Module 3 - SQL, les premiers pas avec MySQL

Objectifs

Pré-requis: MySQL est installé sur votre machine en local.

- Découvrir et utiliser le programme client `mysql` et se familiariser avec la ligne de commande
- Découvrir les instructions SQL basiques (DDL, DML)
- Se familiariser avec la documentation officielle et savoir s'en servir

Travail dirigé

Voir le document en annexe `td.pdf` qui contient des instructions pour explorer et se familiariser avec MySQL, le modèle relationnel et le langage SQL.

Sections:

- Se connecter
- Tests
- Batch mode
- Première relation

> Une correction du TD vous est également fournie en annexe dans le document `td-corrige.pdf`.

Le SGBD MySQL et l'architecture client/serveur

MySQL est un Système de Gestion de Base de données (SGBD) qui est implémenté suivant l'architecture client/serveur.

Pour accéder à votre base de données, vous devez ouvrir une connexion avec un (programme) client dédié: le programme `mysql`. Vous pouvez ouvrir plusieurs connexions.

> L'architecture client/serveur est à la base du web. Un navigateur web comme Firefox *n'est qu'un* client HTTP hyper sophistiqué pour accéder à une machine distante (serveur).

Le SGBD, comme votre système d'exploitation, possède des utilisateurs (identification et authentification) avec des privilèges (autorisations).

MySQL dipose à l'installation d'un unique utilisateur `root`. C'est un super utilisateur qui a tous les droits. Soyez prudents !

> L'utilisateur `root` est réservé à l'administrateur·ice de la base !

Se connecter au serveur de MySQL (*Command Line Interface* ou *CLI*)

Sous Debian/Ubuntu ou macOS

Ouvrir une session avec le SGBD depuis le terminal en vous connectant avec l'utilisateur `root` :

```
mysql -uroot -p
--ou
mysql --user=root --password
```

> Vous avez défini un mot de passe lors de l'installation pour `root`, **gardez-le précieusement** !

Se connecter au serveur de MySQL (CLI)

Sous Windows

Plusieurs options:

- via l'invite de ligne de commande. Pour cela, vous devez ajouter le répertoire `C:\Program Files\MySQL\MySQL Server 8.0\bin` à votre variable d'environnement `Path` (Propriétés du système/Variables d'environnement). Enregistrer. Vous pouvez lancer le programme mysql.exe depuis n'importe quel chemin sur votre machine avec la commande `mysql --user=root --password`
- via le programme `MySQL 8.0 Command Line Client - Unicode` qui est installé avec MySQL. Il vous connecte automatiquement en `root`. Pour vous connecter avec l'utilisateur `user` directement *depuis* la connexion mysql ouverte, saisir la commande suivante : `SYSTEM mysql -uuser -p`

Se connecter au serveur de MySQL via une interface graphique

- MySQL vient avec le programme MySQL Workbench, un client MySQL cross-plateforme (macOS, Linux, Windows) avec une interface graphique (GUI). Sous le capot, c'est toujours mysql qui est utilisé. Environnement plus user-friendly, plus productif.

> Choisissez l'environnement *qui vous convient le mieux* (CLI ou GUI). Je recommande la CLI pour apprendre, car cela vous oblige à bien comprendre ce que vous faites, à réfléchir davantage avant d'écriture ou d'exécuter une requête SQL.

> Il existe de nombreux outils avec interface graphique pour différents OS.

Customiser le prompt de `mysql`

Configurer le client `mysql` via le fichier `INI` `my.cnf` (ou my.ini en fonction de la version/OS)

Sous Debian/Ubuntu macOS

```
sudo vim /etc/mysql/my.cnf
```

Ajouter ceci dans le fichier de configuration

```
[mysql]
prompt=(\\u@\\h) [\\d]\\_mysql>\\_
```

Customiser le prompt de `mysql`

Configurer le client `mysql` via le fichier `INI` `my.cnf` (ou my.ini en fonction de la version/OS)

Sous Windows

```
sudo vim /etc/mysql/my.cnf
```

Ajouter ceci dans le fichier de configuration

```
[mysql]
prompt=(\\u@\\h) [\\d]\\_mysql>\\_
```

Relancer le serveur MySQL.

Utilisation de `mysql` en *Batch Mode*

> Celleux qui utilisent MySQL Workbench sont par défaut en batch mode !

Le *batch mode* c'est le traitement par lots. Au lieu d'écrire vos requêtes une par une directement dans le terminal, vous pouvez les écrire dans un fichier et soumettre tout son contenu à `mysql` pour qu'il exécute toutes les requêtes contenues dans le fichier. Plus pratique, plus puissant et vous permet de fabriquer des scripts (ensembles de requêtes SQL) réutilisables et documentés.

Utilisation de `mysql` en *Batch Mode*

Créer un script SQL (un fichier contenant des instructions SQL), par exemple `mon-script.sql`

```
-- Toutes ces requêtes SQL seront exécutées les unes à la suite des autres
SELECT VERSION(), CURRENT_DATE;
help;
SHOW DATABASES;
```

Puis exécuter l'intégralité des requêtes depuis le terminal:

```
# Envoyer un fichier de requêtes SQL à exécuter
mysql -uuser -p < mon-script.sql
# Ecrire les résultats dans le fichier mysql.out
mysql -uuser -p < mon-script.sql > mysql.out
# Forcer le script à continuer à s'exécuter même s'il y a des erreurs
mysql -uuser -p --force < mon-script.sql
```

> **N'exécutez pas ces instructions depuis une session mysql ouverte**, mais **depuis votre terminal**. La session sera ouverte puis automatiquement refermée lorsque les instructions contenues dans le fichier auront été executées.

Notions *essentielles* d'algèbre relationnelle

L'algèbre de Codd sert de fondation mathématique au modèle relationnel. L'algèbre relationnelle manipule quelques objets mathématiques à définir:

- **relation**: une relation est un objet mathématique porteur d'**informations**. Il contient ou ou plusieurs **attributs valués**. Il possède une **clef** (un ou plusieurs attributs permettant d'identifier chaque information de la relation **de manière unique**). Une relation dispose d'un **nom unique** dans la base de données (ou schéma).
- **attribut**: un attribut dispose d'un **nom unique** dans la relation. Il contient une information **atomique**. Il est **valué** et sa valeur est prise dans un **domaine**.
- **tuple**: ensemble de valeurs pour chacun des attributs de la relation, décrivant un élément particulier de la relation.

Notions *essentielles* d'algèbre relationnel : traduction dans le standard SQL

Voici la traduction algèbre relationnelle (à gauche) en SQL (à droite)

- relation : table
- attribut : colonne
- tuple: ligne d'une table, élément, enregistrement (*record*)

Par exemple la relation `Employé: __matricule__, nom, prénom, date de naissance` a pour clef l'attribut `matricule`, est définie par 4 attributs. Chaque information (tuple) doit donc être définie par 4 valeurs, dont 1 (la clef) qui la distingue de toutes les autres. Par exemple, `{XD1247, Dupont, Harry, 21/04/1992}` est un tuple valide de la relation Employé.

SQL: création/Suppression d'une table avec ses colonnes

Instruction: `CREATE TABLE`

```sql
CREATE TABLE Pilote(
  id INTEGER,
  nom VARCHAR(30),
  prenom VARCHAR(30),
  nb_h_vol INTEGER
);
-- Supprimer une table
DROP TABLE Pilote;
```

Opérations de l'algèbre relationnelle

L'algèbre relationnelle est une **fermeture**. **Chaque opération sur une relation retourne une relation**.

> Exemple: l'addition sur deux entiers est une fermeture, 2 + 3 retourne 5 qui est aussi un entier (sur lequel je peux continuer d'appliquer l'opération d'addition)

Edgard Codd a défini les opérations suivantes:

- **restriction** (ou sélection): conserver que certains tuples de la relation ayant des caractéristiques décrites par le biais d'un *prédicat* SQL: `WHERE (prédicat)`
- **projection** : ne retenir dans la relation résultante que certains attributs et pas d'autres. `SQL: SELECT attribut1, attribut2 ...`
- **union** : concaténer des relations aux caractéristiques similaires (OU en mathématiques) `SQL: UNION`

Opérations de l'algèbre relationnelle

> un prédicat: une expression qui ne peut s'évaluer qu'à vrai ou faux. Par exemple "cette phrase compte six mots" est un prédicat, évalué à faux.

- **intersection**: retourner les éléments communs à deux relations (exactement les mêmes) (ET en mathématiques) `SQL: INTERSECT`
- **différence**: renvoie les éléments d'une relation qui n'existe pas dans l'autre `SQL: EXCEPT`
- **multiplication** (ou produit cartésien): associer à tout élément d'une relation chacun des éléments d'une autre relation `SQL: CROSS JOIN`
- **division**: inverse de la multiplication. **Opération implémentée dans aucun SGBDR à l'heure actuelle**
- **jointure**: associer aux éléments d'une relation des éléments d'une autre relation par le biais d'un prédicat `SQL: JOIN (INNER, OUTER, LEFT, RIGHT)`

Les contraintes: algèbre relationnelle, *cohérence* des données et logique métier

Les contraintes permettent *d'appliquer de la logique métier directement dans la base* (la source de vérité) et alléger le code applicatif.

Elles peuvent être déclarées en même temps que les attributs d'une relation (*inline constraints*) ou à la suite (*out-of-line constraints*)

Les 5 contraintes les plus utilisées (**à connaître**) sont:

- `UNIQUE` : unicité de chaque valeur de l'attribut dans la table (deux lignes ne peuvent pas avoir la même valeur pour cet attribut)
- `PRIMARY KEY` : alias pour contraintes `UNIQUE+NOT NULL+INDEX`. Raccourci pour implémenter la clef (au sens de Codd) de la relation
- `FOREIGN KEY` : permets de référencer un enregistrement dans une autre relation afin de maintenir une base cohérente

Les contraintes: *cohérence* des données et logique métier

- `CHECK` (complètement implémentée en MySQL 8): ajouter une contrainte basée sur un prédicat. Se déclenche au moment de l'INSERT et de l'UPDATE.
- `NOT NULL` : l'attribut doit être valué (une valeur doit être obligatoirement fournie)

Il en existe d'autres (`ENUM` , `SET` , etc.)

Les contraintes, en pratique

Comment définir des contraintes ? Trois possibilités:

1. A la création lors de la définition de l'attribut

```
CREATE TABLE Foo(
  -- Non recommandé ! Mélange définition d'une colonne (nom, type) et contrainte. Plus difficile à maintenir
  id INTEGER PRIMARY KEY
);
```

2. A la fin de la définition de la table (contrainte nommée et séparée)

```
CREATE TABLE Foo(
  id INTEGER
  -- recommandé. Contrainte nommée, séparée de la définition de la colonne
  CONSTRAINT pk_foo PRIMARY KEY(id)
);
```

Les contraintes, en pratique

Trois possibilités:

3. Après création de la table avec l'instruction ALTER TABLE

```
CREATE TABLE Foo(
  id INTEGER
);
-- ajout de la contrainte après création de la table.
ALTER TABLE Foo ADD CONSTRAINT pk_foo PRIMARY KEY(id);
-- ou par exemple, pour ajouter une contrainte NOT NULL sur la colonne id
ALTER TABLE Foo MODIFY id INTEGER NOT NULL;
```

Contrainte de clef étrangère et intégrité référentielle

La contrainte de clef étrangère permet de préserver la cohérence de la base de données en conservant les associations entre tables. Elle contribue à l'**intégrité référentielle**.

L'intégrité référentielle est une situation dans laquelle pour chaque information d'une table A qui fait référence à une information d'une table B, l'information référencée **existe** dans la table B.
L'intégrité référentielle est **un gage de cohérence** du contenu de la base de données.

Contrainte de clef étrangère et intégrité référentielle

> Par exemple, un Pilote travaille pour une Compagnie. Je renseigne donc la compagnie pour laquelle il travaille dans une colonne. Si je supprime la compagnie, cette information sera devenue incohérente (car la compagnie référencée dans la table Pilote n'existe plus !). La contrainte de clef étrangère permet d'empêcher qu'une telle situation se produise.

```
CREATE TABLE Compagnie(
  id INTEGER
  CONSTRAINT pk_compagnie PRIMARY KEY(id)
);

CREATE TABLE Pilote(
  id INTEGER PRIMARY KEY,
  -- attribut pour stocker la référence de l'id de la Compagnie pour laquelle travaille le pilote
  id_compagnie INTEGER,
  -- clef primarie (clef au sens de Codd)
  CONSTRAINT pk_pilote PRIMARY KEY(id)
  -- clef étrangère (intégrité référentielle)
  CONSTRAINT fk_pilote_work_for_compagnie FOREIGN KEY(id_compagnie) REFERENCES Compagnie(id)
);
```

Types essentiels

Dans l'algèbre de Codd, tous les attributs doivent être valués et définis sur un domaine. En informatique, il faut donc leur donner un type pour allouer de la mémoire et valider le domaine de définition de l'attribut.

- Numérique: INTEGER, DECIMAL, TINYINT, etc.
- Caractères: CHAR, VARCHAR, TEXT
- Date, heure: DATE, TIME, DATETIME, TIMESTAMP
- Gros fichiers binaires (Large binary objects comme des fichiers image, audio, vidéo): BLOB
- Autres: JSON, SPATIAL, etc.

Conventions de nommage utilisées dans le cours

- Mots SQL en MAJUSCULES. ex: `SELECT * FROM Foo WHERE id=1;`
- Noms des relations (tables) sans caractère spécial en Camelcase. **Soit tout au singulier, soit tout au pluriel, mais pas les deux**. Soyez cohérent·es. ex: `Personne`, `Employe`
- Noms des attributs (colonnes) et contraintes en minuscule, sans accents snake_case. ex: `id`, `date_de_naissance`

Conventions de nommage utilisées dans le cours

- **Toujours nommer une contrainte** (pas obligatoire en pratique). Important pour la lisibilité et la compréhension, et pour faire évoluer la contrainte (désactivation, réactivation, suppression)
- Préfixer par `pk_` le nom d'une contrainte `PRIMARY KEY`
- Préfixer par `fk_` le nom d'une contrainte `FOREIGN KEY`
- Préfixer par `un_` le nom d'une contrainte `UNIQUE`
- Préfixer par `ck_` le nom d'une contrainte `CHECK`
- Pour le nom d'une contrainte `PRIMARY KEY` : `pk_nom_relation`, pax ex; `pk_foo` pour la table `Foo`.
- Pour le nom d'une contrainte `FOREIGN KEY` : `fk_relation_source_nom_association_relation_cible`. Par ex, une relation entre Pilote et Compagnie `fk_pilote_work_for_compagnie`.

Conventions de nommage utilisées dans le cours

Il n'y a pas de *standard* concernant les conventions de nommage. Cela ouvre la porte à d’éternels débats entre les communautés et les générations.

Ici vous trouverez une bonne liste de standards qu'il est bon de respecter.

Un document de normes plus complet peut se trouver ici.

Un mot sur les clefs

L'attribut qui sert de clef (`PRIMARY KEY`) doit être choisi et défini avec soin (performances).

En effet, **la clef primaire va servir de pivots à de nombreuses jointures**. Elle se doit donc d'être indexée et performante (simple à lire, ne pas changer, etc.)

> Une contrainte PRIMARY KEY ne crée pas automatiquement une séquence (AUTO_INCREMENT). Par contre, elle génère automatiquement les contraintes UNIQUE, NOT NULL et INDEX. C'est un alias pour exprimer ces 3 contraintes.

Un mot sur les clefs

Pour faire une clef primaire performante:

- **un attribut dédié**
- un nombre entier (type `INTEGER`), **surtout pas une chaine de caractères !**
- indépendante du domaine métier
- auto incrémentée
- concise (bâtie sur un seul attribut) et facile à exprimer
- asémantique (pas de sens en dehors du système)
- invariante (ne doit jamais changer, dans aucun cas !)

```
-- Une bonne clef performante
id INTEGER AUTO_INCREMENT PRIMARY KEY
```

Définition des données (DDL: Data Definition Langage)

- CREATE: créer une table
- DROP: supprimer une table
- ALTER: modifier la structure d'une table (colonne, contrainte, nom, type, etc.)

Manipulation des données (DML: Data Manipulation Langage)

- INSERT: insérer des données d'une table
- DELETE: supprimer des données d'une table
- UPDATE: mettre à jour des données d'une table
- TRUNCATE: vider une table de ses données

Inspecter

Dans le doute, on inspecte ses structures de données et ses données

- `DESCRIBE tbl_name` (ou `DESC`) : voir la définition de la table
- `SHOW TABLES | COLUMNS FROM tbl_name | DATABASES` : lister les tables, les colonnes d'une table ou les bases de données
- `SHOW CREATE TABLE tbl_name` : demande à MySQL de nous montrer ce qu'il doit faire pour créer la table. Permets de voir les contraintes.
- `TABLE tbl_name` : lister toutes les valeurs pour toutes les colonnes de la table. Raccourci pour `SELECT * FROM tbl_name`
- `SELECT CURRENT_USER` : afficher l'utilisateur courant
- `SELECT CURRENT_DATABASE` : afficher la base de données courante

Exemple

```sql
-- insertion d'une compagnie (juste un identifiant)
INSERT INTO Compagnie (id) VALUES (1);
-- insertion de 3 pilotes (identifiant, identifiant de la compagnie pour laquelle ils travaillent)
INSERT INTO Pilote(id, id_compagnie) VALUES (1, 1), (2, 1), (3, 1);
-- vider la table Pilote
TRUNCATE Pilote;
-- supprimer la compagnie ayant l'id égal à 1
DELETE FROM Compagnie WHERE id=1;
```

Interrogation, projection des données (DQL: Data Query Langage)

- SELECT
- INTERSECT, UNION, EXCEPT
- JOIN, INNER JOIN, OUTER JOIN, LEFT JOIN, RIGHT JOIN
- GROUP BY, HAVING, ORDER BY

Apprendre à la lire la documentation officielle

Il existe énormément d'instructions en SQL car on fait tout avec (création des structures de données, administration, manipulation des données, interrogations des données, etc.). **Une fois que l'on connaît les opérations de base**, *on ne peut pas retenir et* ***on ne veut pas retenir*** *toutes les possibilités que nous offre chaque instruction, toutes les syntaxes possibles, etc.* (arguments, options, etc.).

Apprendre à la lire la documentation officielle

C'est pourquoi **il est important d'apprendre à se servir de la documentation** pour:

- mettre des choses plus intéressantes dans sa tête à la place
- découvrir les possibilités en fonction des usages. On apprend bien lorsque l'on apprend à faire quelque chose dont on a besoin.
- devenir meilleur•e en SQL et avec MySQL. La documentation est très riche en information, en conseils, etc.
- **gagner en confiance**. Si vous savez lire cette documentation, **vous saurez mieux lire la documentation de la technologie suivante**.
- la *satisfaction est grande* de résoudre un problème rencontré à partir de la compréhension de la documentation
- ne pas dépendre de sources tierces inconnues (Stackoverflow, billets de blog, etc.) (surtout sur des opérations sensibles)

Apprendre à la lire la documentation officielle

La documentation officielle de MySQL v8 est bien faite.

Prenons un exemple, l'instruction ALTER TABLE. Voici ce qu'on y lit

```
ALTER TABLE tbl_name
    [alter_option [, alter_option] ...]
    [partition_options]

alter_option: {
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX | KEY} [index_name]
        [index_type] (key_part,...) [index_option] ...
  | ADD {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name]
        (key_part,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY
        [index_type] (key_part,...)
        [index_option] .
--etc...
```

Apprendre à la lire la documentation officielle

On a d'abord la syntaxe complète de l'instruction. En dessous, un texte nous explique à quoi sert `ALTER TABLE` dans les grandes lignes. On commence par là. Par lire le début du texte. Puis on lit la syntaxe pour se servir de l'instruction.

```
ALTER TABLE tbl_name
    [alter_option [, alter_option] ...]
    [partition_options]
```

ici `tbl_name` est un argument **obligatoire** de `ALTER TABLE`. C'est le nom de la table que l'on souhaite modifier. `[alter_option [, alter_option] ...]` et `[partition_options]` sont entre crochets, ce sont des **options** (donc optionnelles !). `[alter_option [, alter_option] ...]` dit que je peux ajouter autant d'options de ce type les unes à la suite des autres (les 3 petits points).

Apprendre à la lire la documentation officielle

Regardons la première option `alter_option`

```
--...
alter_option: {
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ADD [COLUMN] (col_name column_definition,...)
--...
}
```

ici tout ce qui est entre accolades `{..}` est l'ensemble des valeurs possibles que peut prendre l'option. La barre verticale | indique le OU. C'est soit `ADD [COLUMN] col_name column_definition` OU `ADD [COLUMN] (col_name column_definition,...)`.

Apprendre à la lire la documentation officielle

Je cherche à voir quelle est la syntaxe pour ajouter une colonne `bar` de type `DATE` à ma table `Foo`, car j'ai oublié de l'ajouter au moment du `CREATE TABLE` ou mon modèle de données à évolué avec de nouveaux besoins métier. En lisant la documentation, je vois rapidement que je peux écrire

```
-- soit ça
ALTER TABLE Foo ADD COLUMN bar DATE;
-- ou ça
ALTER TABLE Foo ADD (bar Date);
-- ou ça
ALTER TABLE Foo ADD bar DATE;
-- etc..
```

Pratiquer les bases

Suivre le tutoriel officiel de la documentation.

References

- Tutoriel du manuel officiel
- Manuel officiel en ligne de MySQL v8 (en), le manuel en ligne de MySQL v8. Très complet et très bien fait. N'hésitez pas à le consulter pour éclaircir un doute, ou approfondir un sujet.
- CREATE USER Statement (Manuel)
- Batch Mode (Manuel)
- SHOW Statement (Manuel)
- CREATE TABLE Statement (Manuel)
- Data Types (Manuel)
- Using AUTO_INCREMENT
- DELETE Statement (Manuel)