

M106 : Manipuler des bases de données

Mme JANJARY JIHAD

PLAN

01 Concevoir une base de données

- Analyser le cahier de charges
- Modéliser les données
- Normaliser les données

02 Préparer l'environnement

- Exploiter un outil de modélisation
- Préparer le serveur MySQL

03 Manipuler les données

- Créer une base de données
- Réaliser des requêtes SQL
- Administrer une base de données



PARTIE 3: Manipuler les 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

Création des Bases De Données

Rappel: Définition d'une base de données

Une 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**

Création des 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

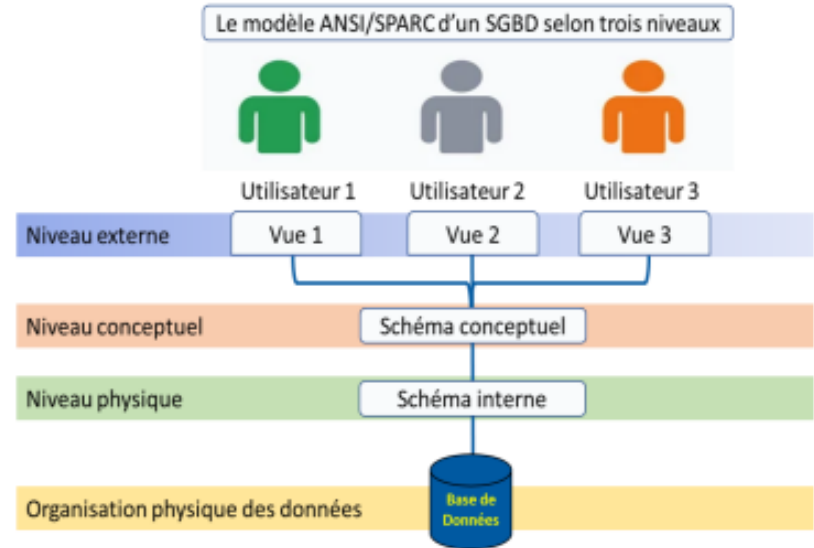
Création des Bases De Données

SGBD (Système de Gestion de Bases de Données)

Le **SGBD** est le logiciel qui va vous permettre de manipuler les données d'une base. C'est ce logiciel qui commande les interactions avec votre base pour y **recupérer**, **ajouter**, **modifier** ou **supprimer** des données.

Pour échanger avec votre base, vous allez donc donner des ordres à votre **SGBD**. Des ordres en français ? Pas exactement ! Ces ordres, vous allez les lui donner dans un langage très simple : le langage **SQL**. En termes d'architecture, et selon le modèle **ANSI/SPARC** d'un SGBD comporte 3 niveaux :

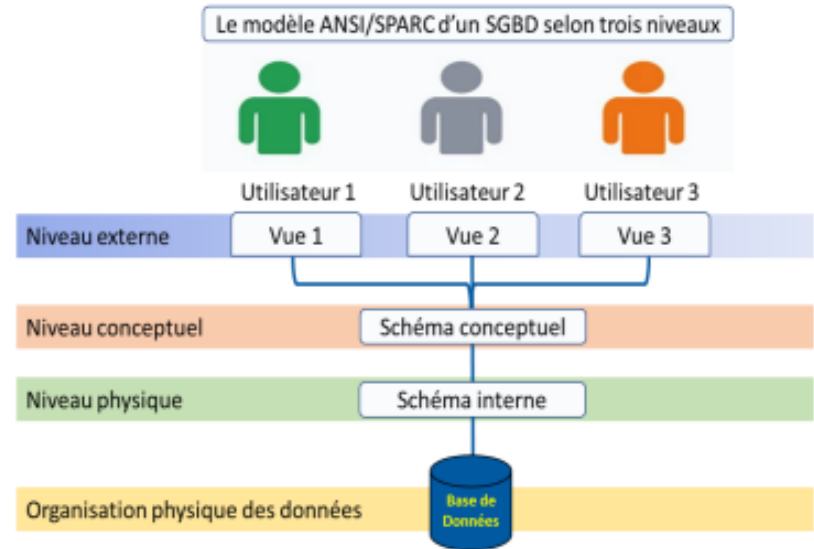
- **Niveau conceptuel** : Le niveau central d'un SGBD. Il correspond à une vue générale de toutes les données existant dans l'entreprise.



Création des Bases De Données

SGBD (Système de Gestion de Bases de Données)

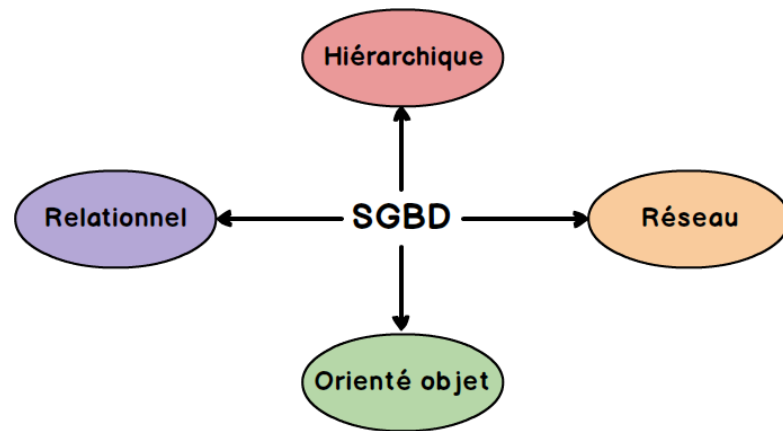
- **Niveau interne ou physique :**
 - Spécification du stockage physique des données (fichiers, disques, etc.) et des méthodes d'accès (index, etc.).
- **Niveau externe :**
 - Il s'agit d'une **vue externe** pour chaque groupe d'utilisateurs sur un sous ensemble de la base ;
 - Chaque **schéma externe**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



Création des Bases De Données

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.



Création des Bases De Données

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.
- **SQLite** SQLite stocke toute la base de données dans un seul et unique fichier. c'est un SGBD très simple à configurer et qui “ne prend pas la tête”. C'est d'ailleurs le SGBD qu'on utilise quand on développe une base de données “**en local**” (sur son ordinateur), et aussi le SGBD utilisé par vos applications Android
- **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.

Création des Bases De Données

	MySQL	Oracle Database	PostgreSQL	SQLite
Popularité	✔ très répandu	✔ utilisé par les grands groupes ayant un grand nombre de données	✗ moins répandu que les autres	✔ répandu pour le prototypage
Prix	✔ gratuit (licence fermée)	✗ très cher (licence fermée)	✔ gratuit (open-source)	✔ gratuit (open-source)
Similarité avec le langage SQL	✗ ne suit pas toujours la syntaxe SQL	✗ ne suit pas toujours la syntaxe SQL	✔ suit la syntaxe SQL de très près	✔ suit bien la syntaxe SQL
Entreprises utilisatrices	📘 utilisé par Facebook	📘 utilisé par Samsung	📘 utilisé par Spotify	📘 utilisé pour les apps Android

Création des Bases De Données

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..

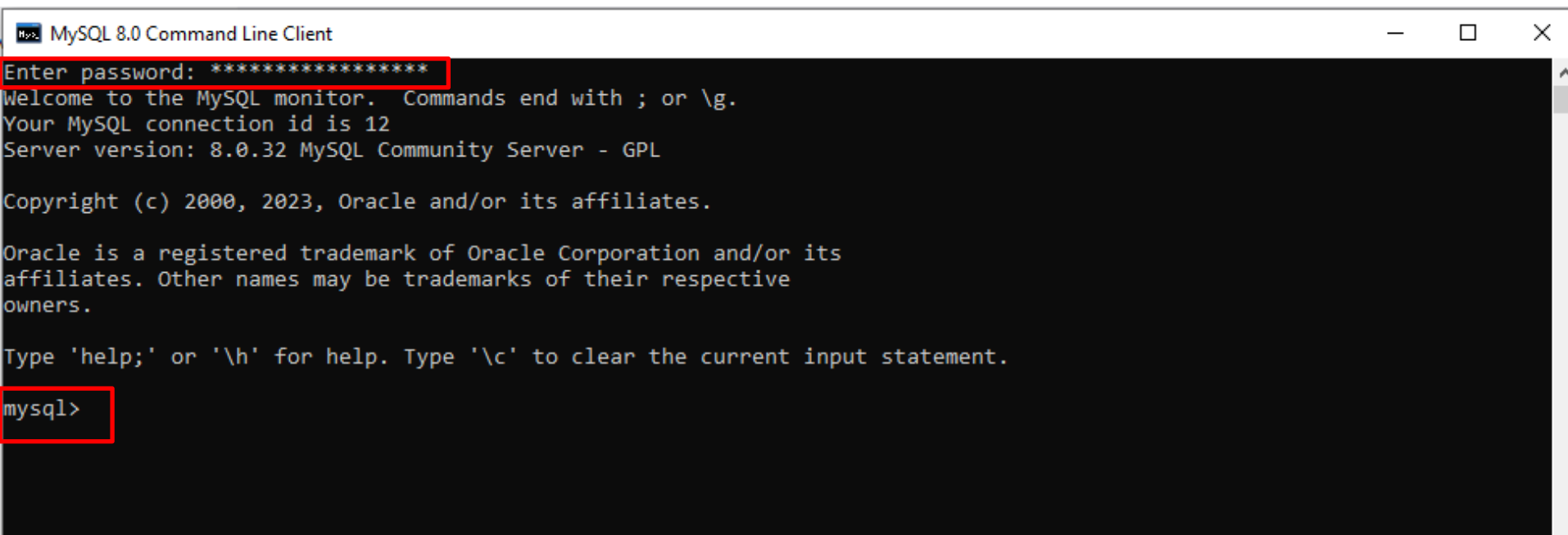
Création des Bases De Données

Le langage de requêtes SQL (Structured Query Language) :

- 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 des Bases De Données

Création d'une base de données



The screenshot shows a terminal window titled "MySQL 8.0 Command Line Client". The text inside the window is as follows:

```
Enter password: *****  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 12  
Server version: 8.0.32 MySQL Community Server - GPL  
  
Copyright (c) 2000, 2023, Oracle and/or its affiliates.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql>
```

Red boxes highlight the password input field and the `mysql>` prompt.

Création des Bases 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';
```

Création des Bases De Données

Création d'une base de données

- **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**

Création des Bases De Données

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.

Création des Bases De Données



Sur MySQL command line Client :

Nous allons créer ensemble une base de données (BDD) pour une application imaginaire, **Foodly**. Cette application permet à des internautes de sélectionner les aliments qu'ils souhaitent acheter pour voir leur composition en protéines, graisses, sucres, etc.

La **BDD** de **Foodly** va donc stocker les données dont l'application a besoin pour fonctionner, à savoir :

- Les utilisateurs inscrits ;
- Les aliments disponibles.

```
mysql> CREATE DATABASE foodly;  
Query OK, 1 row affected (0.01 sec)  
  
mysql>
```

Création des Bases De Données

Exemple : Création d'une base de données sur MySQL

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

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

mysql>
```

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

```
mysql> USE foodly;
Database changed
mysql>
```

- 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

Choix du moteur

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.

Choix du moteur

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
ndbinfo	NO	MySQL Cluster system information storage engine	NULL	NULL	NULL
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
ndbcluster	NO	Clustered, fault-tolerant tables	NULL	NULL	NULL

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

```
mysql>
```

Choix du moteur

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.	<ul style="list-style-type: none">• Très rapide en lecture• Extrêmement rapide pour les opérations COUNT() sur une table entière• Les index FULLTEXT pour la recherche sur des textes	<ul style="list-style-type: none">• 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.	<ul style="list-style-type: none">• Gestion des relations• Gestion des transactions• Verrouillage à la ligne et non à la table	<ul style="list-style-type: none">• 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.	<ul style="list-style-type: none">• Le moteur le plus rapide• Ne consomme pas de place sur le disque dur	<ul style="list-style-type: none">• Les données sont volatiles : un arrêt du serveur et elles disparaissent• Pas de champs BLOB ou TEXT

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

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

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

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

Créer une Base de Données

Définition des colonnes

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

Créer une Base de Données

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

Créer une Base de Données

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
LOBLOB	0-4294967295	Grands données de texte sous forme binaire
LONGTEXT	0-4294967295	Grande données de texte

Créer une Base de Données

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

Contraintes d'intégrité

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 :

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

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 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 :

1. 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**.
2. Une colonne de clé primaire *ne peut pas avoir de valeurs* **NULL**.
3. 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**.

Les contraintes d'intégrité MySQL

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,  
    ...  
);
```

Les contraintes d'intégrité MySQL

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)  
);
```

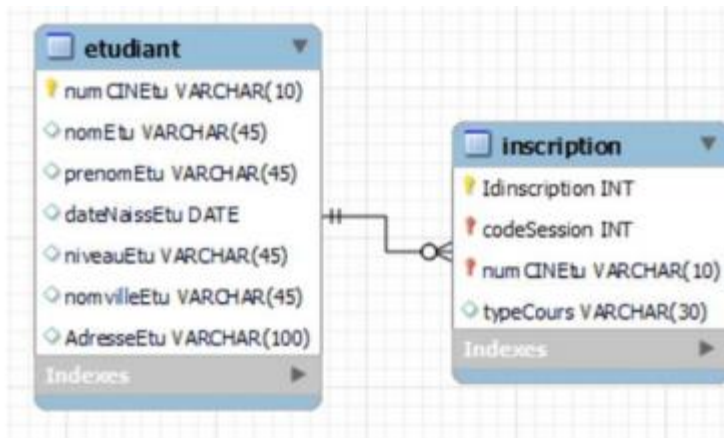
On sépare les colonnes de liste-colonnes par des virgules (,).

Les contraintes d'intégrité MySQL

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 entre les tables étudiant et inscription de notre base de données CentreFormation.



Les contraintes d'intégrité MySQL

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 (référéncée) qui renvoie à la colonne identifiante de la table étudiant: **numCINETu**.
- ✓ La table **étudiant** 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)
```


Les contraintes d'intégrité MySQL

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]
```

Les contraintes d'intégrité MySQL

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;
```

Les contraintes d'intégrité MySQL

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

5- 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

5– UNIQUE (suite)

- 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,...)  
);
```

Les contraintes d'intégrité MySQL

5– UNIQUE (suite)

- 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)
```

Les contraintes d'intégrité MySQL

6– CHECK

- 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]
```

Les contraintes d'intégrité MySQL

6– 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)  
);
```


Les contraintes d'intégrité MySQL

6– 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(prix >= cout)  
);
```

EXEMPLE 3

Créez vos premières tables avec CREATE TABLE (TP)

Pour votre base **Foodly**, vous avez deux tables à créer :

- ✓ Une pour les objets de type utilisateur ;
- ✓ Une autre pour les objets de type aliment.

id	nom	prenom	email
1	FOO	John	sf@gmail.com
2	BAR	Mike	m_bar_123@gmail.com
3	TREE	Sarah	tree.sarah@gmail.com

EXEMPLE 3

Créez vos premières tables avec CREATE TABLE (TP)

```
CREATE TABLE utilisateur (  
  id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  nom VARCHAR(100),  
  prenom VARCHAR(100),  
  email VARCHAR(255) NOT NULL UNIQUE  
);
```

```
Query OK, 0 rows affected (0.03 sec)
```

EXEMPLE 3

Créez vos premières tables avec CREATE TABLE (TP)

Pour votre base **Foodly**, vous avez deux tables à créer :

- ✓ Une pour les objets de type utilisateur ;
- ✓ Une autre pour les objets de type aliment.

id	nom	marque	calories	sucre	graisses	proteines	bio
1	Pomme	Monoprix	65	14,4	0,4	0,4	FALSE
2	Oeuf bio	Carrefour	167	0	11,1	14,2	TRUE
3	Brique de lait	Intermarché	414	43,2	13,5	28,8	FALSE

EXEMPLE 3

Créez vos premières tables avec **CREATE TABLE (TP)**

```
CREATE TABLE aliment (  
  id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  nom VARCHAR(100) NOT NULL,  
  marque VARCHAR(100),  
  sucre FLOAT,  
  calories INTEGER NOT NULL,  
  graisses FLOAT,  
  proteines FLOAT,  
  bio BOOLEAN DEFAULT false  
);
```

```
Query OK, 0 rows affected (0.01 sec)
```

Créer une Base de Données

Vérifiez l'intégrité de votre table avec **SHOW tables** et **SHOW columns**

Pour vérifier que tout ce que vous avez fait fonctionne, On demande à MySQL de nous afficher toutes les tables présentes dans notre base grâce à la commande **SHOW tables**;

```
mysql> SHOW tables;
+-----+
| Tables_in_foodly |
+-----+
| aliment           |
| utilisateur      |
+-----+
2 rows in set (0.01 sec)

mysql>
```

Créer une Base de Données

Vérifiez l'intégrité de votre table avec **SHOW tables** et **SHOW columns**

- On peut même aller encore plus loin en demandant à MySQL de nous afficher le **schéma** de chaque table grâce à la commande **SHOW COLUMNS FROM lenomdematable;**
- Pour lire le schéma de vos tables, il vous faut taper **SHOW COLUMNS FROM utilisateur;** et **SHOW COLUMNS FROM aliment;**

```
mysql> SHOW COLUMNS FROM utilisateur;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
nom	varchar(100)	YES		NULL	
prenom	varchar(100)	YES		NULL	
email	varchar(255)	NO	UNI	NULL	

```
4 rows in set (0.01 sec)
```

Créer une Base de Données

Vérifiez l'intégrité de votre table avec **SHOW tables** et **SHOW columns**

➤ Pour lire le schéma de vos tables, il vous faut taper **SHOW COLUMNS FROM utilisateur;** et

SHOW COLUMNS FROM aliment;

```
mysql> SHOW COLUMNS FROM aliment;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
nom	varchar(100)	NO		NULL	
marque	varchar(100)	YES		NULL	
sucre	float	YES		NULL	
calories	int	NO		NULL	
graisses	float	YES		NULL	
proteines	float	YES		NULL	
bio	tinyint(1)	YES		0	

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

```
mysql>
```


En résumé

- Pour créer une base de données, on utilise la commande **CREATE DATABASE;** .
- Pour utiliser une base en particulier on utilise la commande **USE mabasededonnee;**
- Pour connaître les bases disponibles, on utilise la commande **SHOW DATABASES;**
- Une table est un espace dans votre base de données qui va stocker des objets de même type. On la crée avec la commande **CREATE TABLE;**
- Lors de la création d'une table on spécifie le nom, le type et les options de chaque champ .
- Pour vérifier la création des tables, on utilise les commandes **SHOW tables;** et **SHOW COLUMNS FROM table;** .

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**

Manipulation d'objet table (DROP, ALTER)

Serveur SQL

C

Create
Créer

R

Read
Lire

U

Update
MAJ

D

Delete
Effacer

Manipulation d'objet table (DROP, ALTER)

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.
- 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.

Manipulation d'objet table (DROP, ALTER)

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.

Manipulation d'objet table (DROP, ALTER)

Commande **ALTER TABLE** : Ajouter une ou plusieurs colonnes à une table :

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

Manipulation d'objet table (DROP, ALTER)

Commande ALTER TABLE : Ajouter une ou plusieurs colonnes à une table :

- 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)  
);
```

Manipulation d'objet table (DROP, ALTER)

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],
...;
```


Manipulation d'objet table (DROP, ALTER)

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 );
```

Manipulation d'objet table (DROP, ALTER)

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

Exemple 2 :

- 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)  
);
```

Manipulation d'objet table (DROP, ALTER)

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)  
);
```

Manipulation d'objet table (DROP, ALTER)

Commande ALTER TABLE : Supprimer une colonne d'une table

- Afin de renommer une colonne d'une table, 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 ;
```

Manipulation d'objet table (DROP, ALTER)

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 ;
```

Manipulation d'objet table (DROP, ALTER)

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;
```

Manipulation d'objet table (DROP, ALTER)

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 (column_name,...);
```

- Supprimer une clé étrangère :

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

Manipulation d'objet table (DROP, ALTER)

Commande **ALTER TABLE** : Ajouter et supprimer une contrainte de table

- **UNIQUE:**

- Ajouter la contrainte **UNIQUE** :

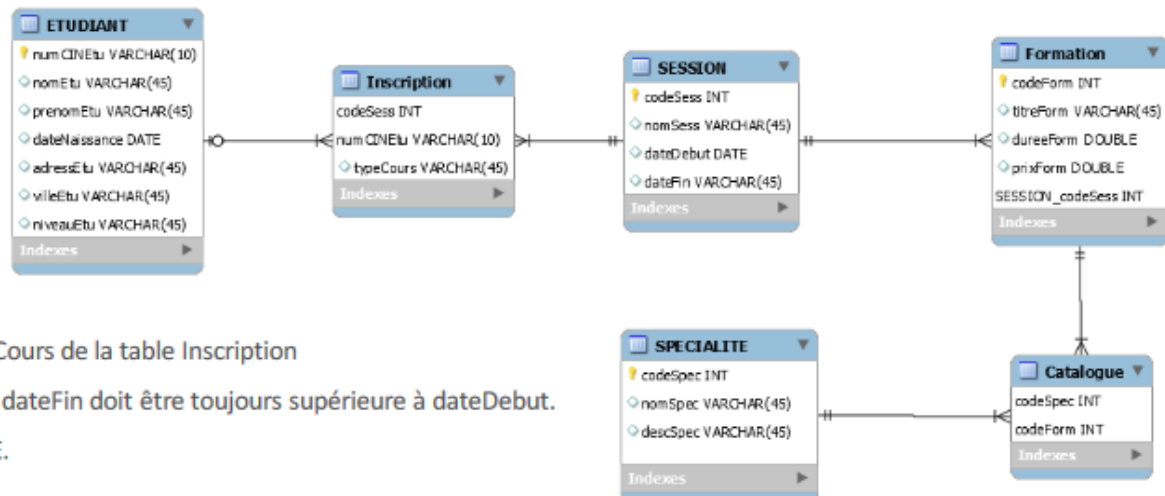
```
ALTER TABLE nom_table  
ADD CONSTRAINT nom_contrainte  
UNIQUE (liste_colonnes);
```

- 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;
```


Exercice 1 (TP)

1. Créer une base de données : « Centre formation »
2. Créer les tables depuis le MLD : « Centre de Formation » (Ne pas oublier les clés primaires et étrangères)



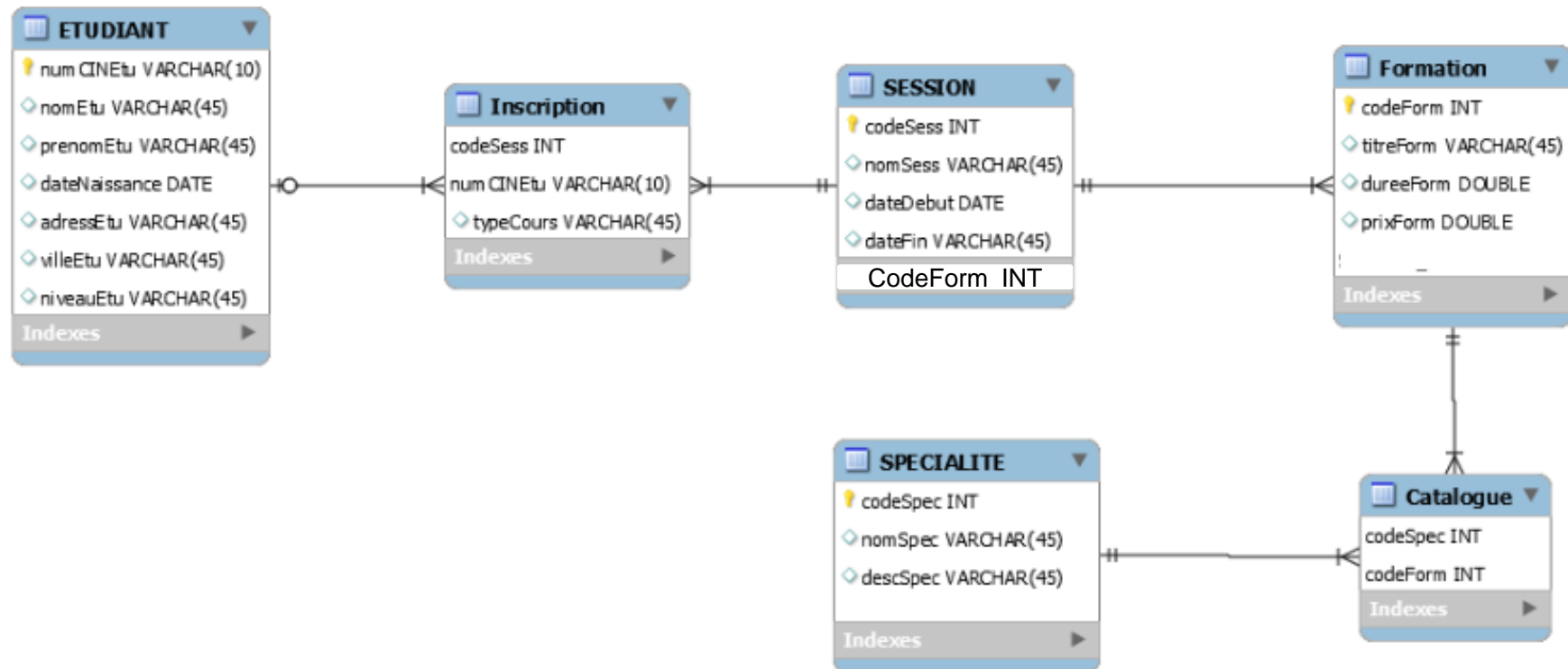
3. Ajouter une contrainte NOT NULL sur la colonne typeCours de la table Inscription
4. Ajouter une contrainte CHECK dans la table SESSION : dateFin doit être toujours supérieure à dateDebut.
5. Ajouter une colonne « Active » sur la table SPECIALITE.



Remarque

- Cette colonne est un flag qui prend la valeur 1 si la spécialité est active, et 0 si elle ne l'est pas.

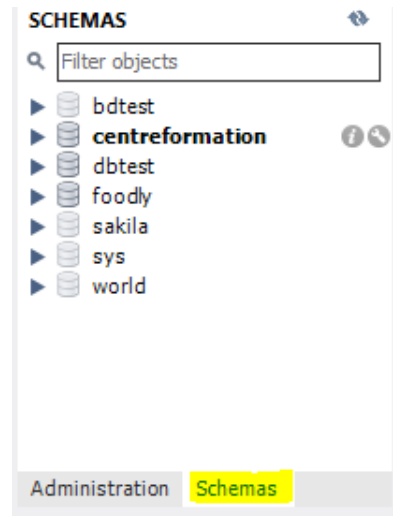
Exercice



Exercice : SOL

1. Création de la base :

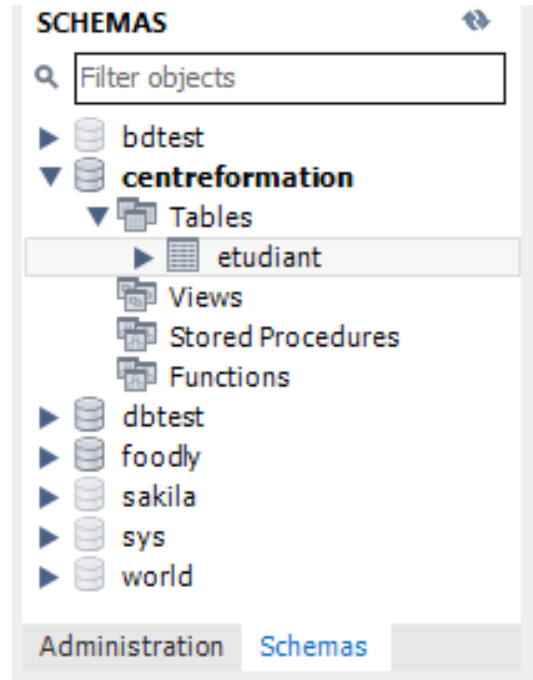
- 1 • `CREATE database IF NOT EXISTS `centreFormation` DEFAULT CHARACTER SET utf8mb4 ;`
- 2 • `USE `centreFormation` ;`



Exercice : SOL

2. Création des tables :

```
5  -----
6  -- Table `centreFormation`.`ETUDIANT`
7  -----
8  ● CREATE TABLE IF NOT EXISTS `centreFormation`.`ETUDIANT` (
9    `numCINETu` VARCHAR(10) NOT NULL,
10   `nomEtu` VARCHAR(45) NULL,
11   `prenomEtu` VARCHAR(45) NULL,
12   `dateNaissance` DATE NULL,
13   `adressEtu` VARCHAR(45) NULL,
14   `villeEtu` VARCHAR(45) NULL,
15   `niveauEtu` VARCHAR(45) NULL,
16   PRIMARY KEY (`numCINETu`))
17   ENGINE = InnoDB;
```



Exercice : SOL

2. Création des tables :

```
-- Table `centreFormation`.`Formation`
```


```
CREATE TABLE IF NOT EXISTS `centreFormation`.`Formation` (  
  `codeForm` INT NOT NULL,  
  `titreForm` VARCHAR(45) NULL,  
  `dureeForm` DOUBLE NULL,  
  `prixForm` DOUBLE NULL,  
  PRIMARY KEY (`codeForm`))  
ENGINE = InnoDB;
```

```
-- Table `centreFormation`.`SESSION`
```

```
CREATE TABLE IF NOT EXISTS `centreFormation`.`SESSION` (  
  `codeSess` INT NOT NULL,  
  `nomSess` VARCHAR(45) NULL,  
  `dateDebut` DATE NULL,  
  `dateFin` VARCHAR(45) NULL,  
  `Formation_codeForm` INT NOT NULL,  
  PRIMARY KEY (`codeSess`, `Formation_codeForm`),  
  CONSTRAINT `fk_Formation_fk1`  
  FOREIGN KEY (`Formation_codeForm`)  
  REFERENCES `centreFormation`.`FORMATION` (`codeForm`)  
  ON DELETE CASCADE  
  ON UPDATE CASCADE)  
ENGINE = InnoDB;
```

Exercice : SOL

2. Création des tables :

```
-----  
-- Table `centreFormation`.`SPECIALITE`  
-----  
 CREATE TABLE IF NOT EXISTS `centreFormation`.`SPECIALITE` (  
  `codeSpec` INT NOT NULL,  
  `nomSpec` VARCHAR(45) NULL,  
  `descSpec` VARCHAR(45) NULL,  
  PRIMARY KEY (`codeSpec`))  
ENGINE = InnoDB;
```

Exercice : SQL

2. Création des tables :

```
-----  
-- Table `centreFormation`.`Inscription`  
-----
```

```
CREATE TABLE IF NOT EXISTS `centreFormation`.`Inscription` (  
  `codeSess` INT NOT NULL,  
  `numCINETu` VARCHAR(10) NOT NULL,  
  `typeCours` VARCHAR(45),  
  PRIMARY KEY (`codeSess`, `numCINETu`),  
  CONSTRAINT `fk_SESSION_has_ETUDIANT_SESSION`  
  FOREIGN KEY (`codeSess`)  
  REFERENCES `centreFormation`.`SESSION` (`codeSess`)  
  ON DELETE CASCADE  
  ON UPDATE CASCADE,  
  CONSTRAINT `fk_SESSION_has_ETUDIANT_ETUDIANT1`  
  FOREIGN KEY (`numCINETu`)  
  REFERENCES `centreFormation`.`ETUDIANT` (`numCINETu`)  
  ON DELETE CASCADE  
  ON UPDATE CASCADE)  
ENGINE = InnoDB;
```

Exercice : SOL

2. Création des tables :

```
-- Table `centreFormation`.`Catalogue`
```

```
CREATE TABLE IF NOT EXISTS `centreFormation`.`Catalogue` (  
  `codeSpec` INT NOT NULL,  
  `codeForm` INT NOT NULL,  
  PRIMARY KEY (`codeSpec`, `codeForm`),  
  CONSTRAINT `fk_SPECIALITE_has_Formation_SPECIALITE1`  
  FOREIGN KEY (`codeSpec`)  
  REFERENCES `centreFormation`.`SPECIALITE` (`codeSpec`)  
  ON DELETE CASCADE  
  ON UPDATE CASCADE,  
  CONSTRAINT `fk_SPECIALITE_has_Formation_Formation1`  
  FOREIGN KEY (`codeForm`)  
  REFERENCES `centreFormation`.`Formation` (`codeForm`)  
  ON DELETE CASCADE  
  ON UPDATE CASCADE)  
ENGINE = InnoDB;
```


Exercice : SOL

3. Ajouter une contrainte NOT NULL sur la colonne typeCours de la table Inscription

```
102 • ALTER TABLE INSCRIPTION
```

```
103     MODIFY typeCours VARCHAR(45) NOT NULL;
```

Info	Columns	Indexes	Triggers	Foreign keys	Partitions	Grants	DDL				
Column		Type	Default Value		Nullable	Character Set	Collation	Privileges			
◇	codeSess	int			NO			select,insert,update,references			
◇	numCINETu	varchar(10)			NO	utf8mb3	utf8mb3_genera...	select,insert,update,references			
◇	typeCours	varchar(45)			NO	utf8mb3	utf8mb3_genera...	select,insert,update,references			

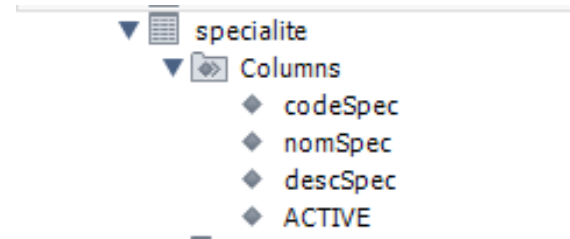
Exercice : SOL

4. Ajouter une contrainte CHECK dans la table SESSION : dateFin doit être toujours supérieure à dateDebut.

```
106 • ALTER TABLE session
107     ADD CONSTRAINT CHK_dateDebut_Fin CHECK (dateFin >= dateDebut);
108
```

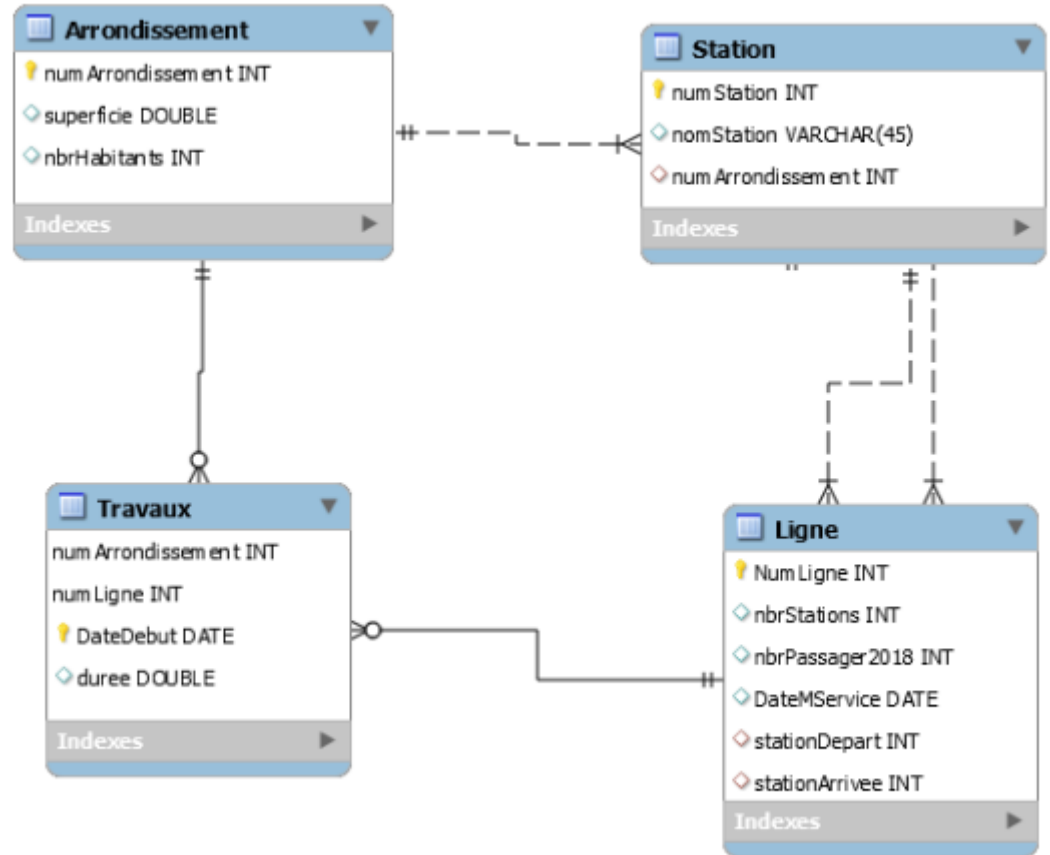
5. Ajouter une colonne « Active » sur la table SPECIALITE.

```
• ALTER TABLE SPECIALITE
  ADD ACTIVE TINYINT;
```



Exercice 2 (TP)

1. Créer une base de données : « **tramwayRabat** »
2. Créer les tables depuis le MLD : « Tramway de Rabat » **(Ne pas oublier les clés primaires et étrangères)**
3. Ajouter une colonne **nomArrond** qui contiendra le nom des arrondissements
4. Renommer la colonne **nbrPassager2018** par **nbrPassager2020**



Exercice 2 : SOL


1. Création de la base :

```
1  -- -----  
2  -- Schema TramwayRabat  
3  -- -----  
4  • CREATE SCHEMA IF NOT EXISTS `TramwayRabat` DEFAULT CHARACTER SET utf8 ;  
5  • USE `TramwayRabat` ;  
6  .
```

Exercice 2 : SOL

2. Création des tables :

```
-----  
-- Table `TramwayRabat`.`Arrondissement`  
-----
```

- 

```
CREATE TABLE IF NOT EXISTS `TramwayRabat`.`Arrondissement` (  
  `numArrondissement` INT NOT NULL,  
  `superficie` DOUBLE NULL,  
  `nbrHabitants` INT NULL,  
  PRIMARY KEY (`numArrondissement`))  
ENGINE = InnoDB;
```

Exercice 2 : SOL

2. Création des tables :(suite)

```
-----  
-- Table `TramwayRabat`.`Station`  
-----
```

- ```
CREATE TABLE IF NOT EXISTS `TramwayRabat`.`Station` (
 `numStation` INT NOT NULL,
 `nomStation` VARCHAR(45) NULL,
 `numArrondissement` INT NULL,
 PRIMARY KEY (`numStation`),
 CONSTRAINT `arrondissement_fk`
 FOREIGN KEY (`numArrondissement`)
 REFERENCES `TramwayRabat`.`Arrondissement` (`numArrondissement`)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

# Exercice 2 : SOL

## 2. Création des tables :(suite) :

```

-- Table `TramwayRabat`.`Ligne`

CREATE TABLE IF NOT EXISTS `TramwayRabat`.`Ligne` (
 `NumLigne` INT NOT NULL,
 `nbrStations` INT NULL,
 `nbrPassager2018` INT NULL,
 `DateMService` DATE NULL,
 `stationDepart` INT NULL,
 `stationArrivee` INT NULL,
 PRIMARY KEY (`NumLigne`),
 CONSTRAINT `FK1`
 FOREIGN KEY (`stationDepart`)
 REFERENCES `TramwayRabat`.`Station` (`numStation`)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION,
 CONSTRAINT `FK2`
 FOREIGN KEY (`stationArrivee`)
 REFERENCES `TramwayRabat`.`Station` (`numStation`)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

## Exercice 2 : SOL

### 2. Création des tables :(suite)

- ```
CREATE TABLE IF NOT EXISTS `TramwayRabat`.`Travaux` (  
  `numArrondissement` INT NOT NULL,  
  `numLigne` INT NOT NULL,  
  `DateDebut` DATE NOT NULL,  
  `duree` DOUBLE NULL,  
  CONSTRAINT `fk1_travaux`  
  FOREIGN KEY (`numArrondissement`)  
  REFERENCES `TramwayRabat`.`Arrondissement` (`numArrondissement`)  
  ON DELETE NO ACTION  
  ON UPDATE NO ACTION,  
  CONSTRAINT `fk2_ligne`  
  FOREIGN KEY (`numLigne`)  
  REFERENCES `TramwayRabat`.`Ligne` (`NumLigne`)  
  ON DELETE NO ACTION  
  ON UPDATE NO ACTION)  
ENGINE = InnoDB;
```


Exercice 2 : SOL

3. Ajouter une colonne nomArrond qui contiendra le nom des arrondissements

```
ALTER TABLE ARRONDISSEMENT  
ADD nomArrond VARCHAR(45);
```

4. Renommer la colonne nbrPassager2018 par nbrPassager2020

- ```
ALTER TABLE LIGNE
CHANGE COLUMN nbrPassager2018 nbrPassager2020 INT;
```

# 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

# Requêtes 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,...);
```

# Requêtes LMD

## 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.

# Requêtes LMD : EXEMPLE

## BD Foodly : Table utilisateur

| Nom du champ  | Descriptif du champ                             | Exemple de valeur |
|---------------|-------------------------------------------------|-------------------|
| <i>id</i>     | identifiant unique de l'utilisateur dans la BDD | 1                 |
| <i>nom</i>    | nom de famille de l'utilisateur                 | Durantay          |
| <i>prenom</i> | prénom de l'utilisateur                         | Quentin           |
| <i>email</i>  | email de l'utilisateur                          | quentin@gmail.com |

```
VALUES ('Durantay', 'Quentin', 'quentin@gmail.com')
Query OK, 1 row affected (0.02 sec)
```

```
INSERT INTO `utilisateur` (`nom`, `prenom`, `email`)
VALUES ('Durantay', 'Quentin', 'quentin@gmail.com');
```

# Requêtes LMD : EXEMPLE

## BD Foodly

- Si vous exécutez cette commande plusieurs fois, vous remarquerez un message d'erreur

```
mysql> INSERT INTO `utilisateur` (`nom`, `prenom`, `email`)
-> VALUES ('Durantay', 'Quentin', 'quentin@gmail.com');
ERROR 1062 (23000): Duplicate entry 'quentin@gmail.com' for key 'utilisateur.email'
mysql>
```

# Requêtes LMD : EXEMPLE

## BD Foodly : Insérer plusieurs lignes

- Si vous exécutez cette commande plusieurs fois, vous remarquerez un message d'erreur

```
INSERT INTO `utilisateur` (`nom`, `prenom`, `email`)
VALUES ('Doe', 'John', 'john@yahoo.fr'),
('Smith', 'Jane', 'jane@hotmail.com'),
('Dupont', 'Sebastien', 'sebastien@orange.fr'),
('Martin', 'Emilie', 'emilie@gmail.com');
```

- Cela devrait donner un message de réponse de ce type :

```
1 Query OK, 4 rows affected (0.00 sec)
2 Records: 4 Duplicates: 0 Warnings: 0
```

# Requêtes LMD : EXEMPLE

## BD Foodly : Table aliment

| Nom du champ     | Descriptif du champ                                     | Exemple de valeur |
|------------------|---------------------------------------------------------|-------------------|
| <i>id</i>        | identifiant unique de l'aliment dans la BDD             | 1                 |
| <i>nom</i>       | nom de l'aliment                                        | poire             |
| <i>marque</i>    | marque de l'aliment                                     | Monoprix          |
| <i>calories</i>  | nombre de calories contenues dans l'aliment (en kcal)   | 72                |
| <i>sucre</i>     | la concentration en sucre de l'aliment (en grammes)     | 19,1              |
| <i>graisses</i>  | la concentration en graisses de l'aliment (en grammes)  | 0,2               |
| <i>proteines</i> | la concentration en protéines de l'aliment (en grammes) | 0,4               |
| <i>bio</i>       | l'aliment est Bio                                       | TRUE              |



# Requêtes LMD : EXEMPLE

## BD Foodly : Table aliment

| Nom du champ     | Descriptif du champ                                     | Exemple de valeur |
|------------------|---------------------------------------------------------|-------------------|
| <i>id</i>        | identifiant unique de l'aliment dans la BDD             | 1                 |
| <i>nom</i>       | nom de l'aliment                                        |                   |
| <i>marque</i>    | marque de l'aliment                                     |                   |
| <i>calories</i>  | nombre de calories                                      |                   |
| <i>sucre</i>     | la concentration en sucre (en grammes)                  |                   |
| <i>graisses</i>  | la concentration en graisses (en grammes)               |                   |
| <i>proteines</i> | la concentration en protéines de l'aliment (en grammes) | 0,4               |
| <i>bio</i>       | l'aliment est Bio                                       | TRUE              |

```
INSERT INTO `aliment` (`nom`, `marque`, `sucre`, `calories`, `graisses`, `proteines`, `bio`)
VALUES
('poire', 'monoprix', 27.5, 134, 0.2, 1.1, FALSE),
('pomme', 'monoprix', 19.1, 72, 0.2, 0.4, FALSE),
('oeuf', 'carrefour', 0.6, 82, 5.8, 6.9, TRUE),
('lait d\'amande', 'bjorg', 4.5, 59, 3.9, 1.1, TRUE);
```

# Requêtes LMD : EXEMPLE

## BD Foodly : Table aliment

```
INSERT INTO `aliment` (`nom`, `marque`, `sucre`, `calories`, `graisses`, `proteines`, `bio`)
VALUES
('poire', 'monoprix', 27.5, 134, 0.2, 1.1, FALSE),
('pomme', 'monoprix', 19.1, 72, 0.2, 0.4, FALSE),
('oeuf', 'carrefour', 0.6, 82, 5.8, 6.9, TRUE),
('lait d\'amande', 'bjorg', 4.5, 59, 3.9, 1.1, TRUE);
```

Noms de tables  
ou colonnes

Backticks (`)

Valeurs de type  
TEXT ou VARCHAR

Guillemets (")

Valeurs de type  
BOOLEAN, INTEGER, FLOAT

Pas de guillemets

# Requêtes LMD : EXEMPLE

## BD Foodly : Respectez la structure des tables

- Si nous essayons de taper la commande :

```
INSERT INTO `utilisateur` (`nom`, `prenom`)
VALUES ('Hello', 'World');
```

- Nous avons un message d'erreur :

```
ERROR 1364 (HY000): Field 'email' doesn't have a default value
```

# Requêtes LMD : EXEMPLE

**BD Foodly : À vous de jouer !**

- Essayez de créer un nouvel aliment :

| nom            | marque   | calories | sucres | graisses | proteines | bio   |
|----------------|----------|----------|--------|----------|-----------|-------|
| haricots verts | Monoprix | 25       | 3      | 0        | 1,7       | FALSE |

# Requêtes LMD : EXEMPLE

## BD Foodly : Solution

Essayez de créer un nouvel aliment :

| nom            | marque   | calories | sucre | graisses | proteines | bio   |
|----------------|----------|----------|-------|----------|-----------|-------|
| haricots verts | Monoprix | 25       | 3     | 0        | 1,7       | FALSE |

```
INSERT INTO `aliment` (`nom`, `marque`, `sucre`, `calories`, `graisses`, `proteines`, `bio`)
VALUES
('haricot vert', 'monoprix', 25, 3, 0, 1.7, FALSE);
```

```
Query OK, 1 row affected (0.00 sec)
```

# Requêtes LMD : EXEMPLE 2

## BD bdtest :

Soit la table « **Produits** » 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 >= 0),
 Date_ajout DATE
);
```

1 - On veut ajouter le produit suivant : Numéro du Produit est **P12**, Son prix est **14** et le cout **12**.

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

# Requêtes LMD : EXEMPLE 2

## BD bdtest :

- **Ajout de plusieurs lignes** : One veut ajouter les produits suivant :
- Numéro du Produit est **P13**, Le cout: 120, le prix 140, ajouté le **01/01/2023**.
- Numéro du Produit est **P100**, Description: Laptop, le cout: 120, le prix 140, ajouté **aujourd'hui**.

# Requêtes LMD : EXEMPLE 2

## BD bdtest :

- **Ajout de plusieurs lignes** : On veut ajouter les produits suivant :
- Numéro du Produit est **P13**, Le cout: 120, le prix 140, ajouté le **01/01/2023**.
- Numéro du Produit est **P100**, Description: Laptop, le cout: 120, le prix 140, ajouté **aujourd'hui**.

```
INSERT INTO Produits(num_produit,description,cout,prix,date_ajout)
VALUES ('P13','',120,140,'2023-01-01'),
('P100','Laptop',5000,6000,CURRENT_DATE)
;
```



# Requêtes LMD : EXEMPLE 2

**BD bdtest : 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 | 2023-04-23 |
| P12         | Non specifie | 12.00   | 14.00   | NULL       |
| P13         |              | 120.00  | 140.00  | 2023-01-01 |

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

```
mysql>
```

# **Activité 2 :**

## **Réaliser les requêtes SQL**

### **(TP)**

# Exercice 1 : « Centre de Formation »

Dans le schéma « **centreformation** », insérer les données suivantes dans les tables correspondantes en utilisant la commande « **INSERT** ».

ETUDIANT :

| numCINEtu | nomEtu        | prenomEtu    | dateNaissance | adressEtu                                 | villeEtu   | niveauEtu  |
|-----------|---------------|--------------|---------------|-------------------------------------------|------------|------------|
| AB234567  | Alami         | Ahmad        | 01/01/1986    | Rue du port, 13                           | Tanger     | bac        |
| G5346789  | Toumi         | Aicha        | 12/03/2000    | N12 immeuble Jaouhara                     | Casablanca | Master     |
| C0987265  | Souni         | Sanaa        | 30/04/1998    | Place du peuple n 2                       | Tanger     | Niveau bac |
| D2356903  | Idrissi Alami | Mohammed     | 5/5/1996      | Lotissement najah,<br>rue n 12 immeuble 3 | Rabat      | Bac+ 4     |
| Y1234987  | Ouled thami   | Ali          | 04/12/1979    | La ville haute, rue<br>chouhada n 6       | Tanger     | Bachelor   |
| J3578902  | Ben Omar      | Abd Allah    | 25/06/1990    | Villa Amina<br>n12 bir rami               | Kenitra    | Phd        |
| F9827363  | Boudiaf       | Fatima Zohra | 10/01/1997    | Immeuble iftikhar,<br>n 13 ettakaddoum    | Rabat      | Bac + 2    |

# Exercice 1 : « Centre de Formation » correction

Les requêtes d'insertions :

```
INSERT INTO ETUDIANT (numCINETu,nomEtu,prenomEtu,dateNaissance,adressEtu,villeEtu,niveauEtu)
values
('AB234567','Alami','Ahmad','1986-01-01','Rue du port, 13','Tanger','bac'),
('G5346789','Toumi','Aicha','2000-12-03','N12 immeuble Jaouhara','Casablanca','Master'),
('C0987265','Souni','Sanaa','1998-04-30','Place du peuple n 2','Tanger','Niveau bac'),
('D2356903','Idrissi Alami','Mohammed','1996-05-05','Lotissement najah, rue n 12 immeuble 3','Rabat','Bac+ 4'),
('Y1234987','Ouled thami','Ali','1979-12-04','La ville haute, rue chouhada n 6','Tanger','Bachelor'),
('J3578902','Ben Omar','Abd Allah','1990-06-25','Villa Amina n12 bir rami','Kenitra','Phd'),
('F9827363','Boudiaf','Fatima Zohra','1997-01-10','Immeuble iftikhar', 'n 13 ettakaddoum','Rabat','Bac + 2');
```

# Exercice 1 : « Centre de Formation »

Dans le schéma « **centreformation** », insérer les données suivantes dans les tables correspondantes en utilisant la commande « **INSERT** ».

**FORMATION :**

| codeForm | titreForm              | dureeForm | prixForm |
|----------|------------------------|-----------|----------|
| 11       | Programming Java       | 12        | 3600     |
| 12       | web developpement      | 14        | 4200     |
| 13       | Anglais technique      | 15        | 3750     |
| 14       | Communication          | 10        | 2500     |
| 15       | Base de données Oracle | 20        | 6000     |
| 16       | Soft skills            | 12        | 3000     |

# Exercice 1 : « Centre de Formation »

## correction

Les requêtes d'insertions :

```
Insert into FORMATION (codeForm,titreForm,dureeForm,prixForm)
values
(11,'Programming Java',12,3600),
(12,'web developpement',14,4200),
(13,'Anglais technique',15,3750),
(14,'Communication',10,2500),
(15,'Base de données Oracle',20,6000),
(16,'Soft skills',12,3000);
```

# Exercice 1 : « Centre de Formation »

SESSION :

Dans le schéma « **centreformation** », insérer les données suivantes dans les tables correspondantes en utilisant la commande « **INSERT** ».

| codeSess | nomSess      | Datedebut  | Datefin    | codeform |
|----------|--------------|------------|------------|----------|
| 1101     | Session1101  | 2022-01-02 | 2022-01-14 | 11       |
| 1102     | Session 1102 | 2022-02-03 | 2022-02-15 | 11       |
| 1201     | Session 1201 | 2022-03-04 | 2022-03-18 | 12       |
| 1202     | Session 1202 | 2022-04-05 | 2022-04-19 | 12       |
| 1301     | Session 1301 | 2022-01-06 | 2022-01-21 | 13       |
| 1302     | Session 1302 | 2022-05-07 | 2022-05-22 | 13       |
| 1303     | Session 1303 | 2022-06-08 | 2022-06-23 | 13       |
| 1401     | Session 1401 | 2022-09-01 | 2022-09-11 | 14       |
| 1402     | Session 1402 | 2022-08-08 | 2022-08-18 | 14       |
| 1501     | Session 1501 | 2022-11-11 | 2022-12-01 | 15       |
| 1502     | Session 1502 | 2022-09-12 | 2022-10-02 | 15       |
| 1601     | Session 1601 | 2022-09-13 | 2022-09-25 | 16       |
| 1602     | Session 1602 | 2022-10-14 | 2022-10-26 | 16       |
| 1104     | Session 1104 | 2022-10-15 | 2022-10-27 | 11       |
| 1203     | Session 1203 | 2022-11-16 | 2022-11-30 | 12       |
| 1204     | Session 1204 | 2022-12-17 | 2022-12-31 | 12       |

# Exercice 1 : « Centre de Formation »

## correction

Les requêtes d'insertions :

```
Insert into session (codesess,nomsess,datedebut,datefin)
values
(1101,'Session 1101','2022-01-02','2022-01-14',11),
(1102,'Session 1102','2022-02-03','2022-02-15',11),
(1201,'Session 1201','2022-03-04','2022-03-18',12),
(1202,'Session 1202','2022-04-05','2022-04-19',12),
(1301,'Session 1301','2022-01-06','2022-01-21',13),
(1302,'Session 1302','2022-05-07','2022-05-22',13),
(1303,'Session 1303','2022-06-08','2022-06-23',13),
(1401,'Session 1401','2022-09-01','2022-09-11',14),
(1402,'Session 1402','2022-08-08','2022-08-18',14),
(1501,'Session 1501','2022-11-11','2022-12-01',15),
(1502,'Session 1502','2022-09-12','2022-10-02',15),
(1601,'Session 1601','2022-09-13','2022-09-25',16),
(1602,'Session 1602','2022-10-14','2022-10-26',16),
(1104,'Session 1104','2022-10-15','2022-10-27',11),
(1203,'Session 1203','2022-11-16','2022-11-30',12),
(1204,'Session 1204','2022-12-17','2022-12-31',12);
```



# Exercice 1 : « Centre de Formation »

Dans le schéma « **centreformation** », insérer les données suivantes dans les tables correspondantes en utilisant la commande « **INSERT** ».

**SPECIALITE :**

| codeSpec | nomSpec       | descSpec                        | Active |
|----------|---------------|---------------------------------|--------|
| 101      | GL            | Genie logiciel et developpement | 1      |
| 102      | Data          | Data engineering                | 1      |
| 103      | Langues       | Anglais, Français               | 1      |
| 104      | Communication | Communication                   | 1      |
| 105      | Securite      | Reseaux et securite             | 0      |

# Exercice 1 : « Centre de Formation » correction

Les requêtes d'insertions :

```
INSERT INTO SPECIALITE (codeSpec,nomSpec,descSpec,Active)
VALUES
(101,'GL', 'Genie logiciel et developpement',1),
(102,'Data', 'Data engineering',1),
(103,'Langues', 'Anglais Français',1),
(104,'Communication','Communication',1),
(105,'Securite','Reseaux et securite',0);
```

# Exercice 1 : « Centre de Formation »

Dans le schéma « **centreformation** », insérer les données suivantes dans les tables correspondantes en utilisant la commande « **INSERT** ».

CATALOGUE :

| CodeSpec | codeForm |
|----------|----------|
| 101      | 11       |
| 101      | 12       |
| 102      | 15       |
| 101      | 15       |
| 103      | 13       |
| 104      | 13       |
| 104      | 14       |
| 104      | 16       |

# Exercice 1 : « Centre de Formation » correction

Les requêtes d'insertions :

```
Insert into catalogue (CodeSpec, codeForm)
Values
(101,11),
(101,12),
(102,15),
(101,15),
(103,13),
(104,13),
(104,14),
(104,16);
```

# Exercice 1 : « Centre de Formation »

Dans le schéma « **centreformation** »,  
insérer les données suivantes dans  
les tables correspondantes en  
utilisant la commande « **INSERT** ».

| codeSess | numCINETu | TypeCours  |
|----------|-----------|------------|
| 1101     | AB234567  | Distanciel |
| 1101     | G5346789  | Distanciel |
| 1101     | C0987265  | Distanciel |
| 1101     | D2356903  | Distanciel |
| 1101     | Y1234987  | Distanciel |
| 1101     | J3578902  | Distanciel |
| 1101     | F9827363  | Distanciel |
| 1201     | AB234567  | Présentiel |
| 1201     | G5346789  | Présentiel |
| 1201     | C0987265  | Présentiel |
| 1201     | D2356903  | Présentiel |
| 1201     | Y1234987  | Présentiel |
| 1201     | J3578902  | Présentiel |
| 1302     | AB234567  | Présentiel |
| 1302     | G5346789  | Distanciel |
| 1302     | C0987265  | Présentiel |
| 1302     | D2356903  | Présentiel |
| 1302     | Y1234987  | Présentiel |
| 1401     | G5346789  | Distanciel |
| 1401     | C0987265  | Distanciel |
| 1401     | D2356903  | Distanciel |
| 1401     | Y1234987  | Distanciel |
| 1401     | J3578902  | Distanciel |
| 1401     | F9827363  | Distanciel |
| 1501     | AB234567  | Distanciel |
| 1501     | G5346789  | Présentiel |
| 1501     | C0987265  | Distanciel |
| 1501     | D2356903  | Présentiel |
| 1501     | Y1234987  | Présentiel |
| 1501     | J3578902  | Présentiel |
| 1501     | F9827363  | Presenciel |

# Exercice 1 : « Centre de Formation » correction

Les requêtes d'insertions :

```
Insert into INSCRIPTION (codeSess,numCINETu,TypeCours)
```

```
Values
```

```
(1101,'AB234567','Distanciel'),
(1101,'G5346789','Distanciel'),
(1101,'C0987265','Distanciel'),
(1101,'D2356903','Distanciel'),
(1101,'Y1234987','Distanciel'),
(1101,'J3578902','Distanciel'),
(1101,'F9827363','Distanciel'),
(1201,'AB234567','Presenciel'),
(1201,'G5346789','Presenciel'),
(1201,'C0987265','Presenciel'),
(1201,'D2356903','Presenciel'),
(1201,'Y1234987','Presenciel'),
(1201,'J3578902','Presenciel'),
(1302,'AB234567','Presenciel'),
...
```

```
(1302,'G5346789','Distanciel'),
(1302,'C0987265','Presenciel'),
(1302,'D2356903','Presenciel'),
(1302,'Y1234987','Presenciel'),
(1401,'G5346789','Distanciel'),
(1401,'C0987265','Distanciel'),
(1401,'D2356903','Distanciel'),
(1401,'Y1234987','Distanciel'),
(1401,'J3578902','Distanciel'),
(1401,'F9827363','Distanciel'),
(1501,'AB234567','Distanciel'),
(1501,'G5346789','Presenciel'),
(1501,'C0987265','Distanciel'),
(1501,'D2356903','Presenciel'),
(1501,'Y1234987','Presenciel'),
(1501,'J3578902','Presenciel'),
(1501,'F9827363','Presenciel');
```

# Requêtes LMD

## 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];
```

# Requêtes LMD

## UPDATE : EXEMPLE BD FOODLY

- On veut changer l'e-mail d'un utilisateur. **UN utilisateur** et non plusieurs. Cela pose la question de **comment identifier** l'utilisateur, et comment **sélectionner la ligne** correspondant.

```
UPDATE `utilisateur` SET `email` = 'quentind@gmail.com' WHERE `id` = '1';
```

- Vous devriez avoir un message qui confirme ce changement :

```
Query OK, 1 row affected (0.00 sec)
```

```
Rows matched: 1 Changed: 1 Warnings: 0
```



# Requêtes LMD

## UPDATE : EXEMPLE BD FOODLY

Mettre à jour  
cette table



Colonne(s) et valeur(s)  
à modifier



```
UPDATE `utilisateur`
```

```
SET `email` = valeurdelemail
```

```
WHERE `id`
```



Filtrer pour ajouter des conditions

La commande UPDATE

# Requêtes LMD

## UPDATE : EXEMPLE BD FOODLY : Mettez à jour tous les objets

➤ AVANT :

```
mysql> select * from utilisateur;
```

| id | nom      | prenom    | email                 |
|----|----------|-----------|-----------------------|
| 1  | Durantay | Quentin   | quentin@gmail.com     |
| 3  | Doe      | John      | john@yahoo.fr         |
| 4  | Smith    | Jane      | jane@hotmail.com      |
| 5  | Dupont   | Sebastien | sebastien@orange.fr   |
| 6  | Martin   | Emilie    | emilie@gmail.com      |
| 7  | Durantay | Quentin   | quentin1111@gmail.com |

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

➤ Après l'exécution de cette requête

```
UPDATE `utilisateur` SET prenom = "quentin";
```

# Requêtes LMD

**UPDATE : EXEMPLE BD FOODLY : Mettez à jour tous les objets**

➤ **RESULTAT :**

```
mysql> UPDATE `utilisateur` SET prenom = "quentin";
Query OK, 6 rows affected (0.00 sec)
Rows matched: 6 Changed: 6 Warnings: 0

mysql> select * from utilisateur;
```

| id | nom      | prenom  | email                 |
|----|----------|---------|-----------------------|
| 1  | Durantay | quentin | quentin@gmail.com     |
| 3  | Doe      | quentin | john@yahoo.fr         |
| 4  | Smith    | quentin | jane@hotmail.com      |
| 5  | Dupont   | quentin | sebastien@orange.fr   |
| 6  | Martin   | quentin | emilie@gmail.com      |
| 7  | Durantay | quentin | quentin1111@gmail.com |

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

# Requêtes LMD

## UPDATE : EXEMPLE BD dbtest

- Rappelons la table Produits :

| Num_Produit | description  | cout    | prix    | Date_ajout |
|-------------|--------------|---------|---------|------------|
| P100        | Laptop       | 5000.00 | 6000.00 | 2023-04-23 |
| P12         | Non specifie | 12.00   | 14.00   | NULL       |
| P13         |              | 120.00  | 140.00  | 2023-01-01 |

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

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

```
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

# Requêtes LMD

## UPDATE : EXEMPLE BD dbtest

- Modifier les données de telle façon que Description = '**Non specifié**' et **Prix = 1.5\*cout** :

```
UPDATE Produits
SET
Description = 'Non specifié',
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 specifié	5000.00	7500.00	2023-04-23
P12	Non specifié	12.00	18.00	2021-12-31
P13	Non specifié	120.00	180.00	2023-01-01
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

# Exercice 2 : « Centre de Formation »

Ecrire et exécuter les requêtes qui permettent d'effectuer les modifications suivantes :

1. Dans la table ETUDIANT, modifier les données de la colonne « **niveauEtu** » comme suit :

| Ancienne valeur | Nouvelle Valeur |
|-----------------|-----------------|
| Master          | Bac+ 5          |
| Bachelor        | Bac+ 4          |
| Phd             | Doctorat        |

2. Renommer le titre de la formation 11 avec « **développement Java** »
3. Ajouter une colonne « **numInscription** » dans la table INSCRIPTION. La valeur de cette colonne sera la concaténation du codeSess et numCINEtu.

**Utiliser la fonction concat()**

4. Il y a eu une erreur lors de l'inscription de l'étudiant dont le nom est « **Alami** », sa date de naissance est le **02 Janvier 1987**, et il est de la ville de **Kenitra**.
5. L'étudiante **Aicha** souhaite rendre tous ses cours en distanciel.

# Solution Exercice 2: « Centre de Formation »

## 1. Q1

```
Update ETUDIANT
```

```
Set niveauEtu = 'Bac+ 5' where niveauEtu='Master';
```

```
Update ETUDIANT
```

```
Set niveauEtu = 'Bac+ 4' where niveauEtu='Bachelor';
```

```
Update ETUDIANT
```

```
Set niveauEtu = 'Phd' where niveauEtu='Doctorat';
```

# Solution Exercice 2: « Centre de Formation »

2. Q2

```
Update FORMATION
```

```
Set titreForm= 'développement Java'
```

```
where codeForm = 11;
```



# Solution Exercice 2: « Centre de Formation »

## 3. Q3

```
ALTER TABLE inscription
```

```
ADD numinscription VARCHAR(40);
```

```
Update inscription
```

```
SET numinscription = concat(codeSess,numCINETu);
```

# Solution Exercice 2: « Centre de Formation »

4. Q4

```
Update ETUDIANT
Set dateNaissance = '1987-01-02',
villeEtu = 'Kenitra'
where nomEtu='Alami';
```

# Solution Exercice 2: « Centre de Formation »

5. Q5

```
Update inscription
set Typecours = 'Distanciel'
where numCINETu = 'G5346789';
```

# Requêtes LMD

## 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;
```

- 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**.

# Requêtes LMD

## DELETE : EXEMPLE BD FOODLY

- Admettons qu'un utilisateur souhaite se **désinscrire** de **Foodly**. Il faudrait alors le supprimer de votre BDD

```
DELETE FROM `utilisateur` WHERE `id` = '2';
```

- Si on tape la requête suivante : Quel serait votre réflexe ?

```
DELETE FROM utilisateur;
```

# Requêtes LMD

## DELETE : EXEMPLE BD dbtest

- On veut supprimer les Produits dont le **cout**  $\leq 12$  .

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

```
mysql> use dbtest;
Database changed
mysql> DELETE FROM Produits WHERE cout<=12;
Query OK, 1 row affected (0.00 sec)

mysql> select * from produits;
```

| Num_Produit | description  | cout    | prix    | Date_ajout |
|-------------|--------------|---------|---------|------------|
| P100        | Non specifie | 5000.00 | 7500.00 | 2023-04-23 |
| P13         | Non specifie | 120.00  | 180.00  | 2023-01-01 |

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

# CHAPITRE 2 : Réaliser des requêtes SQL

1. Requetes LMD

**2. Requetes de sélection**

3. Expression du SGBD

4. Fonctions d'agrégation du SGBD

5. Sous requêtes

6. Requetes de l'union

7. Jointures

# Requêtes de sélection

## **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;
```

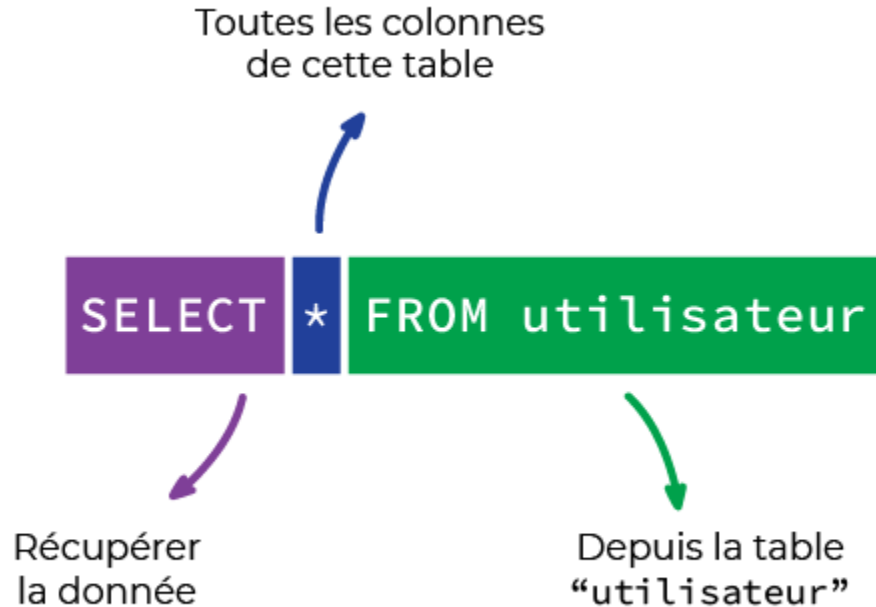
- **N.B :** Lors de l'exécution de l'instruction SELECT, MySQL évalue la clause FROM avant la clause SELECT :





# Requêtes de sélection

## SELECT :



# Requêtes de sélection

## SELECT :

```
mysql> SELECT `nom`, `prenom`, `email` FROM utilisateur;
```

| nom      | prenom  | email                 |
|----------|---------|-----------------------|
| Durantay | quentin | quentin@gmail.com     |
| Doe      | quentin | john@yahoo.fr         |
| Smith    | quentin | jane@hotmail.com      |
| Dupont   | quentin | sebastien@orange.fr   |
| Martin   | quentin | emilie@gmail.com      |
| Durantay | quentin | quentin1111@gmail.com |

6 rows in set (0.00 sec)

```
mysql> SELECT * FROM utilisateur;
```

|   | id       | nom     | prenom                | email |
|---|----------|---------|-----------------------|-------|
| 1 | Durantay | quentin | quentin@gmail.com     |       |
| 3 | Doe      | quentin | john@yahoo.fr         |       |
| 4 | Smith    | quentin | jane@hotmail.com      |       |
| 5 | Dupont   | quentin | sebastien@orange.fr   |       |
| 6 | Martin   | quentin | emilie@gmail.com      |       |
| 7 | Durantay | quentin | quentin1111@gmail.com |       |

6 rows in set (0.00 sec)

# Requêtes de sélection

## BD FOODLY : SELECT

```
20 • SELECT * FROM aliment;
```

[illegible]

# Requêtes de sélection

## BD FOODLY : SELECT

[illegible]

```
22 • SELECT * FROM aliment WHERE id = 4;
```

[illegible]



# Requêtes de sélection

## BD FOODLY : SELECT

[illegible]

```
27 • SELECT * FROM aliment WHERE calories < 90;
```

[illegible]

# Requêtes de sélection

**BD FOODLY : SELECT => Il existe un autre mot clé pour effectuer des comparaisons sur du texte :  
il s'agit du mot clé en dehors de l'opérateur égal**

**LIKE** permet de sélectionner les objets dont le texte d'une colonne répond à un modèle spécifique.

```
29 • SELECT * FROM utilisateur;
```

30

31

| Result Grid     Filter Rows: <input type="text"/> Edit: |      |          |         |                       |
|---------------------------------------------------------|------|----------|---------|-----------------------|
|                                                         | id   | nom      | prenom  | email                 |
| ▶                                                       | 1    | Durantay | quentin | quentin@gmail.com     |
|                                                         | 3    | Doe      | quentin | john@yahoo.fr         |
|                                                         | 4    | Smith    | quentin | jane@hotmail.com      |
|                                                         | 5    | Dupont   | quentin | sebastien@orange.fr   |
|                                                         | 6    | Martin   | quentin | emilie@gmail.com      |
|                                                         | 7    | Durantay | quentin | quentin1111@gmail.com |
| *                                                       | NULL | NULL     | NULL    | NULL                  |

31

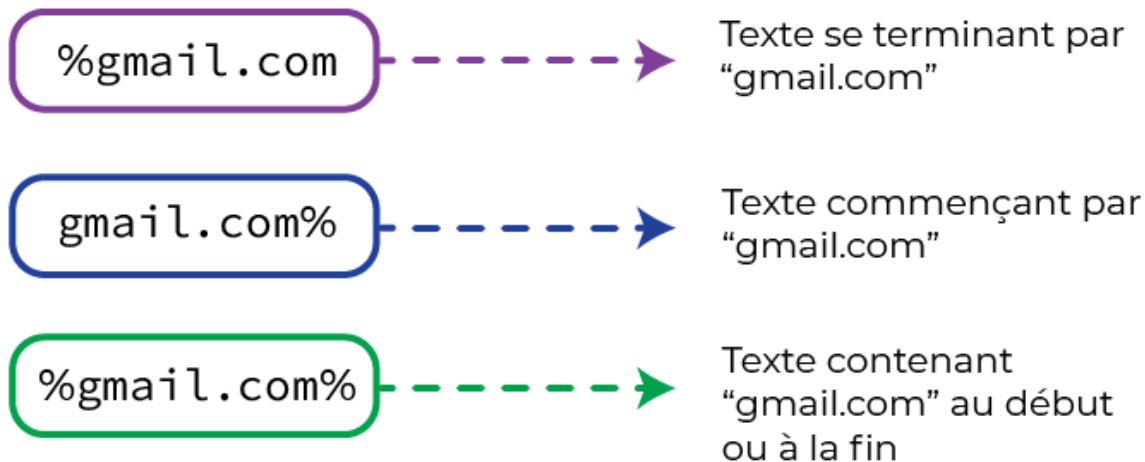
```
32 • SELECT * FROM utilisateur WHERE email LIKE "%gmail.com";
```

| Result Grid     Filter Rows: <input type="text"/> Edit:    Export/Import |      |          |         |                       |
|--------------------------------------------------------------------------|------|----------|---------|-----------------------|
|                                                                          | id   | nom      | prenom  | email                 |
| ▶                                                                        | 1    | Durantay | quentin | quentin@gmail.com     |
|                                                                          | 6    | Martin   | quentin | emilie@gmail.com      |
|                                                                          | 7    | Durantay | quentin | quentin1111@gmail.com |
| *                                                                        | NULL | NULL     | NULL    | NULL                  |

# Requêtes de sélection

**BD FOODLY : SELECT => Il existe un autre mot clé pour effectuer des comparaisons sur du texte :  
il s'agit du mot clé en dehors de l'opérateur égal**

Le caractère `%` que nous avons écrit à un rôle très spécifique. Il va permettre de faire correspondre des schémas spécifiques, on parle parfois de **pattern**, dans les données textuelles. Voyez plutôt :





# Requêtes de sélection

## SELECT :

- Exemples de requêtes **SELECT** simples :

```
mysql> select * from produits;
```

| Num_Produit | description  | cout    | prix    | Date_ajout |
|-------------|--------------|---------|---------|------------|
| P100        | Non specifie | 5000.00 | 7500.00 | 2023-04-23 |
| P13         | Non specifie | 120.00  | 180.00  | 2023-01-01 |

2 rows in set (0.00 sec)

# Requêtes de sélection

## SELECT :

- Exemples de requêtes **SELECT** simples :

```
mysql> select Num_Produit,description from produits;
```

```
+-----+-----+
| Num_Produit | description |
+-----+-----+
| P100 | Non specifie |
| P13 | Non specifie |
+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql>
```

# Requêtes de sélection

## SELECT :

- Exemples de requêtes **SELECT** simples :

```
mysql> select Num_Produit,description from produits;
```

```
+-----+-----+
| Num_Produit | description |
+-----+-----+
| P100 | Non specifie |
| P13 | Non specifie |
+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql>
```

# Requêtes de sélection

## 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 :



# Requêtes de sélection

## **DISTINCT :**

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

```
1 • SELECT description FROM dbtest.produits;
2
```

<

Result Grid

|   | description  |
|---|--------------|
| ▶ | Laptop       |
|   | Non specifie |
|   | Laptop       |
|   |              |

```
1 • SELECT distinct description FROM dbtest.produits;
2
```

<

Result Grid

|   | description  |
|---|--------------|
| ▶ | Laptop       |
|   | Non specifie |
|   |              |

# Requêtes de sélection

## 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
```

|   | 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
```

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

# Exercice 3 : « Centre de Formation »

1. Consulter le contenu de toutes les tables du schéma « CentreFormation »
2. Donner la liste des étudiants qui sont de la ville de Tanger.
3. Donner la liste des Formations qui coutent plus de 3000 Dhs.
4. Pour chaque formation, donner le prix journalier. (Supposant que la durée renseignée sur la table est en jours)
5. Donner la liste des sessions ouvertes de la formation dont le code est 11.
6. Donner la liste des numéros CIN des étudiants inscrits dans la session 1302 classés par ordre alphabétique.
7. Donner la liste des spécialités Actives.



# Solution Exercice 3 : « Centre de Formation »

1. Consulter le contenu de toutes les tables du schéma « CentreFormation »

```
Select * from (nom_de _la_table) ;
```

2. Donner la liste des étudiants qui sont de la ville de Tanger.

```
Select * from ETUDIANT where villeEtu ='Tanger';
Select * from ETUDIANT where villeEtu like 'Tanger';
```

3. Donner la liste des Formations qui coutent plus de 3000 Dhs.

```
SELECT * FROM centreformation.formation
Where prixForm >3000;
```



# Solution Exercice 3 : « Centre de Formation »

4. Pour chaque formation, donner le prix journalier. (Supposant que la durée renseignée sur la table est en jours)

```
SELECT codeForm, titreForm, prixForm/dureeForm FROM centreformation.formation;
```

5. Donner la liste des sessions ouvertes de la formation dont le code est 11.

```
SELECT * FROM centreformation.session
where codeForm = 11;
```

6. Donner la liste des numéros CIN des étudiants inscrits dans la session 1302 classés par ordre alphabétique.

```
SELECT numCINETu FROM centreformation.inscription Where codeSess =1302;
```

7. Donner la liste des spécialités Actives.

```
SELECT * FROM centreformation.specialite where Active =1;
```

# Requêtes de sélection

## 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 :

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

Result Grid | Filter Rows: | Export: | Wrap

|   | description | Prix-Cout |
|---|-------------|-----------|
| ▶ | Laptop      | 1000.00   |
|   | Non specfie | 2.00      |
|   | Laptop      | 1000.00   |
|   |             | 20.00     |
|   |             | 20.00     |

```
1
2 • SELECT description, Prix-Cout from produits
3 GROUP BY description;
4
5
```

Result Grid | Filter Rows: | Export: | Wrap

|   | description | Prix-Cout |
|---|-------------|-----------|
| ▶ | Laptop      | 1000.00   |
|   | Non specfie | 2.00      |

# Requêtes de sélection

## GROUP BY :

- En pratique, on utilise souvent la clause **GROUP BY** avec des fonctions d'agrégation telles que **SUM**, **AVG**, **MAX**, **MIN** et **COUNT**



# Requêtes de sélection

## GROUP BY :

```
select * from test group by category;
```

| day        | category | value |
|------------|----------|-------|
| 2020-03-01 | Red      | 62    |
| 2020-12-01 | Blue     | 63    |
| 2020-01-01 | Green    | 55    |
| 2020-04-01 | Blue     | 41    |
| 2020-06-01 | Red      | 60    |
| 2020-10-01 | Blue     | 60    |
| 2020-07-01 | Green    | 41    |
| 2020-11-01 | Green    | 60    |
| 2020-08-01 | Red      | 46    |

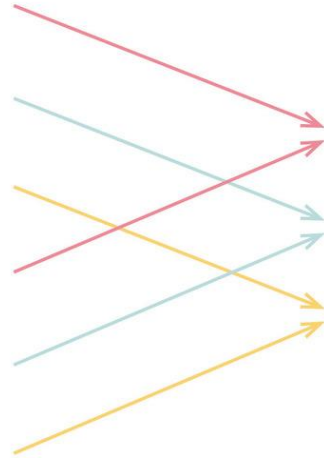


| day        | category | value |
|------------|----------|-------|
| 2020-04-01 | Blue     | 63    |
| 2020-10-01 | Blue     | 41    |
| 2020-12-01 | Blue     | 60    |
| 2020-01-01 | Green    | 55    |
| 2020-07-01 | Green    | 41    |
| 2020-11-01 | Green    | 60    |
| 2020-03-01 | Red      | 62    |
| 2020-06-01 | Red      | 60    |
| 2020-08-01 | Red      | 46    |

# Requêtes de sélection

**GROUP BY :**

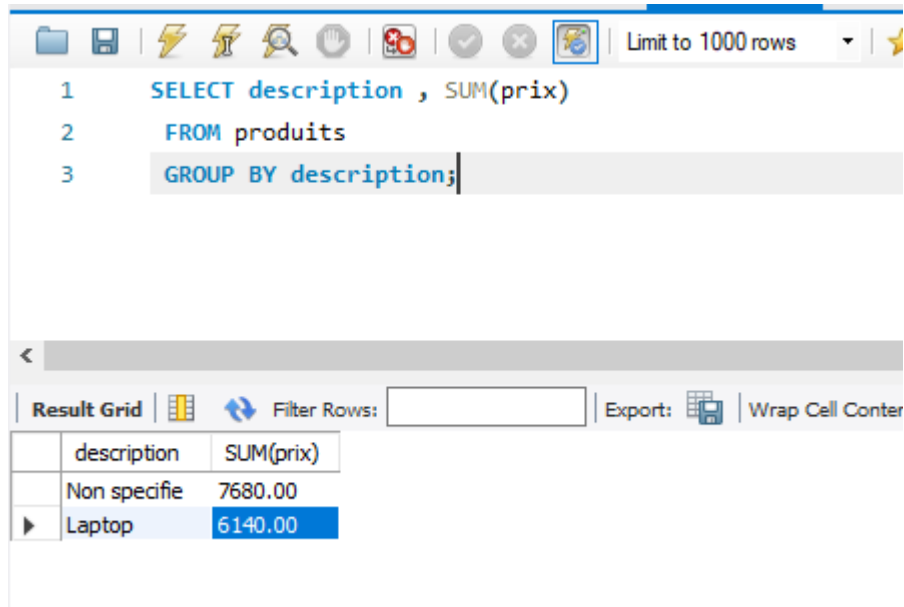
| title  | genre     | price |
|--------|-----------|-------|
| book 1 | adventure | 11.90 |
| book 2 | fantasy   | 8.49  |
| book 3 | romance   | 9.99  |
| book 4 | adventure | 9.99  |
| book 5 | fantasy   | 7.99  |
| book 6 | romance   | 5.88  |



| genre     | avg_price                 |
|-----------|---------------------------|
| adventure | $(11.90 + 9.99)/2$ 10.945 |
| fantasy   | $(8.49 + 7.99)/2$ 8.24    |
| romance   | $(9.99 + 5.88)/2$ 7.935   |

# Requêtes de sélection

## GROUP BY :SUM



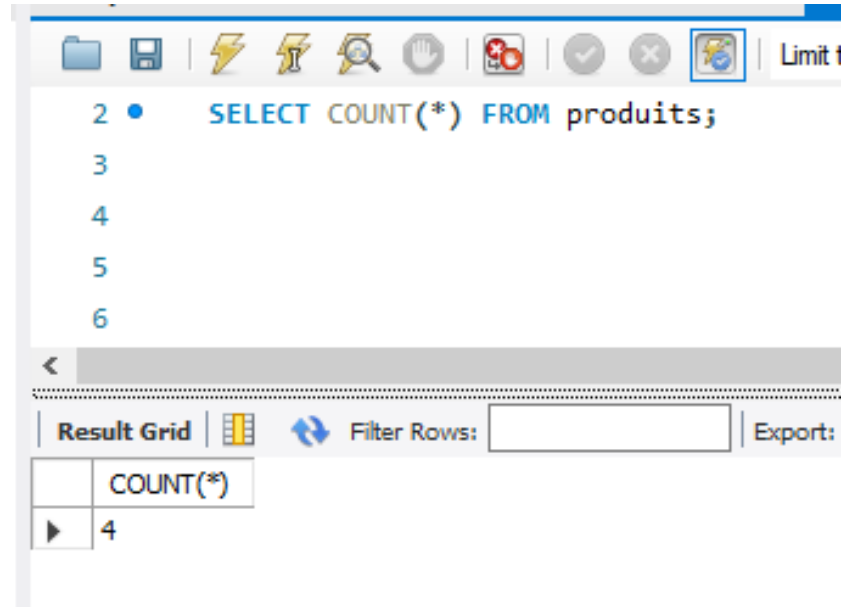
The screenshot shows a SQL query editor with a toolbar at the top containing icons for file operations, execution, and search. The query text is as follows:

```
1 SELECT description , SUM(prix)
2 FROM produits
3 GROUP BY description;
```

Below the query editor, the results are displayed in a table with the following data:

| description | SUM(prix) |
|-------------|-----------|
| Non specife | 7680.00   |
| Laptop      | 6140.00   |

## GROUP BY :COUNT



The screenshot shows a SQL query editor with a toolbar at the top. The query text is as follows:

```
2 • SELECT COUNT(*) FROM produits;
```

Below the query editor, the results are displayed in a table with the following data:

| COUNT(*) |
|----------|
| 4        |

# Requêtes de sélection

**GROUP BY :COUNT => exemple avec la BD**

**FOODLY**

Vous souvenez-vous de la commande que nous avons effectuée pour récupérer uniquement les utilisateurs dont l'e-mail était un Gmail ? Comment l'adapter pour connaître le **nombre d'utilisateurs avec une adresse Gmail dans la base ?**



# Requêtes de sélection

**GROUP BY :COUNT => exemple avec la BD**

**FOODLY**

Vous souvenez-vous de la commande que nous avons effectuée pour récupérer uniquement les utilisateurs dont l'e-mail était un Gmail ? Comment l'adapter pour connaître le **nombre d'utilisateurs avec une adresse Gmail dans la base ?**

```
35 • SELECT COUNT(*)
36 FROM utilisateur
37 WHERE email LIKE "%gmail.com";
38
39
```



| Result Grid |          | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|----------|--------------|---------|--------------------|
|             | COUNT(*) |              |         |                    |
| ▶           | 3        |              |         |                    |



# Requêtes de sélection

**GROUP BY :COUNT => exemple avec la BD**

**FOODLY**


```
39 • SELECT COUNT(email)
40 FROM utilisateur
41 WHERE email LIKE "%gmail.com";
42
43
```



# Requêtes de sélection

**GROUP BY :COUNT => exemple avec la BD FOODLY**

```
39 • SELECT COUNT(email)
40 FROM utilisateur
41 WHERE email LIKE "%gmail.com";
42
43
```

|                                                                                             |              |
|---------------------------------------------------------------------------------------------|--------------|
| <                                                                                           |              |
| Result Grid                                                                                 |              |
| Filter Rows: <input type="text"/>                                                           |              |
| Export:  |              |
|                                                                                             | COUNT(email) |
| ▶                                                                                           | 3            |

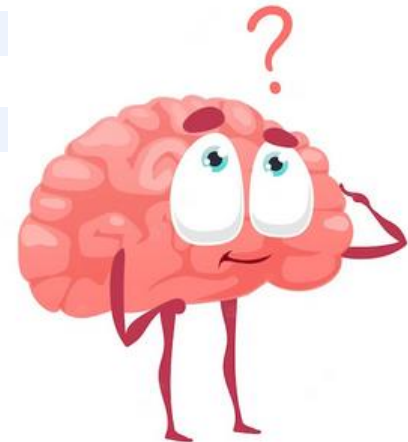
# Requêtes de sélection

**GROUP BY :COUNT => exemple avec la BD FOODLY**

**Table aliment :**

|   | id   | nom           | marque    | sucres | calories | graisses | proteines | bio  |
|---|------|---------------|-----------|--------|----------|----------|-----------|------|
| ▶ | 1    | haricot vert  | monoprix  | 25     | 3        | 0        | 1.7       | 0    |
|   | 2    | poire         | monoprix  | 27.5   | 134      | 0.2      | 1.1       | 0    |
|   | 3    | pomme         | monoprix  | 19.1   | 72       | 0.2      | 0.4       | 0    |
|   | 4    | oeuf          | carrefour | 0.6    | 82       | 5.8      | 6.9       | 1    |
|   | 5    | lait d'amande | bjorg     | 4.5    | 59       | 3.9      | 1.1       | 1    |
| • | NULL | NULL          | NULL      | NULL   | NULL     | NULL     | NULL      | NULL |

```
47 • SELECT COUNT(DISTINCT nom)
48 FROM aliment
49 WHERE nom LIKE "%pomme%";
50
```



# Requêtes de sélection

**GROUP BY :COUNT => exemple avec la BD FOODLY**

**Table aliment :**

|   | id   | nom           | marque    | sucres | calories | graisses | proteines | bio  |
|---|------|---------------|-----------|--------|----------|----------|-----------|------|
| ▶ | 1    | haricot vert  | monoprix  | 25     | 3        | 0        | 1.7       | 0    |
|   | 2    | poire         | monoprix  | 27.5   | 134      | 0.2      | 1.1       | 0    |
|   | 3    | pomme         | monoprix  | 19.1   | 72       | 0.2      | 0.4       | 0    |
|   | 4    | oeuf          | carrefour | 0.6    | 82       | 5.8      | 6.9       | 1    |
|   | 5    | lait d'amande | bjorg     | 4.5    | 59       | 3.9      | 1.1       | 1    |
| • | NULL | NULL          | NULL      | NULL   | NULL     | NULL     | NULL      | NULL |

```
47 • SELECT COUNT(DISTINCT nom)
48 FROM aliment
49 WHERE nom LIKE "%pomme%";
50
```

|             |            |
|-------------|------------|
| <           |            |
| Result Grid |            |
|             | COUNT(nom) |
| ▶           | 1          |





# Requêtes de sélection

**GROUP BY :COUNT => exemple avec la BD FOODLY**

**Table aliment :**

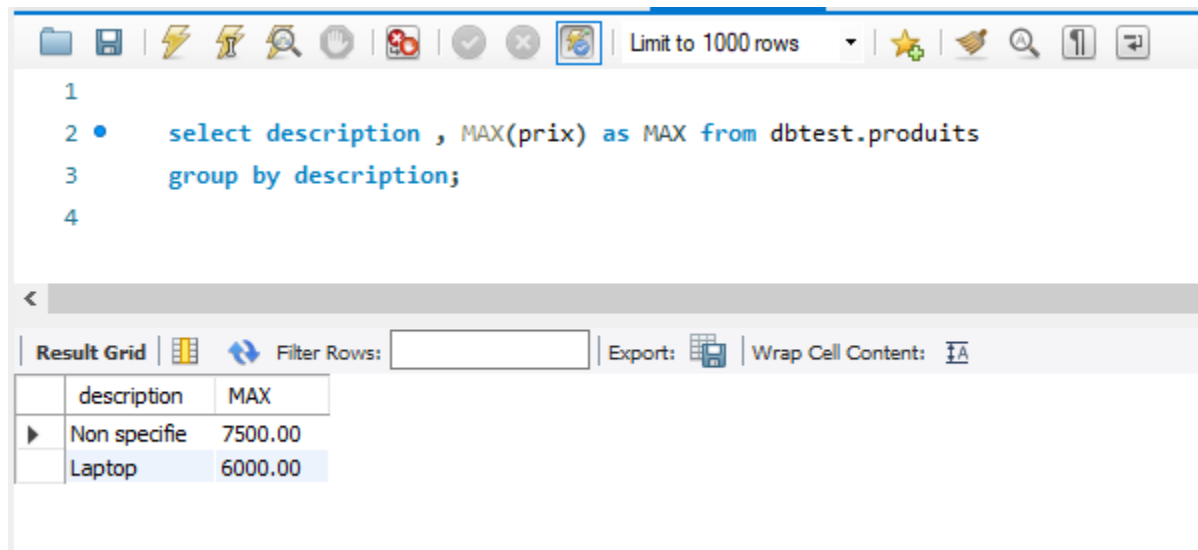
|   | id   | nom           | marque    | sucre | calories | graisses | proteines | bio  |
|---|------|---------------|-----------|-------|----------|----------|-----------|------|
| ▶ | 1    | haricot vert  | monoprix  | 25    | 3        | 0        | 1.7       | 0    |
|   | 2    | poire         | monoprix  | 27.5  | 134      | 0.2      | 1.1       | 0    |
|   | 3    | pomme         | monoprix  | 19.1  | 72       | 0.2      | 0.4       | 0    |
|   | 4    | oeuf          | carrefour | 0.6   | 82       | 5.8      | 6.9       | 1    |
|   | 5    | lait d'amande | bjorg     | 4.5   | 59       | 3.9      | 1.1       | 1    |
|   | 6    | pomme         | test      | 18.1  | 72       | 0.2      | 0.4       | 0    |
|   | 7    | golden pomme  | test      | 18.1  | 72       | 0.2      | 0.4       | 0    |
| * | NULL | NULL          | NULL      | NULL  | NULL     | NULL     | NULL      | NULL |

```
47 • SELECT COUNT(DISTINCT nom)
48 FROM aliment
49 WHERE nom LIKE "%pomme%";
50
```

| Result Grid |                     |  |  Filter Rows: |
|-------------|---------------------|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
|             | COUNT(DISTINCT nom) |                                                                                     |                                                                                                  |
| ▶           | 2                   |                                                                                     |                                                                                                  |

# Requêtes de sélection

## GROUP BY :MAX



The screenshot shows a database query editor interface. The top toolbar contains various icons for file operations, execution, and viewing. The main text area contains a SQL query:

```
1
2 • select description , MAX(prix) as MAX from dbtest.produits
3 group by description;
4
```

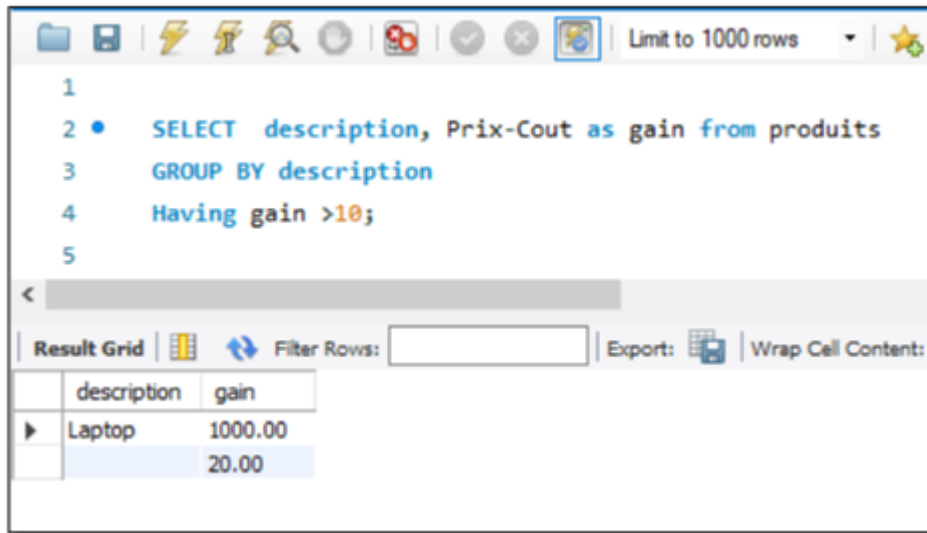
Below the query editor is a "Result Grid" section. It includes a "Filter Rows:" input field, an "Export:" button, and a "Wrap Cell Content:" checkbox. The result grid displays the following data:

|   | description | MAX     |
|---|-------------|---------|
| ▶ | Non specfie | 7500.00 |
|   | Laptop      | 6000.00 |

# Requêtes de sélection

## 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**.





# Requêtes de sélection

## 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**.

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

<

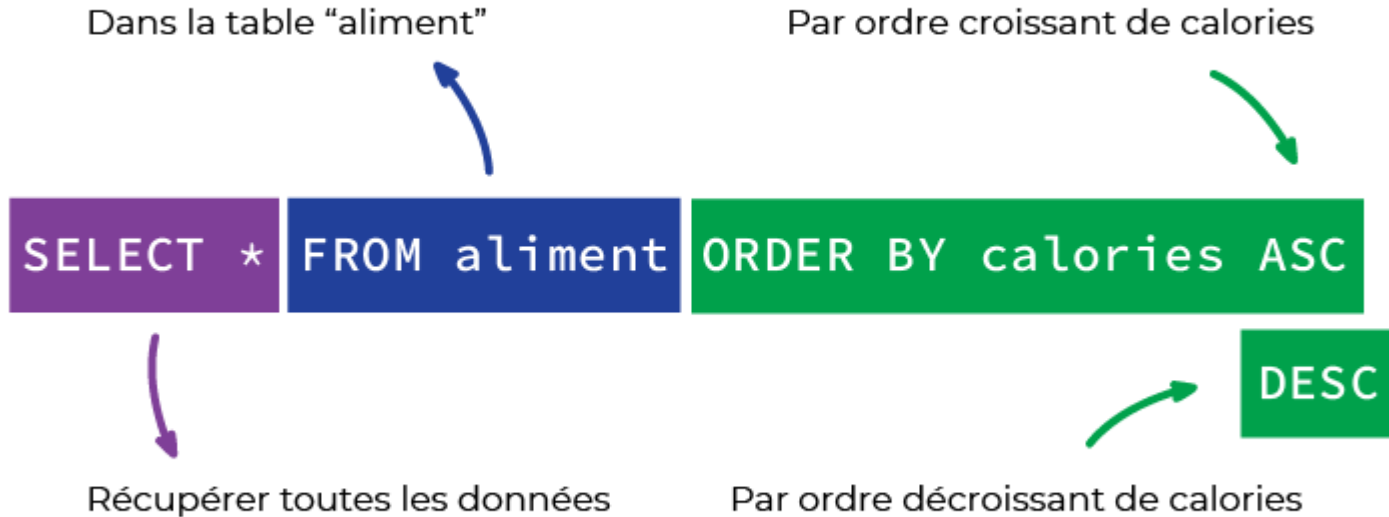
Result Grid Filter Rows:  Export: Wrap Cell Content:

|   | num_produit | gain    |
|---|-------------|---------|
| ▶ | P12         | 2.00    |
|   | P13         | 20.00   |
|   | P14         | 20.00   |
|   | P100        | 1000.00 |
|   | P120        | 1000.00 |

# Requêtes de sélection

ORDER BY :

X



Le mot clé ORDER BY

# CHAPITRE 2 : Réaliser des requêtes SQL

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

# Expression du SGBD

- 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 chaînes de caractères en date ou en nombre selon le besoin.

# Expression du SGBD

## 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**

# Expression du SGBD

## 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

# Expression du SGBD

## Les opérateurs MySQL : Opérateurs arithmétiques

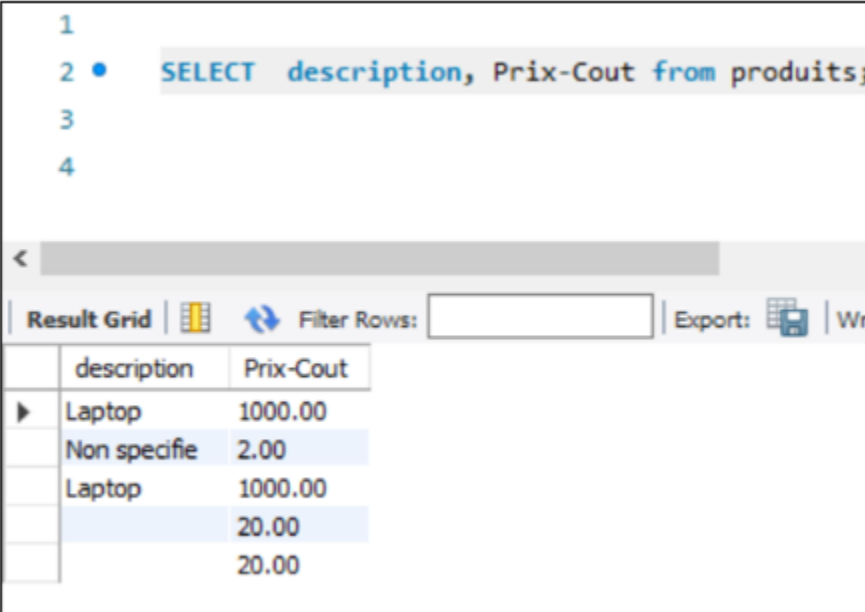
### Priorité des opérateurs :

- 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.

# Expression du SGBD

## Les opérateurs MYSQL : Opérateurs arithmétiques

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



The screenshot shows a MySQL query editor with a query window at the top and a result grid at the bottom. The query window contains the following SQL statement:

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

The result grid displays the output of the query. It has two columns: 'description' and 'Prix-Cout'. The data is as follows:

|   | description  | Prix-Cout |
|---|--------------|-----------|
| ▶ | Laptop       | 1000.00   |
|   | Non specifié | 2.00      |
|   | Laptop       | 1000.00   |
|   |              | 20.00     |
|   |              | 20.00     |



# Expression du SGBD

## 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:
  1. = (Égal à)
  2. > (Supérieur à),
  3. < (Inférieur à),
  4. >= (Supérieur ou égal à),
  5. <= (Inférieur ou égal à),
  6. <> (Différent de)

# Expression du SGBD

## 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 => diapo suivante ) sont utilisés pour comparer une valeur scalaire (**unique**) avec une sous requête.

# Expression du SGBD

## Les opérateurs MYSQL : Opérateurs logiques

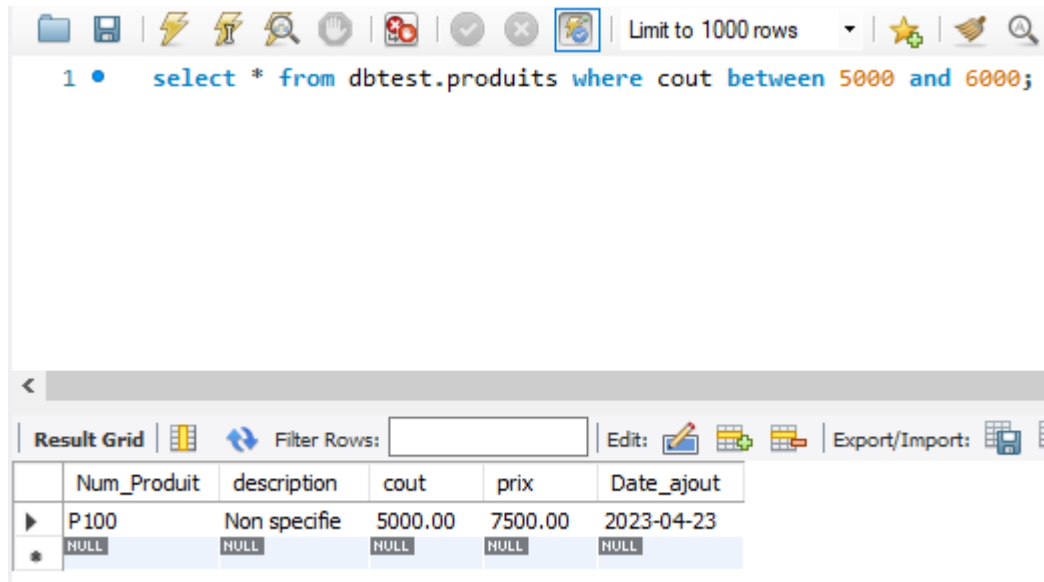
| Opérateur  | Description                                                        |
|------------|--------------------------------------------------------------------|
| ALL (*)    | TRUE si tous les éléments dun jeu de comparaisons sont TRUE.       |
| AND        | TRUE les deux expressions booléennes 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                                    |

# Expression du SGBD

## 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



The screenshot shows a MySQL query editor interface. At the top, there is a toolbar with various icons. Below the toolbar, a query is entered in a text area:

```
1 • select * from dbtest.produits where cout between 5000 and 6000;
```

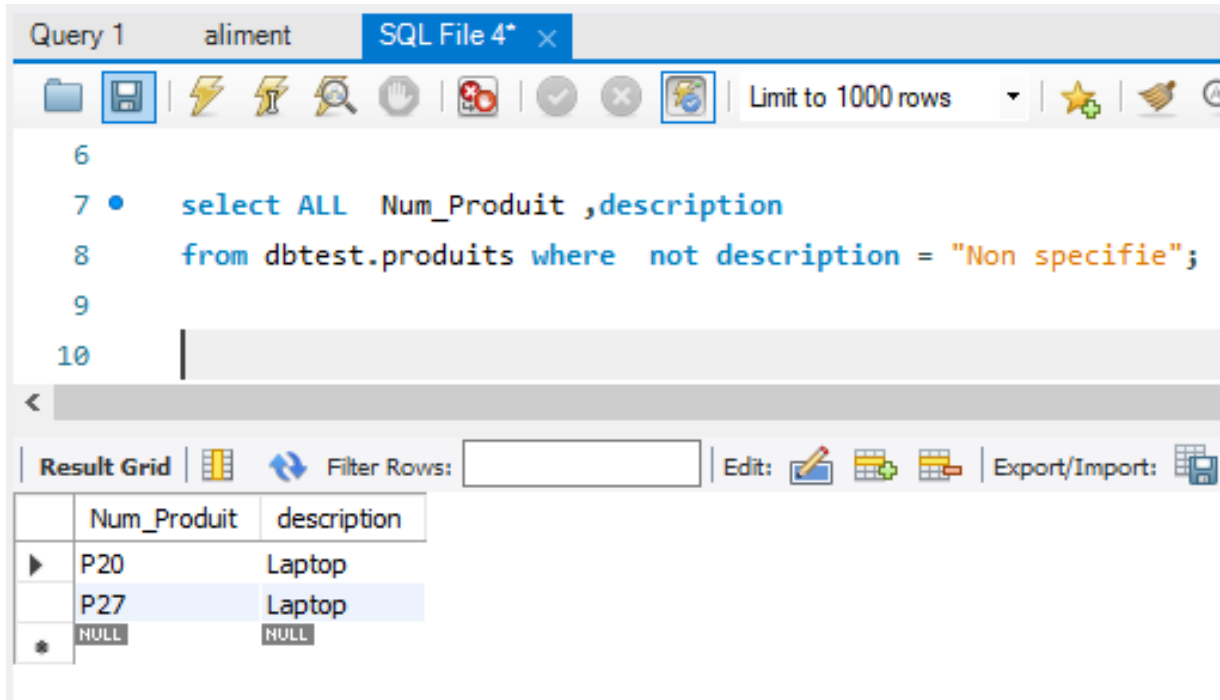
Below the query editor, there is a section for the results. It includes a toolbar with icons for 'Result Grid', 'Filter Rows', 'Edit', and 'Export/Import'. Below this toolbar is a table with the following data:

|   | Num_Produit | description  | cout    | prix    | Date_ajout |
|---|-------------|--------------|---------|---------|------------|
| ▶ | P100        | Non specifié | 5000.00 | 7500.00 | 2023-04-23 |
| * | NULL        | NULL         | NULL    | NULL    | NULL       |

# Expression du SGBD

## Les opérateurs MYSQL : Opérateurs logiques

### Opérateur not et ALL



The screenshot shows a MySQL query editor window with a tab labeled "SQL File 4\*". The query text is as follows:

```
6
7 • select ALL Num_Produit ,description
8 from dbtest.produits where not description = "Non specifie";
9
10
```

Below the query editor, the "Result Grid" is displayed, showing the results of the query. The grid has two columns: "Num\_Produit" and "description". The results are as follows:

| Num_Produit | description |
|-------------|-------------|
| P20         | Laptop      |
| P27         | Laptop      |
| NULL        | NULL        |






# Expression du SGBD

## Les opérateurs MYSQL : Opérateurs logiques

### Opérateur OR

```
6
7 • select *
8 from dbtest.produits where prix = 7500 OR prix = 6000;
9
10
```

<

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

|   | Num_Produit | description | cout    | prix    | Date_ajout |
|---|-------------|-------------|---------|---------|------------|
| ▶ | P100        | Non specife | 5000.00 | 7500.00 | 2023-04-23 |
|   | P27         | Laptop      | 5000.00 | 6000.00 | 2023-04-23 |
| * | NULL        | NULL        | NULL    | NULL    | NULL       |

# Expression du SGBD

## 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 l'intervalle ([a-f]) ou de l'ensemble spécifié ([abcdef])                                |
| [^]                 | Tout caractère en dehors de l'intervalle ([^a-f]) ou de l'ensemble spécifié (^abcdef). ^ représente le NOT |

# Expression du SGBD

## Les opérateurs MYSQL : Opérateurs logiques

### Opérateur LIKE/NOT LIKE

#### • Exemples :




```
1 • select * from dbtest.produits
2 where description like "L%";
3
```

Result Grid

|   | Num_Produit | description | cout    | prix    | Date_ajout |
|---|-------------|-------------|---------|---------|------------|
| ▶ | P20         | Laptop      | 120.00  | 140.00  | 2023-01-01 |
|   | P27         | Laptop      | 5000.00 | 6000.00 | 2023-04-23 |
| * | NULL        | NULL        | NULL    | NULL    | NULL       |

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



```
1 • select * from dbtest.produits
2 where Num_Produit like "%2_";
3
```

Result Grid

|   | Num_Produit | description | cout    | prix    | Date_ajout |
|---|-------------|-------------|---------|---------|------------|
| ▶ | P20         | Laptop      | 120.00  | 140.00  | 2023-01-01 |
|   | P27         | Laptop      | 5000.00 | 6000.00 | 2023-04-23 |
| * | NULL        | NULL        | NULL    | NULL    | NULL       |

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



# Expression du SGBD

## 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>

# Expression du SGBD

## Les fonctions intégrées MySQL : Fonctions Mathématiques

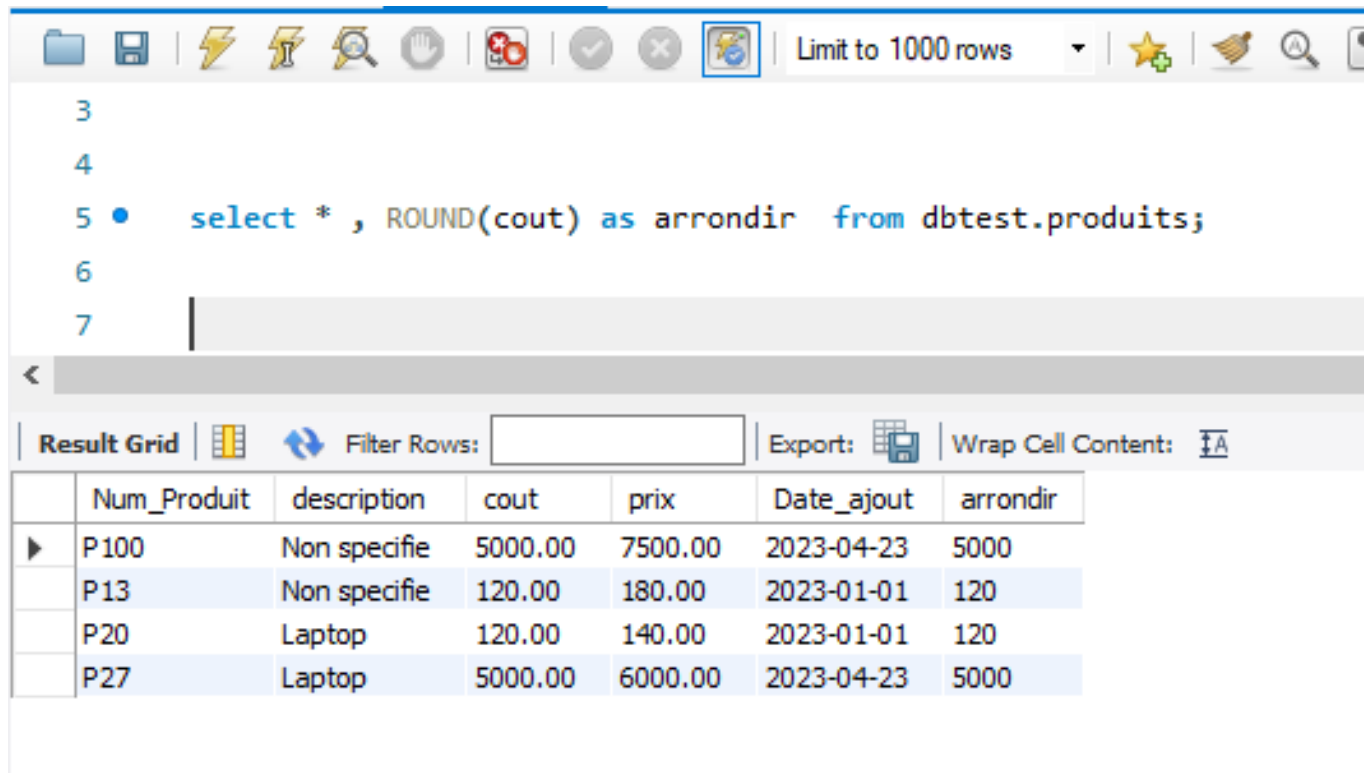
- Ce sont des fonctions ayant un ou plusieurs nombres comme arguments, et qui renvoient une valeur numérique

| 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.                   |

# Expression du SGBD

## Les fonctions intégrées MySQL : Fonctions Mathématiques

### Exemple : ROUND



The screenshot shows a MySQL query editor interface. The query being executed is:

```
select * , ROUND(cout) as arrondir from dbtest.produits;
```

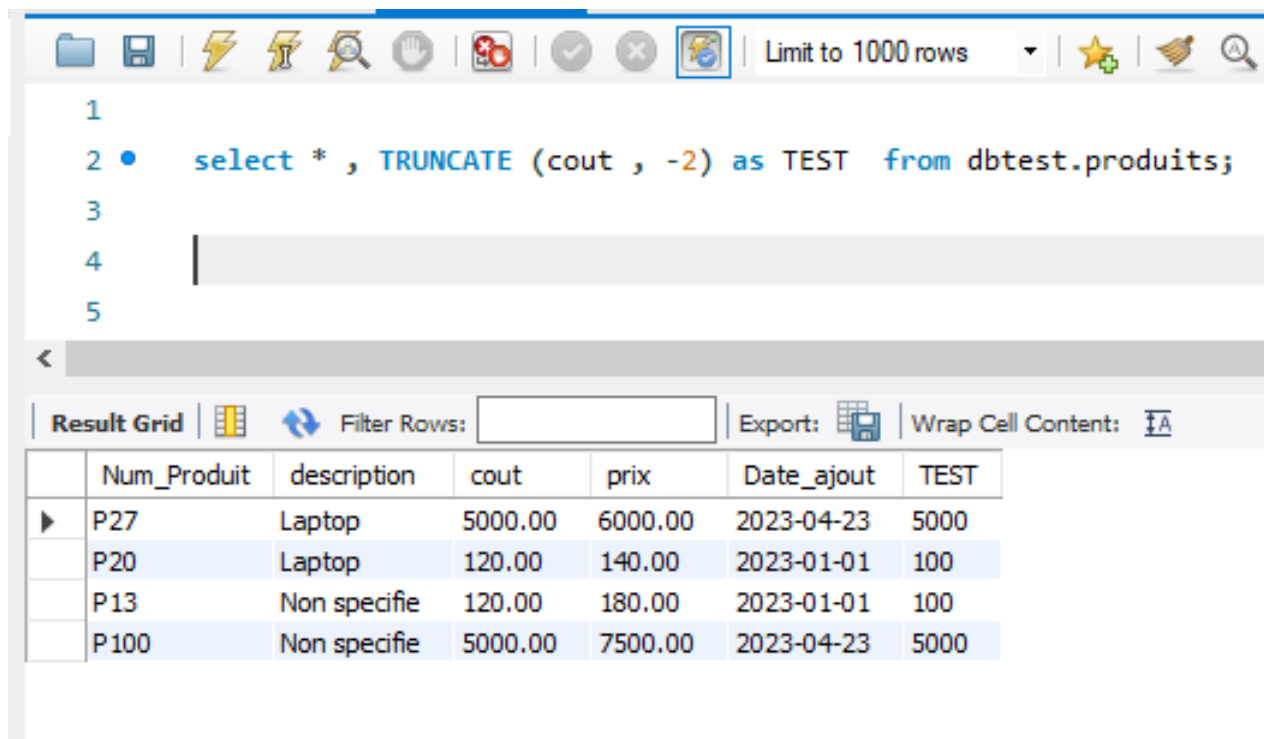
The results are displayed in a table with the following columns: Num\_Produit, description, cout, prix, Date\_ajout, and arrondir. The data is as follows:

|   | Num_Produit | description | cout    | prix    | Date_ajout | arrondir |
|---|-------------|-------------|---------|---------|------------|----------|
| ▶ | P100        | Non specife | 5000.00 | 7500.00 | 2023-04-23 | 5000     |
|   | P13         | Non specife | 120.00  | 180.00  | 2023-01-01 | 120      |
|   | P20         | Laptop      | 120.00  | 140.00  | 2023-01-01 | 120      |
|   | P27         | Laptop      | 5000.00 | 6000.00 | 2023-04-23 | 5000     |

# Expression du SGBD

## Les fonctions intégrées MYSQL : Fonctions Mathématiques

### Exemple : TRUNCATE



The screenshot shows a MySQL query editor interface. The query editor has a toolbar with various icons and a "Limit to 1000 rows" dropdown. The query being executed is:

```
1
2 • select * , TRUNCATE (cout , -2) as TEST from dbtest.produits;
3
4
5
```

Below the query editor, the "Result Grid" is displayed, showing the results of the query. The grid has columns: Num\_Produit, description, cout, prix, Date\_ajout, and TEST. The results are as follows:

|   | Num_Produit | description | cout    | prix    | Date_ajout | TEST |
|---|-------------|-------------|---------|---------|------------|------|
| ▶ | P27         | Laptop      | 5000.00 | 6000.00 | 2023-04-23 | 5000 |
|   | P20         | Laptop      | 120.00  | 140.00  | 2023-01-01 | 100  |
|   | P13         | Non specife | 120.00  | 180.00  | 2023-01-01 | 100  |
|   | P100        | Non specife | 5000.00 | 7500.00 | 2023-04-23 | 5000 |

# Expression du SGBD

## 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.

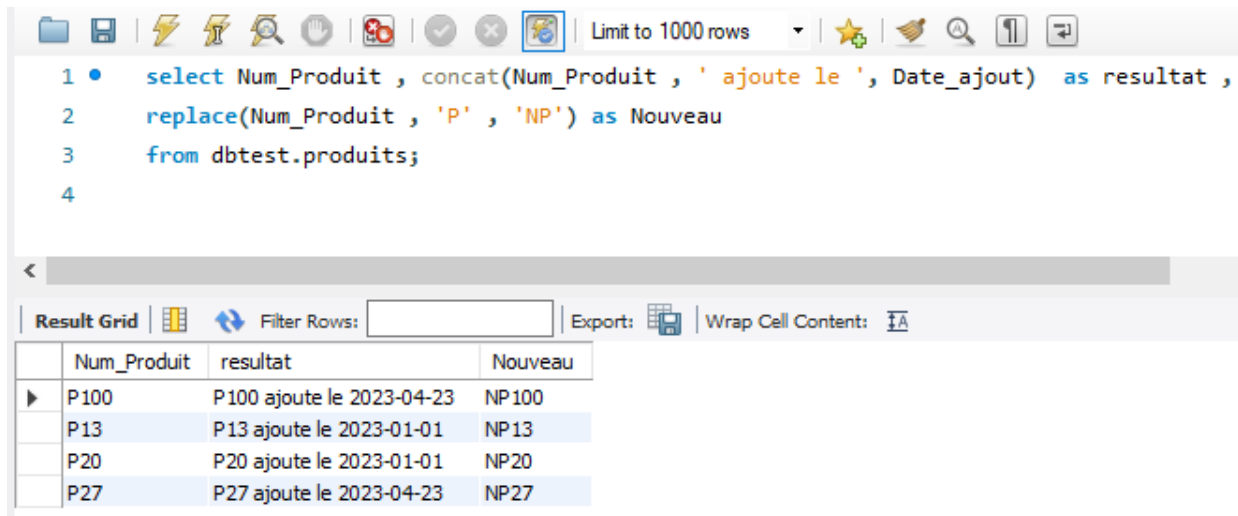
| Name            | Description                                                                                          |
|-----------------|------------------------------------------------------------------------------------------------------|
| CONCAT          | Concaténer deux ou plusieurs chaînes 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                                                                    |

# Expression du SGBD

## 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'.



The screenshot shows a MySQL query editor interface. The query is as follows:

```
1 • select Num_Produit , concat(Num_Produit , ' ajoute le ', Date_ajout) as resultat ,
2 replace(Num_Produit , 'P' , 'NP') as Nouveau
3 from dbtest.produits;
4
```

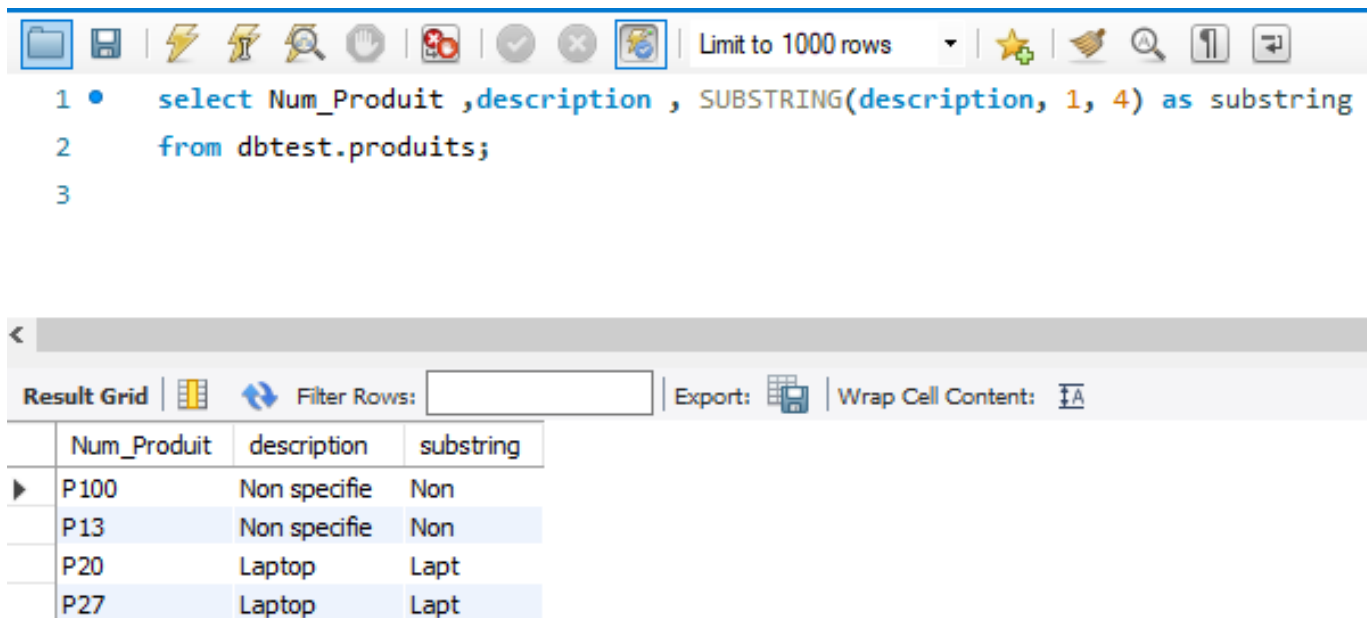
Below the query, the results are displayed in a table with the following columns: Num\_Produit, resultat, and Nouveau. The table contains four rows of data:

|   | Num_Produit | resultat                  | Nouveau |
|---|-------------|---------------------------|---------|
| ▶ | P100        | P100 ajoute le 2023-04-23 | NP100   |
|   | P13         | P13 ajoute le 2023-01-01  | NP13    |
|   | P20         | P20 ajoute le 2023-01-01  | NP20    |
|   | P27         | P27 ajoute le 2023-04-23  | NP27    |

# Expression du SGBD

## Les fonctions intégrées **MYSQL** : Fonctions de traitement de chaînes

### Exemple :SUBSTRING



The screenshot shows a MySQL query editor interface. The top toolbar includes icons for file operations, execution, and search. The query text is as follows:

```
1 • select Num_Produit ,description , SUBSTRING(description, 1, 4) as substring
2 from dbtest.produits;
3
```

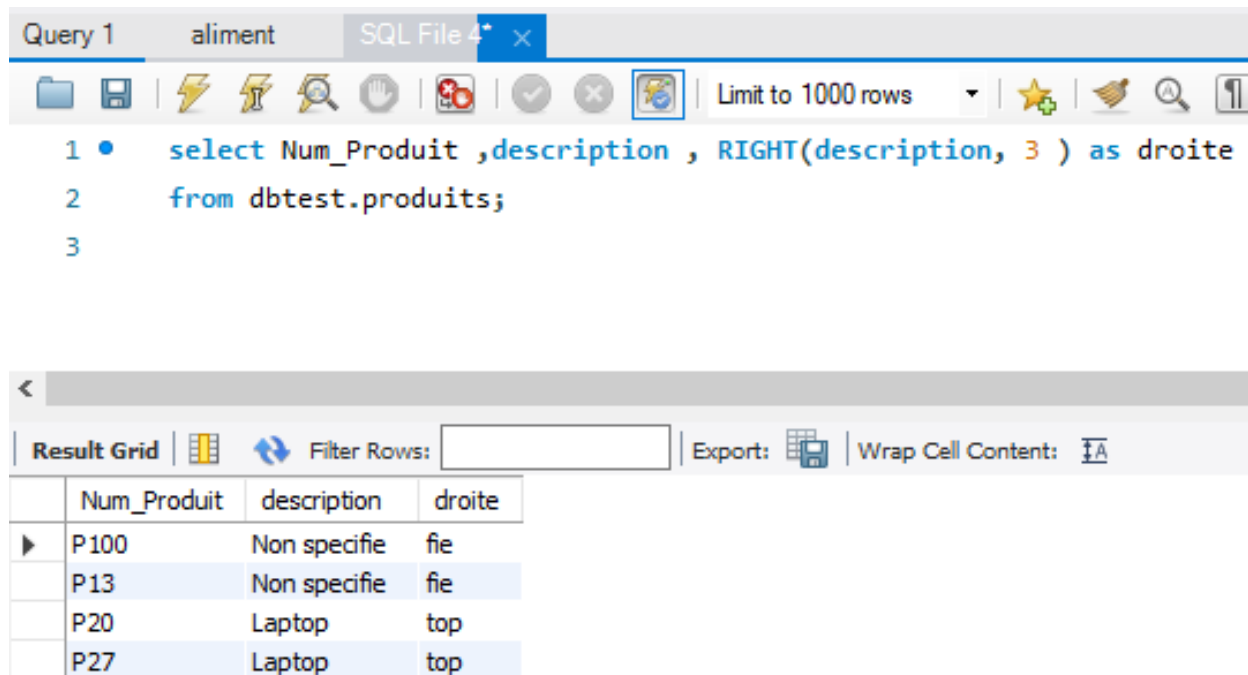
Below the query editor, the 'Result Grid' is displayed. It includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The result grid contains the following data:

|   | Num_Produit | description | substring |
|---|-------------|-------------|-----------|
| ▶ | P100        | Non specife | Non       |
|   | P13         | Non specife | Non       |
|   | P20         | Laptop      | Lapt      |
|   | P27         | Laptop      | Lapt      |

# Expression du SGBD

## Les fonctions intégrées MySQL : Fonctions de traitement de chaînes

### Exemple : RIGHT



The screenshot shows a MySQL query editor window with a tab labeled 'Query 1'. The query is as follows:

```
1 • select Num_Produit ,description , RIGHT(description, 3) as droite
2 from dbtest.produits;
3
```

Below the query editor, the 'Result Grid' is displayed, showing the results of the query. The grid has four columns: 'Num\_Produit', 'description', and 'droite'. The first two rows are highlighted in blue.

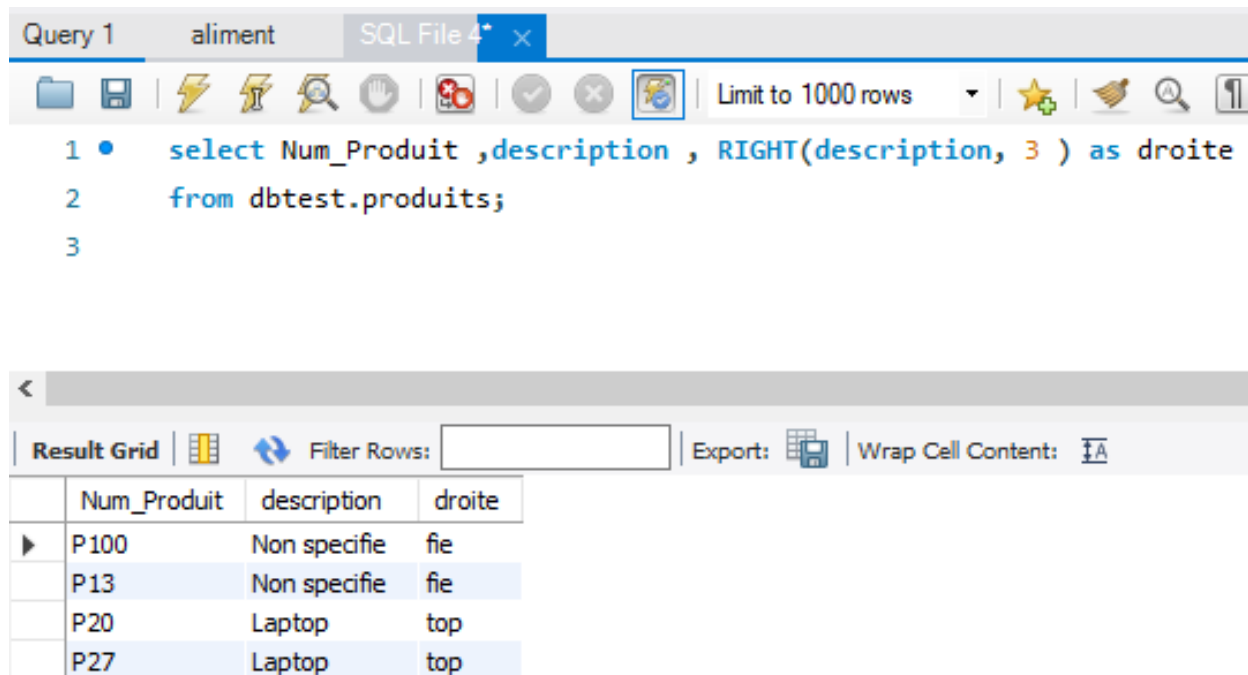
|   | Num_Produit | description | droite |
|---|-------------|-------------|--------|
| ▶ | P100        | Non specfie | fie    |
|   | P13         | Non specfie | fie    |
|   | P20         | Laptop      | top    |
|   | P27         | Laptop      | top    |



# Expression du SGBD

## Les fonctions intégrées MySQL : Fonctions de traitement de chaînes

### Exemple : RIGHT



The screenshot shows a MySQL query editor window with the following components:

- Query Editor:** Contains the SQL query:

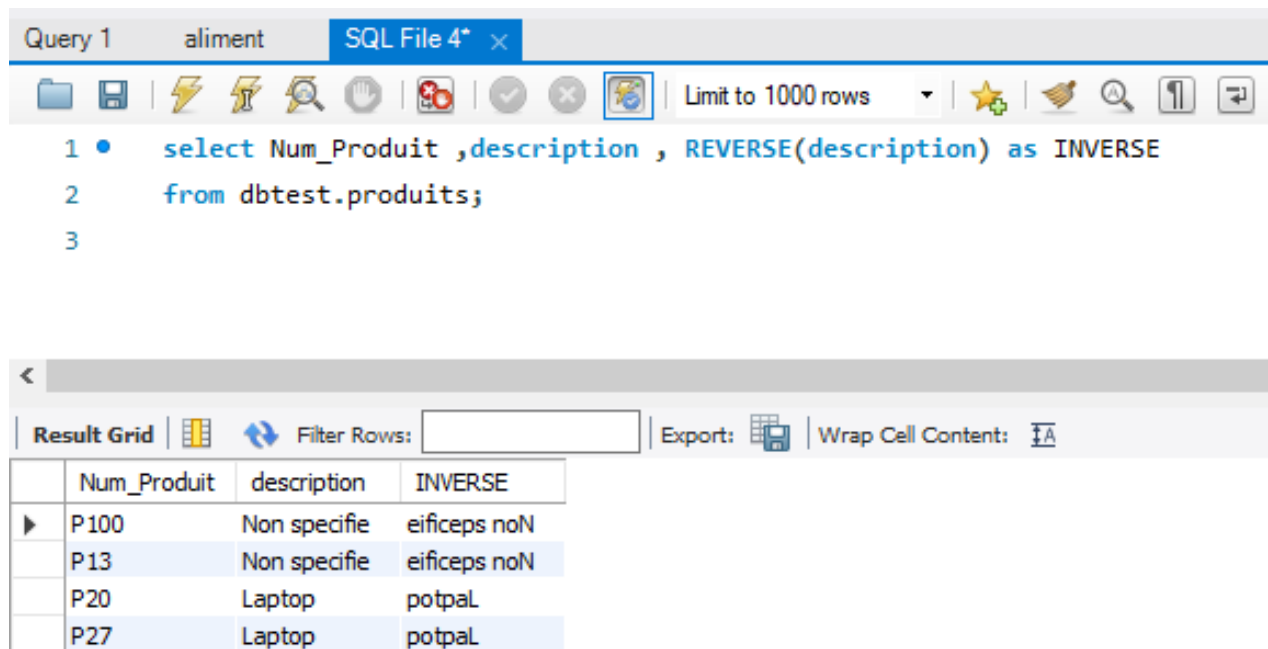
```
1 • select Num_Produit ,description , RIGHT(description, 3) as droite
2 from dbtest.produits;
3
```
- Toolbar:** Includes icons for file operations, execution, and a dropdown menu set to "Limit to 1000 rows".
- Result Grid:** Displays the query results in a table with 4 columns: Num\_Produit, description, and droite. The first two rows are highlighted in blue.

|   | Num_Produit | description | droite |
|---|-------------|-------------|--------|
| ▶ | P100        | Non specfie | fie    |
|   | P13         | Non specfie | fie    |
|   | P20         | Laptop      | top    |
|   | P27         | Laptop      | top    |

# Expression du SGBD

## Les fonctions intégrées **MYSQL** : Fonctions de traitement de chaînes

### Exemple : **REVERSE**



The screenshot shows a MySQL query editor window with a tab labeled "aliment" and "SQL File 4\* x". The query is as follows:

```
1 • select Num_Produit ,description , REVERSE(description) as INVERSE
2 from dbtest.produits;
3
```

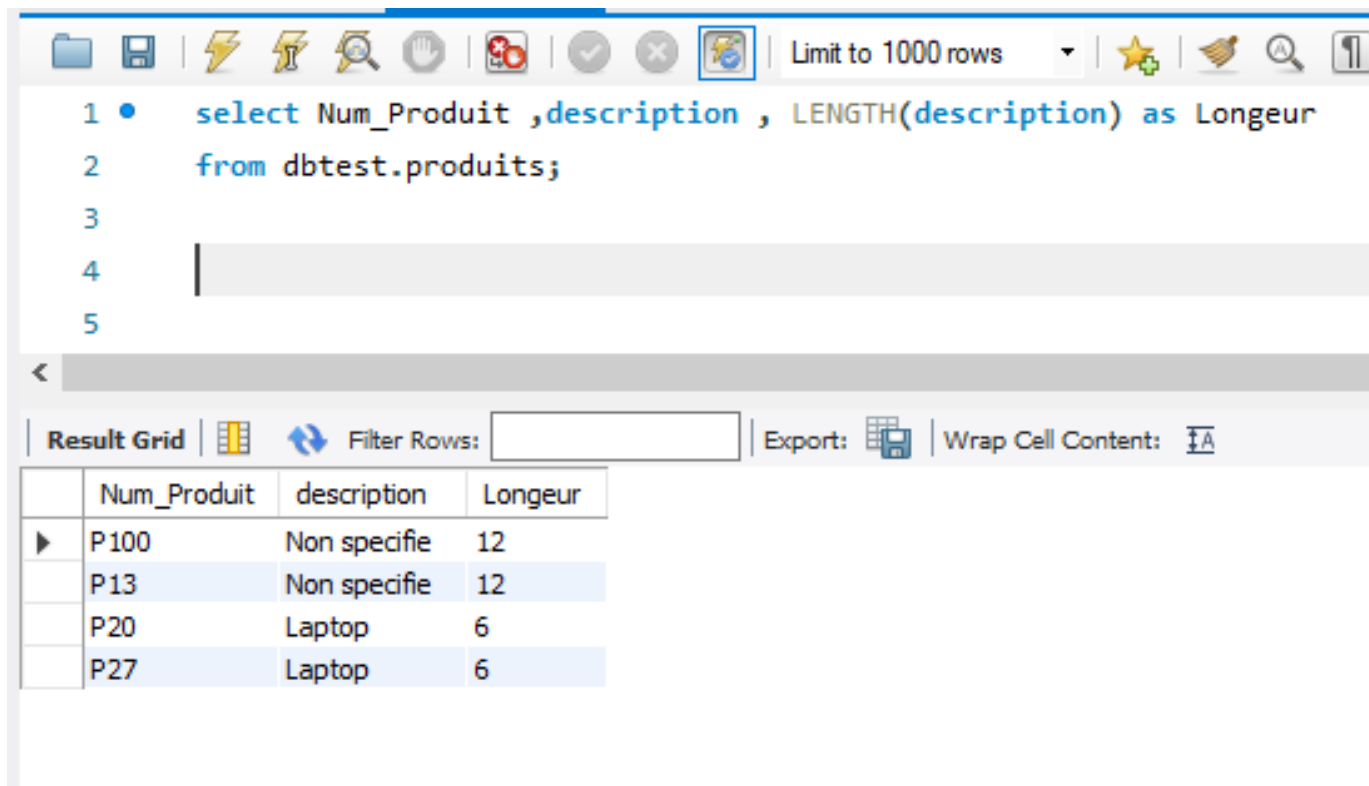
Below the query editor, the "Result Grid" is displayed, showing the results of the query. The grid has four columns: Num\_Produit, description, and INVERSE. The results are as follows:

|   | Num_Produit | description | INVERSE     |
|---|-------------|-------------|-------------|
| ▶ | P100        | Non specife | eficeps noN |
|   | P13         | Non specife | eficeps noN |
|   | P20         | Laptop      | potpaL      |
|   | P27         | Laptop      | potpaL      |

# Expression du SGBD

## Les fonctions intégrées **MYSQL** : Fonctions de traitement de chaînes

### Exemple : **LENGTH**



The screenshot shows a MySQL IDE interface. The top toolbar includes icons for file operations, execution, and search, along with a dropdown menu set to "Limit to 1000 rows". The SQL editor contains the following query:

```
1 • select Num_Produit ,description , LENGTH(description) as Longueur
2 from dbtest.produits;
3
4
5
```

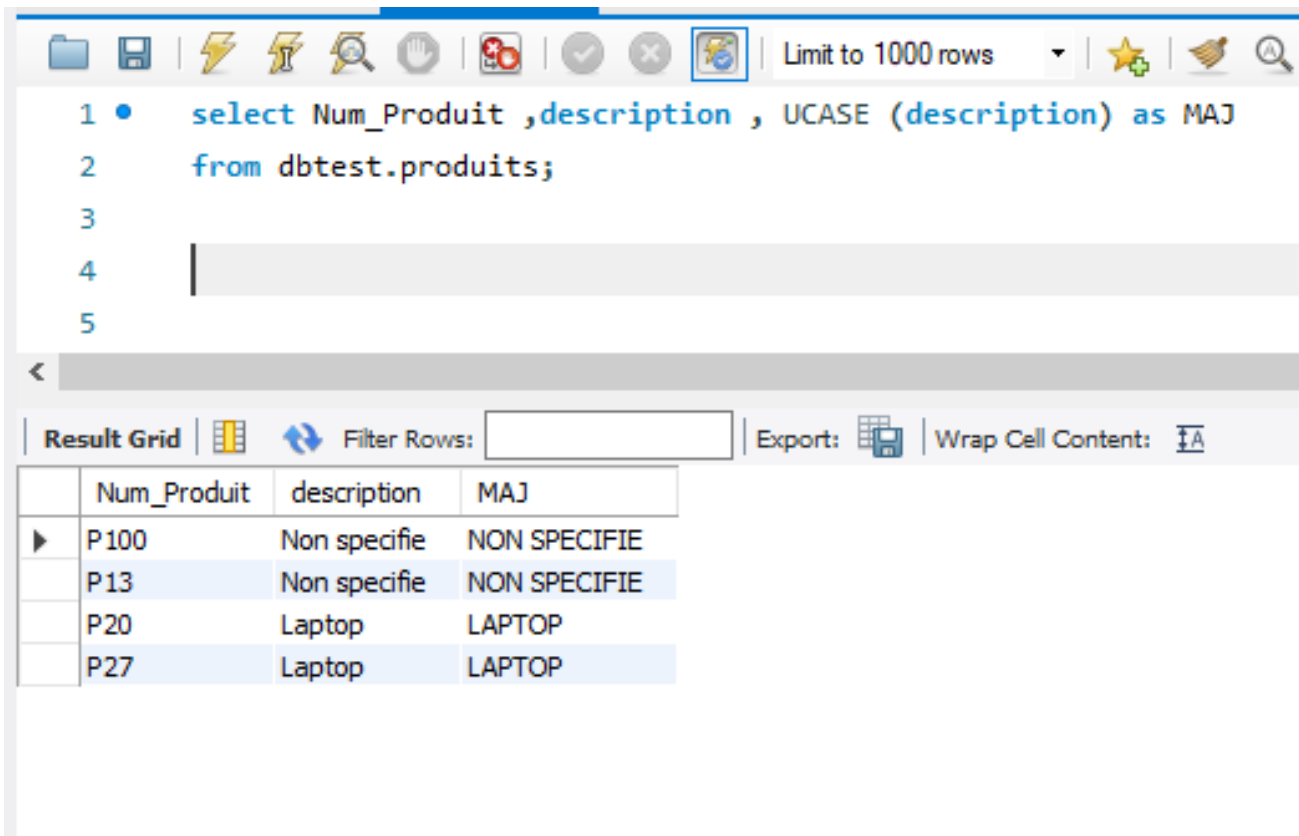
Below the editor, the "Result Grid" tab is active, displaying the query results in a table. The table has four columns: Num\_Produit, description, and Longueur. The first two rows (P100 and P13) are highlighted in blue. The table contains the following data:

|   | Num_Produit | description | Longueur |
|---|-------------|-------------|----------|
| ▶ | P100        | Non specife | 12       |
|   | P13         | Non specife | 12       |
|   | P20         | Laptop      | 6        |
|   | P27         | Laptop      | 6        |

# Expression du SGBD

## Les fonctions intégrées MYSQL : Fonctions de traitement de chaînes

### Exemple : UCASE



The screenshot shows a MySQL query editor interface. The query entered is:

```
1 • select Num_Produit ,description , UCASE (description) as MAJ
2 from dbtest.produits;
3
4
5
```

Below the query editor, the results are displayed in a table. The table has four columns: Num\_Produit, description, and MAJ. The first two columns are visible, and the third column (MAJ) contains the uppercase version of the description.

| Num_Produit | description | MAJ          |
|-------------|-------------|--------------|
| P100        | Non specife | NON SPECIFIE |
| P13         | Non specife | NON SPECIFIE |
| P20         | Laptop      | LAPTOP       |
| P27         | Laptop      | LAPTOP       |

# Expression du SGBD

## 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 :

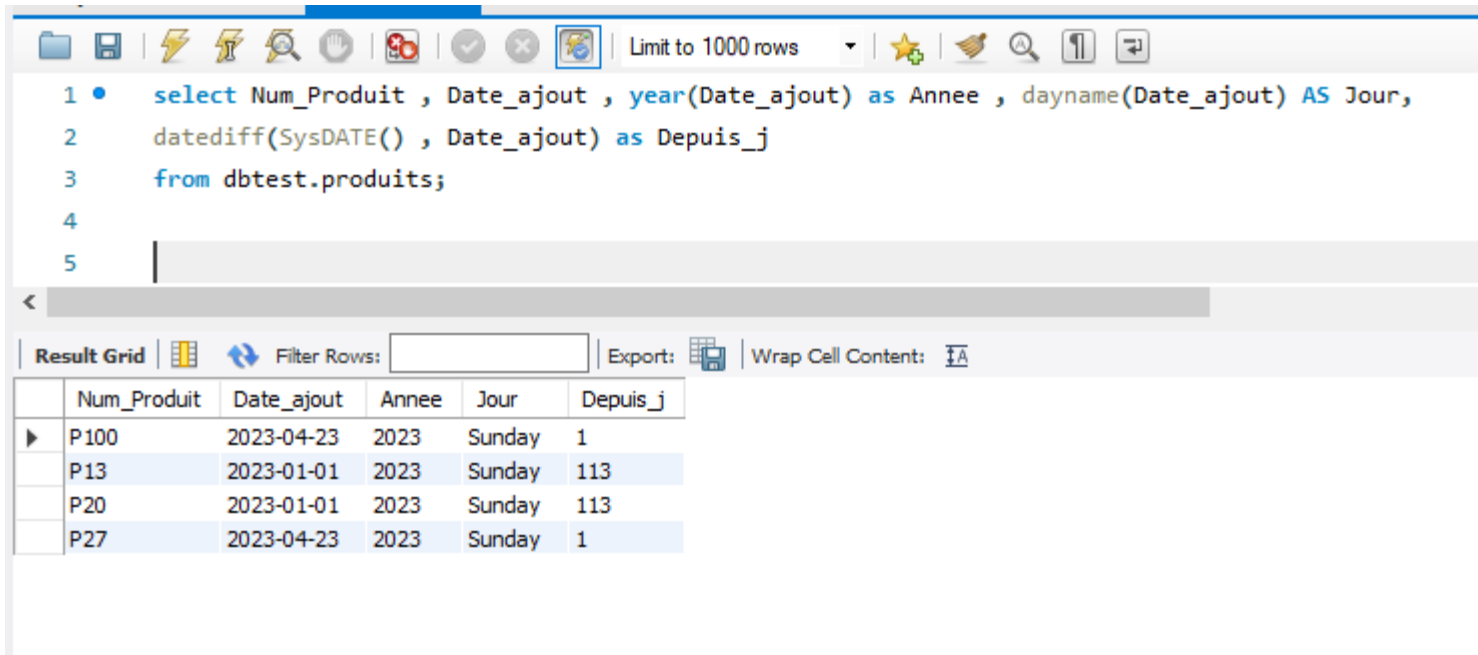
| Name          | Description                                                                         |
|---------------|-------------------------------------------------------------------------------------|
| CURDATE       | Renvoie la date actuelle.                                                           |
| 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                                             |

# Expression du SGBD

## 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.



The screenshot shows a MySQL query editor interface. The query is as follows:

```
1 • select Num_Produit , Date_ajout , year(Date_ajout) as Annee , dayname(Date_ajout) AS Jour,
2 datediff(SysDATE() , Date_ajout) as Depuis_j
3 from dbtest.produits;
4
5
```

Below the query editor, the results are displayed in a table. The table has the following columns: Num\_Produit, Date\_ajout, Annee, Jour, and Depuis\_j. The results are as follows:

|   | Num_Produit | Date_ajout | Annee | Jour   | Depuis_j |
|---|-------------|------------|-------|--------|----------|
| ▶ | P100        | 2023-04-23 | 2023  | Sunday | 1        |
|   | P13         | 2023-01-01 | 2023  | Sunday | 113      |
|   | P20         | 2023-01-01 | 2023  | Sunday | 113      |
|   | P27         | 2023-04-23 | 2023  | Sunday | 1        |

# Expression du SGBD

## Les fonctions intégrées MySQL : Fonctions de manipulation de dates

### Exemple DATE\_FORMAT():

- Syntaxe :

```
SELECT DATE_FORMAT(date, format)
```

```
SELECT DATE_FORMAT("2018-09-24", "%D %b %Y");
```

```
-- résultat : "24th Sep 2018"
```

```
SELECT DATE_FORMAT("2018-09-24", "%M %d %Y");
```

```
-- résultat : "September 24 2018"
```

```
SELECT DATE_FORMAT("2018-09-24 22:21:20", "%W %M %e %Y");
```

```
-- résultat : "Monday September 24 2018"
```

```
SELECT DATE_FORMAT("2018-09-24", "%d/%m/%Y");
```

```
-- résultat : "24/09/2018"
```

```
SELECT DATE_FORMAT("2018-09-24", "Message du : %d/%m/%Y");
```

```
-- résultat : "Message du : 24/09/2018"
```

```
SELECT DATE_FORMAT("2018-09-24 22:21:20", "%H:%i:%s");
```

```
-- résultat : "22:21:20"
```

# Expression du SGBD

## Les fonctions intégrées MySQL : Fonctions de manipulation de dates

### Exemple DATEDIFF()

- Syntaxe : `SELECT DATEDIFF( date1, date2 );`

```
SELECT DATEDIFF('2014-01-09', '2014-01-01'); -- retourne : 8
```

```
SELECT DATEDIFF('2014-01-09 00:00:00', '2014-01-01 00:00:00'); -- retourne : 8
```

```
SELECT DATEDIFF('2014-01-01', '2014-01-09'); -- retourne : -8
```

```
SELECT DATEDIFF('2015-01-01', '2014-01-01'); -- retourne : 365
```

```
SELECT DATEDIFF('2014-01-02', '2014-01-01'); -- retourne : 1
```

```
SELECT DATEDIFF('2014-01-02 23:59:59', '2014-01-01'); -- retourne : 1
```



# Expression du SGBD

## Les fonctions intégrées MySQL : Fonctions de manipulation de dates

### Exemple DAYOFWEEK()

- Syntaxe :

```
SELECT DAYOFWEEK(colonne_date)
FROM test;
```

```
SELECT DAYOFWEEK(NOW());
```

```
SELECT DAYOFWEEK('2018-09-15'); -- Résultat : 7 => Le 15 septembre 2018 est un samedi (samedi = 7).
```

# Expression du SGBD

## Les fonctions intégrées MySQL : Fonctions de manipulation de dates

### Exemple TIMEDIFF()

- Syntaxe : `SELECT TIMEDIFF(heure1, heure2);`

```
SELECT TIMEDIFF("HH:MM", "HH:MM"); -- format de la réponse : "HH:MM"
SELECT TIMEDIFF("HH:MM:SS", "HH:MM:SS"); -- format de la réponse : "HH:MM:SS"
SELECT TIMEDIFF("YYYY-MM-DD HH:MM", "YYYY-MM-DD HH:MM"); -- format de la réponse : "HH:MM"
SELECT TIMEDIFF("YYYY-MM-DD HH:MM:SS", "YYYY-MM-DD HH:MM:SS"); -- format de la réponse : "HH:MM:SS"

SELECT TIMEDIFF("10:12:13", "10:11:12"); -- résultat : 00:01:01
SELECT TIMEDIFF("10:11:12", "10:12:13"); -- résultat : -00:01:01
SELECT TIMEDIFF("10:15", "10:05"); -- résultat : 00:10
SELECT TIMEDIFF("2018-09-16 10:12:13", "2018-09-13 10:11:12"); -- résultat : 72:01:01
```

# Expression du SGBD

## 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(chaine,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

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

# Fonctions d'agrégation du SGBD

- 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.

# Fonctions d'agrégation du SGBD

## 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     |



$\Sigma$

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

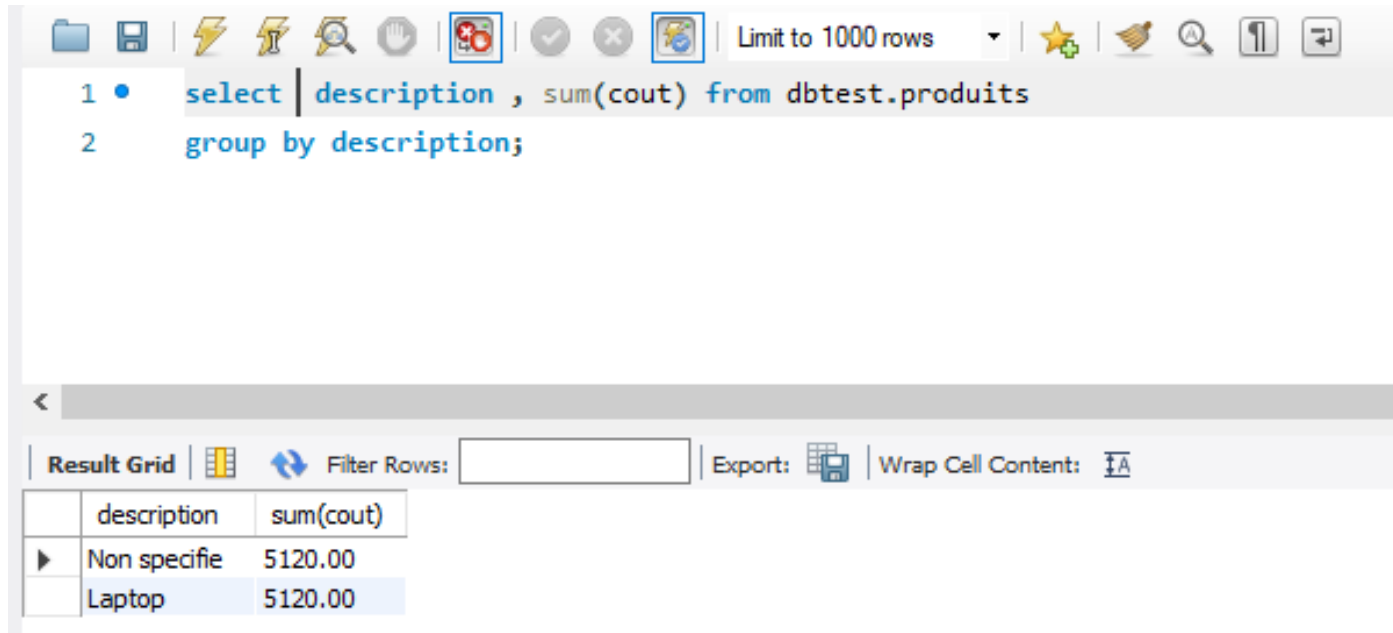


| Nom | SUM (Valeur) |
|-----|--------------|
| A   | 30           |
| B   | 40           |
| C   | 70           |

# Fonctions d'agrégation du SGBD

La fonction **SUM()** :

• Exemple :



The screenshot shows a database query editor interface. At the top, there is a toolbar with various icons for file operations, execution, and viewing. Below the toolbar, the SQL query is entered in a text area:

```
1 • select description , sum(cout) from dbtest.produits
2 group by description;
```

Below the query editor, there is a section for the results. It includes a "Result Grid" tab, a "Filter Rows" input field, and an "Export" button. The result grid displays the following data:

|   | description | sum(cout) |
|---|-------------|-----------|
| ▶ | Non specife | 5120.00   |
|   | Laptop      | 5120.00   |

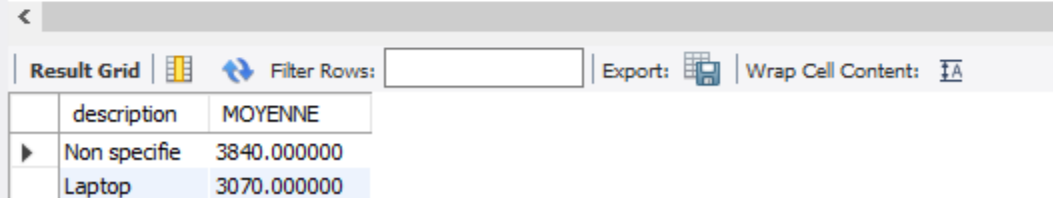
# Fonctions d'agrégation du SGBD

## La fonction AVG() :

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

- **Exemple :**

```
2 • select description , AVG(prix) as MOYENNE from dbtest.produits
3 group by description;
```



The screenshot shows a database query interface. At the top, there is a text area containing the SQL query: `select description , AVG(prix) as MOYENNE from dbtest.produits group by description;`. Below the text area is a toolbar with icons for 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. The 'Result Grid' icon is active. Below the toolbar is a table with the following data:

|   | description  | MOYENNE     |
|---|--------------|-------------|
| ▶ | Non specifié | 3840.000000 |
|   | Laptop       | 3070.000000 |



# Fonctions d'agrégation du SGBD

## La fonction **COUNT()** :

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

```
2 • select description , count(Num_produit) from dbtest.produits
3 group by description;
4
5
6
7
8
9
```

| Result Grid |             | Filter Rows:       | Export: | Wrap Cell Content: |
|-------------|-------------|--------------------|---------|--------------------|
|             | description | count(Num_produit) |         |                    |
| ▶           | Non specife | 2                  |         |                    |
|             | Laptop      | 2                  |         |                    |

# Fonctions d'agrégation du SGBD

## La fonction MAX/MIN ::

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

```
1
2 • select max(prix) from dbtest.produits ;
3
4
5
```

Result Grid

|   | max(prix) |
|---|-----------|
| ▶ | 7500.00   |

Limit to 1000

```
1
2 • select min(prix) from dbtest.produits ;
3
4
5
```

Result Grid

|   | min(prix) |
|---|-----------|
| ▶ | 140.00    |

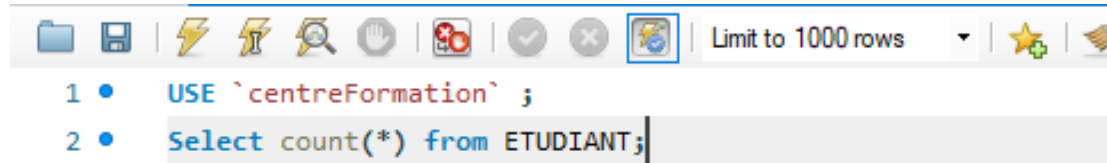
# Exercice 6 : « Centre de Formation »

1. Combien y a-t'il d'étudiants ?
2. Donner l'âge de chacun des étudiants (utiliser les fonctions comme : `datediff()`, `CURDATE()`, `timestampdiff()`, `now()`...)
3. Quelle est la plus chère des formations ? et la moins chère ?
4. Si un étudiant est inscrit dans toutes les formations, combien il doit payer ?
5. Donner le nombre des étudiants inscrits dans chacune des sessions
6. Donner la liste des numéros CIN des étudiants qui sont inscrits au moins une fois.
7. Donner pour chacun des étudiants le nombre d'inscriptions.
8. Donner pour chaque session le nombre d'inscriptions distantielles et présentiels



# Solution exercice 6 : « Centre de Formation »

## 1. Combien y a-t'il d'étudiants ?



The screenshot shows a toolbar with various icons for file operations, execution, and settings. The code area contains two lines of SQL code:

```
1 • USE `centreFormation` ;
2 • Select count(*) from ETUDIANT;
```

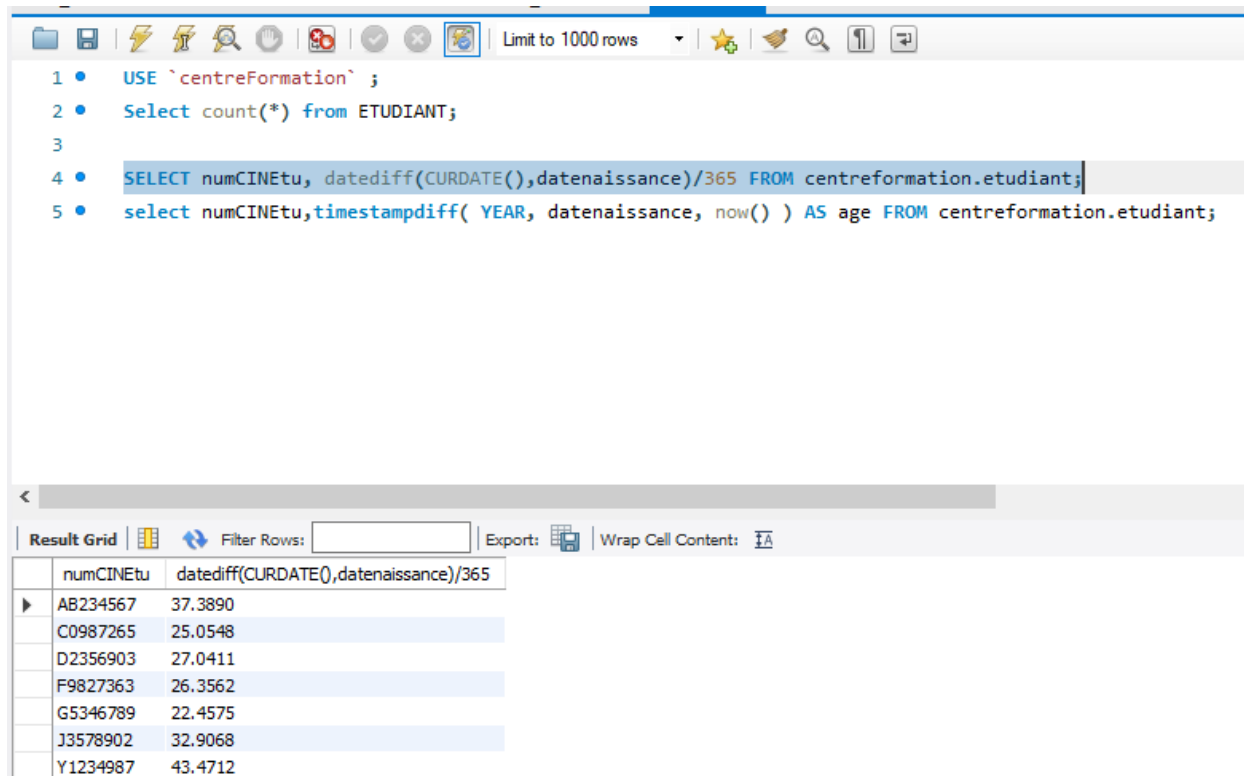


The screenshot shows a result grid with a single row and column. The column header is 'count(\*)' and the value is 7.

| count(*) |
|----------|
| 7        |

# Solution exercice 6 : « Centre de Formation »

2. Donner l'âge de chacun des étudiants (utiliser les fonctions comme :  
**datediff()**, **CURDATE()**, **timestampdiff()**, **now()**...)



The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
1 • USE `centreFormation` ;
2 • Select count(*) from ETUDIANT;
3
4 • SELECT numCINETu, datediff(CURDATE(),datenaissance)/365 FROM centreformation.etudiant;
5 • select numCINETu,timestampdiff(YEAR, datenaissance, now()) AS age FROM centreformation.etudiant;
```

The result grid at the bottom displays the output of the query. It has two columns: **numCINETu** and **datediff(CURDATE(),datenaissance)/365**. The data rows are as follows:

| numCINETu | datediff(CURDATE(),datenaissance)/365 |
|-----------|---------------------------------------|
| AB234567  | 37.3890                               |
| C0987265  | 25.0548                               |
| D2356903  | 27.0411                               |
| F9827363  | 26.3562                               |
| G5346789  | 22.4575                               |
| J3578902  | 32.9068                               |
| Y1234987  | 43.4712                               |

## Solution exercice 6 : « Centre de Formation »

2. Donner l'âge de chacun des étudiants (utiliser les fonctions comme :  
datediff(), CURDATE(), timestampdiff(), now()...)

```
5 • select numCINETu,timestampdiff(YEAR, datenaissance, now()) AS age FROM centreformation.etudiant;
```

| Result Grid |     | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|-----|--------------|---------|--------------------|
| numCINETu   | age |              |         |                    |
| AB234567    | 37  |              |         |                    |
| C0987265    | 25  |              |         |                    |
| D2356903    | 27  |              |         |                    |
| F9827363    | 26  |              |         |                    |
| G5346789    | 22  |              |         |                    |
| J3578902    | 32  |              |         |                    |
| Y1234987    | 43  |              |         |                    |

## Solution exercice 6 : « Centre de Formation »

### 3. Quelle est la plus chère des formations ? et la moins chère ?

```
8 • SELECT codeForm, titreForm, Max(prixForm) FROM centreformation.formation GROUP BY codeForm, titreForm;
9 • select codeForm, titreForm, Min(prixForm) FROM centreformation.formation GROUP BY codeForm, titreForm;
```

10

11

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

|   | codeForm | titreForm             | Max(prixForm) |
|---|----------|-----------------------|---------------|
| ▶ | 15       | Base de données Orade | 6000          |
|   | 12       | web developpment      | 4200          |
|   | 13       | Anglais technique     | 3750          |
|   | 11       | Programming Java      | 3600          |
|   | 16       | Soft skills           | 3000          |
|   | 14       | Communication         | 2500          |

# Solution exercice 6 : « Centre de Formation »

3. Quelle est la plus chère des formations ? et la moins chère ?

```
9 • select codeForm, titreForm, Min(prixForm) FROM centreformation.formation GROUP BY codeForm, titreForm;
```

|   | codeForm | titreForm              | Min(prixForm) |
|---|----------|------------------------|---------------|
| ▶ | 14       | Communication          | 2500          |
|   | 16       | Soft skills            | 3000          |
|   | 11       | Programming Java       | 3600          |
|   | 13       | Anglais technique      | 3750          |
|   | 12       | web developpment       | 4200          |
|   | 15       | Base de données Oracle | 6000          |






## Solution exercice 6 : « Centre de Formation »

4. Si un étudiant est inscrit dans toutes les formations, combien il doit payer ?

```
11 • SELECT SUM(prixForm) FROM centreformation.formation;
```

<



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

|   | SUM(prixForm) |
|---|---------------|
| ▶ | 23050         |

## Solution exercice 6 : « Centre de Formation »

5. Donner le nombre des étudiants inscrits dans chacune des sessions

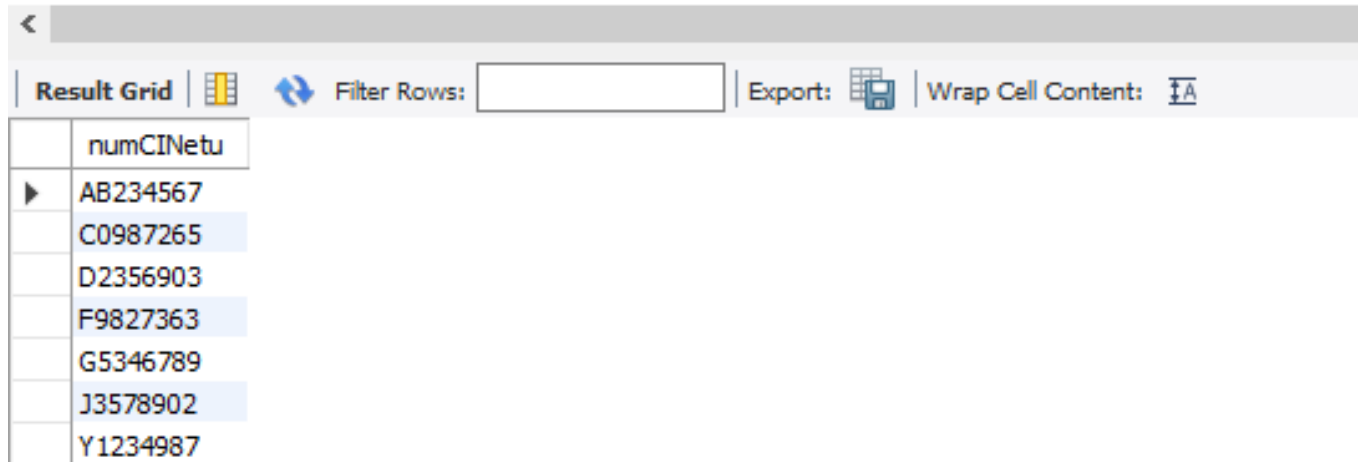
```
13 • SELECT codeSess, count(numCINetu) FROM centreformation.inscription
14 group by codeSess;
```

| <                                                                                                      |          |                  |
|--------------------------------------------------------------------------------------------------------|----------|------------------|
| Result Grid                                                                                            |          |                  |
| Filter Rows: <input type="text"/>                                                                      |          |                  |
| Export:             |          |                  |
| Wrap Cell Content:  |          |                  |
|                                                                                                        | codeSess | count(numCINetu) |
| ▶                                                                                                      | 1101     | 7                |
|                                                                                                        | 1201     | 6                |
|                                                                                                        | 1302     | 5                |
|                                                                                                        | 1401     | 6                |
|                                                                                                        | 1501     | 7                |

## Solution exercice 6 : « Centre de Formation »

6. Donner la liste des numéros CIN des étudiants qui sont inscrits au moins une fois.

```
16 • SELECT distinct numCINetu FROM centreformation.inscription ;
```






The screenshot shows a database query result interface. At the top, there is a toolbar with a back arrow, a 'Result Grid' button, a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' button. Below the toolbar is a table with one column labeled 'numCINetu'. The table contains eight rows of distinct student CIN numbers, each preceded by a small right-pointing triangle icon.

| numCINetu |
|-----------|
| AB234567  |
| C0987265  |
| D2356903  |
| F9827363  |
| G5346789  |
| J3578902  |
| Y1234987  |

## Solution exercice 6 : « Centre de Formation »

7. Donner pour chacun des étudiants le nombre d'inscriptions..

```
19 • SELECT numCINetu, count(codeSess) FROM centreformation.inscription
20 group by numCINetu ;
21
22
```

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

|   | numCINetu | count(codeSess) |
|---|-----------|-----------------|
|   | AB234567  | 4               |
|   | C0987265  | 5               |
|   | D2356903  | 5               |
|   | F9827363  | 3               |
| ▶ | G5346789  | 5               |
|   | J3578902  | 4               |
|   | Y1234987  | 5               |

## Solution exercice 6 : « Centre de Formation »

8. Donner pour chaque session le nombre d'inscriptions distantielles et présentielles

21

22 • `Select codeSess, typeCours, count(numinscription)`

23 `from inscription`

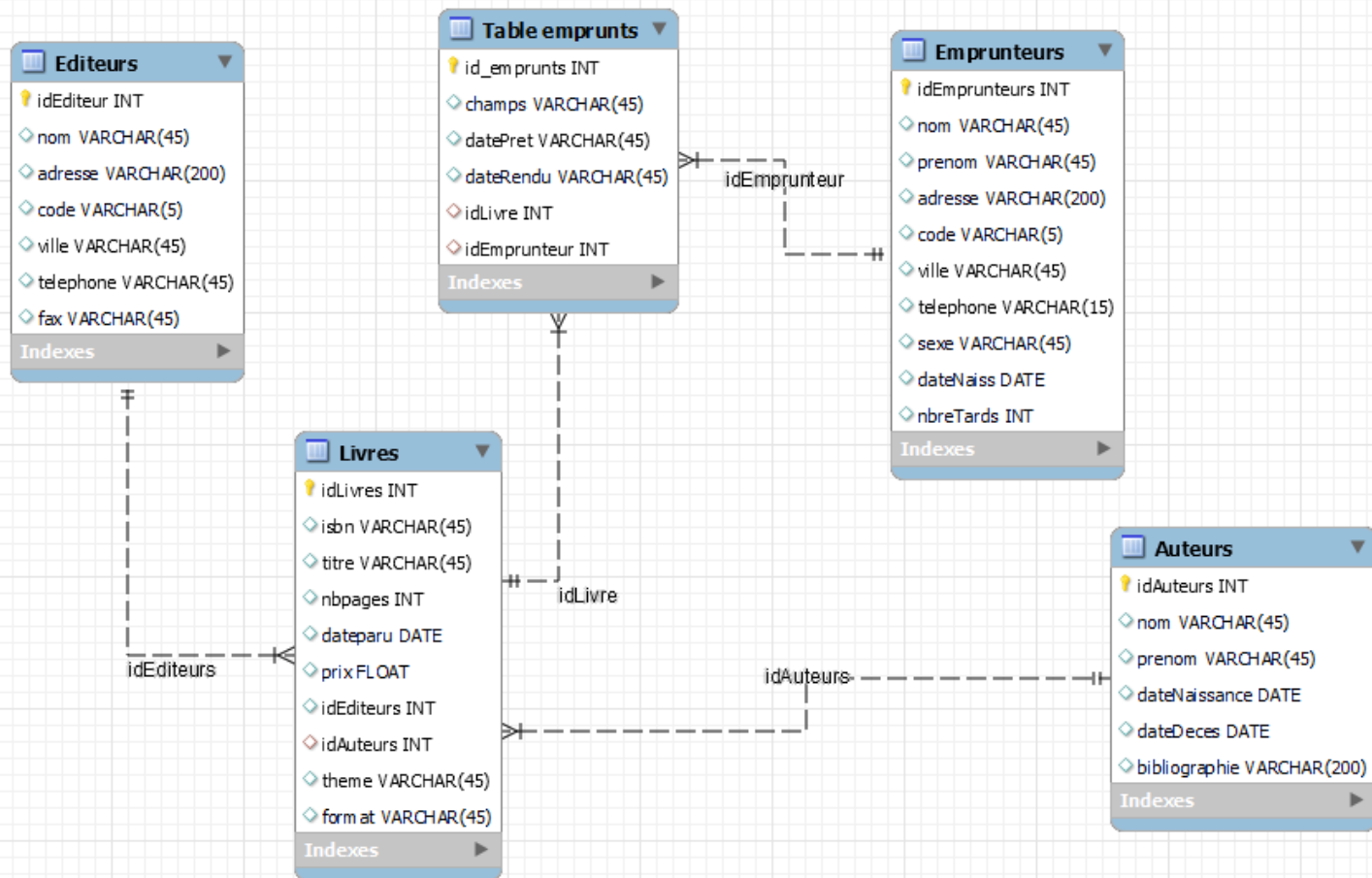
24 `group by codeSess,typeCours;`

|   | codeSess | typeCours  | count(numinscription) |
|---|----------|------------|-----------------------|
| ► | 1101     | Distanciel | 7                     |
|   | 1201     | Presenciel | 6                     |
|   | 1302     | Presenciel | 4                     |
|   | 1302     | Distanciel | 1                     |
|   | 1401     | Distanciel | 6                     |
|   | 1501     | Distanciel | 2                     |
|   | 1501     | Presenciel | 5                     |

Result 19 ×

# Exercice 7 : «Bibliothèque»

On considère le schéma relationnel suivant qui modélise une application sur la **gestion d'une Bibliothèque**



# Exercice 7 : «Bibliothèque»

1- Créez une base de données sous le nom **Bibliothèque**

2 - Créez les tables depuis le MLD : «**Bibliothèque**» (Ne pas oublier les clés primaires et étrangères)

3 – Exécutez le script **biblio-insertion.sql**

Créer les requêtes suivantes:

1 - Affichez tous les titres de livres contenant le mot "**conte**" ou « **livre** »

2 - On veut afficher des titres de livres qui contiennent à la fois "**conte**" et "**légende**"

3 - Affichez des livres dont le prix est supérieur à **20 euros** et qui ont été publiés après **2010** .

4- Afficher les titres de tous les livres dont le thème est "**Biographie**"

5 - Calculer le nombre moyen de pages de tous les livres

6 - Afficher les adresses complètes (**adresse, code postal et ville**) de tous les éditeurs

7 - Afficher les titres des livres publiés entre le **1er janvier 2010** et le **31 décembre 2015**

# Exercice 7 : «Bibliothèque» : suite

- 8- Afficher les noms des auteurs dont le nom commence par la lettre "J"
- 9 - Afficher les dates de naissance et de décès des auteurs, triées par ordre décroissant de date de décès
- 10- Afficher le nombre de livres empruntés par chaque emprunteur, en ordre décroissant du nombre de livres empruntés
- 11 - Afficher tous les éditeurs qui ont publié au moins 5 livres
- 12 - Afficher le nombre de livres empruntés par chaque emprunteur, mais seulement pour les emprunteurs qui ont emprunté plus de **3** livres
- 13 - Afficher le nombre d'emprunteurs qui ont emprunté au moins un livre **chaque mois de l'année en cours**
- 14 - Afficher le nombre de livres différents par auteur
- 15 - Afficher les auteurs ayant écrit plus de 3 livres
- 16 - Afficher les auteurs ayant écrit des livres dont le nombre de pages moyen est inférieur à 300 pages



# Solution Exercice 7 : «Bibliothèque»

1- Créez une base de données sous le nom **Bibliothèque**

```

-- Schema Bibliothèque

CREATE SCHEMA IF NOT EXISTS `Bibliothèque` DEFAULT CHARACTER SET utf8 ;
USE `Bibliothèque` ;
```

2 - Créez les tables depuis le MLD : «**Bibliothèque**» (Ne pas oublier les clés primaires et étrangères)

```

-- Table `Bibliothèque`.`Editeurs`

CREATE TABLE IF NOT EXISTS `Bibliothèque`.`Editeurs` (
 `idEditeur` INT NOT NULL,
 `nom` VARCHAR(45) NULL,
 `adresse` VARCHAR(200) NULL,
 `code` VARCHAR(5) NULL,
 `ville` VARCHAR(45) NULL,
 `telephone` VARCHAR(45) NULL,
 `fax` VARCHAR(45) NULL,
 PRIMARY KEY (`idEditeur`))
ENGINE = InnoDB;
```

# Solution Exercice 7 : «Bibliothèque»

2 - Créez les tables depuis le MLD : «Bibliothèque» (Ne pas oublier les clés primaires et étrangères

```
-- Table `Bibliothèque`.`Auteurs`
```

```
CREATE TABLE IF NOT EXISTS `Bibliothèque`.`Auteurs`
 `idAuteurs` INT NOT NULL,
 `nom` VARCHAR(45) NULL,
 `prenom` VARCHAR(45) NULL,
 `dateNaissance` DATE NULL,
 `dateDeces` DATE NULL,
 `bibliographie` VARCHAR(200) NULL,
 PRIMARY KEY (`idAuteurs`))
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `Bibliothèque`.`Livres` (
 `idLivres` INT NOT NULL,
 `isbn` VARCHAR(45) NULL,
 `titre` VARCHAR(45) NULL,
 `nbpages` INT NULL,
 `dateparu` DATE NULL,
 `prix` FLOAT NULL,
 `idEditeurs` INT NULL,
 `idAuteurs` INT NULL,
 `theme` VARCHAR(45) NULL,
 `format` VARCHAR(45) NULL,
 PRIMARY KEY (`idLivres`),
 INDEX `idEditeurs_idx` (`idAuteurs` ASC) VISIBLE,
 CONSTRAINT `idEditeurs`
 FOREIGN KEY (`idAuteurs`)
 REFERENCES `Bibliothèque`.`Editeurs` (`idEditeur`),
 CONSTRAINT `idAuteurs`
 FOREIGN KEY (`idAuteurs`)
 REFERENCES `Bibliothèque`.`Auteurs` (`idAuteurs`))
ENGINE = InnoDB;
```

# Solution Exercice 7 : «Bibliothèque»

2 - Créez les tables depuis le MLD : «Bibliothèque» (Ne pas oublier les clés primaires et étrangères)

```
-- Table `Bibliothèque`.`Emprunteurs`
```

```
CREATE TABLE IF NOT EXISTS `Bibliothèque`.`Emprunteurs` (
 `idEmprunteurs` INT NOT NULL,
 `nom` VARCHAR(45) NULL,
 `prenom` VARCHAR(45) NULL,
 `adresse` VARCHAR(200) NULL,
 `code` VARCHAR(5) NULL,
 `ville` VARCHAR(45) NULL,
 `telephone` VARCHAR(15) NULL,
 `sexe` VARCHAR(45) NULL,
 `dateNaiss` DATE NULL,
 `nbreTards` INT NULL,
 PRIMARY KEY (`idEmprunteurs`))
ENGINE = InnoDB;
```

```
98 -----
99 • CREATE TABLE IF NOT EXISTS `Bibliothèque`.`emprunts` (
100 `id_emprunts` INT NOT NULL,
101 `champs` VARCHAR(45) NULL,
102 `datePret` VARCHAR(45) NULL,
103 `dateRendu` VARCHAR(45) NULL,
104 `idLivre` INT NULL,
105 `idEmprunteur` INT NULL,
106 PRIMARY KEY (`id_emprunts`),
107 INDEX `idLivre_idx` (`idLivre` ASC) VISIBLE,
108 INDEX `idEmprunteur_idx` (`idEmprunteur` ASC) VISIBLE,
109 CONSTRAINT `idLivre`
110 FOREIGN KEY (`idLivre`)
111 REFERENCES `Bibliothèque`.`Livres` (`idLivres`)
112 ON DELETE NO ACTION
113 ON UPDATE NO ACTION,
114 CONSTRAINT `idEmprunteur`
115 FOREIGN KEY (`idEmprunteur`)
116 REFERENCES `Bibliothèque`.`Emprunteurs` (`idEmprunteurs`)
117 ON DELETE NO ACTION
118 ON UPDATE NO ACTION)
119 ENGINE = InnoDB;
```

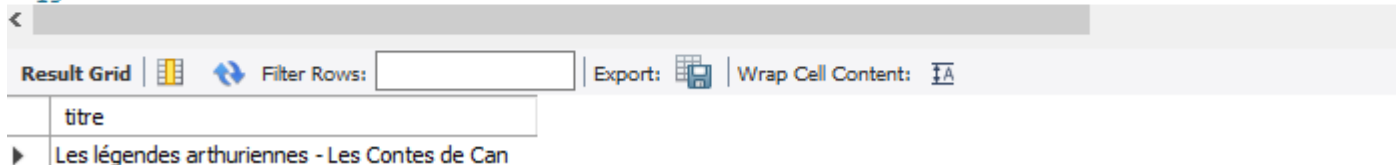
# Solution Exercice 7 : «Bibliothèque»

1 - Affichez tous les titres de livres contenant le mot "**conte**" ou « **livre** »

```
1 -- 0 - db
2 • use bibliothèque;
3
4 -- 1 - Sélection de tous les titres de livres contenant le mot "conte" ou "livre"
5 • SELECT titre FROM Livres WHERE titre LIKE '%conte%' OR titre LIKE '%livre%';
6
```

2 - On veut afficher des titres de livres qui contiennent à la fois "**conte**" et "**légende**"

```
10 -- 2 - Sélection des titres de livres qui contiennent à la fois "conte" et "légende"
11 • SELECT titre FROM Livres WHERE titre LIKE '%conte%' AND titre LIKE '%légende%';
12
13
14
15
```



The screenshot shows a database application interface. At the top, there is a toolbar with icons for 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. Below the toolbar, a table with one row is displayed. The table has a column header 'titre' and a single data row containing the text 'Les légendes arthuriennes - Les Contes de Can'.

| titre                                         |
|-----------------------------------------------|
| Les légendes arthuriennes - Les Contes de Can |

## Solution Exercice 7 : «Bibliothèque»

3 - Affichez des livres dont le prix est supérieur à **20 euros** et qui ont été publiés après **2010**.

18 -- 3 - Sélection des livres dont le prix est supérieur à 20 euros et qui ont été publiés après 2010

```
19 • SELECT * FROM Livres WHERE prix > 20 AND dateparu > '2010-12-31';
```

20

[illegible]

# Solution Exercice 7 : «Bibliothèque»

4- Afficher les titres de tous les livres dont le thème est "Biographie«

```
30 -- 4 - Afficher les titres de tous les livres dont le thème est "Biographie"
31 • SELECT titre
32 FROM Livres
33 WHERE theme = 'Biographie';
34
```

|                                               |  |              |         |                    |
|-----------------------------------------------|--|--------------|---------|--------------------|
| Result Grid                                   |  | Filter Rows: | Export: | Wrap Cell Content: |
| titre                                         |  |              |         |                    |
| Histoire de la vie                            |  |              |         |                    |
| Les légendes arthuriennes - Les Contes de Can |  |              |         |                    |

5 - Calculer le nombre moyen de pages de tous les livres

```
38 -- 5 - Calculer le nombre moyen de pages de tous les livres
39 • SELECT AVG(nbpages) AS nb_pages_moyen
40 FROM Livres;
41
```

|                |  |              |         |                    |
|----------------|--|--------------|---------|--------------------|
| Result Grid    |  | Filter Rows: | Export: | Wrap Cell Content: |
| nb_pages_moyen |  |              |         |                    |
| 527.0000       |  |              |         |                    |

# Solution Exercice 7 : «Bibliothèque»

6 - Afficher les adresses complètes (**adresse, code postal et ville**) de tous les éditeurs

```
48 -- 6 - Afficher les adresses complètes (adresse, code postal et ville) de tous les éditeurs
49 • SELECT CONCAT(adresse, ', ', code, ' ', ville) AS adresse_complete
50 FROM Editeurs;
51
```

| Result Grid |                                          | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|------------------------------------------|--------------|---------|--------------------|
|             | adresse_complete                         |              |         |                    |
| ▶           | 58 rue Jean Bleuzen, 92170 Vanves        |              |         |                    |
|             | 5 Rue Gaston Gallimard, 75007 Paris      |              |         |                    |
|             | 87 Quai Panhard et Levassor, 75013 Paris |              |         |                    |

7 - Afficher les titres des livres publiés entre le **1er janvier 2010** et le **31 décembre 2015**

```
61 -- 7 - Afficher les titres des livres publiés entre le 1er janvier 2010 et le 31 décembre 2015
62 • SELECT titre
63 FROM Livres
64 WHERE dateparu BETWEEN '2010-01-01' AND '2015-12-31';
65
```

| Result Grid |                                               | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|-----------------------------------------------|--------------|---------|--------------------|
|             | titre                                         |              |         |                    |
|             | Les contes de Perrault                        |              |         |                    |
|             | Histoire de la vie                            |              |         |                    |
|             | Les légendes arthuriennes - Les Contes de Can |              |         |                    |

# Solution Exercice 7 : «Bibliothèque»

8- Afficher les noms des auteurs dont le nom commence par la lettre « J »

```
73 -- 8 - Afficher les noms des auteurs dont le nom commence par la lettre "J"
74 • SELECT nom
75 FROM Auteurs
76 WHERE nom LIKE 'J%';
77
78
```

<

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

|   | nom         |
|---|-------------|
| ▶ | Jules Verne |

9 - Afficher les dates de naissance et de décès des auteurs, triées par ordre décroissant de date de décès

```
85 -- 9 - Afficher les dates de naissance et de décès des auteurs, triées par ordre décroissant de date de décès
86 • SELECT dateNaissance, dateDeces
87 FROM Auteurs
88 ORDER BY dateDeces DESC;
89
90
```

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

|   | dateNaissance | dateDeces  |
|---|---------------|------------|
| • | 1828-02-08    | 1905-03-24 |
|   | 1840-04-02    | 1902-09-29 |
|   | 1802-02-26    | 1885-05-22 |



# Solution Exercice 7 : «Bibliothèque»

10- Afficher le nombre de livres empruntés par chaque emprunteur, en ordre décroissant du nombre de livres empruntés

```
96 -- 10 - Afficher le nombre de livres empruntés par chaque emprunteur, en ordre décroissant du nombre de livres empruntés
97 • SELECT idEmprunteur, COUNT(*) AS nombre_livres_empruntes
98 FROM emprunts
99 GROUP BY idEmprunteur
100 ORDER BY nombre_livres_empruntes DESC;
101
```

| Result Grid |              | Filter Rows:            | Export: | Wrap Cell Content: |
|-------------|--------------|-------------------------|---------|--------------------|
|             | idEmprunteur | nombre_livres_empruntes |         |                    |
| ▶           | 1            | 15                      |         |                    |
|             | 2            | 2                       |         |                    |
|             | 3            | 1                       |         |                    |
|             | 4            | 1                       |         |                    |
|             | 5            | 1                       |         |                    |

# Solution Exercice 7 : «Bibliothèque»

11 - Afficher tous les éditeurs qui ont publié au moins 5 livres

```

108 -- 11 - Afficher tous les éditeurs qui ont publié au moins 5 livres
109 • SELECT idediteurs , COUNT(*)
110 FROM livres
111 GROUP BY idediteurs
112 HAVING COUNT(*) >= 5;
113
```

| Result Grid |            |          | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|------------|----------|--------------|---------|--------------------|
|             | idediteurs | COUNT(*) |              |         |                    |
| ▶           | 1          | 5        |              |         |                    |

# Solution Exercice 7 : «Bibliothèque»

12 - Afficher le nombre de livres empruntés par chaque emprunteur, mais seulement pour les emprunteurs qui ont emprunté plus de **3** livres

```
116 -- 12 - Afficher le nombre de livres empruntés par chaque emprunteur, mais seulement pour les emprunteurs qui ont emprunté plus de 3 livres
117 • SELECT idEmprunteur, COUNT(*) AS nombre_livres_empruntes
118 FROM emprunts
119 GROUP BY idEmprunteur
120 HAVING COUNT(*) > 3;
121
122
```

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

|   | idEmprunteur | nombre_livres_empruntes |
|---|--------------|-------------------------|
| ▶ | 1            | 15                      |

# Solution Exercice 7 : «Bibliothèque»

13 - Afficher le nombre d'emprunteurs qui ont emprunté au moins un livre **chaque mois** de l'année en

cours

```
122
123 -- 13 - Afficher le nombre d'emprunteurs qui ont emprunté au moins un livre chaque mois de l'année en cours
124 • SELECT MONTH(datePret) AS mois, COUNT(DISTINCT idEmprunteur) AS nombre_emprunteurs
125 FROM emprunts
126 WHERE YEAR(datePret) = YEAR(NOW())
127 GROUP BY mois
128 HAVING COUNT(*) >= 1;
129
```

| Result Grid |      |                    | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|------|--------------------|--------------|---------|--------------------|
|             | mois | nombre_emprunteurs |              |         |                    |
| ▶           | 1    | 1                  |              |         |                    |
|             | 2    | 1                  |              |         |                    |
|             | 3    | 1                  |              |         |                    |
|             | 4    | 1                  |              |         |                    |
|             | 5    | 1                  |              |         |                    |
|             | 6    | 1                  |              |         |                    |
|             | 7    | 1                  |              |         |                    |
|             | 8    | 1                  |              |         |                    |
|             | 9    | 1                  |              |         |                    |
|             | 10   | 1                  |              |         |                    |
|             | 11   | 1                  |              |         |                    |
|             | 12   | 1                  |              |         |                    |

# Solution Exercice 7 : «Bibliothèque»

14 - Afficher le nombre de livres différents par auteur

```
133 -- 14 - Afficher le nombre de livres différents par auteur
134 • SELECT idauteurs, COUNT(DISTINCT titre) AS nbLivresDiff
135 FROM Livres
136 GROUP BY idauteurs;
137
138
```





| Result Grid |           |              | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|-----------|--------------|--------------|---------|--------------------|
|             | idauteurs | nbLivresDiff |              |         |                    |
| ▶           | 1         | 5            |              |         |                    |
|             | 2         | 3            |              |         |                    |
|             | 3         | 2            |              |         |                    |

# Solution Exercice 7 : «Bibliothèque»

15 - Afficher les auteurs ayant écrit plus de 3 livres

```
137
138
139 -- 15 - Afficher les auteurs ayant écrit plus de 3 livres
140 • SELECT idauteurs, COUNT(*) AS nbLivres
141 FROM Livres
142 GROUP BY idauteurs
143 HAVING COUNT(*) > 3;
144
```

<

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

|   | idauteurs | nbLivres |
|---|-----------|----------|
| ▶ | 1         | 5        |

# Solution Exercice 7 : «Bibliothèque»

16 - Afficher les auteurs ayant écrit des livres dont le nombre de pages moyen est inférieur à 200 pages

```
144
145 -- 16 - Afficher les auteurs ayant écrit des livres dont le nombre de pages moyen est inférieur à 200 pages
146 • SELECT idauteurs, AVG(nbPages) AS pagesMoyennes
147 FROM Livres
148 GROUP BY idauteurs
149 HAVING AVG(nbPages) < 300;
150
151
```

<   Filter Rows:  | Export:  | Wrap Cell Content: 

|   | idauteurs | pagesMoyennes |
|---|-----------|---------------|
| ▶ | 2         | 240.0000      |