre base existe avec le même nom, MySQL retourne une erreur.
- Utiliser l'option **IF NOT EXISTS** pour créer conditionnellement une base de données si elle n'existe pas.
- On peut spécifier les options **CHARACTER SET** et **COLLATE** et le classement de la nouvelle base de données. Sinon MySQL utilisera les valeurs par défaut pour la nouvelle base de données.

Sur MySQL command line Client :

- Connectez-vous au serveur MySQL avec un compte utilisateur disposant du privilège CREATE DATABASE.
- Exécutez la commande **CREATE DATABASE dbTest** ; et appuyez sur Entrée :

```
CREATE DATABASE testdb;
```

- MySQL retourne le résultat suivant :

```
Query OK, 1 row affected (0.04 sec)
```

- On peut utiliser la commande **SHOW CREATE DATABASE** pour examiner la base de données créée :

```
SHOW CREATE DATABASE testdb;
```

- Voici le résultat de cette commande :

```
mysql> SHOW CREATE DATABASE dbTest;
+----------+-----------------------------------------------------------------------------------------------------------------------------------------+
| Database | Create Database                                                                                                                         |
+----------+-----------------------------------------------------------------------------------------------------------------------------------------+
| dbTest   | CREATE DATABASE `dbTest` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */ |
+----------+-----------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

- Enfin, sélectionnez la base de données nouvellement créée avec laquelle on veut travailler en utilisant la commande USE :

```
USE testdb;
```

- Le système confirme par le message suivant :

```
Database changed
```

- Utiliser la commande **EXIT** pour quitter le programme.

CHAPITRE 1

Créer une Base de Données

1. Création des Bases de Données
2. **Choix de moteur**
3. Création des tables
4. Définition des colonnes
5. Typage des colonnes
6. Contraintes d’intégrité
7. Manipulation d’objet table

Moteurs de stockage

Le moteur de stockage d'un Système de Gestion de Base de Données (SGBD), aussi appelé **moteur de table, est l'ensemble d'algorithmes qui permettent** de stocker et d'accéder aux données. En général, les SGBD utilisent un seul moteur de stockage qui est optimisé au mieux pour la lecture, l'écriture et la suppression de données.

MySQL se distingue des autres SGBD par le fait de donner à l'utilisateur le libre choix d'utiliser un moteur de table parmi plusieurs moteurs différents. Ces moteurs de stockage peuvent être transactionnels ou non-transactionnels.

On se retrouve ainsi avec des bases où plusieurs moteurs peuvent coexister. C'est un choix de conception qui a ses avantages et inconvénients.

Moteurs de stockage

Afin de consulter la liste des engins mis à notre disposition par MySQL, on peut exécuter la commande :

- SHOW ENGINES sur la ligne de commande MySQL comme suit :

```
mysql> SHOW ENGINES;
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   | NO         |
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
9 rows in set (0.01 sec)
```

Moteurs de stockage

- Chacun de ces moteurs de table possèdent des caractéristiques propres qui peuvent représenter des atouts ou des inconvénients selon le type d'application qui aura besoin d'une base de données.
- A partir de la version MySQL 5.5 , le moteur par défaut est InnoDB.
- On peut choisir le moteur de stockage au moment de la création d'une table.

La syntaxe est la suivante :

```
CREATE TABLE <nomTable> (...) ENGINE <nomMoteur>
```

Quel moteur choisir?

Les moteurs MySQL les plus utilisés sont MyISAM et InnoDb et Memory, le tableau suivant présente une comparaison entre ces moteurs :

Moteur	Avantages	Inconvénients
MyISAM : un moteur non transactionnel assez rapide en écriture et très rapide en lecture.	• Très rapide en lecture • Extrêmement rapide pour les opérations COUNT() sur une table entière • Les index FULLTEXT pour la recherche sur des textes	• Pas de gestion des relations • Pas de gestion des transactions • Bloque une table entière lors d'opérations d'insertions, suppressions ou mise à jour des données
InnoDB : un moteur relationnel performant faisant partie de la famille des moteurs transactionnels.	• Gestion des relations • Gestion des transactions • Verrouillage à la ligne et non à la table	• Plus lent que MyISAM • Occupe plus de place sur le disque dur • Occupe plus de place en mémoire vive
Memory : un moteur de stockage permettant de créer des tables directement dans la mémoire vive, sans passer par le disque dur pour stocker les données. Ceci en fait le moteur de stockage le plus rapide que propose MySQL, mais aussi le plus dangereux.	• Le moteur le plus rapide • Ne consomme pas de place sur le disque dur	• Les données sont volatiles : un arrêt du serveur et elles disparaissent • Pas de champs BLOB ou TEXT

Afin de faciliter le choix, pour une petite base, sans liens entre les tables, et dont la cohérence des données n'est pas vitale, on peut opter pour MyISAM .

Dans les autres cas, choisissez InnoDB.

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE

CHAPITRE 1

Créer une Base de Données

1. Création des Bases de Données
2. Choix de moteur
3. **Création des tables**
4. Définition des colonnes
5. Typage des colonnes
6. Contraintes d'intégrité
7. Manipulation d'objet table

01 - Créer une Base de Données

Création des tables

La création des tables est possible à partir de la ligne de commande.

- **Syntaxe :** Pour créer une table « nom_table », il suffit d'utiliser la requête suivante:

```
CREATE TABLE nom_table
```

- **Exemple :** Création d'une table dans une base de données MySQL
    - Pour créer une nouvelle base de données sur MySQL, nous utilisons la commande CREATE DATABASE avec la syntaxe suivante :

```
CREATE TABLE [IF NOT EXISTS] table_name(
   column_1_definition,
   column_2_definition,
   ...,
   table_constraints
) ENGINE=storage_engine;
```

Sur cette syntaxe, on définit :

- Le nom de la table. « IF NOT EXISTS » est une commande optionnelle, et permet de vérifier si une table avec le même nom existe déjà.
- La liste des colonnes de cette table, séparées par des virgules.
- La liste des contraintes comme UNIQUE, CHECK, PRIMARY KEY et FOREIGN KEY.
- Le moteur de stockage avec la clause : ENGINE. Si on n'utilise pas cette option, MySQL va utiliser le moteur définit par default : InnoDB.

CHAPITRE 1

Créer une Base de Données

1. Création des Bases de Données
2. Choix de moteur
3. Création des tables
4. **Définition des colonnes**
5. Typage des colonnes
6. Contraintes d’intégrité
7. Manipulation d’objet table

La syntaxe de définition d'une colonne est la suivante :

```
nom_colonne type_donnee Liste_contraintes;
```

Avec :

- Nom_colonne : le nom qu'on va donner à cette colonne.
- Type_donnee : le type et taille de données qui vont être stockées dans cette colonne (numériques, caractères, date..).
- Liste_contraintes : la liste des contraintes sur cette colonne.

CHAPITRE 1

Créer une Base de Données

1. Création des Bases de Données
2. Choix de moteur
3. Création des tables
4. Définition des colonnes
5. **Typage des colonnes**
6. Contraintes d’intégrité
7. Manipulation d’objet table

Les types de données qui représentent la nature des valeurs stockées dans une colonne appartiennent à trois catégories : Numérique, date/heure et chaîne (caractères). Voici les listes des types supportés par MySQL.

1 - Type de données numériques

Type	Taille	Utilisation
TINYINT	1 octet	petites valeurs entières/ Boolean
SMALLINT	2 octets	valeur entière
MEDIUMINT	3 octets	valeur entière
INT ou INTEGER	4 octets	valeur entière
BIGINT	8 octets	Valeur maximale entier
FLOAT	4 octets	Simple précision valeurs à virgule flottante
DOUBLE	8 octets	Double-précision valeurs à virgule flottante
DECIMAL	De DECIMAL (M, D), si M> D, M + 2 est par ailleurs D + 2	valeur décimale

2 - Type de données caractères

Type	Taille (octets)	Utilisation
CHAR	0-255	chaîne longueur fixe
VARCHAR	0-65535	chaînes de longueur variable
TINYBLOB	0-255	Pas plus de 255 caractères dans une chaîne binaire
TINYTEXT	0-255	Courtes chaînes de texte
BLOB	0-65535	données textuelles longues sous forme binaire
TEXTE	0-65535	Longue données de texte
MEDIUMBLOB	0-16777215	forme binaire de longueur moyenne des données de texte
MEDIUMTEXT	0-16777215	longueur moyenne des données de texte
LONGBLOB	0-4294967295	Grands données de texte sous forme binaire
LONGTEXT	0-4294967295	Grande données de texte

3 - Type de données date/heure

Type	Taille (Byte)	Format	Utilisation
DATE	3	AAAA-MM-JJ	Les valeurs de date
TIME	3	HH: MM: SS	Valeur temps ou la durée
ANNÉE	1	AAAA	Année Valeur
DATETIME	8	AAAA-MM-JJ HH: MM: SS	Mixage valeurs date et heure
TIMESTAMP	4	AAAAMMJJ HHMMSS	Date de mélange et la valeur temps, un horodatage

CHAPITRE 1

Créer une Base de Données

1. Création des Bases de Données
2. Choix de moteur
3. Création des tables
4. Définition des colonnes
5. Typage des colonnes
6. **Contraintes d'intégrité**
7. Manipulation d'objet table

Définition

- Une contrainte d'intégrité est une règle qui définit la cohérence d'une donnée ou d'un ensemble de données de la BD. Les contraintes peuvent avoir une portée sur une colonne ou sur une table lorsque la contrainte porte sur plusieurs colonnes.
- Il existe trois types de contraintes d'intégrité : (STP sous forme de schéma)
    - **Intégrité de domaine : (NOT NULL, DEFAULT, UNIQUE…)**
        - Spécifie un ensemble de valeurs pour une colonne
        - Détermine si les valeurs nulles sont autorisées
        - Implémentation par contrôle de validité
    - **Intégrité des entités : (PRIMARY KEY)**
        - Précise la clé primaire de chaque table
    - **Intégrité référentielle : (FOREIGN KEY/REFERENCES)**
        - Assure l'intégrité des relations entre les clés primaires et les clés étrangères.

Les contraintes d'intégrité MySQL :

1 - PRIMARY KEY

- Une clé primaire est une colonne ou un ensemble de colonnes qui identifie de manière unique chacune des ligne de la table. Cette colonne doit vérifier les conditions suivantes :
    - Une clé primaire doit contenir des valeurs uniques. Si la clé primaire se compose de plusieurs colonnes, la combinaison de valeurs dans ces colonnes doit être unique.
    - Une colonne de clé primaire ne peut pas avoir de valeurs NULL. Toute tentative d'insertion ou de mise à jour de NULL dans les colonnes de clé primaire entraînera une erreur. Notez que MySQL ajoute implicitement une contrainte **NOT NULL** aux colonnes de clé primaire.
    - Une table ne peut avoir qu'une et une seule clé primaire.
- Lorsque vous définissez une clé primaire pour une table, MySQL crée automatiquement un index appelé **PRIMARY**.

1 - PRIMARY KEY (suite)

En général, la clé primaire d'une table est définie dans l'instruction **CREATE TABLE**.

Si la clé primaire est une seule colonne, **PRIMARY KEY** (contrainte de colonne) est utilisée avec la syntaxe suivante :

```
CREATE TABLE nom_table(
    primary_key_colonne  Type_donnee  PRIMARY KEY,
    ...
);
```

1 - PRIMARY KEY (suite)

Si la clé primaire est composée de plusieurs colonnes, **PRIMARY KEY** (contrainte de table) est utilisée avec la syntaxe suivante :

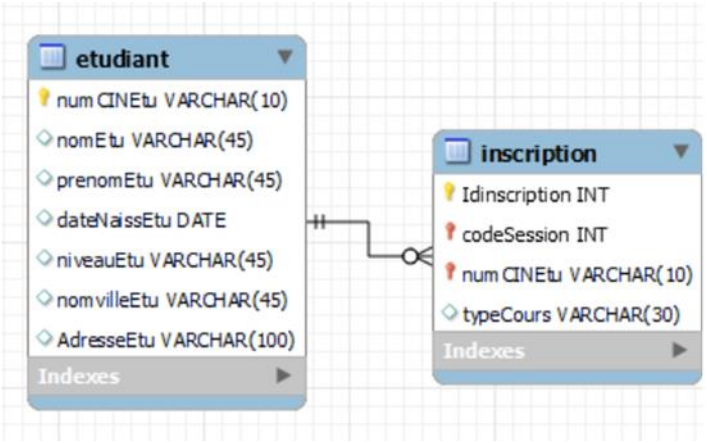
```
CREATE TABLE nom_table(
    primary_key_colonne1 type_donnee,
    primary_key_colonne type_donnee,
    ...,
    PRIMARY KEY(liste_colonnes)
);
```

On sépare les colonnes de liste_colonnes par des virgules (,).

2 - FOREIGN KEY

Une clé étrangère est une colonne ou un groupe de colonnes d'une table qui renvoie à une colonne ou à un groupe de colonnes d'une autre table. La clé étrangère impose des contraintes sur les données des tables associées, ce qui permet à MySQL de maintenir l'intégrité référentielle.

Exemple : Rappelons la relation entres les tables **étudiant** et **inscription** de notre base de données **CentreFormation**.



2 - FOREIGN KEY (suite)

Chaque étudiant peut avoir aucune ou plusieurs inscriptions, et chaque inscription appartient à un étudiant unique.

Cette relation est traduite au niveau du MLD par une clé étrangère dans la table inscription (référencée) qui renvoie à la colonne identifiante de la table etudiant: numCINEtu.

La table etudiant est appelée table parent ou table référencée, et la table inscription est appelée table enfant ou table de référencement.

Afin de Créer une contrainte FOREIGN KEY pour la table inscription:

```
CONSTRAINT fk1_inscription
FOREIGN KEY (numCINEtu)
REFERENCES etudiant(numCINEtu)
```

2 - FOREIGN KEY (suite)

Voici la syntaxe générale qu'on utilise pour la définition d'une contrainte de clé étrangère

```
CONSTRAINT nom_contrainte
FOREIGN KEY (nom_colonne)
REFERENCES table_parent(nom_colonne_p)
[ON DELETE reference_option]
[ON UPDATE reference_option]
```

Avec :

- **nom_contrainte :** le nom qu'on va donner à la contrainte FOREIGN KEY
- **nom_colonne :** la colonne dans la table de référencement
- **table_parent :** la table référencée
- **nom_colonne_p :** clé primaire de la table référencée
- **reference_option :** détermine l'action que MySQL effectuera lorsque les valeurs des colonnes de clé parent seront supprimées (**ON DELETE**) ou mises à jour (**ON UPDATE**).

3 - NOT NULL

- La contrainte NOT NULL est une contrainte de colonne qui garantit que les valeurs stockées dans une colonne ne sont pas NULL.
- La syntaxe d'une contrainte NOT NULL est la suivante :

```
nom_colonne type_donnee NOT NULL;
```

4 - DEFAULT

- La contrainte MySQL DEFAULT permet de spécifier une valeur par défaut pour une colonne :

```
nom_colonne   type_donnee  DEFAULT valeur_par_defaut;
```

- Dans cette syntaxe, le mot clé DEFAULT est suivi par la valeur par défaut de la colonne. Le type de la valeur par défaut doit correspondre au type de données de la colonne.

Les contraintes d'intégrité MySQL : UNIQUE

- UNIQUE est une contrainte d'intégrité qui garantit que les valeurs d'une colonne ou d'un groupe de colonnes sont uniques. Ainsi elle peut être une contrainte de colonne ou une contrainte de table.
- Afin de définir la contrainte UNIQUE pour une colonne lors de la création de la table, on utilise cette syntaxe :

```
nom_colonne type_donnee UNIQUE;
```

Les contraintes d'intégrité MySQL : UNIQUE

- Pour définir une contrainte UNIQUE pour plusieurs colonnes (contrainte de table), on utilise la syntaxe suivante :

```
CREATE TABLE nom_table(
   nom_colonne1 definition_colonne,
   nom_colonne2 definition_colonne,
   ...,
   UNIQUE(nom_colonne1, nom_colonne2,..)
);
```

- Si on définit la contrainte UNIQUE sans la nommer, MySQL génère automatiquement un nom pour celle-ci. Pour définir une contrainte UNIQUE avec un nom, on utilise cette syntaxe :

```
[CONSTRAINT nom_contrainte]
UNIQUE(liste_colonne)
```

5 - CHEC

- La contrainte **CHECK** assure que les valeurs stockées dans une colonne ou un groupe de colonnes satisfont à une expression booléenne. Elle est ainsi considérée comme une contrainte de colonne et une contrainte de table.
- A partir de la version MySQL 8.0.16, la contrainte CHECK de table et de colonne est prise en charge lors de la création de la table, et ce pour tous les moteurs de stockage.
- Voici la syntaxe à utiliser :

```
[CONSTRAINT [nom_contrainte]] CHECK(expression) [[NOT] ENFORCED]
```

- Où il faudra:
  - Spécifier le nom de la contrainte à créer. Si le nom de la contrainte est omis, MySQL génère automatiquement un nom avec la convention suivante : **nom_table_chk_n** (n étant un entier).
  - Spécifiez une expression booléenne qui doit être évaluée à **TRUE** ou **FALSE** pour chaque ligne de la table. Si l'expression est évaluée à **FALSE**, les valeurs entrées violent la contrainte.
  - En option, spécifier la clause d'application pour indiquer si la contrainte de vérification est appliquée :
  - Utilisez **ENFORCED** ou omettez simplement la clause **ENFORCED** pour créer et appliquer la contrainte.
  - Utilisez **NOT ENFORCED** pour créer la contrainte mais ne pas l'appliquer.

5 - CHECK (suite) - Exemple 1 : Contrainte de colonne

- Contrainte MySQL CHECK - exemple de contrainte de colonne

```
CREATE TABLE Produits (
    Num_Produit VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40),
    cout DECIMAL(10,2 ) NOT NULL CHECK (cout >= 0),
    prix DECIMAL(10,2) NOT NULL CHECK (prix >= 0)
);
```

- La contrainte check assure que lors de l'insertion, la valeur de chacune des colonnes cout et prix sera >= 0 .

5 - CHECK (suite) - Exemple 2 : Contrainte de table

- Contrainte MySQL CHECK - exemple de contrainte de table

```
CREATE TABLE Produits (

    Num_Produit VARCHAR(18) PRIMARY KEY,

    description VARCHAR(40),

    cout DECIMAL(10,2 ) NOT NULL CHECK (cout >= 0),

    prix DECIMAL(10,2) NOT NULL CHECK (prix >= 0),

CONSTRAINT produits_chk_prix_et_cout
        CHECK(price >= cost)

);
```

- Dans cette requête, nous avons ajouté une contrainte CHECK nommée **produits_chk_prix_et_cout** comme contrainte de table, et qui assure que la valeur du prix doit toujours être supérieure à la valeur du coût pour chaque élément de la table produit.

CHAPITRE 1

Créer une Base de Données

1. Création des Bases de Données
2. Choix de moteur
3. Création des tables
4. Définition des colonnes
5. Typage des colonnes
6. Contraintes d’intégrité
7. **Manipulation d’objet table**

Commande DROP TABLE

- La commande DROP TABLE supprime définitivement une table et ses données de la base de données. Dans MySQL, nous pouvons également supprimer plusieurs tables à la fois en suivant la syntaxe de base :

```
DROP [TEMPORARY] TABLE [IF EXISTS] nom_table1, nom_table2
...
```

- L'option **TEMPORARY** permet de supprimer uniquement les tables temporaires. Ceci prévient que l'utilisateur supprime accidentellement des tables non temporaires.

Commande DROP TABLE

- L'option **IF EXISTS** fait que MySQL supprime une table uniquement si elle existe. Si vous supprimez une table inexistante avec l'option **IF EXISTS**, MySQL génère une NOTE, qui peut être récupérée à l'aide de l'instruction SHOW WARNINGS.
- Notez que l'instruction **DROP TABLE** supprime uniquement les tables. Il ne supprime pas les privilèges utilisateur spécifiques associés aux tables. Par conséquent, si vous créez une table avec le même nom que celle supprimée, MySQL appliquera les privilèges existants à la nouvelle table, ce qui peut poser un risque de sécurité.
- Enfin, pour exécuter **DROP TABLE**, l'utilisateur doit disposer des privilèges **DROP** pour la table qu'il va supprimer.

Commande ALTER TABLE

- ALTER TABLE est la commande utilisée pour changer la structure d'une table :
  - Ajouter, modifier, renommer et supprimer une colonne
  - Renommer une table
  - Ajouter et supprimer une contrainte d'intégrité.

Ajouter une ou plusieurs colonnes à une table :

```
ALTER TABLE nom_table
    ADD nouvelle_colonne1 [definition1],
    ADD nouvelle_colonne2 [definition2],
    ...;
```

Commande ALTER TABLE : Ajouter une ou plusieurs colonnes à une table (suite)

- **Exemples :**
  - Ajouter une colonne « num_produit » à la table Produits :

```
ALTER TABLE Produits (
    ADD num_Produit VARCHAR(18)
);
```

  - Ajouter plusieurs colonnes à la table Produits :

```
ALTER TABLE Produits (
    ADD Num_Produit VARCHAR(18),
    cout DECIMAL(10,2 ) NOT NULL CHECK (cout >= 0)
 );
```

Commande ALTER TABLE : Modifier une ou plusieurs colonnes d'une table

- Cette modification peut porter sur le type ainsi que les contraintes associées à cette colonne en utilisant la syntaxe suivante :

```
ALTER TABLE nom_table
    MODIFY nouvelle_colonne1 [definition1],
    MODIFY nouvelle_colonne2 [definition2],
    ...;
```

Commande ALTER TABLE : Modifier une ou plusieurs colonnes d'une table (suite)

Exemples 1 :

- Modifier une colonne de la table « Produits » :
  - La colonne « num_produit » est déjà définit comme suit : **num_Produit VARCHAR(18).**

```
ALTER TABLE Produits (
    MODIFY num_Produit VARCHAR(20) PRIMARY KEY );
```

  - Permet de modifier cette colonne de telle façon à :
    - Augmenter la taille du varchar(18) à varchar (20).
    - Définir cette colonne comme clé primaire de la table « Produits » via l'ajout de la contrainte **PRIMARY KEY.**

Commande ALTER TABLE : Modifier une ou plusieurs colonnes d'une table (Suite)

Exemple 2 :

- Afin de Afin de modifier deux colonnes de la table « Produits » :

```
ALTER TABLE Produits (
    MODIFY num_Produit VARCHAR(20) PRIMARY KEY,
    MODIFY cout DECIMAL(10,2 ) CHECK (cout >= 0)
 );
```

- Cette commande a permit de :
  - Changer le type de la colonne num_Produit et la définir comme clé primaire.
  - Ajouter la contrainte CHECK sur la colonne Cout.

Commande ALTER TABLE : Renommer une ou plusieurs colonnes d'une table

- Afin de renommer une colonne d'une table, utilise la syntaxe suivante :

```
ALTER TABLE nom_table
    CHANGE COLUMN nom_original nouveau_nom [definition] ;
```

Exemple :

- Pour renommer la colonne « cout » en « cout_produit » on exécute la commande suivante:

```
ALTER TABLE Produits (
 CHANGE COLUMN cout cout_produit DECIMAL(10,2 ) CHECK (cout >= 0)
 );
```

Commande ALTER TABLE : Supprimer une colonne d'une table

- Afin de supprimer une colonne d'une table, on utilise la syntaxe suivante :

```
ALTER TABLE nom_table
    DROP COLUMN nom_colonne ;
```

Exemple :

- Supprimer la colonne « description » de la table « Produits»:

```
ALTER TABLE Produits
    DROP COLUMN description ;
```

Commande ALTER TABLE : Renommer une table

- Pour renommer une table, on utilise la syntaxe suivante :

```
ALTER TABLE nom_table
RENAME TO nouveau_nom_table ;
```

Exemple :

- Renommer la table « Produits » en « Articles » :

```
ALTER TABLE Produits
RENAME TO Articles ;
```

Commande ALTER TABLE : Ajouter et supprimer une contrainte de table

PRIMARY KEY :

- Ajouter une clé primaire:

```
ALTER TABLE nom_table
ADD PRIMARY KEY(column_list);
```

- Supprimer une clé primaire:

```
ALTER TABLE nom_table
DROP PRIMARY KEY;
```

Commande ALTER TABLE : Ajouter et supprimer une contrainte de table

FOREIGN KEY :

- Ajouter une clé étrangère :

```
ALTER TABLE nom_table
ADD CONSTRAINT constraint_name
FOREIGN KEY (column_name, ...)
REFERENCES parent_table (colunm_name,...);
```

- Supprimer une clé étrangère :

```
ALTER TABLE nom_table
DROP FOREIGN KEY nom_contrainte;
```

Commande ALTER TABLE : Ajouter et supprimer une contrainte de table

UNIQUE :

- Ajouter la contrainte UNIQUE :

```
ALTER TABLE nom_table
ADD CONSTRAINT nom_contrainte
UNIQUE (liste_colonnes);
```

- Supprimer la contrainte UNIQUE : Il faut supprimer l'index que MYSQL crée pour cette contrainte et qui porte le même nom.

```
ALTER TABLE nom_table
DROP INDEX nom_contrainte;
```

CHAPITRE 2

Réaliser des requêtes SQL

Ce que vous allez apprendre dans ce chapitre :

- Maitriser les principales requêtes SQL
- Gérer les fonctions du SGBD

CHAPITRE 2

Réaliser des requêtes SQL

Dans ce qui suit, nous allons apprendre à utiliser les requêtes SQL –LMD : Langage de manipulation de données(LMD).

INSERT :

- La commande INSERT permet d'insérer une ou plusieurs lignes de données dans une table selon la syntaxe suivante :

```
INSERT INTO nom_table(colonne1,colonne2,...)
VALUES (valeur1,valeur2,...);
```

On doit spécifier :

- Le nom de la table ainsi qu'une liste de ses colonnes séparées par des virgules entre parenthèses.
- La liste de valeurs à insérer dans ces colonnes, séparées par des virgules.

Il faut noter que :

- Le nombre de colonnes et de valeurs doit être le même. De plus, les positions des colonnes doivent correspondre aux positions de leurs valeurs.
- Pour les colonnes qui ne figurent pas dans la liste spécifies MySQL insert les valeurs par default ou alors NULL si elles ne sont pas spécifiées.
- Si une colonne est définie comme une colonne AUTO_INCREMENT, MySQL génère un entier séquentiel chaque fois qu'une ligne est insérée dans la table.

INSERT : Insérer plusieurs lignes

- Pour insérer plusieurs lignes dans une table à l'aide d'une seule instruction **INSERT**, on utilise la syntaxe suivante :

```
INSERT INTO nom_table(c1,c2,...)
VALUES (v01,v02,...),
(v11,v22,...),
..
(vn1,vn2,...)
;
```

- Les lignes à ajouter dans la table sont séparées par des virgules dans la clause VALUES.

INSERT : Exemples

- Soit la table « Produit » créée à partir de la requête suivante :

```
CREATE TABLE Produits (
    Num_Produit VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40) DEFAULT 'Non specifie',
    cout DECIMAL(10,2 ) NOT NULL CHECK (cout >= 0),
    prix DECIMAL(10,2) NOT NULL CHECK (prix >= cout),
    Date_ajout DATE
);
```

- 1 - One veut ajouter le produit suivant : Numéro du Produit est P12, Son prix est 14 et le cout 12.
- Voici la commande à exécuter :

```
INSERT INTO Produits(num_produit,cout,prix)
VALUES ('P12',12,14);
```

INSERT : Exemples

- Ajout de plusieurs lignes : One veut ajouter les produits suivant :
    - Numéro du Produit est P13, Le cout: 120, le prix 140, ajouté le 01/01/2022.
    - Numéro du Produit est P100, Description: Laptop, le cout: 120, le prix 140, ajouté aujourd'hui.
- Voici la commande à exécuter :

```
INSERT INTO Produits(num_produit,description,cout,prix,date_ajout)
VALUES ('P13','',120,140,'2022-01-01'),
       ('P100','Laptop',5000,6000,CURRENT_DATE)
;
```

INSERT : Exemples

- Voici le contenu de la table après l'exécution de ces requêtes :

```
mysql> select * from Produits;
+-------------+--------------+---------+---------+------------+
| Num_Produit | description  | cout    | prix    | Date_ajout |
+-------------+--------------+---------+---------+------------+
| P100        | Laptop       | 5000.00 | 6000.00 | 2022-01-14 |
| P12         | Non specifie |   12.00 |   14.00 | NULL       |
| P13         |              |  120.00 |  140.00 | 2021-01-01 |
+-------------+--------------+---------+---------+------------+
3 rows in set (0.00 sec)
```

- **On remarque que :**
  - La description non spécifiée dans le premier insert a été remplacée par la valeur par défaut : 'Non spécifié'
  - Pour insérer une valeur de date dans une colonne, il faut utiliser le format 'YYYY-MM-DD' ou :
    - YYYY représente une année à quatre chiffres.
    - MM représente un mois à deux chiffres.
    - DD représente un jour à deux chiffres.
  - Nous avons utilisé la fonction CURRENT_DATE()qui retourne la date système.

UPDATE :

- L'instruction **UPDATE** permet de mettre à jour les données d'une table. Elle sert à modifier les valeurs dans une ou plusieurs colonnes d'une seule ligne ou de plusieurs lignes, selon la syntaxe suivante :

```
UPDATE nom_table
SET
    nom_colonne1 = expr1,
    nom_colonne2 = expr2,
    ...
[WHERE
    condition];
```

- **On doit spécifier :**
  - Le nom de la table dont on souhaite mettre à jour les données.
  - La colonne qui va être mise à jour et la nouvelle valeur dans la clause SET. Pour mettre à jour les valeurs dans plusieurs colonnes, on utilise une liste d'affectations séparées par des virgules.
  - En option, Définir une condition dans la clause WHERE. Si cette étape est omise, l'instruction UPDATE modifiera **toutes les lignes de la table**.

UPDATE : Exemples

- Rappelons la table Produits :

```
mysql> select * from Produits;
+-------------+--------------+---------+---------+------------+
| Num_Produit | description  | cout    | prix    | Date_ajout |
+-------------+--------------+---------+---------+------------+
| P100        | Laptop       | 5000.00 | 6000.00 | 2022-01-14 |
| P12         | Non specifie |   12.00 |   14.00 | NULL       |
| P13         |              |  120.00 |  140.00 | 2021-01-01 |
+-------------+--------------+---------+---------+------------+
3 rows in set (0.00 sec)
```

- Modifier la date d'ajout du produit P12 en 31/12/2021 :

```
UPDATE Produits
SET
  Date_ajout = '2021-12-31'
 WHERE Num_Produit='P12';
```

UPDATE : Exemples

- Modifier les données de telle façon que Description =’Non specifie’ et Prix = 1.5*cout :

```
UPDATE Produits
SET
  Description = 'Non specifie',
  Prix = 1.5*Cout ;
 WHERE Num_Produit='P12';
```

- Résultat suivant les deux modifications :

```
mysql> select * from Produits;
+-------------+--------------+---------+---------+------------+
| Num_Produit | description  | cout    | prix    | Date_ajout |
+-------------+--------------+---------+---------+------------+
| P100        | Non specifie | 5000.00 | 7500.00 | 2022-01-14 |
| P12         | Non specifie |   12.00 |   18.00 | 2021-12-31 |
| P13         | Non specifie |  120.00 |  180.00 | 2021-01-01 |
+-------------+--------------+---------+---------+------------+
```

DELETE :

- L'instruction **DELETE** permet de supprimer une ou plusieurs lignes d'une table en utilisant la syntaxe suivante :

```
DELETE FROM nom_table
WHERE Conditions;
```

- Cette instruction permet d'utiliser ,en option, une condition pour spécifier les lignes à supprimer dans la clause WHERE, si cette dernière est omise, l'instruction DELETE supprimera toutes les lignes de la table.
- Pour une table qui a une contrainte de clé étrangère, lorsque vous supprimez des lignes de la table parent, les lignes de la table enfant peuvent aussi être supprimées automatiquement à l'aide de l'option ON DELETE CASCADE.

DELETE : Exemples

- De la table Produit :

```
mysql> select * from Produits;
+-------------+-------------+---------+---------+------------+
| Num_Produit | description | cout    | prix    | Date_ajout |
+-------------+-------------+---------+---------+------------+
| P100        | Non specifie | 5000.00 | 7500.00 | 2022-01-14 |
| P12         | Non specifie |   12.00 |   18.00 | 2021-12-31 |
| P13         | Non specifie |  120.00 |  180.00 | 2021-01-01 |
+-------------+-------------+---------+---------+------------+
```

- On veut supprimer les Produits dont le cout <=12 .

```
DELETE FROM Produits
WHERE cout<=12;
```

- Voici la table après l'exécution de cette requête

```
mysql> DELETE FROM Produits
    -> WHERE cout<=12;
Query OK, 1 row affected (0.01 sec)

mysql> select * from Produits;
+-------------+-------------+---------+---------+------------+
| Num_Produit | description | cout    | prix    | Date_ajout |
+-------------+-------------+---------+---------+------------+
| P100        | Non specifie | 5000.00 | 7500.00 | 2022-01-14 |
| P13         | Non specifie |  120.00 |  180.00 | 2021-01-01 |
+-------------+-------------+---------+---------+------------+
2 rows in set (0.00 sec)
```

CHAPITRE 2

Réaliser des requêtes SQL

1. Requêtes LMD
2. **Requêtes de sélection**
3. Expression du SGBD
4. Fonctions d'agrégation du SGBD
5. Sous requêtes
6. Requêtes de l'union
7. Jointures

SELECT :

- L'instruction **SELECT** permet de consulter les données et de les présenter triées et/ou regroupées suivant certains critères.
- L'instruction **SELECT** basique est comme suit :

```
SELECT [Liste_select]
FROM nom_table;
```

- Dans cette syntaxe, il faut :
    - Spécifier une ou plusieurs colonnes à partir desquelles on veut sélectionner des données après le mot-clé SELECT : Pour sélectionner toute les colonnes de la table, on utilise « SELECT * » , Sinon on spécifie les noms des colonnes séparés par une virgule (,).
    - Spécifier le nom de la table à partir de laquelle on veut sélectionner des données après le mot-clé FROM.
- **N.B :** Lors de l'exécution de l'instruction SELECT, MySQL évalue la clause FROM avant la clause SELECT :

SELECT :

- **Exemples de requêtes SELECT simples :**

```
mysql> select * from Produits;
+-------------+-------------+--------+--------+------------+
| Num_Produit | description | cout   | prix   | Date_ajout |
+-------------+-------------+--------+--------+------------+
| P100        | Laptop      | 5000.00 | 6000.00 | 2022-01-14 |
| P12         | Non specifie |   12.00 |   14.00 | NULL       |
| P13         |             |  120.00 |  140.00 | 2022-01-01 |
+-------------+-------------+--------+--------+------------+
3 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> Select Num_produit, Date_ajout from Produits;
+-------------+------------+
| Num_produit | Date_ajout |
+-------------+------------+
| P100        | 2022-01-14 |
| P12         | NULL       |
| P13         | 2022-01-01 |
+-------------+------------+
3 rows in set (0.00 sec)
```

SELECT :

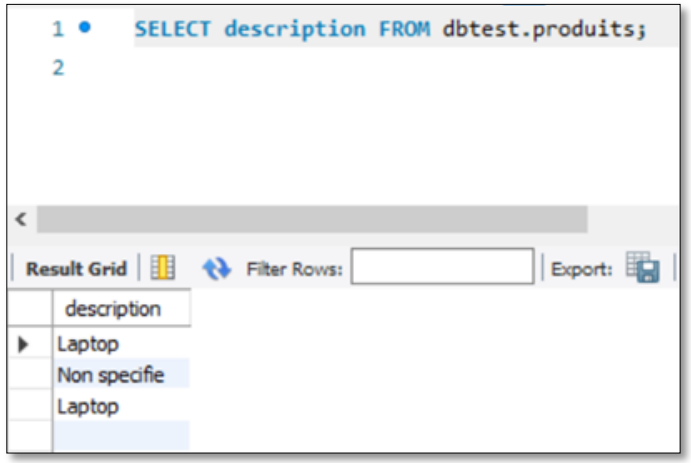
- **Options de la Requête SELECT :**
  - La requête SQL plus avancées prend la forme suivante :

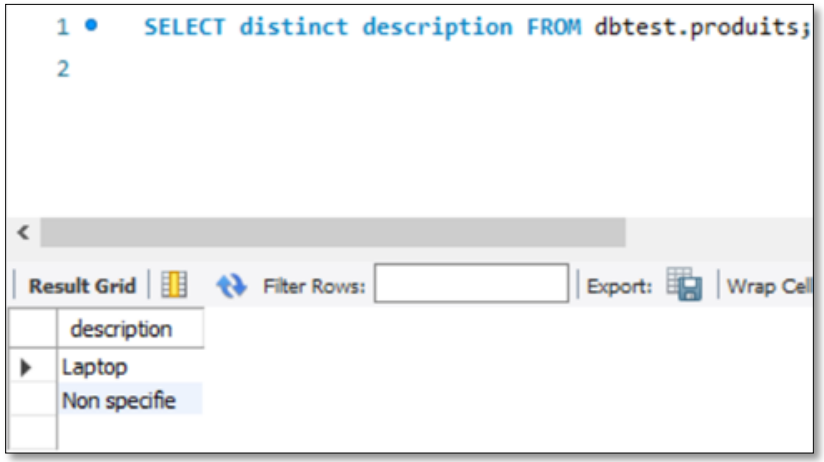
```
SELECT [DISTINCT] Liste_Select
FROM   Liste_Tables
WHERE  Liste_Conditions_Recherche
GROUP BY Liste_regroupement
HAVING  Liste_Conditions_regroupement
ORDER BY liste_Tri
```

  - MySQL exécute cette requête dans cet ordre :

DISTINCT :

- DISTINCT est une option qui permet de supprimer les lignes en double.





WHERE :

- WHERE définit la liste de conditions que les données recherchées doivent vérifier. La condition de recherche est une combinaison d'une ou plusieurs expressions utilisant l'opérateur logique **AND**, **OR** et **NOT**. L'instruction **SELECT** inclura toute ligne qui satisfait la condition de recherche dans le jeu de résultats.
- **WHERE** est aussi utilisé dans **UPDATE** ou **DELETE** pour spécifier les lignes à mettre à jour ou à supprimer.

```
1 •  SELECT * FROM dbtest.produits
2    WHERE Date_ajout >= '2022-01-14';
3
```

Result Grid | Filter Rows: | Edit:

Num_Produit	description	cout	prix	Date_ajout
P100	Laptop	5000.00	6000.00	2022-01-14
P120	Laptop	6000.00	7000.00	2022-01-14
NULL	NULL	NULL	NULL	NULL

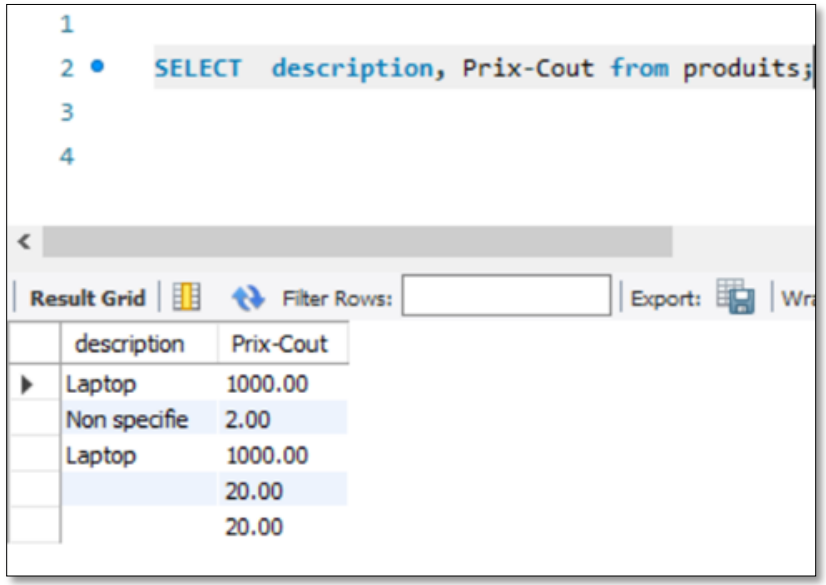
```
1 •  SELECT * FROM dbtest.produits
2    WHERE Date_ajout >= '2022-01-14'
3    AND Cout = 6000;
4
```

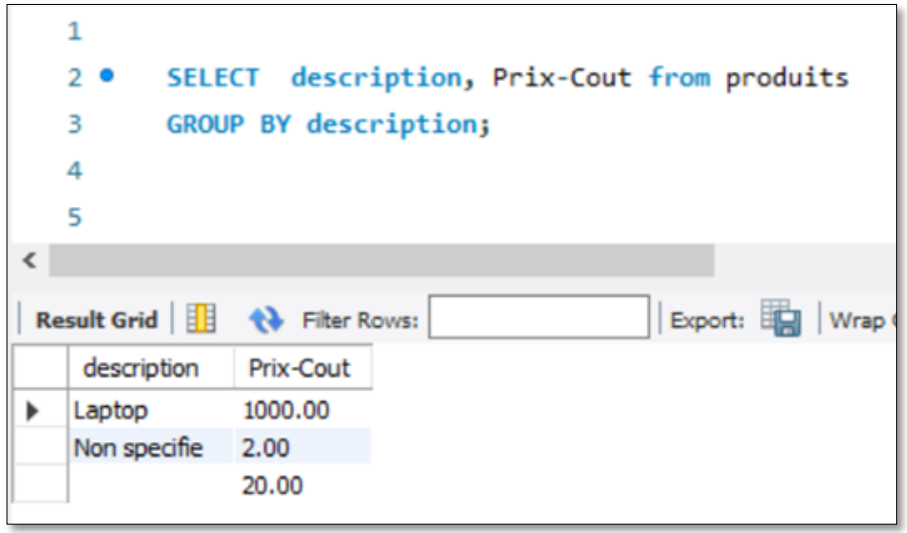
Result Grid | Filter Rows: | Edit:

Num_Produit	description	cout	prix	Date_ajout
P120	Laptop	6000.00	7000.00	2022-01-14
NULL	NULL	NULL	NULL	NULL

GROUP BY :

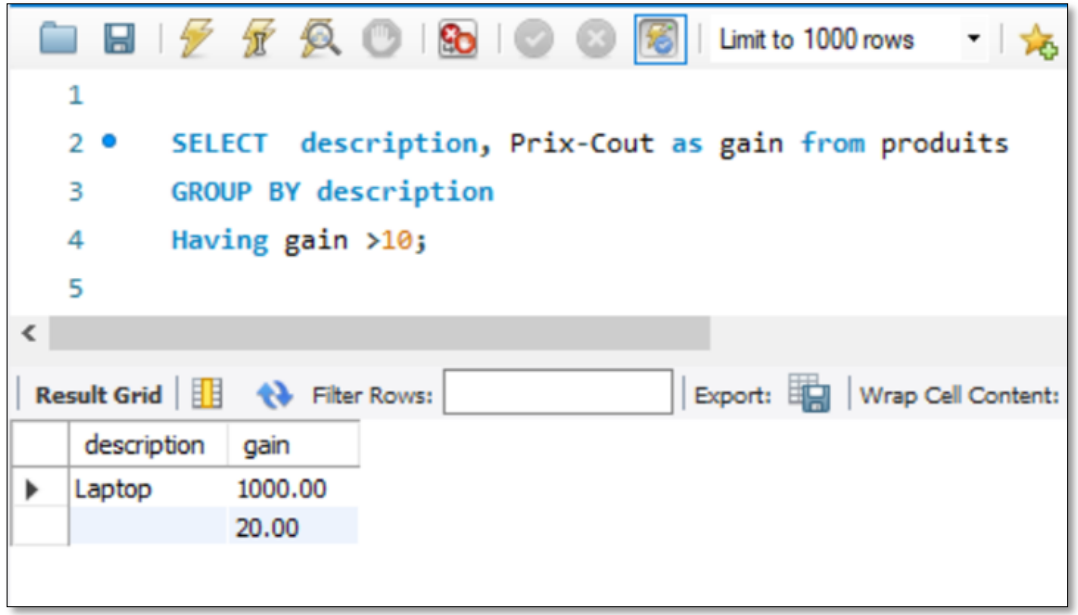
- La clause **GROUP BY** regroupe un ensemble de lignes dans un ensemble de lignes récapitulatives par valeurs de colonnes ou d'expressions. La clause **GROUP BY** renvoie une ligne pour chaque groupe, ceci réduit le nombre de lignes dans le jeu de résultats.
- En pratique, on utilise souvent la clause **GROUP BY** avec des fonctions d'agrégation telles que **SUM**, **AVG**, **MAX**, **MIN** et **COUNT**. La fonction d'agrégation qui apparaît dans la clause **SELECT** fournit les informations de chaque groupe.
- **Exemple :**





HAVING :

La clause **HAVING** est utilisée dans l'instruction **SELECT** pour spécifier des conditions de filtre pour un groupe de lignes ou d'agrégats. Elle est souvent utilisée avec **GROUP BY** pour filtrer les groupes en fonction d'une condition spécifiée. Si la clause **GROUP BY** est omise, **HAVING** se comporte comme la clause **WHERE**.



ORDER BY :

- La clause **ORDER BY** est utilisée pour trier les lignes du jeu de résultats. Elle peut porter sur plusieurs colonnes, chacune suivie, en option, de l'ordre de tri utilise croissant **ASC** ou décroissant **DESC**. L'ordre de tri par default étant **ASC**.
- **Exemple :**

```
1
2 •   SELECT  num_produit, Prix-Cout as gain from produits
3     order by gain ASC;
4
5
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

num_produit	gain
P12	2.00
P13	20.00
P14	20.00
P100	1000.00
P120	1000.00

PARTIE 3

CHAPITRE 2

Réaliser des requêtes SQL

1. Requêtes LMD
2. Requêtes de sélection
3. **Expression du SGBD**
4. Fonctions d'agrégation du SGBD
5. Sous requêtes
6. Requêtes de l'union
7. Jointures

- Une expression se compose d'ensemble de colonnes, constantes et fonctions combinées au moyen d'opérateurs. On trouve les expressions dans les différentes parties du SELECT : en tant que colonne résultat, dans les clauses **WHERE**, **HAVING**, et **ORDER BY**.
- Il existe trois types d'expressions selon le type de données de SQL :
    - **Expressions arithmétiques**
    - **Expressions de chaînes de caractères**
    - **Expressions de dates.**
- A chaque type correspondent des opérateurs et des fonctions spécifiques.
- Afin d'utiliser des données de types différents dans la même expression, on peut utiliser les fonctions de conversion dont dispose le langage **SQL**, celle-ci permettent de convertir des chaines de caractères en date ou en nombre selon le besoin.

Les opérateurs MYSQL :

- Un opérateur est un symbole spécifiant une action exécutée sur une ou plusieurs expressions. Nous trouvons en SQL, différentes catégories d'opérateurs.
- Voici les principaux utilisables dans les requêtes de sélection.
    - **Opérateurs arithmétiques**
    - **Opérateurs de comparaison**
    - **Opérateurs logiques**

Les opérateurs MYSQL : Opérateurs arithmétiques

- Les opérateurs arithmétiques présents dans SQL sont les suivants :
  - + addition
  - - soustraction
  - * multiplication
  - / division
- **Remarque :** la division par 0 provoque une fin avec code d'erreur.

Les opérateurs MYSQL : Opérateurs arithmétiques

- **Priorité des opérateurs :**
  - Une expression arithmétique peut comporter plusieurs opérateurs. Dans ce cas, le résultat de l'expression peut varier selon l'ordre dans lequel sont effectuées les opérations. Les opérateurs de multiplication et de division sont prioritaires par rapport aux opérateurs d'addition et de soustraction. Des parenthèses peuvent être utilisées pour forcer l'évaluation de l'expression dans un ordre différent de celui découlant de la priorité des opérateurs.
  - Au moyen des opérateurs arithmétiques + et - il est possible de construire les expressions suivantes :
    - date +/- nombre : le résultat est une date obtenue en ajoutant le nombre de jours nombre à la date.
    - date2 - date1 : le résultat est le nombre de jours entre les deux dates.

Les opérateurs MYSQL : Opérateurs arithmétiques

Exemple : Calcul du Gain = Prix-Cout pour chacun des produits :

```
1
2 •   SELECT  description, Prix-Cout from produits;
3
4
```

Result Grid | Filter Rows: | Export: | Wra

description	Prix-Cout
Laptop	1000.00
Non specifie	2.00
Laptop	1000.00
	20.00
	20.00

PARTIE 3

Les opérateurs MYSQL : Opérateurs de comparaison

- Les opérateurs de comparaison testent si deux expressions sont identiques.
- Ils peuvent s'utiliser sur toutes les expressions composées de données structurées.
- Ces opérateurs sont: = (Égal à) > (Supérieur à), < (Inférieur à), >= (Supérieur ou égal à), <= (Inférieur ou égal à), <> (Différent de)

Les opérateurs MYSQL : Opérateurs logiques

- Les opérateurs logiques testent la valeur logique d'une condition. Les opérateurs logiques, comme les opérateurs de comparaison, retournent un type de données booléen de valeur TRUE ou FALSE.
- Un certain nombre d'entre eux (signalés par un * dans le tableau) sont utilisés pour comparer une valeur scalaire (unique) avec une sous requête.

Opérateur	Description
ALL (*)	TRUE si tous les éléments dun jeu de comparaisons sont TRUE.
AND	TRUE les deux expressions booléennes sont TRUE.
ANY (*)	TRUE si n'importe quel élément d'un jeu de comparaison est TRUE.
BETWEEN	TRUE si l'opérande est situé dans une certaine plage.
EXISTS (*)	TRUE si une sous-requête contient des lignes.
IN (*)	TRUE si l'opérande est égal à un élément dune liste d'expressions.
LIKE	TRUE si l'opérande correspond à un modèle.
NOT	Inverse la valeur de tout autre opérateur booléen.
OR	TRUE si lune ou l'autre expression booléenne est TRUE.
SOME (*)	TRUE si certains éléments d'un jeu de comparaisons sont TRUE.
*	Utilisés avec des sous-requêtes

Les opérateurs MYSQL : Opérateurs logiques

Opérateur BETWEEN

- On utilise BETWEEN pour tester si une valeur est comprise entre une valeur minimale et une autre maximale. La syntaxe de l'opérateur BETWEEN : **valeur BETWEEN Minimum AND Maximum**

Exemple :

```
2 •   SELECT  * from produits
3     WHERE cout BETWEEN 5000 and 6000;
4
5
```

Result Grid | Filter Rows: | Edit:

Num_Produit	description	cout	prix	Date_ajout
P100	Laptop	5000.00	6000.00	2022-01-14
P120	Laptop	6000.00	7000.00	2022-01-14
NULL	NULL	NULL	NULL	NULL

Les opérateurs MYSQL : Opérateurs logiques

Opérateur LIKE/NOT LIKE

- On Utilise l'opérateur LIKE pour tester si une valeur correspond à un modèle spécifique. l'opérateur NOT sert à annuler l'opérateur LIKE.
- Le modèle est définit en utilisant les caractères génériques suivants :

Caractère générique	Description
%	Toute chaîne de zéro caractère ou plus
_	Nimporte quel caractère à cet emplacement
[ ]	Tout caractère de lintervalle ([a-f]) ou de lensemble spécifié ([abcdef])
[^]	Tout caractère en dehors de lintervalle ([^a-f]) ou de lensemble spécifié (^abcdef]). ^ représente le NOT

Exemples :

```
2 •   SELECT  * from produits
3     WHERE description like 'L%';
4
5
```

Result Grid | Filter Rows: | Edit:

Num_Produit	description	cout	prix	Date_ajout
P100	Laptop	5000.00	6000.00	2022-01-14
P120	Laptop	6000.00	7000.00	2022-01-14
NULL	NULL	NULL	NULL	NULL

La liste des produits dont la description commence par 'L'

```
2 •   SELECT  * from produits
3     WHERE Num_Produit like '%2_';
4
5
```

Result Grid | Filter Rows: | Edit:

Num_Produit	description	cout	prix	Date_ajout
P120	Laptop	6000.00	7000.00	2022-01-14
NULL	NULL	NULL	NULL	NULL

La liste des produits qui ont '2' dans la case avant dernière de leur Num_Produit.

PARTIE 3

Les fonctions intégrées MYSQL :

- MYSQL, comme les autres SGBD, propose de nombreuses fonctions intégrées qui permettent de manipuler les données utilisateurs ou les données du système.

Il existe plusieurs catégories de fonctions :

- Fonctions Mathématiques
- Fonctions de traitement de chaînes
- Fonctions de manipulation de dates
- Fonctions de conversion

- La liste exhaustive des fonctions MySQL est disponible sur le lien : https://dev.mysql.com/doc/refman/8.0/en/functions.html

Les fonctions intégrées MYSQL : Fonctions Mathématiques

- Ce sont des fonctions ayant un ou plusieurs nombres comme arguments, et qui renvoient une valeur numérique.

Voici quelques exemples :

Fonction	Description
ABS()	Retourne la valeur absolue d'un nombre.
CEIL()	Renvoie la plus petite valeur entière supérieure ou égale au nombre d'entrée.
FLOOR()	Renvoie la plus grande valeur entière non supérieure au nombre d'entrée.
MOD()	Renvoie le reste d'un nombre divisé par un autre
ROUND()	Arrondit un nombre à un nombre spécifié de places décimales.
TRUNCATE()	Tronque un nombre à un nombre spécifié de places décimales.

Les fonctions intégrées MYSQL : Fonctions de traitement de chaînes

Voici une liste contenant les fonctions de chaîne MySQL les plus utilisées qui permettent de manipuler efficacement les données de chaîne de caractères.

Name	Description
CONCAT	Concaténer deux ou plusieurs chaînes en une seule chaîne
INSTR	Renvoie la position de la première occurrence d'une sous-chaîne dans une chaîne
LENGTH	Obtenir la longueur d'une chaîne en octets et en caractères
LEFT	Obtient un nombre spécifié de caractères les plus à gauche d'une chaîne
LOWER	Convertir une chaîne en minuscule
LTRIM	Supprimer tous les espaces du début d'une chaîne
REPLACE	Recherche et remplace une sous-chaîne dans une chaîne
RIGHT	retourne un nombre spécifié de caractères les plus à droite d'une chaîne
RTRIM	Supprime tous les espaces de la fin d'une chaîne
SUBSTRING	Extraire une sous-chaîne à partir d'une position avec une longueur spécifique.
SUBSTRING_INDEX	Renvoie une sous-chaîne à partir d'une chaîne avant un nombre spécifié d'occurrences d'un délimiteur
TRIM	Supprime les caractères indésirables d'une chaîne
FIND_IN_SET	Rechercher une chaîne dans une liste de chaînes séparées par des virgules
FORMAT	Mettre en forme un nombre avec une locale spécifique, arrondi au nombre de décimales
UPPER	Convertir une chaîne en majuscule

Les fonctions intégrées MYSQL : Fonctions de traitement de chaînes

Exemple :

- Pour chaque produit, Retourner les valeurs suivantes :
  - Num_Produit
  - Une colonne « resultat » qui contient : Num_Produit 'Ajoute le' Date_ajout
  - Une colonne « nouveau » qui contient : Remplacer 'P' dans Num_Produit par 'NP'.

```
7
8  Select Num_Produit, CONCAT(Num_Produit,' ajoute le ', Date_ajout) as resultat,
9  REPLACE(Num_Produit,'P','NP') as Nouveau
10 from produits;
11
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Num_Produit	resultat	Nouveau
P100	P100 ajoute le 2022-01-14	NP100
P12	NULL	NP12
P120	P120 ajoute le 2022-01-14	NP120
P13	P13 ajoute le 2022-01-01	NP13
P14	P14 ajoute le 2022-01-01	NP14

PARTIE 3

Les fonctions intégrées MYSQL : Fonctions de manipulation de dates

Voici une liste contenant les fonctions de manipulation de dates MySQL les plus utilisées qui permettent de manipuler efficacement les données date :

Name	Description
CURDATE	Renvoie la date actuelle.
DATEDIFF	Calcule le nombre de jours entre deux valeurs DATE.
DAY	Obtient le jour du mois d'une date spécifiée.
DATE_ADD	Ajoute une valeur de temps à la valeur de date.
DATE_SUB	Soustrait une valeur d'heure d'une valeur de date.
DATE_FORMAT	Formate une valeur de date en fonction d'un format de date spécifié.
DAYNAME	Obtient le nom d'un jour de la semaine pour une date spécifiée.
DAYOFWEEK	Renvoie l'index des jours de la semaine pour une date.
EXTRACT	Extrait une partie d'une date.
LAST_DAY	Renvoie le dernier jour du mois d'une date spécifiée
NOW	Renvoie la date et l'heure actuelles d'exécution de l'instruction.
MONTH	Renvoie un entier qui représente un mois d'une date spécifiée.
STR_TO_DATE	Convertit une chaîne en une valeur de date et d'heure basée sur un format spécifié.
SYSDATE	Renvoie la date actuelle.
TIMEDIFF	Calcule la différence entre deux valeurs TIME ou DATETIME.
TIMESTAMPDIFF	Calcule la différence entre deux valeurs DATE ou DATETIME.
WEEK	Renvoie le numéro de semaine d'une date
WEEKDAY	Renvoie un index des jours de la semaine pour une date.
YEAR	Renvoie l'année pour une date spécifiée

Les fonctions intégrées MYSQL : Fonctions de manipulation de dates

Exemple :

- La liste des produits, leur date d'ajout, l'année d'ajout, Le jour de la semaine, et depuis combien de jours ils ont été ajoutés.

```
1
2 •  SELECT  Num_Produit, Date_ajout, Year(Date_ajout) as Annee, DAYNAME(Date_ajout) as Jour,
3    DATEDIFF(SYSDATE(), Date_ajout) as Depuis_j
4    from produits
5    ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Num_Produit	Date_ajout	Annee	Jour	Depuis_j
P100	2022-01-14	2022	Friday	2
P12	NULL	NULL	NULL	NULL
P120	2022-01-14	2022	Friday	2
P13	2022-01-01	2022	Saturday	15
P14	2022-01-01	2022	Saturday	15

PARTIE 3

Les fonctions intégrées MYSQL : Fonctions de conversion

- **TO_CHAR(nombre,format)** - Renvoie la chaîne de caractères en obtenue en convertissant nombre en fonction de format.
- **TO_CHAR(date,format)** - Renvoie conversion d'une date en chaîne de caractères. Le format indique quelle partie de la date doit apparaître.
- **TO_DATE(chaîne,format)** - Permet de convertir une chaîne de caractères en donnée de type date. Le format est identique à celui de la fonction TO_CHAR.
- **TO_NUMBER(chaine)** - Convertit chaine en sa valeur numérique.

CHAPITRE 2

Réaliser des requêtes SQL

- Une fonction d'agrégation effectue un calcul sur plusieurs valeurs et renvoie une seule valeur.
- Les fonctions d'agrégation les plus utilisés : SUM, AVG, COUNT, MAX et MIN.
- Les fonctions d'agrégation sont souvent utilisées avec la clause GROUP BY pour calculer une valeur agrégée pour chaque groupe, par exemple la valeur moyenne par groupe ou la somme des valeurs dans chaque groupe.

La fonction SUM() :

- La fonction SUM() retourne la somme des valeurs d'une colonne.
- L'image suivante illustre l'utilisation de la fonction d'agrégation SUM() avec une clause GROUP BY :

Nom	Valeur
A	10
A	20
B	40
C	20
C	50

Σ

```
SELECT
        Nom,
        SUM(Valeur)
FROM
        Sample_table
GROUP BY
        Nom;
```

Nom	SUM (Valeur)
A	30
B	40
C	70

La fonction SUM() :

Exemple :

```
12 •  select description, Sum(cout) from Produits
13    group by description;
14
15
```

Result Grid | Filter Rows: | Export: | Wrap

description	Sum(cout)
Laptop	11000.00
Non specifie	12.00
	140.00

La fonction AVG() :

- La fonction AVG() retourne la moyenne des valeurs d'une colonne.

Exemple :

```
18 •  select description, AVG(cout) from Produits
19    group by description;
20
```

Result Grid | Filter Rows: | Export: | Wrap Cell

description	AVG(cout)
Laptop	5500.000000
Non specifie	12.000000
Accessoires	70.000000

PARTIE 3

La fonction COUNT() :

- La fonction COUNT() retourne le nombre d'enregistrements sélectionnés.

Exemple :

```
18 •  select description, count(Num_produit) from Produits
19    group by description;
20
21
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

description	count(Num_produit)
Laptop	2
Non specifie	1
Accessoires	2

La fonction MAX/MIN :

- Les fonctions **MAX**() et **MIN**() retournent respectivement le maximum et le minimum des valeurs des enregistrements sélectionnés.

Exemple :

```
18 •    select  MAX(Prix) from Produits;
19
```

Result Grid | Filter Rows: | Export:

MAX(Prix)
7000.00

MAX (Prix)

```
18 •    select  MIN(Prix) from Produits;
19
```

Result Grid | Filter Rows: | Export:

MIN(Prix)
14.00

MIN (Prix)

CHAPITRE 2

Réaliser des requêtes SQL

- Une sous-requête est une requête imbriquée dans une autre requête telle que SELECT, INSERT, UPDATE ou DELETE.
- Une sous-requête est appelée « requête interne » tandis que la requête qui contient la sous-requête est appelée une requête externe. La requête dite interne est évaluée pour chaque ligne de la requête externe.
- Une sous-requête peut être utilisée partout où une expression est utilisée et doit être fermée entre parenthèses.

Exemples :

- La table Produits :

Num_Produit	description	cout	prix	Date_ajout	Max(Prix)
P100	Laptop	5000.00	6000.00	2022-01-14	6000.00
P12	Non specifie	12.00	14.00	NULL	14.00
P120	Laptop	6000.00	7000.00	2022-01-14	7000.00
P13	Accessoires	120.00	140.00	2022-01-01	140.00
P14	Accessoires	20.00	40.00	2022-01-01	40.00

Sous-Requête Mono-ligne :

- On utilise des opérateurs de comparaison, par exemple =, >, < pour comparer une seule valeur renvoyée par la sous-requête avec l'expression dans la clause WHERE.
- Afficher le Produit ayant le Prix minimale :
  - **Select Num_produit, Description, Prix from Produits**
  - **where Prix = (Select Min(prix) from Produits)**

```
18 •  select  Num_produit, Description, Prix from Produits
19    where Prix = (Select Min(prix) from Produits);
20
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Num_produit	Description	Prix
P12	Non specifie	14.00

Sous-Requête avec IN/NOT IN:

- Si une sous-requête renvoie plusieurs valeurs, on peut utiliser d'autres opérateurs tels que l'opérateur IN ou NOT IN dans la clause WHERE.
- Afficher la liste des produits de la table Produits qui existent sur la table Sales :
  - **Select * from Produits**
  - **where Num_produit IN (Select Num_produits from Sales)**

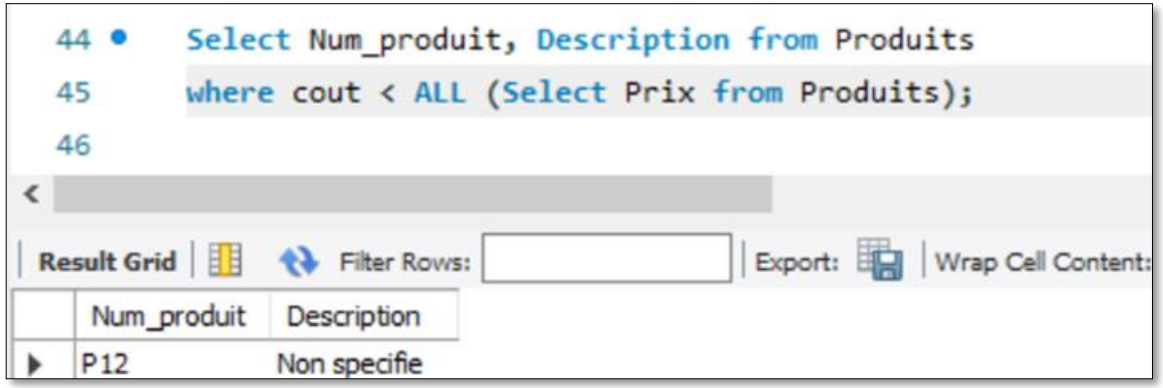
```
21 •    select  * from Produits
22      where Num_produit IN  (Select Num_produits from Sales);
23
```

Result Grid | Filter Rows: | Edit: | Export/Imp

Num_Produit	description	cout	prix	Date_ajout
P100	Laptop	5000.00	6000.00	2022-01-14
P12	Non specifie	12.00	14.00	NULL
P120	Laptop	6000.00	7000.00	2022-01-14

Sous-Requête avec ALL/ANY

- On peut utiliser les opérateurs **ANY**, ou **ALL** pour comparer la valeur d'une expression avec les valeurs d'un attribut d'une requête interne.
- Le mot-clé ALL spécifie tous les éléments retournés tandis que le mot-clé **ANY** spécifie l'un d'entre eux.
- Afficher les produits dont le cout est Inferieur à tous les prix des produits.
    - **Select Num_produit, Description from Produits where cout < ALL (Select Prix from Produits)**



Sous-Requête avec ALL/ANY

- On peut utiliser les opérateurs **ANY**, ou **ALL** pour comparer la valeur d'une expression avec les valeurs d'un attribut d'une requête interne.
- Le mot-clé ALL spécifie tous les éléments retournés tandis que le mot-clé **ANY** spécifie l'un d'entre eux.
- Afficher les Produits dont le prix est inferieur à un des couts de produits :
    - **Select Num_produit, Description from Produits**
    - **where prix < ANY (Select cout from Produits)**

```
44 •  Select Num_produit, Description from Produits
45    where prix < ANY (Select cout from Produits);
46
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Num_produit	Description
P12	Non specifie
P13	Accessoires
P14	Accessoires

Sous-Requête avec EXISTS/NOT EXISTS

- Lorsqu'une sous-requête est utilisée avec l'opérateur EXISTS ou NOT EXISTS, une sous-requête renvoie une valeur booléenne TRUE ou FALSE.
- Afficher les produits avec une colonne supplémentaire indiquant si le produit est présent la table Sales :
  - **Select P.Num_produit, P.Description,**
  - **EXISTS(Select * from Sales S Where P.Num_produit = S.Num_Produit) as existe**
  - **From Produits P**

```
25 •   Select  P.Num_produit, P.Description,
26     EXISTS(Select * from Sales S
27     Where P.Num_produit = S.Num_Produit) as existe
28     From Produits P;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Num_produit	Description	existe
P100	Laptop	1
P12	Non specifie	1
P120	Laptop	1
P13	Accessoires	1
P14	Accessoires	1

CHAPITRE 2

Réaliser des requêtes SQL

- La combinaison des résultats des requêtes consiste à transformer deux ou plusieurs jeux de résultat en un seul.
- Les jeux de résultats combinés doivent tous avoir la même structure : Même nombre de colonnes et même types de données.
- Il existe trois types de combinaison : UNION (UNION All), INTERSECT, MINUS.

Opérateur UNION

- L'opérateur UNION permet de combiner deux ou plusieurs ensembles de résultats de requêtes en un seul ensemble de résultats. Voici la syntaxe d'utilisation de l'opérateur UNION :

```
SELECT column_list1
UNION [DISTINCT | ALL]
SELECT column_list2
UNION [DISTINCT | ALL]
SELECT column_list3
...
```

Opérateur UNION

- Par défaut, l'opérateur **UNION** supprime les lignes en double même si on ne spécifie pas l'opérateur DISTINCT.
- Soient les deux tables :

GROUP1
ID
1
2
3

GROUP2
ID
2
3
4

- La requête UNION :

```
1    SELECT ID FROM Group1
2    UNION
3    SELECT ID FROM Group2;
4
```

Result Grid | Filter Rows: | Export:

ID
1
2
3
4

PARTIE 3

Opérateur UNION

- La requête UNION ALL :

```
1    SELECT ID FROM Group1
2    UNION ALL
3    SELECT ID FROM Group2;
```

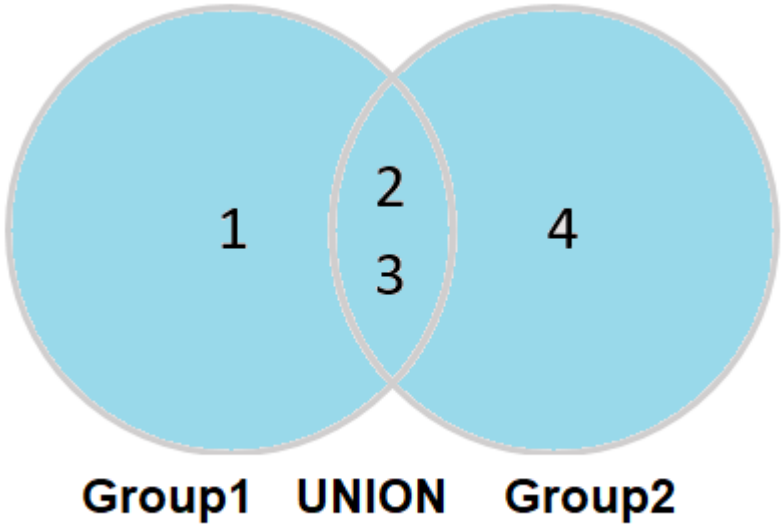
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID
1
2
3
2
3
4

- Si on utilise UNION ALL explicitement, les lignes dupliquées, sont affichées dans le résultat. Du fait que UNION ALL ne gère pas les doublons, il s'exécute plus rapidement que UNION DISTINCT.

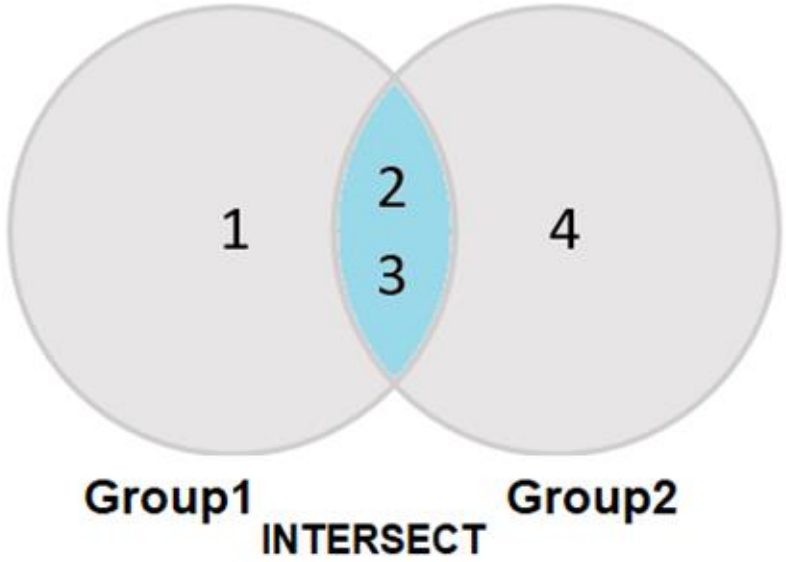
Opérateur UNION

- Le diagramme de Venn suivant illustre l'union de deux ensembles de résultats provenant des tables Group1 et Group2 :



Opérateur INTERSECT

- L'opérateur INTERSECT compare les ensembles de résultats de deux requêtes ou plus et renvoie les lignes distinctes générées par les deux requêtes.
- MySQL ne prend pas en charge l'opérateur INTERSECT. Cependant, On peut l'émuler en utilisant les jointures.
- Le schéma suivant illustre l'opérateur INTERSECT :



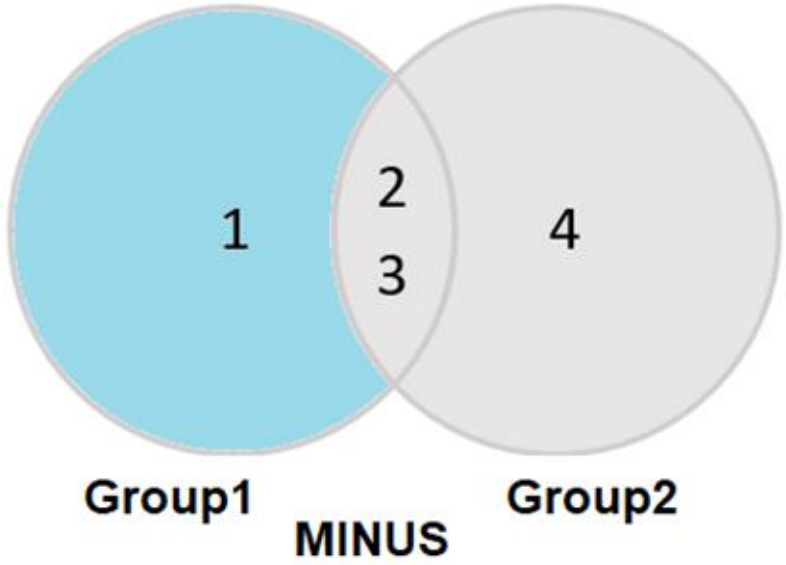
Opérateur INTERSECT

- Contrairement à l'opérateur UNION, l'opérateur INTERSECT renvoie l'intersection entre deux cercles.
- **Donc la requête :**
  - (SELECT ID FROM Group1)
  - INTERSECT
  - (SELECT ID FROM Group2)
- **Retourne :**

ID
2
3

Opérateur MINUS (EXCEPT)

- L'opérateur MINUS compare les résultats de deux requêtes et renvoie des lignes distinctes du jeu de résultats de la première requête qui n'apparaissent pas dans le jeu de résultats de la deuxième requête. MySQL ne prend pas en charge l'opérateur MINUS. Cependant, On peut l'émuler en utilisant les jointures.
- Le schéma suivant illustre l'opérateur MINUS :



- **Donc la requête :**
  - (SELECT ID FROM Group1)
  - MINUS
  - (SELECT ID FROM Group2)
- **Retourne :**

ID
1

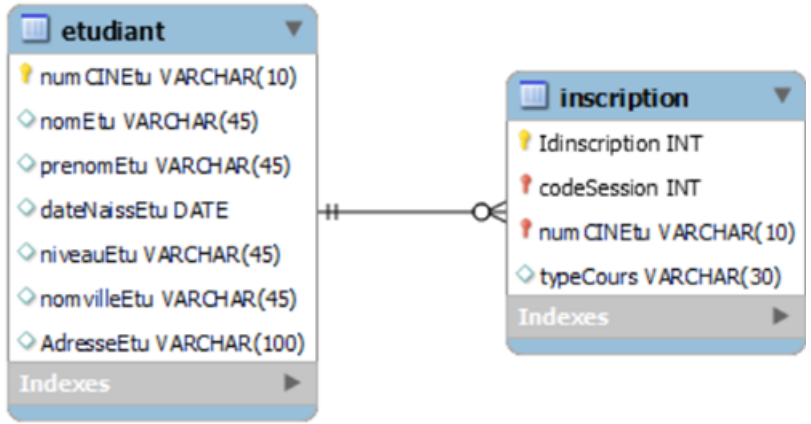
PARTIE 3

CHAPITRE 2

Réaliser des requêtes SQL

1. Requêtes LMD
2. Requêtes de sélection
3. Expression du SGBD
4. Fonctions d’agrégation du SGBD
5. Sous requêtes
6. Requêtes de l’union
7. **Jointures**

- Une base de données relationnelle se compose de plusieurs tables liées entre elles à l'aide de colonnes communes, appelées clés étrangères.
- Dans l'exemple Centre de Formation déjà présenté dans ce cours nous trouvons que les deux tables « Etudiant » et « Inscription » sont liées a l'aide de la colonne : numCINEtu



- Afin d'avoir plus d'information sur les étudiants ou les inscriptions, on a besoin de chercher dans les deux tables à la fois. D'où la nécessité des jointures.
- La **jointure** consiste à rechercher entre deux tables ayant un attribut commun (même type et même domaine de définition) tous les tuples (toutes les lignes) pour lesquels ces attributs ont la même valeur.
    - **MySQL** prend en charge les types de jointures suivants :
        - **INNER JOIN** (Jointure interne)
        - **LEFT JOIN** (Joint gauche)
        - **RIGHT JOIN** (Joindre à droite)
        - **CROSS JOIN** (Jointure croisée)
- Pour joindre des tables, On utilise la clause de jointure dans l'instruction SELECT après la clause FROM.
- Notez que MySQL ne prend pas en charge la jointure FULL OUTER JOIN.

INNER JOIN

- INNER JOIN joint deux tables en fonction d'une condition connue sous le nom de prédicat de jointure.
- Elle spécifie ainsi toutes les paires correspondantes de lignes renvoyées et ignore les lignes n'ayant pas de correspondance entre les deux tables. La clause INNER JOIN ne retient que les lignes des deux tables pour lesquelles l'expression exprimée au niveau de ON se vérifie.

Exemple :

Num_Produit	description	cout	prix	Date_ajout
P100	Laptop	5000.00	6000.00	2022-01-14
P12	Non specifie	12.00	14.00	NULL
P120	Laptop	6000.00	7000.00	2022-01-14
P13	Accessoires	120.00	140.00	2022-01-01
P14	Accessoires	20.00	40.00	2022-01-01

Produits

IdSales	Num_Produit	Quantite	Client	Date_Vente
1195	P120	2	Client 8	2021-10-20
1196	P13	50	Client 8	2021-10-20
1198	P100	6	Client 1	2021-11-30
1201	P12	12	Client 1	2022-01-15

Sales

INNER JOIN

- Afin d'avoir pour chaque produit vendu, son prix, sa description et la quantité qui a été vendue, nous avons besoin de joindre les deux tables comme suit :

```
SELECT
    P.Num_Produit,
    P.Description,
    P.prix,
    S.quantite

FROM
    Produits P
INNER JOIN Sales S ON P.Num_Produit=S.Num_Produit;
```

INNER JOIN

- Résultat de la requête :

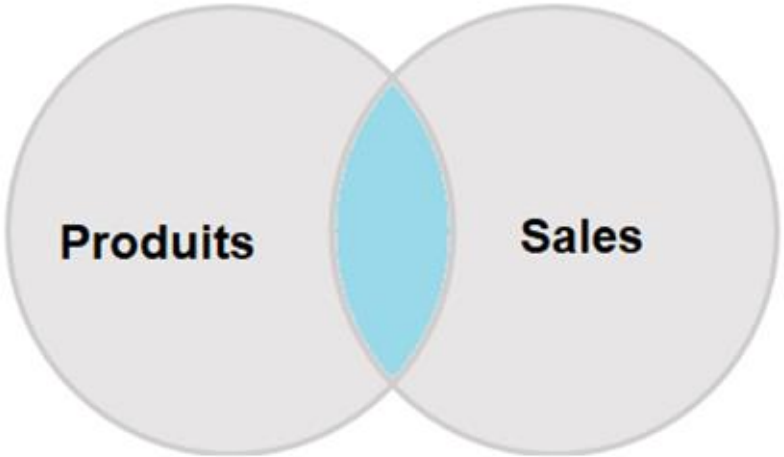
```
1    SELECT
2        P.Num_Produit,
3        P.Description,
4        P.prix,
5        S.Quantite
6    FROM
7        Produits P
8    INNER JOIN Sales S ON P.Num_Produit=S.Num_Produit;
9
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Num_Produit	Description	prix	Quantite
P120	Laptop	7000.00	2
P13	Accessoires	140.00	50
P100	Laptop	6000.00	6
P12	Non specifie	14.00	12

INNER JOIN

- Dans cet exemple, INNER JOIN utilise les valeurs des colonnes de « Num_Produit » dans les tables « Produits » et « Sales » pour faire la correspondance.
- Le diagramme de Venn suivant illustre la jointure interne :



INNER JOIN

- Si la colonne de jointure a le même nom dans les deux tables objets de la jointure, on peut utiliser la syntaxe suivante :

```
SELECT
    P.Num_Produit,
    P.Description,
    P.prix,
    S.quantite

FROM
    Produits P
INNER JOIN Sales S USING (Num_Produit);
```

LEFT JOIN

- Lors de la jointure de deux tables à l'aide d'une jointure gauche, les concepts de tables gauche et droite sont introduits.
- La jointure gauche sélectionne les données à partir de la table de gauche. Pour chaque ligne de la table de gauche, la jointure de gauche est comparée à chaque ligne de la table de droite.
- Si les valeurs des deux lignes satisfont la condition de jointure, la clause de jointure gauche crée une nouvelle ligne dont les colonnes contiennent toutes les colonnes des lignes des deux tables et inclut cette ligne dans le jeu de résultats.
- Si les valeurs des deux lignes ne correspondent pas, la clause de jointure gauche crée toujours une nouvelle ligne dont les colonnes contiennent les colonnes de la ligne de la table de gauche et NULL pour les colonnes de la ligne de la table de droite.
- En d'autres termes, la jointure gauche sélectionne toutes les données de la table de gauche, qu'il existe ou non des lignes correspondantes dans la table de droite.

LEFT JOIN

Exemple :

```
SELECT
    P.Num_Produit,
    P.Description,
    P.prix,
    S.quantite

FROM
    Produits P
LEFT JOIN Sales S ON P.Num_Produit=S.Num_Produit;
```

- Ou Alors : **LEFT JOIN Sales S USING(Num_Produit).**

LEFT JOIN

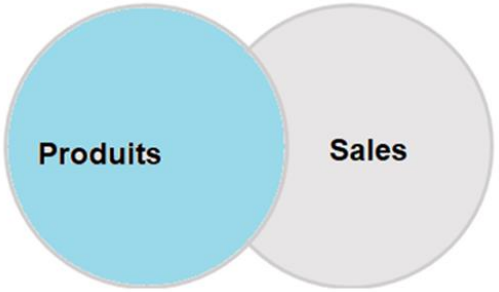
- Le résultat est le suivant :

```
1   SELECT
2       P.Num_Produit,
3       P.Description,
4       P.prix,
5       S.Quantite
6   FROM
7       Produits P
8   Left JOIN Sales S ON P.Num_Produit=S.Num_Produit;
9
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Num_Produit	Description	prix	Quantite
P120	Laptop	7000.00	2
P13	Accessoires	140.00	50
P100	Laptop	6000.00	6
P12	Non specifie	14.00	12
P14	Accessoires	40.00	NULL

- Le diagramme de Venn suivant illustre la jointure gauche :



RIGHT JOIN

- La clause de jointure droite est similaire à la clause de jointure gauche sauf que le traitement des tables gauche et droite est inversé. La jointure droite commence à sélectionner les données de la table de droite au lieu de la table de gauche : RIGHT JOIN sélectionne toutes les lignes de la table de droite et fait correspondre les lignes de la table de gauche. Si une ligne de la table de droite n'a pas correspondances dans la table de gauche, la colonne de la table de gauche aura NULL dans le jeu de résultats final.

RIGHT JOIN

Exemple :

```
SELECT
    S.Num_Produit,
    S.Quantite,
    P.prix
FROM
    Sales S
Right JOIN Produits P ON P.Num_Produit=S.Num_Produit;
```

- Ou Alors : **RIGHT JOIN Produits P USING(Num_Produit).**

RIGHT JOIN

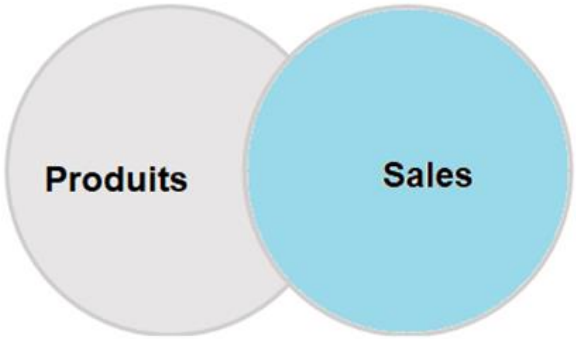
- Le résultat est le suivant :

```
1    SELECT
2        S.Num_Produit,
3        S.Quantite,
4        P.prix
5    FROM
6        Sales S
7    Right JOIN Produits P ON P.Num_Produit=S.Num_Produit;
8
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Num_Produit	Quantite	prix
P120	2	7000.00
P13	50	140.00
P100	6	6000.00
P12	12	14.00
NULL	NULL	40.00

- Le diagramme de Venn suivant illustre la jointure droite :



CROSS JOIN

- Contrairement à autres types de jointures, la jointure croisée n'a pas de condition de jointure. Elle crée le produit cartésien des lignes des tables jointes. La jointure croisée combine chaque ligne de la première table avec chaque ligne de la table de droite pour créer le jeu de résultats.

Exemple :

```
1    SELECT
2        P.Num_Produit,
3        P.description,
4        P.prix
5    FROM
6      Produits P
7    Cross JOIN Sales S;
```

Result Grid | Filter Rows: | Export:

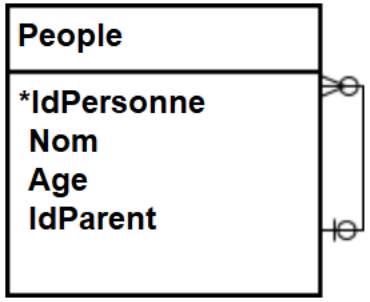
Num_Produit	description	prix
P100	Laptop	6000.00
P100	Laptop	6000.00
P100	Laptop	6000.00
P100	Laptop	6000.00
P12	Non specifie	14.00
P12	Non specifie	14.00
P12	Non specifie	14.00
P12	Non specifie	14.00
P120	Laptop	7000.00
P120	Laptop	7000.00
P120	Laptop	7000.00
P120	Laptop	7000.00
P13	Accessoires	140.00
P13	Accessoires	140.00

SELF JOIN

- L'auto-jointure est souvent utilisée pour interroger des données hiérarchiques ou pour comparer une ligne avec d'autres lignes dans la même table.
- Pour effectuer une auto-jointure, on utilise des alias de la table pour ne pas répéter deux fois le même nom de table dans la même requête.
- **N.B :** Référencer une table deux fois ou plus dans une requête sans utiliser d'alias de table provoquera une erreur.

Exemple :

- La table « People » est définie ainsi :



idPersonne	Nom	Age	idParent
12	Mohammad	35	2
2	Abdullah	54	1
14	Ibrahim	1	12
5	Ali	12	3
3	Talib	56	1
1	Mottalib	70	1

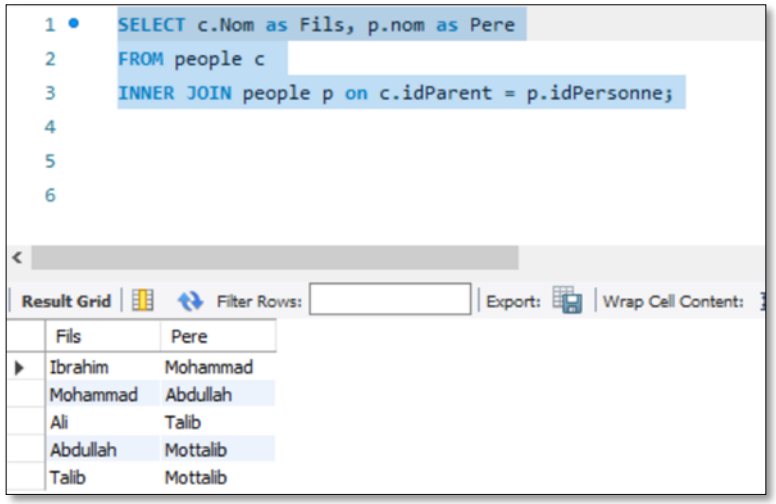
- La colonne IdParent définit le père de chaque personne, qui est aussi un élément de la table » People »

SELF JOIN

- Afin d'avoir la liste des personnes et leurs parents, il faut utiliser une auto-jointure.
- Au moyen de **INNER JOIN** :

```
SELECT c.Nom as Fils, p.nom as Pere
FROM people c
INNER JOIN people p on c.idParent = p.idPersonne;
```

- Le résultat nous donne uniquement les Personnes ayant un parent définit :



SELF JOIN

- Afin d'avoir la liste des personnes et leurs parents, il faut utiliser une auto-jointure.
- Au moyen de **LEFT JOIN** :
- Le résultat comprend même les personnes qui n'ont pas un père défini.

Exemple :

```
1 •   SELECT c.Nom as Fils, p.nom as Pere
2     FROM people c
3     Left JOIN people p on c.idParent = p.idPersonne;
4
5
6
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Fils	Pere
Ibrahim	Mohammad
Mohammad	Abdullah
Ali	Talib
Abdullah	Mottalib
Talib	Mottalib
Mottalib	NULL

PARTIE 3

CHAPITRE 3

Administrer une base de données

Ce que vous allez apprendre dans ce chapitre :

- Maîtriser les différentes fonctions liées à l'administration des BDD
- Vous initier avec les commandes de gestion des comptes et de privilèges de base

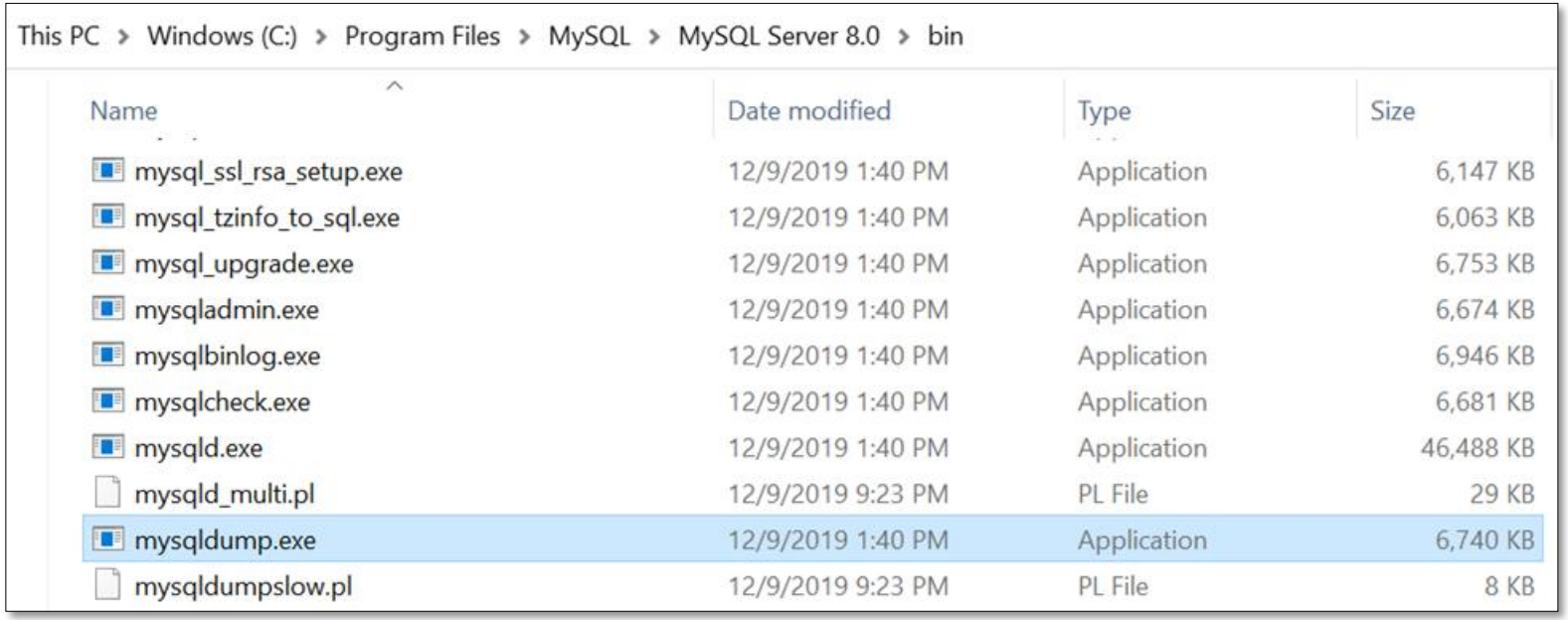
CHAPITRE 3

Administrer une base de données

1. **Backup/Restore**
2. Importation
3. Exportation
4. Commandes de création des comptes utilisateurs
5. Commandes de gestion des privilèges de base

Backup

L'outil mysqldump permet de sauvegarder une ou plusieurs bases de données. Il génère un fichier texte contenant les instructions SQL qui peuvent recréer les bases de données. mysqldump est situé dans le répertoire root/bin du répertoire d'installation de MySQL.



Backup d'une ou plusieurs bases de données :

- La commande pour faire un backup avec mysqldump :

```
mysqldump --user=<username>
--password=<password>
--result-file=<Lien_Fichier_Backup>
--databases <Liste_des_databases>
```

- Dans cette syntaxe on doit définir :
  - Username et password: le nom et le mot de passe de l'utilisateur qui est connecté sur MySQL
  - Lien du fichier du backup
  - Le ou les noms des bases de données qu'on veut sauvegarder.
- Pour faire un backup de plusieurs bases de données à la fois on suit la syntaxe suivante : **--databases Nom_Base1, Nom_Base2, …**
- Si on veut faire un backup de toutes les bases de données d'une instance MySql, on remplace l'option : **--databases <Liste_des_databases>**
- Par : **--all-databases**

Backup d'une ou plusieurs bases de données

Exemples : Backup de la base de données « dbtest »

PARTIE 3

Backup d'une ou plusieurs bases de données

Exemples : Backup de la base de données « dbtest »

Le contenu du fichier dbtest_backup.sql après exécution :

dbtest_backup.sql

```
18  --
19  -- Current Database: `dbtest`
20  --
21
22  CREATE DATABASE /*!32312 IF NOT EXISTS*/ `dbtest` /*!40100 DEFAULT CHARACTER SET utf8mb4
23
24  USE `dbtest`;
25
26  --
27  -- Table structure for table `group1`
28  --
29
30  DROP TABLE IF EXISTS `group1`;
31  /*!40101 SET @saved_cs_client     = @@character_set_client */;
32  /*!50503 SET character_set_client = utf8mb4 */;
33  CREATE TABLE `group1` (
34    `id` int NOT NULL,
35    PRIMARY KEY (`id`)
36  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
37  /*!40101 SET character_set_client = @saved_cs_client */;
38
39  --
40  -- Dumping data for table `group1`
41  --
42
43  LOCK TABLES `group1` WRITE;
44  /*!40000 ALTER TABLE `group1` DISABLE KEYS */;
45  INSERT INTO `group1` VALUES (1),(2),(3);
46  /*!40000 ALTER TABLE `group1` ENABLE KEYS */;
47  UNLOCK TABLES;
48
```

Backup d'une ou plusieurs tables d'une base de données

- Afin de faire un backup de tables spécifiques d'une base de données, on exécute :

mysqldump --user=<username> --password=<password> --result-file=<path_to_backup_file> <Nom_Base> <table1> <table2> <table3>..

Exemple :

- Réaliser le backup des tables « Produits » et « Sales » de la base de données « dbtest »

```
mysqldump --user=root --password=Mypassword4@
--result-file=c:\backup\backup_tables.sql dbtest produits sales
```

Backup de la structure ou les données d'une bases de données

- L'outil mysqldump permet aussi de sauvegarder juste la structure ou juste les données d'une base de données en utilisant respectivement les options : **--no-data et –-no-create-info**
- **Structure seulement :**

```
mysqldump --user=<username>
--password=<password>
--result-file=<Lien_Fichier_Backup>
--no-data
--databases <Liste_des_databases>
```

- **Données seulement :**

```
mysqldump --user=<username>
--password=<password>
--result-file=<Lien_Fichier_Backup>
--no-create-info
--databases <Liste_des_databases>
```

CHAPITRE 3

Administrer une base de données

1. Backup/Restore
2. **Importation**
3. Exportation
4. Commandes de création des comptes utilisateurs
5. Commandes de gestion des privilèges de base

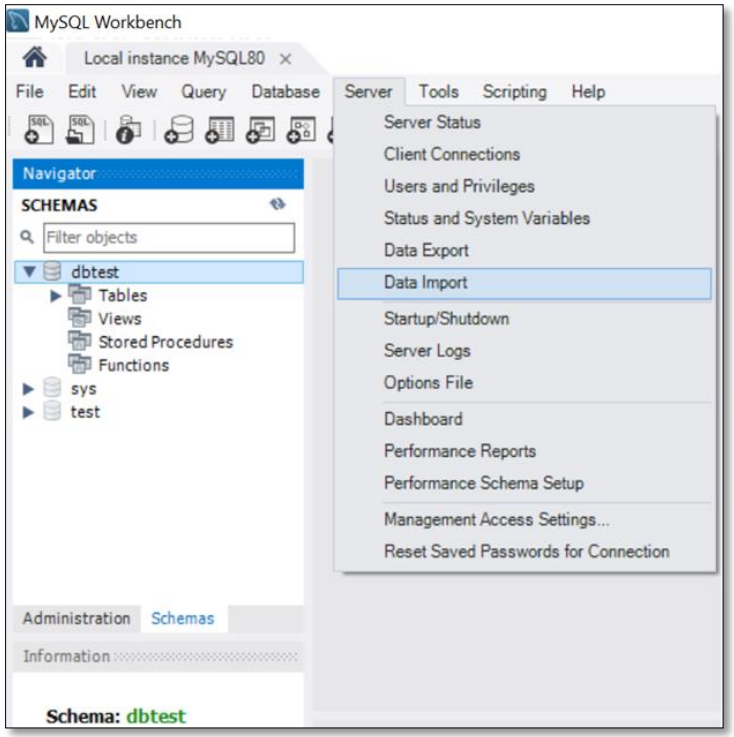
- Afin d'importer des données sous forme de fichier sql sur MySQL à partir de la ligne de commande, on peut utiliser la commande SOURCE comme dans le cas du Restore.

Exemple :

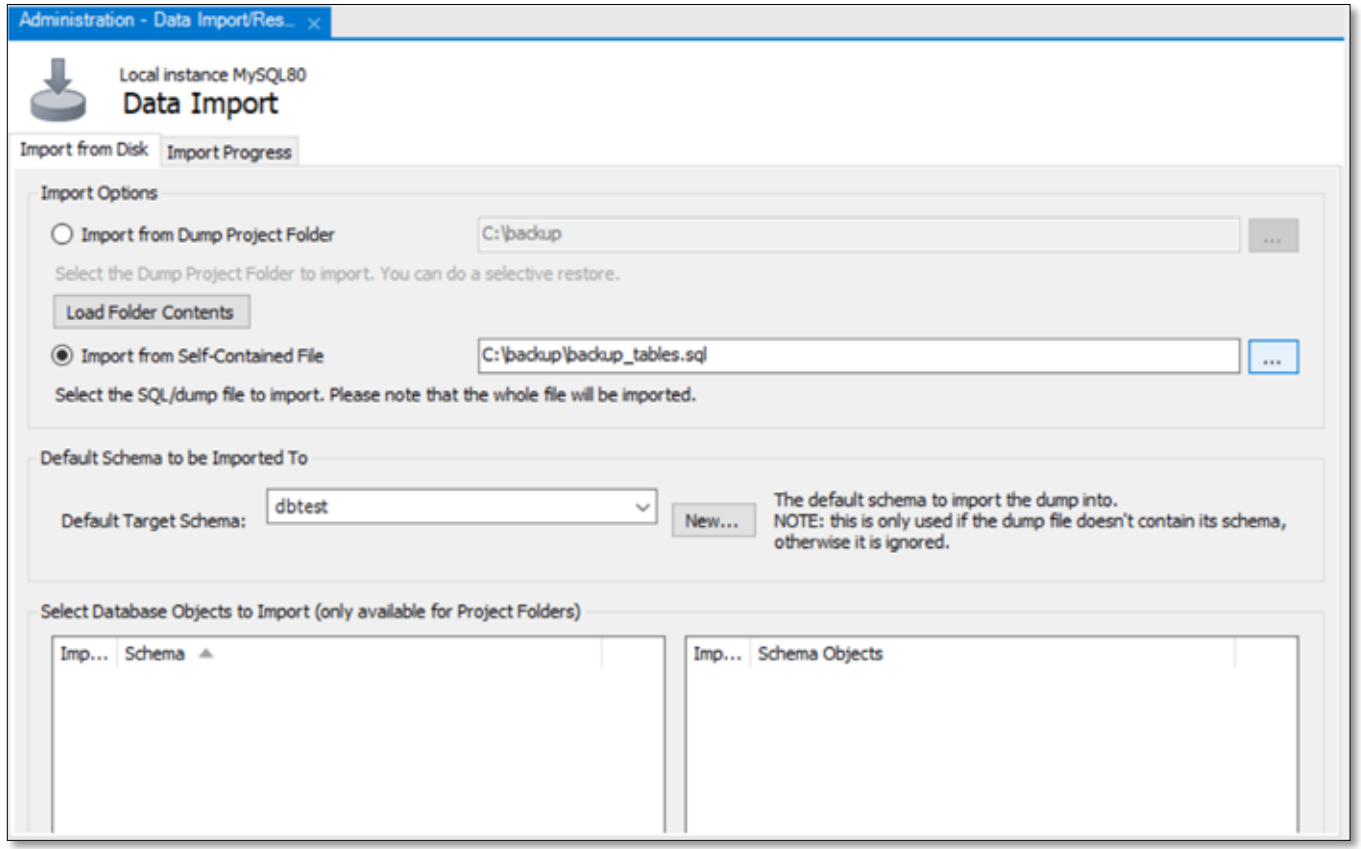
```
mysql>source c:\backup\fichier_backup.sql
```

- Il est recommandé d'utiliser la commande SOURCE pour restaurer une base de données car elle renvoie des informations très détaillées sur le processus, notamment des avertissements et des erreurs.

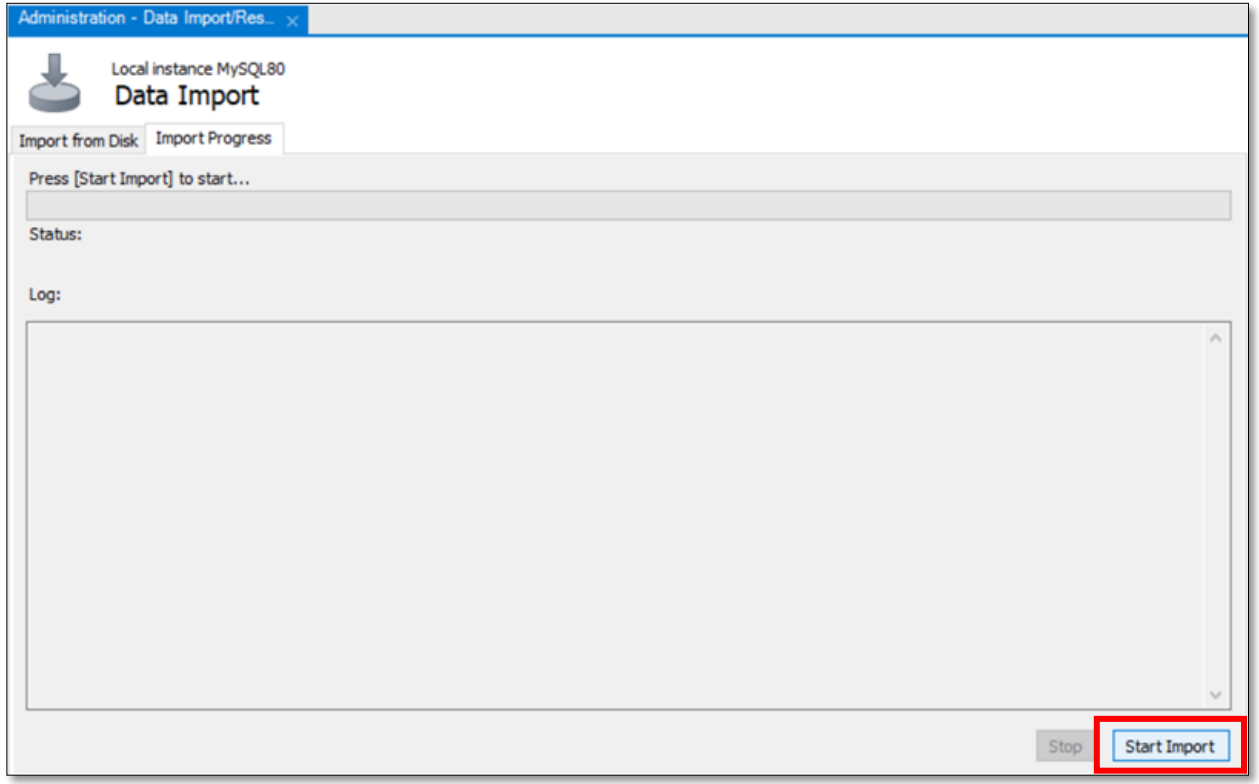
- L'utilitaire Import data dans Workbench permet aussi de réaliser cette tâche en suivant ces étapes :
    1. Ouvrez MySQL Workbench
    2. Dans la liste des MySQL Connexions, choisissez votre base de données
    3. Cliquer sur **Data Import** à partir de l'élément **Server** dans le menu de navigation



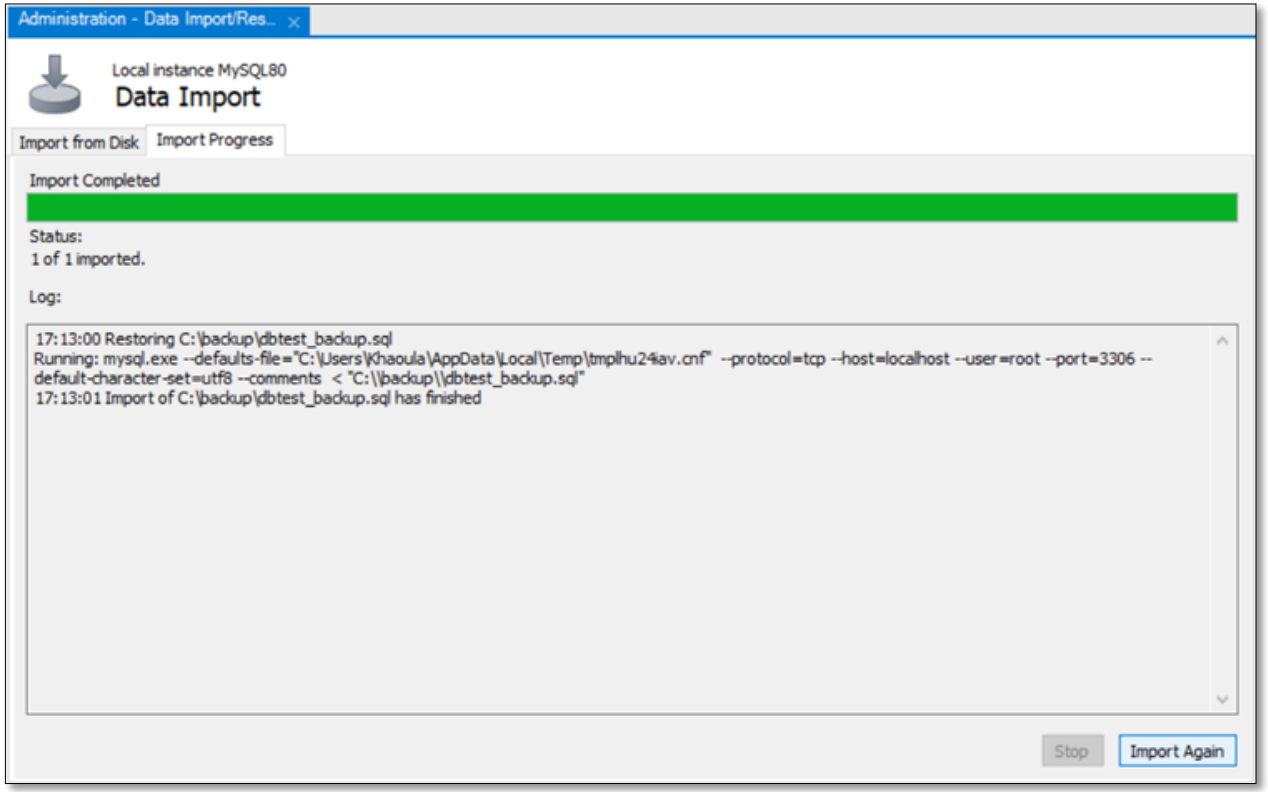
4. Dans le volet « Data Import from Disk », Section « Import Options », choisissez « Import from Self-Contained File », et sélectionner le fichier SQL qui contient les données à importer.



5. Choisissez le schéma cible par défaut (Default Target Schema) ou les données seront importées. Vous pouvez également créer une nouvelle base de données en choisissant New (Nouveau)
6. Passez sur le volet : Import Progress. Choisissez Start Import (Démarrer l'importation) pour lancer l'import



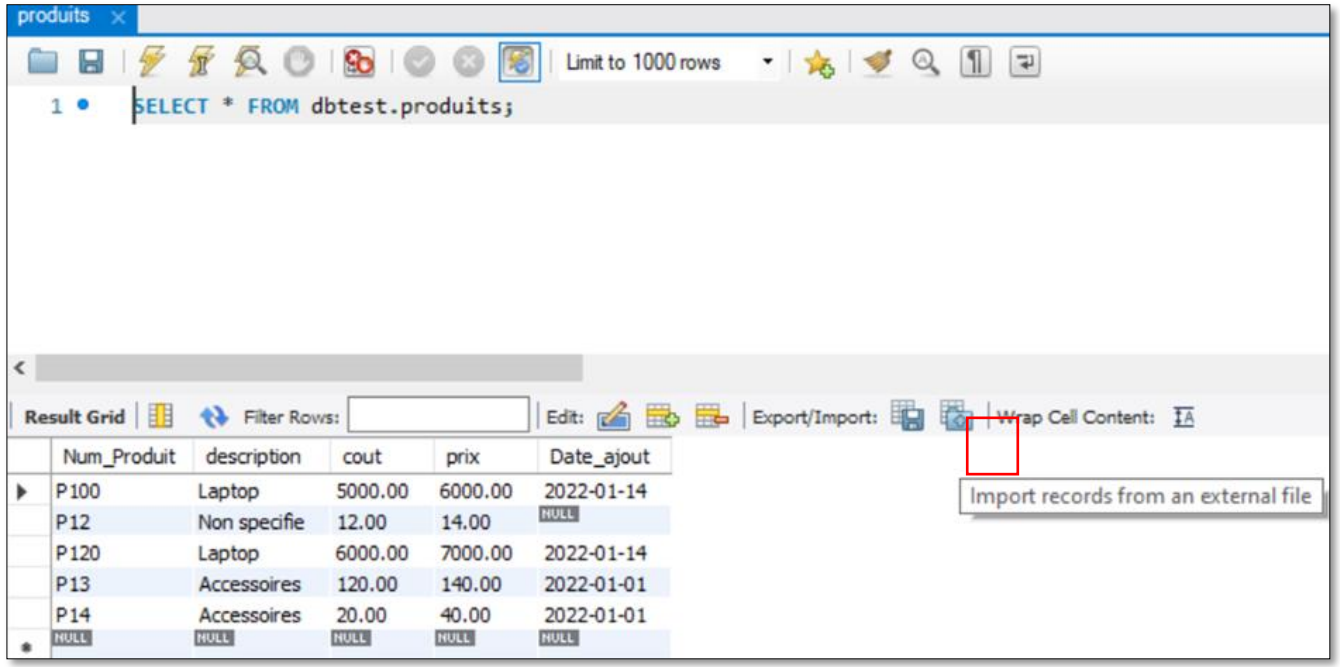
7. Votre importation peut prendre quelques minutes ou plus en fonction de la taille du fichier .SQL. Une fois l'importation terminée, vous devez voir un message semblable à ce qui suit :



PARTIE 3

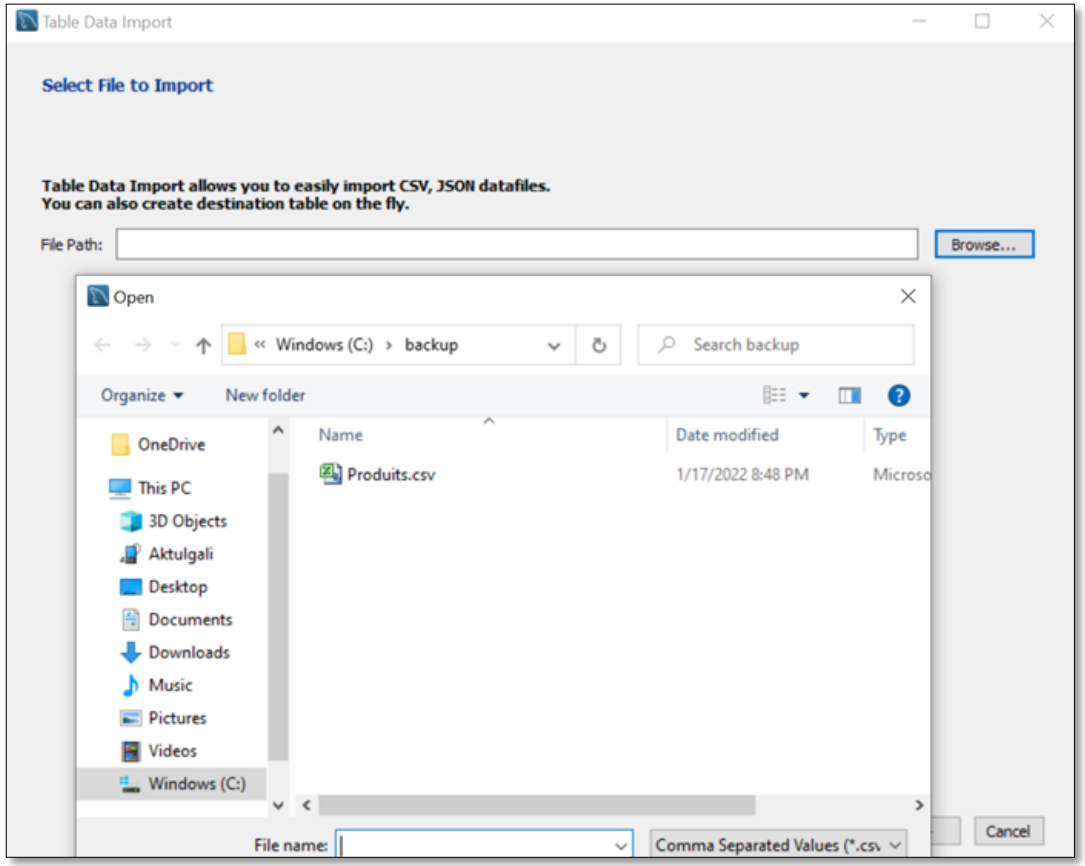
Importer des données vers une table :

- MySQL Workbench fournit un outil pour importer des données dans une table. Il permet de modifier les données avant de les charger.
- Voici les étapes à suivre pour importer des données dans une table :
    1. Ouvrez la table dans laquelle vous voulez importer des données
    2. Cliquer sur l'icône « Import records from an external file »



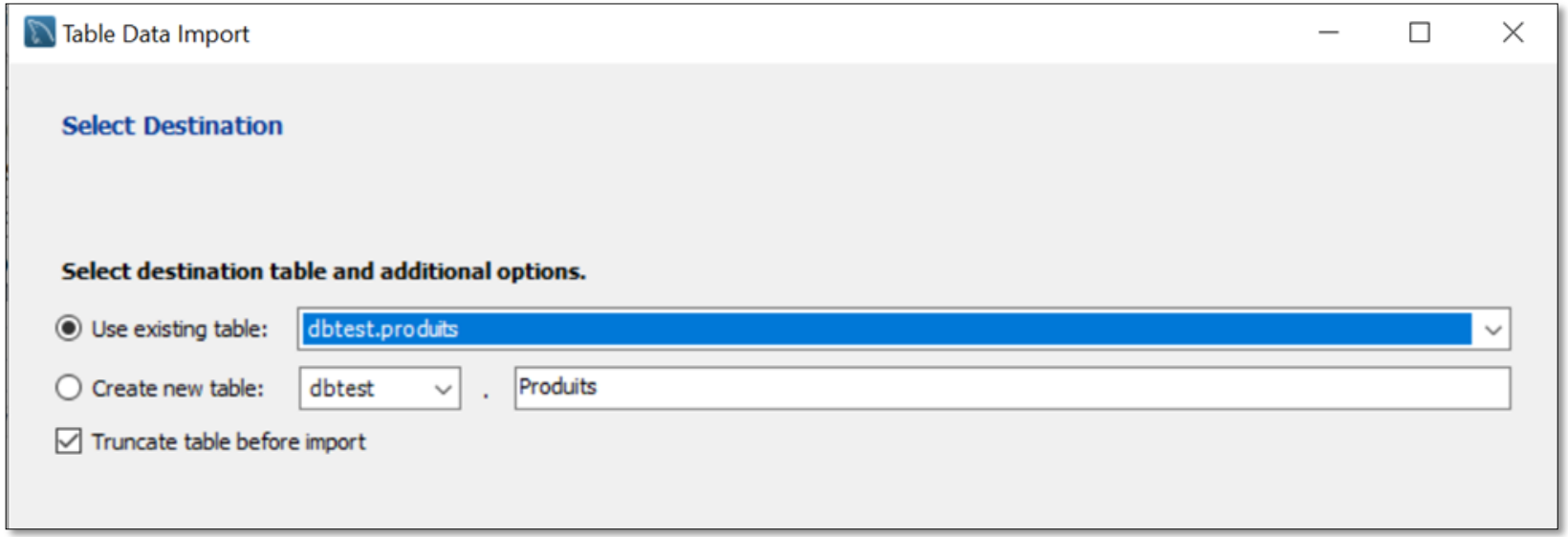
Importer des données vers une table :

3. Naviguer vers le fichier qui contient les données.



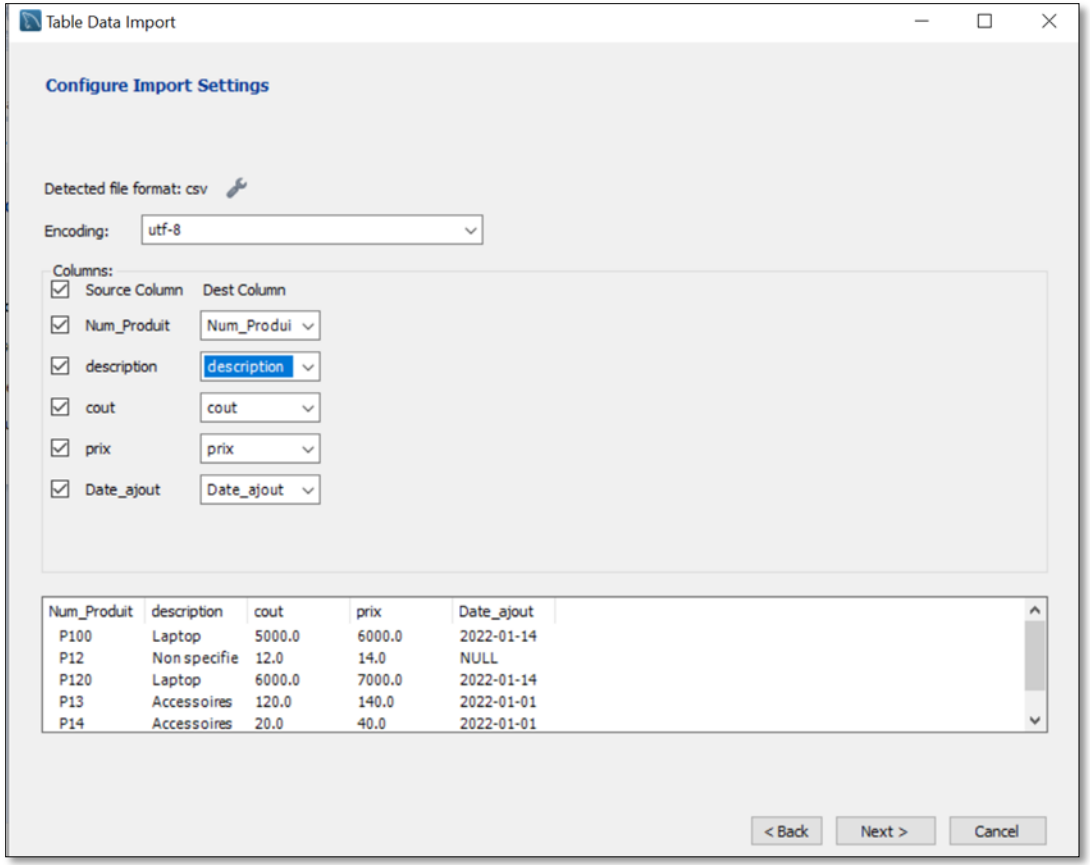
Importer des données vers une table :

4. Vous pouvez choisir d'importer les données vers une table qui existe déjà, ou en créer une nouvelle.



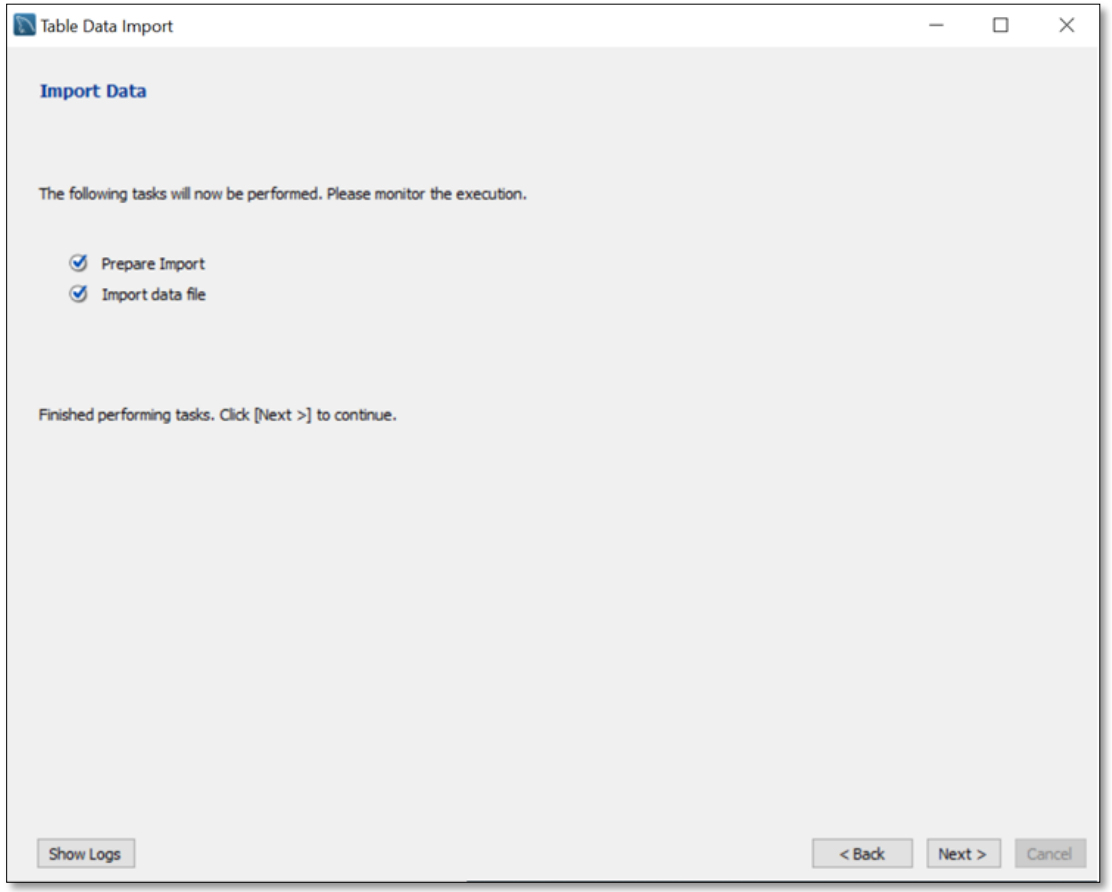
Importer des données vers une table :

5. Ensuite vous pouvez vérifier que le type du ficher choisis a été bien détecté et réaliser le mapping des colonnes du fichier avec celle de la table.



Importer des données vers une table :

6. Cliquer sur Next pour réaliser le Import.



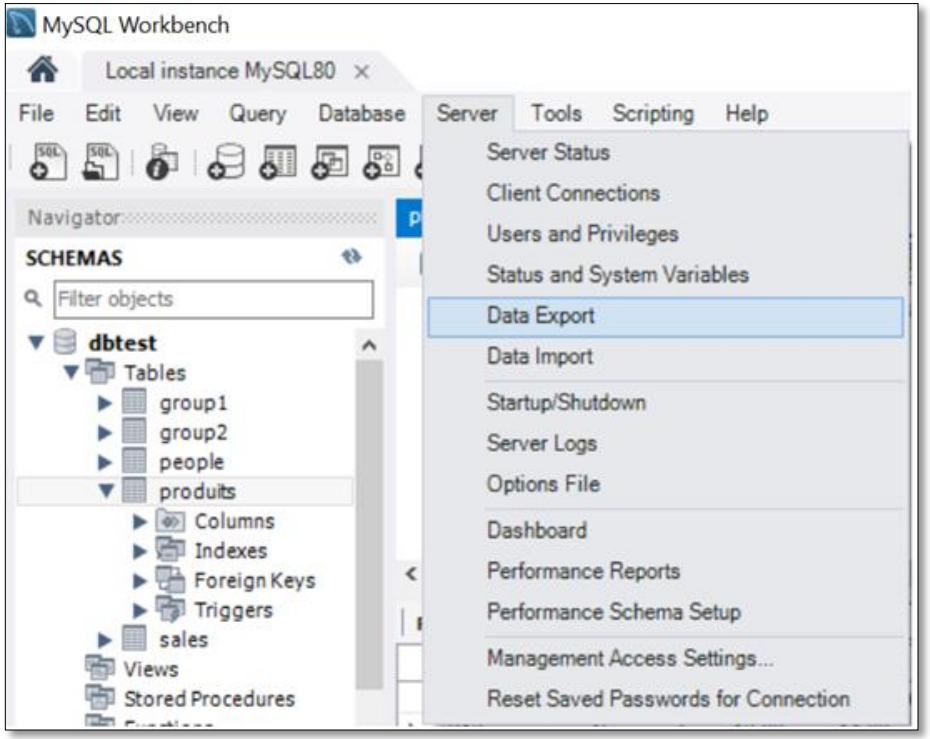
CHAPITRE 3

Administrer une base de données

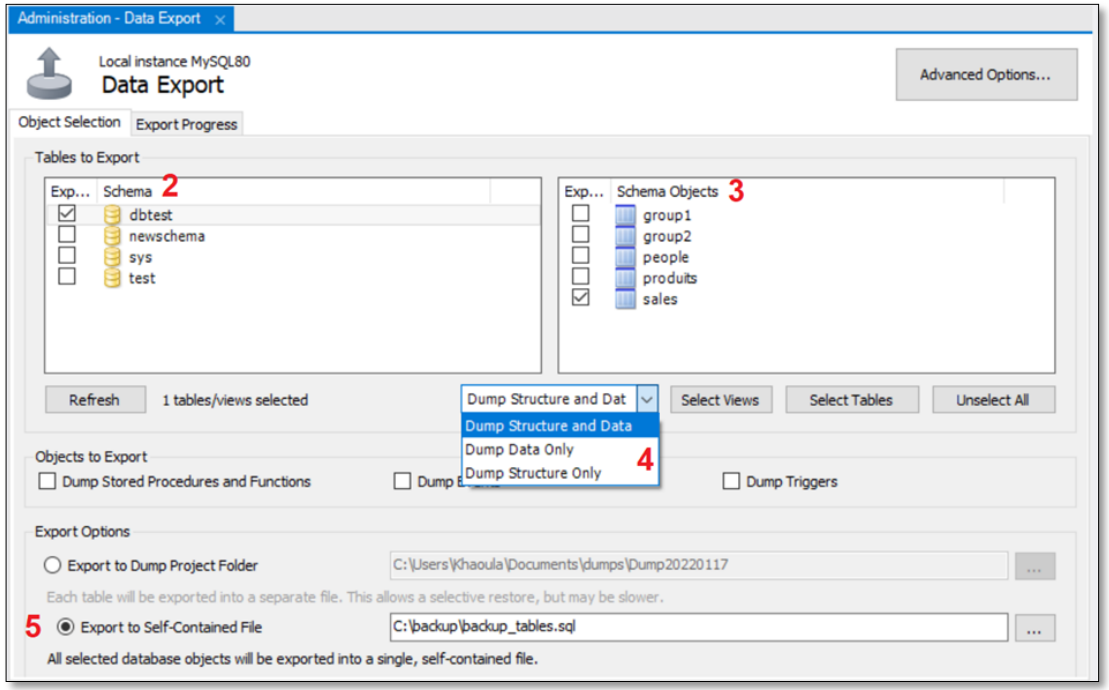
1. Backup/Restore
2. Importation
3. **Exportation**
4. Commandes de création des comptes utilisateurs
5. Commandes de gestion des privilèges de base

- L'utilitaire Export data dans Workbench permet d'exporter les données d'une base de données en suivant ces étapes :

1. Ouvrez MySQL Workbench. Dans la liste des MySQL Connexions, choisissez votre base de données, puis naviguer dans le menu Server et cliquer sur « Export Data »



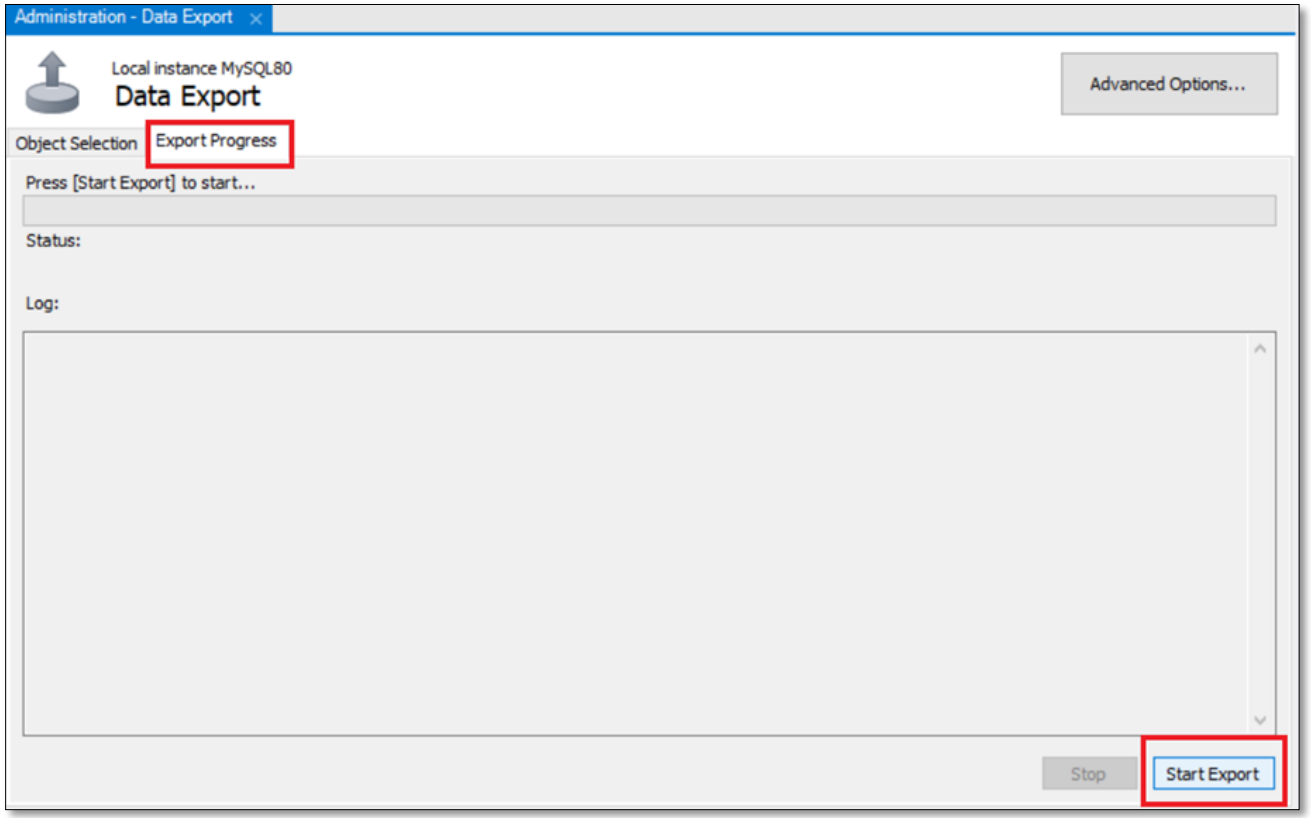
- Sur le volet « Objet Sélection »
    2. Choisissez le Schéma qui contient les données à exporter
    3. Choisissez les objets à exporter (Tables, Vues..)
    4. La liste déroulante permet de préciser s'il s'agit d'exporter la structure des objets sélectionnés, les données qu'ils contiennent ou les deux. On peut aussi choisir d'exporter d'autres objets comme les procédures stockées, les Triggers..
    5. Sélectionnez le fichier cible qui va contenir le script des données exportées



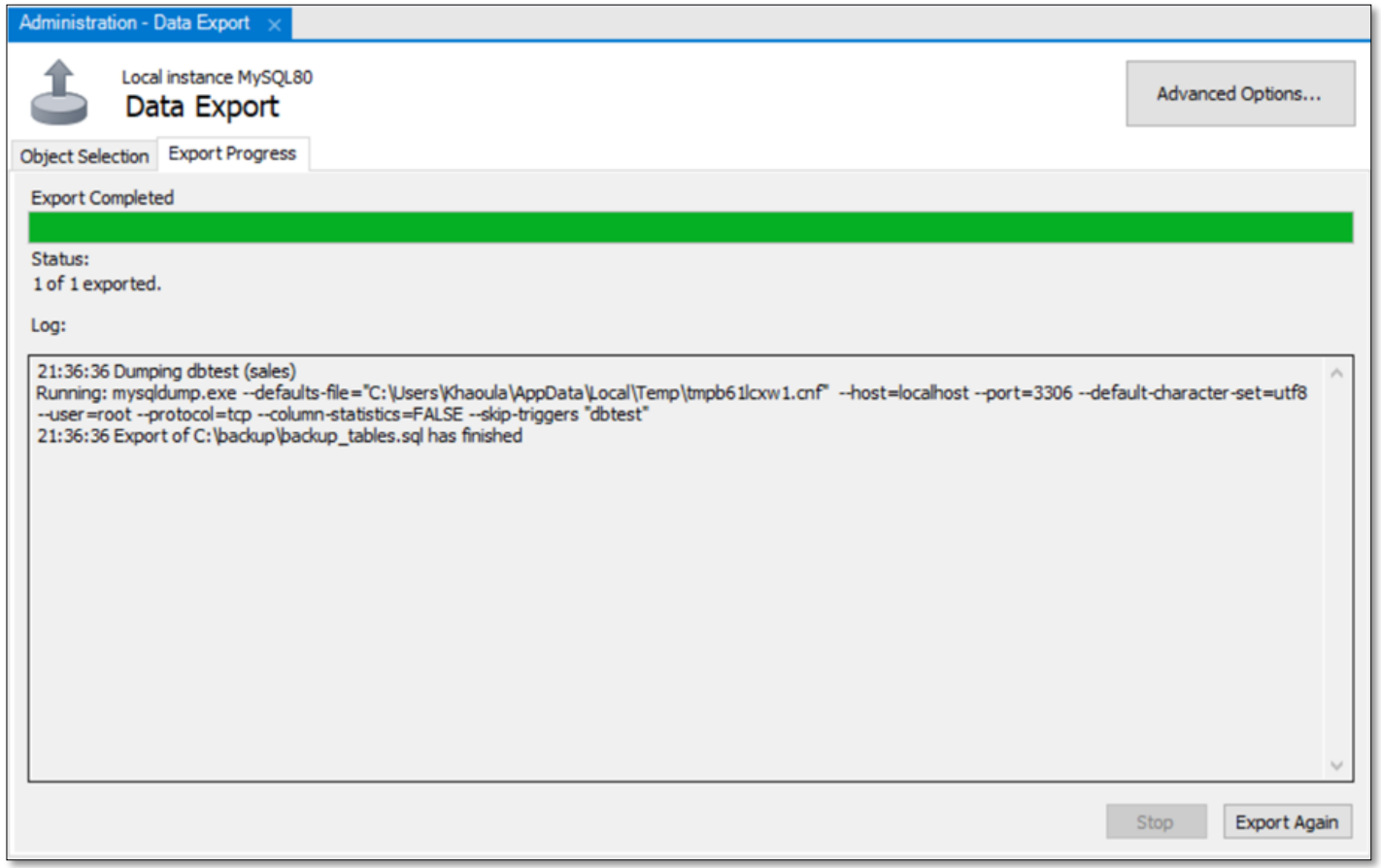
PARTIE 3

- Sur le volet « Export Progress »

6. Cliquez sur « Start Export » pour commencer le process



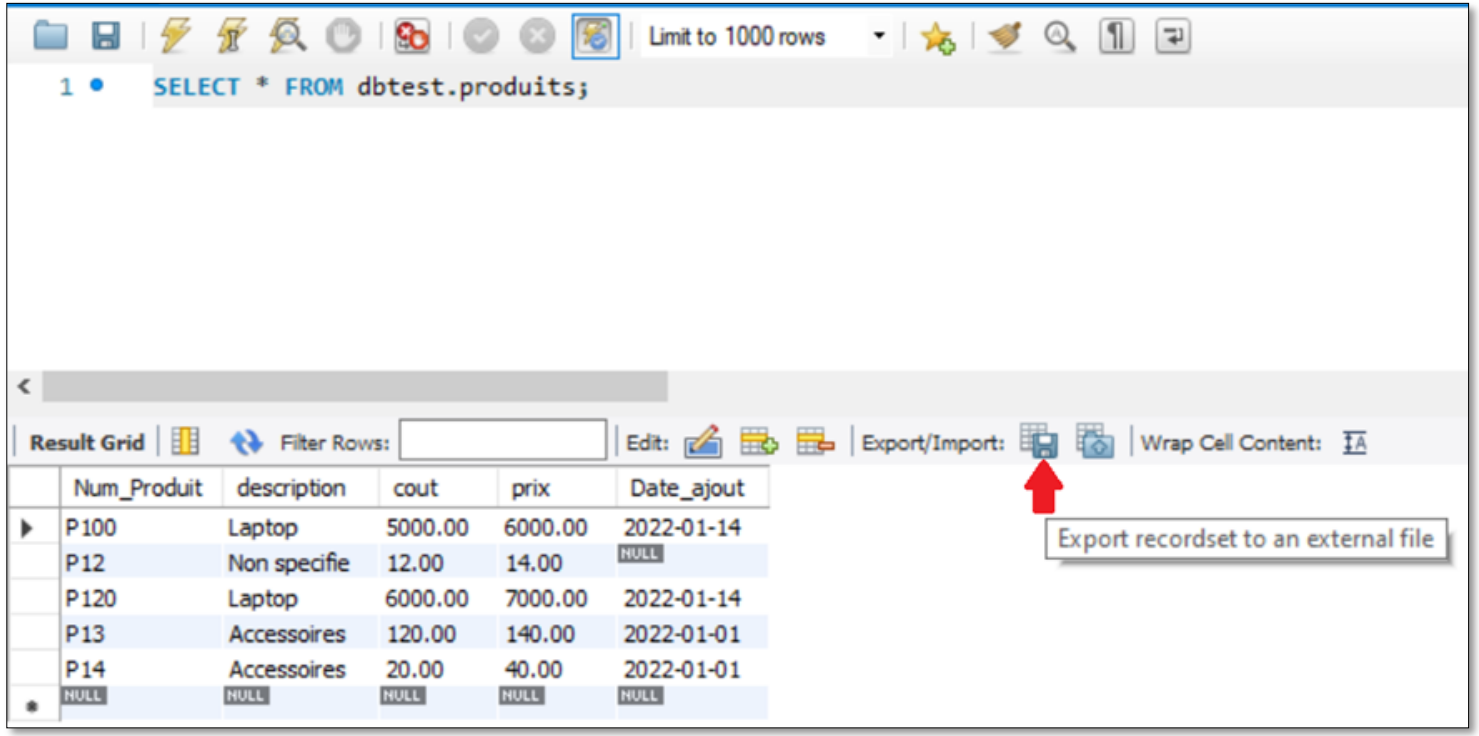
7. Le système confirme l'export :



PARTIE 3

Exporter les données d'une table :

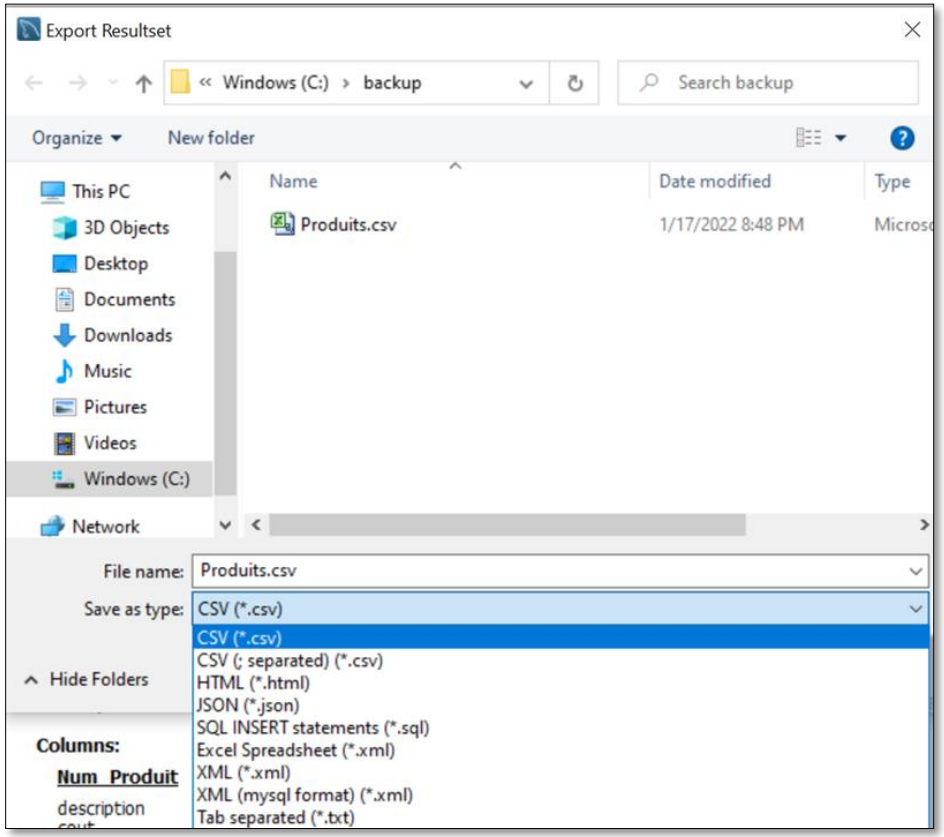
- MySQL Workbench fournit un outil pour exporter les données d'une table. Voici les étapes à suivre :
    1. Ouvrez la table de laquelle vous voulez exporter des données
    2. Cliquer sur l'icône « Export recordset to an external file »



Num_Produit	description	cout	prix	Date_ajout
P100	Laptop	5000.00	6000.00	2022-01-14
P12	Non specifie	12.00	14.00	NULL
P120	Laptop	6000.00	7000.00	2022-01-14
P13	Accessoires	120.00	140.00	2022-01-01
P14	Accessoires	20.00	40.00	2022-01-01
NULL	NULL	NULL	NULL	NULL

Exporter les données d'une table :

3. Choisissez le type et l'emplacement du fichier data à créer, et cliquer sur Ok



PARTIE 3

CHAPITRE 3

Administrer une base de données

1. Backup/Restore
2. Importation
3. Exportation
4. **Commandes de création des comptes utilisateurs**
5. Commandes de gestion des privilèges de base

Commandes de création des comptes utilisateurs

- On utilise la commande **CREATE USER** pour créer de nouveaux utilisateurs dans le serveur de base de données MySql.
- Voici la syntaxe de base de **CREATE USER** :

```
CREATE USER [IF NOT EXISTS] nom_compte
IDENTIFIED BY 'mot_de_passe';
```

- Dans cette expression il faut spécifier :
  - **nom_compte :** Il s'agit du nom du compte a créer et se compose en général de deux partie sous cette forme : nom_utilisateur@nom_host
  - **« nom_utilisateur »** est le nom de l'utilisateur. Et **« nom_host »** est le nom de l'hôte à partir duquel l'utilisateur se connecte au serveur MySQL. La partie nom d'hôte du nom de compte est optionnelle. Si elle est omise, l'utilisateur peut se connecter depuis n'importe quel hôte.
  - **Mot_de_passe :** Le mot de passe relatif au nouveau compte.
- **CREATE USER** crée un nouvel utilisateur sans aucun privilège.

Exemple :

- Sur la ligne de commande MySQL, on liste les utilisateurs existants : **mysql> select user from mysql.user**

Résultat :

```
mysql> select user from mysql.user;
+------------------+
| user             |
+------------------+
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| root             |
+------------------+
4 rows in set (0.20 sec)
```

mysql> create user Ahmad@localhost identified by 'Monmot2p@ss'

- On peut vérifier sur la table mysql.user la création du nouveau compte :

```
mysql> select user from mysql.user;
+------------------+
| user             |
+------------------+
| Ahmad            |
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| root             |
+------------------+
5 rows in set (0.00 sec)
```

Exemple :

- Afin de tester le login, on ouvre une autre session MySql avec le compte « Ahmad » , en utilisant la commande suivante : **Mysql – u Ahmad –p**
- Puis on saisit le mot de passe : **Monmot2p@ss**

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u Ahmad -p
Enter password: ************
```

- On peut vérifier les bases de données auxquelles Ahmad peut accéder :

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
+--------------------+
1 row in set (0.01 sec)
```

CHAPITRE 3

Administrer une base de données

1. Backup/Restore
2. Importation
3. Exportation
4. Commandes de création des comptes utilisateurs
5. **Commandes de gestion des privilèges de base**

Attribution des privilèges (GRANT)

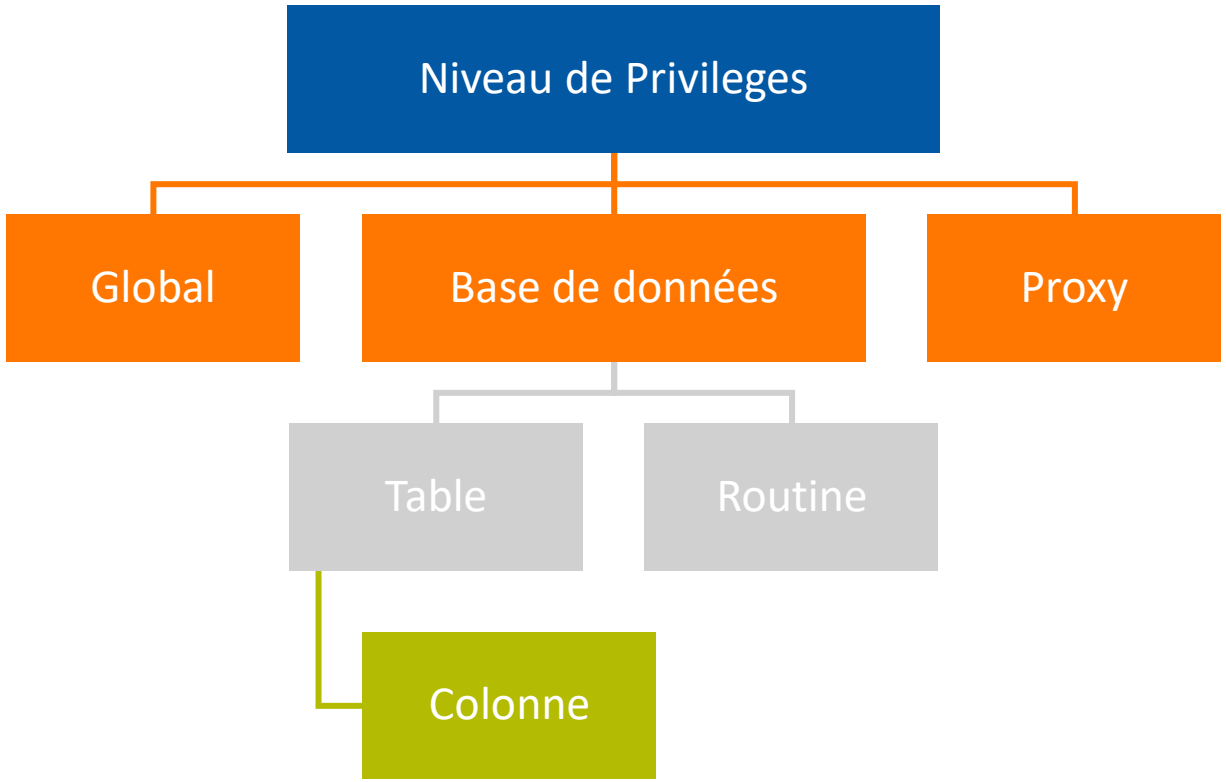
- La commande CREATE USER crée un ou plusieurs comptes d'utilisateurs sans privilèges. Pour qu'un utilisateur puisse accéder aux objets de base de données, il faut d'abord lui accorder des privilèges . Ceci se fait à l'aide de la commande GRANT.
- Voici la syntaxe générique de GRANT :

```
GRANT privilege [,privilege],..

ON privilege_level

TO nom_utilisateur;
```

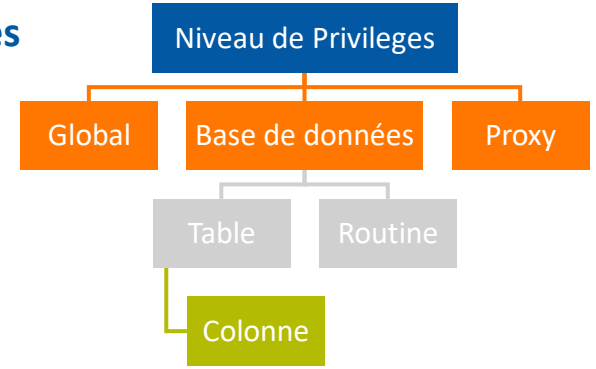
- Pour donner des privilèges au compte « nom_utilisateur », il faut spécifier le ou les privilèges à donner (SELECT, DELETE, UPDATE, ALL…) et sur quel niveau les appliquer.

Attribution des privilèges (GRANT) : Les niveaux de privilèges

Il existe différents niveaux de privilèges dans MySql :



Attribution des privilèges (GRANT) : Les niveaux de privilèges

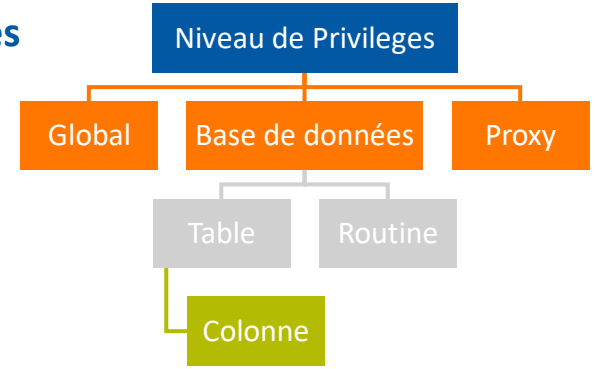


1. Niveau global :
    - Les privilèges globaux s'appliquent à toutes les bases de données d'un serveur MySQL. Pour attribuer des privilèges globaux, vous utilisez la syntaxe *.*.

    Exemple :

```
GRANT UPDATE
ON *.*
TO Ahmad@localhost;
```

Attribution des privilèges (GRANT) : Les niveaux de privilèges

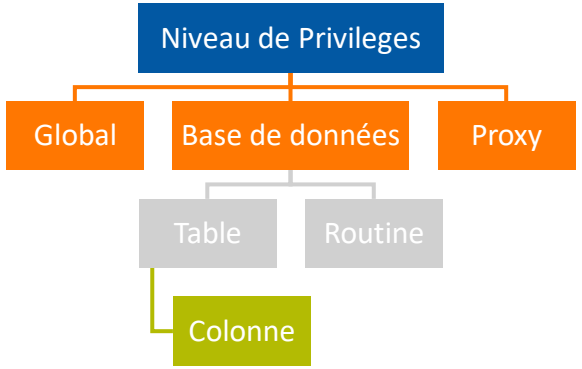


2. Niveau Base de données :
    - Ces privilèges s'appliquent à tous les objets d'une base de données. Pour attribuer des privilèges au niveau de la base de données, on utilise la syntaxe : ON nom_base_de_données.*

Exemple :

```
GRANT INSERT
ON dbtest.*
TO Ahmad@localhost;
```

Attribution des privilèges (GRANT): Les niveaux de privilèges



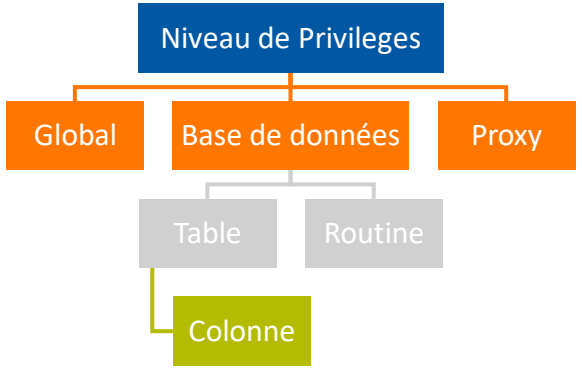
3. Niveau Table :

   - Les privilèges de table s'appliquent à toutes les colonnes d'une table. Pour attribuer des privilèges au niveau de la table, on utilise la syntaxe ON nom_base_de_données.nom_table.

   Exemple :

```
GRANT INSERT,DELETE
ON dbtest.Produits
TO Ahmad@localhost;
```

Attribution des privilèges (GRANT): Les niveaux de privilèges

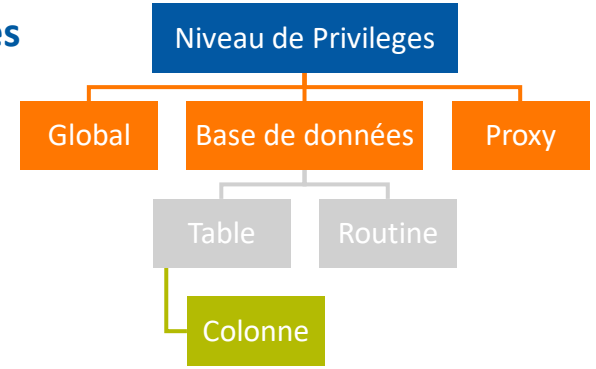


4. Niveau Colonne :
    - Les privilèges de colonne s'appliquent à des colonnes uniques dans une table. Vous devez spécifier la ou les colonnes pour chaque privilège.

Exemple :

```
GRANT
    SELECT (Num_Produit,Description, Date_ajout),
    UPDATE(Prix)
ON dbtest.Produits
TO Ahmad@localhost;
```

Attribution des privilèges (GRANT) : Les niveaux de privilèges

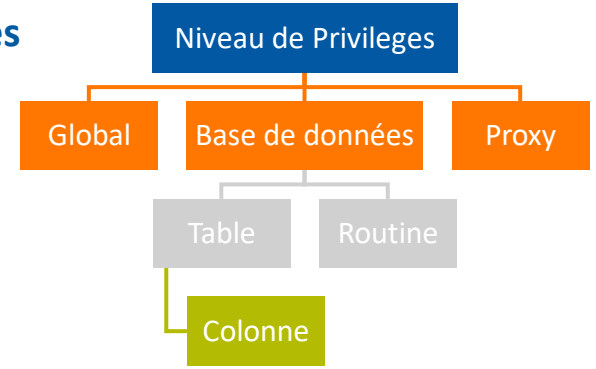


5. Niveau Routine :
    - Les privilèges de routine s'appliquent aux procédures et fonctions stockées.

    Exemple :

```
GRANT EXECUTE
ON PROCEDURE CalculPrix
TO Ahmad@localhost;
```

Attribution des privilèges (GRANT) : Les niveaux de privilèges



6. Niveau Proxy :

   - Les privilèges d'utilisateur proxy permettent à un utilisateur externe d'être un proxy pour un autre, c'est-à-dire, d'avoir les privilèges du deuxième utilisateur. En d'autres termes, l'utilisateur externe est un "utilisateur proxy" (un utilisateur qui peut usurper l'identité ou devenir un autre utilisateur) et le deuxième utilisateur est un "utilisateur mandaté" (un utilisateur dont l'identité peut être reprise par un utilisateur proxy).

   Exemple :

```
GRANT PROXY

ON root

TO Ahmad@localhost;
```

Attribution des privilèges (GRANT): Les niveaux de privilèges

Le tableau suivant illustre les privilèges autorisés les plus utilisés pour les instructions GRANT et REVOKE La liste exhaustives des privilèges MySql est consultable sur le lien suivant : https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html

Privilège	Description	Niveau					
		Global	B. Données	Table	Colonne	Routine	Proxy
LL [PRIVILEGES]	Accorde tous les privilèges au niveau spécifié sauf GRANT OPTION						
ALTER	Autorise l'utilisation de ALTER TABLE	X	X	X			
ALTER ROUTINE	Autorise Alter et Drop des procédures et fonctions stockées	X	X			X	
CREATE	Autorise la création des bases de données et tables	X	X	X			
CREATE ROUTINE	Autorise la création des procédures et fonctions stockées	X	X				
CREATE TEMPORARY TABLES	Autorise la création des tables temporaires: CREATE TEMPORARY TABLE	X	X				
CREATE USER	Autorise l'utilisation de: `CREATE USER, DROP USER, RENAME USER et REVOKE ALL PRIVILEGES`	X					
CREATE VIEW	Autorise la création et modification des vues.	X	X	X			
DELETE	Autorise l'utilisation de DELETE.	X	X	X			
DROP	Autorise la suppression des bases de données et des tables.	X	X	X			
EXECUTE	Autorise l'exécution des routines.	X	X	X			
GRANT OPTION	Autorise l'utilisateur a accorder ou révoquer des privilèges d'autres comptes	X	X	X		X	X
INDEX	Autorise la création et suppression des indexes	X	X	X			
INSERT	Autorise l'utilisation de `INSERT`	X	X	X	X		
PROCESS	Autorise l'exécution de `SHOW PROCESSLIST.`	X					
PROXY	Autorise l'utilisation de PROXY						
REFERENCES	Autorise la création des clés étrangères	X	X	X	X		
SELECT	Autorise l'utilisation SELECT	X	X	X	X		
SHUTDOWN	Autorise l'utilisation de la commande mysqladmin shutdown	X					
UPDATE	Autorise l'utilisation de `UPDATE`	X	X	X	X		

Révocation des privilèges (REVOKE)

- Afin de supprimer un ou plusieurs privilèges donnés à des utilisateurs, on utilise la commande REVOKE suivant cette syntaxe :

```
REVOKE
   Privilege1 [,privilege2]..
ON [type_objet] privilege_level
FROM utilisateur1 [, utilisateur2] ..;
```

- Il faut spécifier :
  - Une liste de privilèges séparés par des virgules qu'on veut révoquer d'un compte d'utilisateur après le mot-clé REVOKE ;
  - Le type d'objet et le niveau de privilège après le mot-clé ON ;
  - Un ou plusieurs comptes d'utilisateur dont vous souhaitez révoquer les privilèges dans la clause FROM.

Révocation des privilèges (REVOKE)

Exemples :

```
REVOKE
ALL , GRANT OPTION
FROM Ahmad@localhost;
```

```
REVOKE SELECT,UPDATE,DELETE
ON dbtest.*
FROM Ahmad@localhost, str@localhost;
```

- L'instruction REVOKE prend effet selon le niveau de privilège :
  - **Niveau global :** Les modifications prennent effet lorsque le compte utilisateur se connecte au serveur MySQL dans les sessions suivantes. Les modifications ne sont pas appliquées à tous les utilisateurs actuellement connectés.
  - **Niveau base de données :** Les modifications prennent effet après la prochaine instruction USE.
  - **Niveaux table et colonne :** Les modifications prennent effet sur toutes les requêtes suivantes.