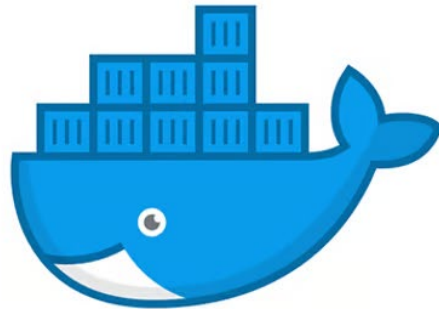


# P\_DB-106

---



William Lonfat – CIN2A  
ETML – Vennes - Lausanne  
32 périodes  
Antoine Mveng

# Table des matières

<b>1</b>	<b>SPÉCIFICATIONS.....</b>	<b>3</b>
1.1	TITRE.....	3
1.2	DESCRIPTION.....	3
1.3	MATÉRIEL ET LOGICIELS À DISPOSITION .....	3
1.4	CAHIER DES CHARGES.....	3
1.4.1	Travail à réaliser par l'apprenti.....	3
1.4.2	Contraintes.....	4
1.5	POINTS ÉVALUÉS .....	5
1.6	VALIDATION ET CONDITIONS DE RÉUSSITE.....	5
<b>2</b>	<b>ANALYSE.....</b>	<b>5</b>
2.1	DOCUMENT D'ANALYSE ET CONCEPTION .....	5
<b>3</b>	<b>RÉALISATION .....</b>	<b>10</b>
3.1	MISE EN PLACE DE L'ENVIRONNEMENT DOCKER.....	10
3.2	CRÉATION DU SCRIPT DE CRÉATION DE LA DB ET DES TABLES.....	11
3.3	CRÉATION DU SCRIPT POUR IMPORTER LES DONNÉES.....	12
3.4	MARCHE À SUIVRE POUR CRÉER LA DB.....	13
3.5	MARCHE À SUIVRE POUR IMPORTER LES DONNÉES .....	15
3.6	CRÉATION DU SCRIPT POUR CRÉER LES UTILISATEURS ET LES RÔLES.....	16
3.7	MARCHE À SUIVRE POUR CRÉER LES UTILISATEURS ET LES RÔLES .....	17
3.8	REQUÊTES SQL .....	18
3.9	CRÉATION DES INDEX .....	30
3.10	VÉRIFIER QUE LES REQUÊTES UTILISENT LES INDEX .....	31
3.11	SCÉNARIO DE TRANSACTION .....	32
3.12	BACKUP DE LA BASE DE DONNÉES .....	35
<b>4</b>	<b>TESTS.....</b>	<b>35</b>
4.1	TABLEAU DES TESTS .....	35
<b>5</b>	<b>CONCLUSION.....</b>	<b>36</b>
5.1	BILAN DES FONCTIONNALITÉS DEMANDÉES.....	36
5.2	BILAN PERSONNEL .....	36
<b>6</b>	<b>ANNEXES .....</b>	<b>36</b>

# 1 SPÉCIFICATIONS

## 1.1 Titre

Gestion d'une pizzeria avec livraisons

## 1.2 Description

Modéliser et implémenter la base de données d'une pizzeria (commandes, pizzas, ingrédients, clients, livreurs, zones de livraison, paiements). Charger des données, effectuer des requêtes SQL, définir des rôles et transactions, et mettre en place une stratégie de sauvegardes/restauration.

## 1.3 Matériel et logiciels à disposition

- Docker, conteneurs MySQL et PhpMyAdmin
- Accès à Internet

## 1.4 Cahier des charges

### 1.4.1 Travail à réaliser par l'apprenti

#### 1. Modélisation (Merise) :

Modélisation du MCD de la base de données selon le cahier des charges

#### 2. Création des tables de la base de données :

Création des différentes tables nécessaires au fonctionnement de la DB

#### 3. Charger des données :

Effectuer des LOAD DATA pour pouvoir charger les données fournies (en format .tsv) dans la base de données.

**4. Mettre en place une stratégie de sauvegarde et de restauration :**

Effectuer un backup complet une fois par semaine, et effectuer un backup différentiel chaque jour de la semaine à minuit.

**5. Effectuer des requêtes SQL :**

Effectuer les différentes requêtes SQL demandées, pour pouvoir récupérer des informations sur les données importantes comme les produits vendus, les commandes, les prix. (Liste non exhaustive).

**6. Créer des index :**

Créer des index en fonction des requêtes données pour pouvoir accélérer le temps de réponses.

**7. Créer des utilisateurs et des rôles :**

Créer des rôles et des utilisateurs pour pouvoir donner les droits adaptés aux différents utilisateurs de la base de données. (Droits attribués selon la fonction de l'utilisateur dans la pizzeria.)

**8. Effectuer un scénario d'une transaction :**

Créer une requête permettant de faire une transaction, par exemple pour un cas concret comme l'annulation d'une commande et la restitution du stock. S'assurer que la commande a bien été annulée avant de pouvoir restituer le stock.

(Plus de détails concernant le travail à réaliser est disponible dans le cahier des charges.)

### 1.4.2 Contraintes

1. Les droits octroyés à l'utilisateur dans la base de données doivent correspondre à sa fonction dans la pizzeria.
2. L'intégrité des données importées avec le LOAD DATA doit être vérifié et doit correspondre aux données des fichiers TSV. (Adaptés pour le MCD.)

3. La stratégie de backup doit sauvegarder la base de données automatiquement une fois par jour (en différentiel) et une fois par semaine. (En sauvegarde complète). Pouvoir restaurer la base de données avec le dump automatiquement créé.
4. Les index mis en place pour améliorer le temps de réponse des requêtes données, doivent montrer un changement au niveau du temps de réaction

## 1.5 Points évalués

- Le rapport
- Le journal de travail
- Le code et les commentaires
- Les documentations de mise en œuvre et d'utilisation

## 1.6 Validation et conditions de réussite

- Compréhension du travail
- État de fonctionnement du produit livré

# 2 ANALYSE

## 2.1 Document d'analyse et conception

1<sup>ère</sup> version du Modèle conceptuel de données (MCD) :

Pour commencer à créer une base de données pour une pizzeria il faut d'abord créer un MCD. Il contient des entités et sont reliées entre-elles grâce à des associations (avec les clés primaires PK et des clés étrangères PK). Le MCD contient différentes tables / entités, comme adresse, client, paiement, commande, ligne commande, article, livraison et livreur.

### Associations :

1. La table CLIENT est reliée à la table ADRESSE grâce à l'association vivre. Grâce aux cardinalités 1,n / 1,n on peut dire qu'un client peut posséder au minimum une ou plusieurs adresses et qu'une adresse peut être liée à 1 ou plusieurs clients.

2. La table COMMANDE est reliée à la table ADRESSE grâce à l'association *etre\_livrer*. Les cardinalités 0,n / 0,1 précisent qu'une commande peut être livrée à aucune adresses ou au maximum une seule adresse et qu'une adresse peut recevoir plusieurs adresses.
3. La table COMMANDE est également liée à la table CLIENT grâce à l'association *commander* qui précise qu'une commande peut appartenir à un seul client et qu'un client peut commander plusieurs commandes. (1,n et 1,1)
4. La table LIVRAISON est liée à la table COMMANDE avec l'association *avoir*. Les cardinalités 0,n et 1,n permette de dire qu'une commande peut avoir 0 (Sur place ou à emporter) ou plusieurs livraisons (grande commande) et qu'une livraison peut être liée à plusieurs livraisons. (Avec des adresses différentes).
5. La table LIVRAISON est également liée à la table LIVREUR avec l'association *livrer* qui montre qu'un livreur peut livrer plusieurs livraisons et qu'une livraison peut avoir plusieurs livreurs. (0,n et 1,n).
6. La table COMMANDE est aussi liée à la table LIGNE\_COMMANDE avec l'association *contenir*. (LIGNE\_COMMANDE représente ici les lignes présentes dans une addition.) Une commande peut contenir plusieurs lignes et une ligne est liée à une seule commande. (1,n et 1,1).
7. La table LIGNE\_COMMANDE est liée à la table ARTICLE par l'association *etre\_dans*. Les cardinalités montrent qu'une ligne (de commande) peut être liée à un seul article, et qu'un article peut se retrouver une seule, ou plusieurs fois dans des lignes de commandes.
8. La table LIGNE\_COMMANDE est liée également à **elle-même**, pour pouvoir gérer les cas des toppings. En ayant une association 1,1 et 0,n cela permet de dire qu'un topping peut être lié à une seule autre ligne de commande, et qu'une ligne de commande (une pizza par exemple) peut posséder 0 ou plusieurs toppings.
9. La table PAIEMENT est liée à la table COMMANDE avec l'association *payer* qui possède les cardinalités (1,1 et 1,n). Un paiement est lié à une seule

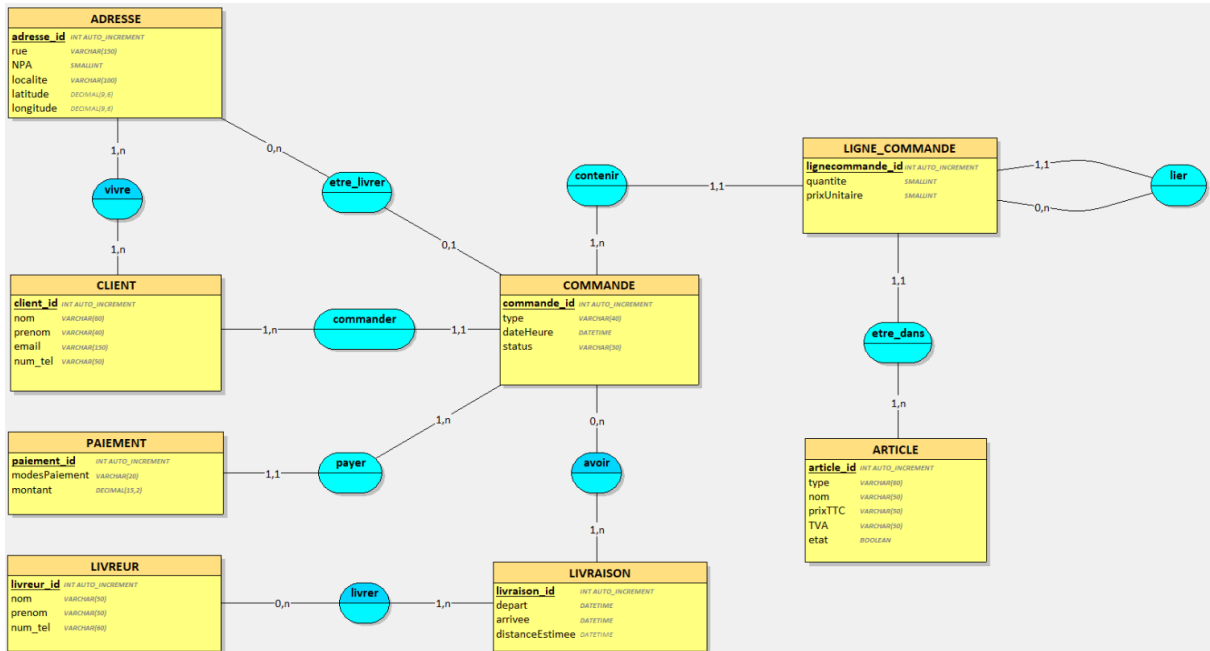
commande et une commande peut avoir un ou plusieurs paiements (acomptes possibles)

### **Contenu des tables :**

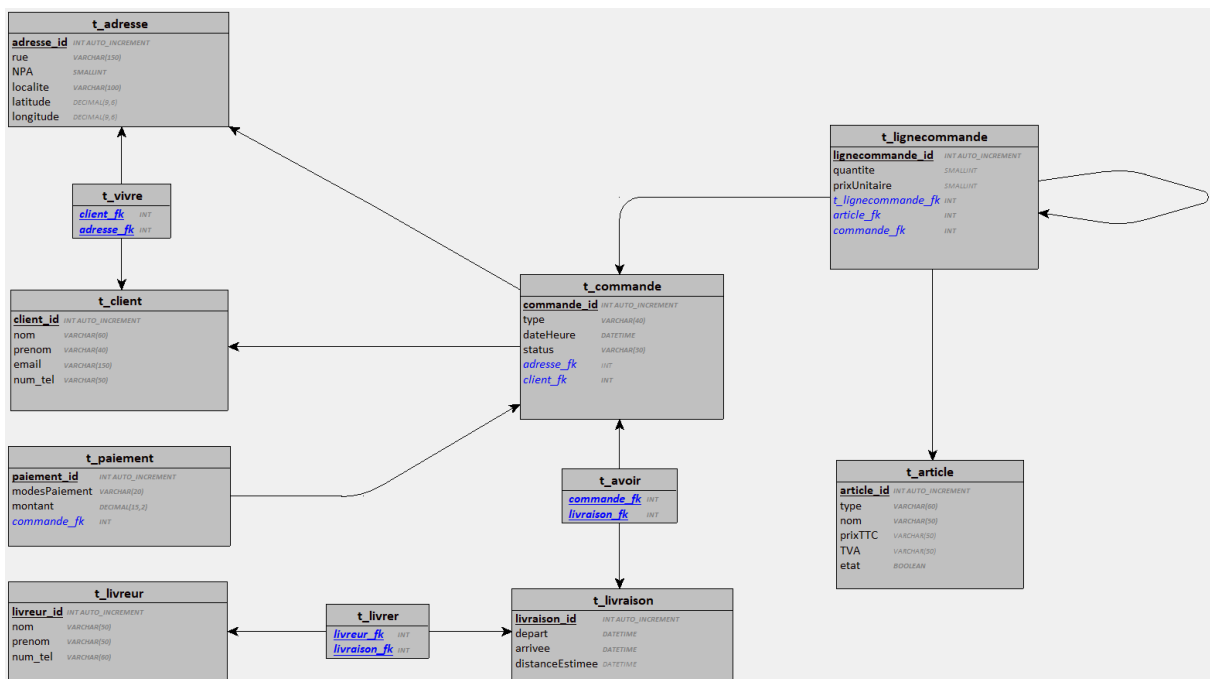
(Toutes les tables contiennent un ID)

- La table CLIENT contient les champs nom, prenom, email, et num\_tel, pour pouvoir récupérer et mettre dans une seule tables les données des clients.
- La table ADRESSE contient les champs rue, NPA, localite, latitude et longitude, pour pouvoir regrouper les informations sur les adresses et pouvoir avoir des informations précises pour pouvoir livrer des commandes.
- La table COMMANDE contient les champs type, dateHeure et status, pour pouvoir afficher les types de commandes (Sur place ou livraison), pouvoir attribuer une date et une heure, ainsi qu'avoir des informations sur son statut pour savoir si la commande a été annulée, ou validée.
- La table LIVRAISON contient les champs depart et arrive, pour savoir quand la livraison a débutée et s'est terminée (Le champ distanceEstimee servait à avoir des indications sur la distance pour pouvoir répartir les livreurs par zones pour pouvoir être plus efficace, ce qui n'est pas géré ici. Le champ a été supprimé à la 2<sup>ème</sup> version du MCD.)
- La table LIVREUR contient les champs nom, prenom et num\_tel pour pouvoir avoir des informations sur le livreur et l'appeler en cas de besoin.
- La table PAIEMENT contient les champs modesPaiement et montant pour pouvoir avoir des informations sur le paiement en cas de problème.
- La table LIGNE\_COMMANDE contient les champs quantite et PrixUnitaire.
- La table ARTICLE contient les champs type, nom, prixTTC, TVA et etat, qui permette de calculer le prix des produits, en ayant leurs noms et leur stock.

Cette première version a été faite pour poser les bases de la conception de la base de données. Elle contient des erreurs qui ont été corrigées dans la 2<sup>ème</sup> version du MCD, qui a été faite pour pouvoir accueillir les données des fichiers TSV fournis.



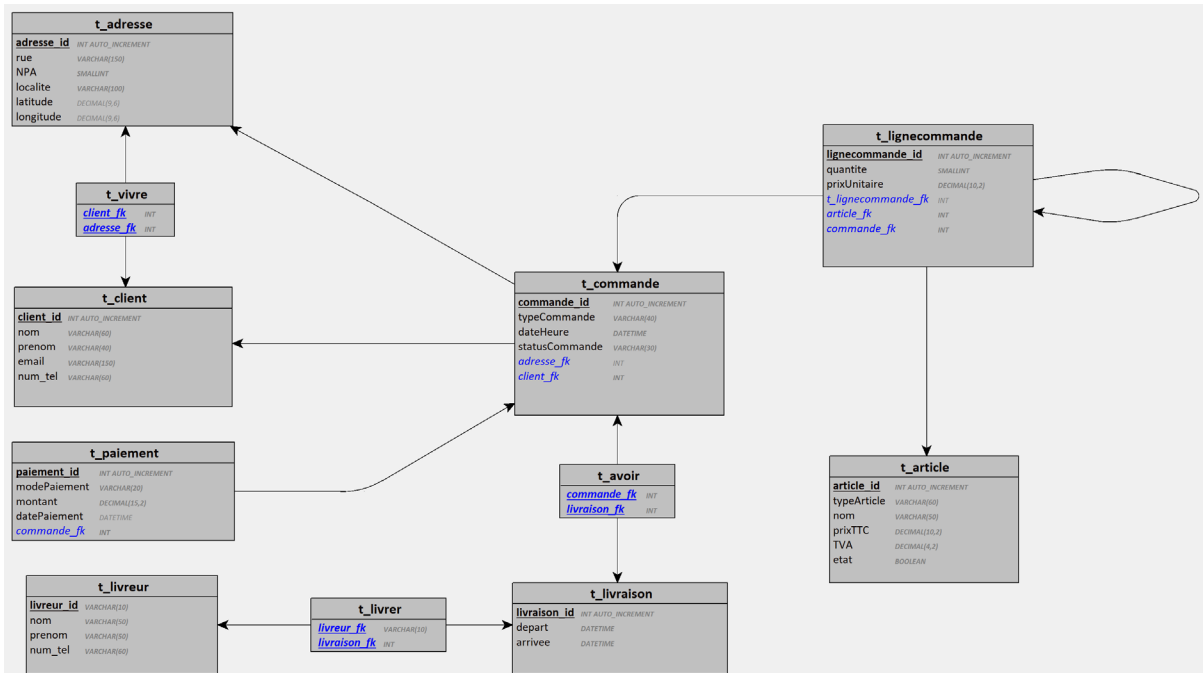
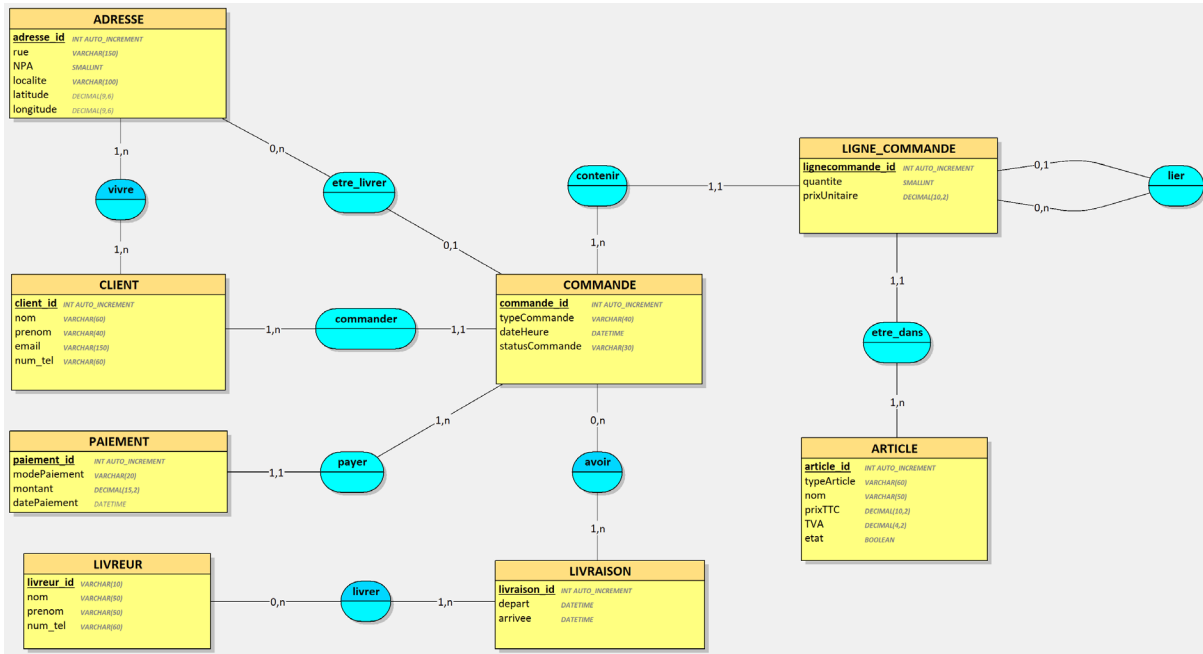
Voici le MLD de la première version, qui a été généré automatiquement par Looping :



Voici les modifications apportées avec la 2<sup>ème</sup> version du MCD :

1. Les champs type, status de la table COMMANDE ainsi que le champ type de la table ARTICLE ont été remplacés par typeArticle, typeCommande et statusCommande, en raison d'une confusion de noms avec MySQL, ce qui ne permettait pas d'importer les données.
2. Les types de prixTTC et de TVA ont été changés en type DECIMAL à la place de VARCHAR, pour pouvoir mieux contrôler le contenu des données et pouvoir vérifier les virgules, décimales.
3. Des petites améliorations ont été faites sur les champs qui ne devrait pas être NULL.
4. Pour la table t\_livreur l'ID a été modifié en type VARCHAR, pour pouvoir prendre l'ID unique des livreurs comme LIV002 à la place de chiffres. Cela permet d'identifier plus facilement le livreur.
5. Pour la table t\_paiement, le nom du champ modesPaiement et été changé en modePaiement. (Il y a un seul mode de paiement par paiement.) et le champ datePaiement a été rajouté.
6. Pour la table t\_lignecommande le NOT NULL a été rajouté au champ quantite. Le champ prixUnitaire a été modifié en type DECIMAL a la place de SMALLINT (plus adapté).
7. La modification très importante de cette version est le changement des cardinalités de l'association lier. Grace à 0,1 et 0,n, les cardinalités sont maintenant correctes pour pouvoir gérer **l'option d'avoir des toppings**, car 1,1 dit qu'un topping doit être lié à un parent ce qui met t\_lignecommande\_fk NOT NULL, ce qui pose un problème. Mettre 0,1 à la place permet de dire que la fk est optionnelle, et donc qu'on puisse commander une pizza sans toppings.

Voici les nouveau schémas MCD et MLD de la base de données :






## 3 RÉALISATION

### 3.1 Mise en place de l'environnement Docker

#### Prérequis :

- Docker est installé sur votre machine
- Vérifier que Docker Desktop est lancé

1. Récupérer le fichier **P\_DB-106.zip** accessible via cette [url](#).
2. Le dézipper le fichier dans C (et pas dans votre disque externe)
3. Se rendre dans le dossier P\_DB-106

Nom	Modifié le	Type	Taille
 scripts	24.01.2023 15:19	Dossier de fichiers	
 .env	19.01.2023 13:36	Fichier ENV	1 Ko
 docker-compose.yml	19.01.2023 13:36	Fichier YML	1 Ko

4. Ouvrir **git bash** et se positionner dans le répertoire **P\_DB-106**

```
willi@ACER_Lonfat MINGW64 ~
$ cd "C:\Users\willi\Documents\P_DB-106"

willi@ACER_Lonfat MINGW64 ~/Documents/P_DB-106
$ ls
docker-compose.yml  scripts/
```

5. Lancer la création des conteneurs Docker avec **docker-compose up -d**
6. A la fin de l'exécution, vous pouvez faire la commande `docker ps` pour voir les 2 conteneurs :

```
✓ Container db          Started
✓ Container pma         Started

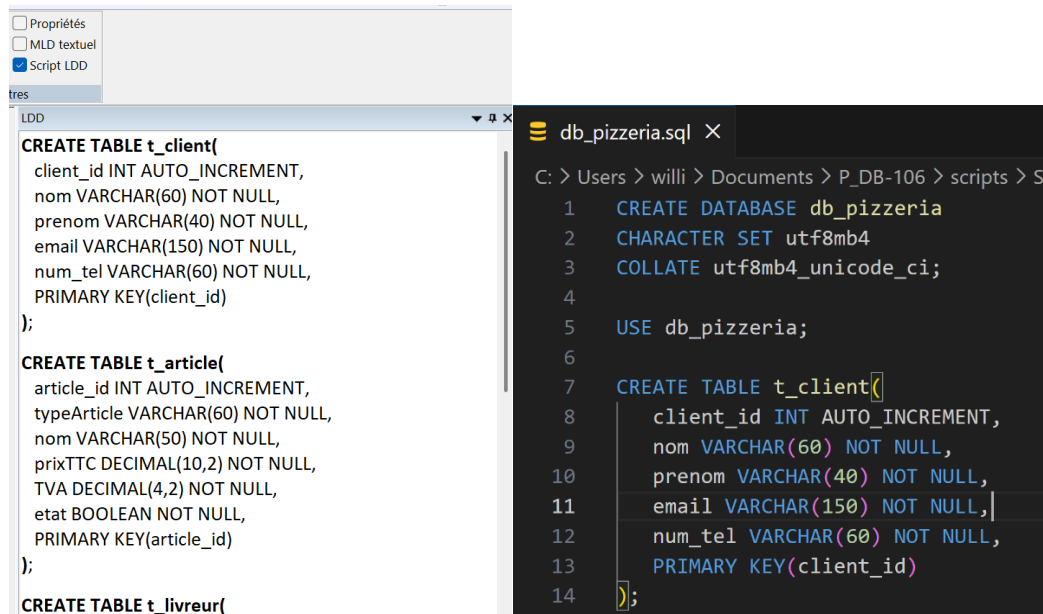
willi@ACER_Lonfat MINGW64 ~/Documents/P_DB-106
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED     STATUS      PORTS                               NAMES
0dc00f82d5ba   phpmyadmin:5.2.0  "/docker-entrypoint...."  5 seconds ago  Up 4 seconds  0.0.0.0:8081->80/tcp              pma
8fff3db1769f   mysql:8.0.30    "docker-entrypoint.s..."  5 seconds ago  Up 4 seconds  33060/tcp, 0.0.0.0:6033->3306/tcp  db
```

(Le seul changement apporté ici sont l'ajout des fichiers de création ainsi que les fichiers de données (en format .csv) dans le répertoire scripts, et la modification du docker-compose, pour pouvoir lier le répertoire /scripts qui est en local au répertoire /var/lib/mysql-files pour que les commandes de LOAD DATA puissent fonctionner.)

## 3.2 Création du script de création de la DB et des tables

Pour créer le MPD, Il suffit de cliquer sur Script LDD sur looping pour y accéder. On peut ensuite le copier-coller dans un fichier nommé `db_pizzeria.sql`. Pour la base de données `db_pizzeria`, j'ai choisi l'encodage `utf8mb4` avec comme `COLLATE utf8mb4_unicode_ci`. Cela permet de gérer tout type d'accent pour les prénoms, noms, etc... Ce collate est également case insensitive, ce qui permet de mettre des majuscules comme des minuscules sans aucun problème. Comme le collate est Unicode, il est même possible d'ajouter des émojis dans la base de données. (Cela peut arriver si on place des emojis à

côté des noms de produit par exemple.) Le script pour créer la base se trouve juste avant le MPD avec les CREATE TABLE.



### 3.3 Création du script pour importer les données

Pour importer les données traitées (sous format .csv), on peut utiliser LOAD DATA. Comme les données ont .csv comme format toutes les colonnes sont séparées par des ; chaque ; représente un changement de colonne et chaque retour à la ligne un changement de ligne. Voici le LOAD DATA pour la table t\_client :

```

LOAD DATA INFILE '/var/lib/mysql-files/DATA/CSV/t_client.csv'
INTO TABLE t_client
CHARACTER SET utf8mb4
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(client_id, nom, prenom, email, num_tel);

```

On commence par préciser le chemin du fichier, on décrit le nom de la table avec le bon encodage puis on précise que les champs se termine par ; On doit également préciser que le retour à la ligne (\n) représente la fin de la ligne. Puis on ignore la première ligne du fichier, car elle représente l'ordre dans lequel les tables sont misent. Il est très important de vérifier l'ordre des tables dans le LOAD DATA ainsi que l'ordre des données dans le fichier .csv, car ce sont elles qui vont

préciser quelles données se trouve dans quelle colonne. Le LOAD DATA contient également **SET FOREIGN\_KEY\_CHECKS = 0;** qui signifie à MySQL de ne pas vérifier les clés étrangères lors de toutes les importations des données. Ce paramètre sera égal à 1 à la fin du script pour pouvoir réactiver les contrôles. Cela n'empêche pas qu'il faut toujours commencer par les tables qui n'ont pas de dépendances en premier, avant de mettre les données des tables qui contiennent des FK. (Sinon ça créé évidemment des erreurs car si MySQL ne trouve pas de FK cela ne vas pas marcher.)

Le paramètre **SET** en dessous de l'ordre des colonnes peut être utilisé pour remplacer les colonnes vide par NULL. Par exemple dans une ligne ou la colonne des FK ne contient rien (;;) le paramètre SET permet de détecter les cas vides puis de remplacer par NULL les champs grâce à NULLIF. (Si il y a rien remplacer par NULL) :

```
(commande_id, typeCommande, dateHeure, statusCommande, @adresse_fk, client_fk)
SET adresse_fk = NULLIF(@adresse_fk, '');
```

Il ne faut également pas oublier de mettre **USE db\_pizzeria;** au début du fichier pour pouvoir mettre les données des fichiers dans la bonne DB et dans les bonnes tables.

### 3.4 Marche à suivre pour créer la DB

Ouvrez d'abord un terminal CMD dans **P\_DB-106** et tapez la commande suivante **docker exec -it db /bin/bash** (après avoir démarrer Docker Desktop) pour pouvoir ouvrir le terminal du container docker.

Ensuite accédez au répertoire `/var/lib/mysql-files` avec la commande **cd /var/lib/mysql-files**. Vous pourrez constater avec **ls**, que vous vous trouvez dans le même dossier que `P_DB-106/scripts`. (Le dossier local est lié avec le container.)

```
C:\Users\willi\Documents\P_DB-106>docker exec -it db /bin/bash
bash-4.4# cd /var/lib/mysql-files
bash-4.4# ls
DATA  Scripts
bash-4.4#
```

Entrez ensuite la commande **cd /var/lib/mysql-files/Scripts/Create-DB-Tables/** pour aller chercher le script de création de la base de données et des tables. Vous pourrez voir le script en tapant **ls**. Pour importer le script il suffit d'entrer **mysql -u root -proot < db\_pizzeria-Create-DB-Tables.sql**. On précise le user et le mot de passe puis on importe le fichier SQL (le sens du chevron < doit être correct.)

```
bash-4.4# cd /var/lib/mysql-files/Scripts/Create-DB-Tables/
bash-4.4# ls
OLD db_pizzeria-Create-DB-Tables.sql
bash-4.4# mysql -u root -proot < db_pizzeria-Create-DB-Tables.sql
mysql: [Warning] Using a password on the command line interface can be insecure.
bash-4.4#
```

Pour vérifier la création de la base de données et des tables, il suffit de se connecter au serveur SQL avec l'utilisateur root (**mysql -u root -proot**). On peut ensuite taper la commande **SHOW DATABASES;** pour pouvoir voir les bases de données existantes. Il faudra ensuite utiliser la base de données db\_pizzeria avec **USE** puis finalement pouvoir voir les tables avec **SHOW TABLES;**

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| db_pizzeria |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.01 sec)

mysql> USE db_pizzeria
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES;
+-----+
| Tables_in_db_pizzeria |
+-----+
| t_adresse |
| t_article |
| t_avoir |
| t_client |
| t_commande |
| t_lignecommande |
| t_livraison |
| t_livreur |
| t_paiement |
| t_vivre |
+-----+
11 rows in set (0.00 sec)

mysql>
```

Il est également possible de voir s'il y a des données avec **SELECT \***

```
mysql> SELECT * FROM t_client;
Empty set (0.00 sec)

mysql> |
```

### 3.5 Marche à suivre pour importer les données

Rendez-vous dans le terminal bash du container, (on peut sortir de MySQL avec **exit**), puis rendez-vous dans le répertoire qui contient le fichier des LOAD DATA avec **cd /var/lib/mysql-files/Scripts/LoadData** puis tapez **ls** pour pouvoir voir le fichier db\_pizzeria-LOAD\_DATA.sql. Il est donc possible d'importer ce fichier avec la même commande que pour importer la création de la DB et des tables :

```
bash-4.4# cd /var/lib/mysql-files/Scripts/LoadData
bash-4.4# ls
db_pizzeria-LOAD_DATA.sql
bash-4.4# mysql -u root -proot < db_pizzeria-LOAD_DATA.sql
mysql: [Warning] Using a password on the command line interface can be insecure.
bash-4.4# |
```

On peut donc ensuite vérifier que les données ont bien été importée en refaisant la même marche à suivre :

```
mysql> USE db_pizzeria
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT * FROM t_client;
```

client_id	nom	prenom	email	num_tel
1	Durand	Liam	liam.durand@example.ch	+41 79 859 45 41
2	Rossi	Ethan	ethan.rossi@example.ch	+41 79 854 23 96
3	Morel	Tom	tom.morel@example.ch	+41 79 704 64 14
4	Morel	Lina	lina.morel@example.ch	+41 79 323 39 74
5	Durand	Lina	lina.durand@example.ch	+41 79 674 35 93
6	Berger	Tom	tom.berger@example.ch	+41 79 325 67 85
7	Durand	Luca	luca.durand@example.ch	+41 79 877 30 99
8	Lambert	Elya	elya.lambert@example.ch	+41 79 384 29 37
9	Favre	Nina	nina.favre@example.ch	+41 79 194 58 22
10	Favrod	Jules	jules.favrod@example.ch	+41 79 718 43 15
11	Meyer	Sofia	sofia.meyer@example.ch	+41 79 227 58 20
12	Girard	Tom	tom.girard@example.ch	+41 79 949 90 89
13	Fischer	Jules	jules.fischer@example.ch	+41 79 296 18 15
14	Girard	Ethan	ethan.girard@example.ch	+41 79 181 39 22
15	Schmid	Lina	lina.schmid@example.ch	+41 79 564 91 56
16	Favrod	Leo	leo.favrod@example.ch	+41 79 463 36 95
17	Morel	Luca	luca.morel@example.ch	+41 79 723 91 31
18	Mercier	Tom	tom.mercier@example.ch	+41 79 267 69 58

### 3.6 Création du script pour créer les utilisateurs et les rôles

Pour créer les utilisateurs et les rôles, on peut servir les commande **CREATE ROLE** et **CREATE USER**. On peut également utiliser la commande **GRANT** pour pouvoir attribuer des droits. Voici un bout du fichier :

```
-- Créer le role pizzaiolo
CREATE ROLE 'pizzaiolos_pizzeria';

-- Lecture des commandes
GRANT SELECT ON db_pizzeria.t_commande TO 'pizzaiolos_pizzeria';|
```

Ici on a créé un nouveau rôle pour les pizzaiolos, puis on définit avec GRANT les droits qu'a le rôle. On définit ici que les pizzaiolos ont le droit de voir (SELECT) la table t\_commande de la base de données db\_pizzeria. On peut gérer très finement les droits, comme ici avec un seul champ que les pizzaiolos peuvent modifier :

```
-- Mise à jour du statut des commandes
GRANT UPDATE (statusCommande) ON db_pizzeria.t_commande TO 'pizzaiolos_pizzeria';
```

Maintenant pour créer les utilisateurs et leur attribuer un rôle, cela est également similaire :

```
-- Utilisateur Pizzaiolo
CREATE USER 'antonio_pizzeria'@'localhost' IDENTIFIED BY 'PizzaioloPizzThanos!';
GRANT 'pizzaiolos_pizzeria' TO 'antonio_pizzeria'@'localhost';
SET DEFAULT ROLE 'pizzaiolos_pizzeria' TO 'antonio_pizzeria'@'localhost';
```

On définit le nom de l'utilisateur (chaque utilisateur est unique dans le serveur de la base de données, c'est pour cela qu'il est préfixé par \_pizzeria) ainsi que le serveur (ici en localhost) puis on utilise IDENTIFIED BY pour entrer son mot de passe. (Il faut faire attention car il est en clair.) Puis on attribue le rôle souhaité à l'utilisateur avec GRANT. On peut ensuite définir le rôle par défaut de l'utilisateur avec SET DEFAULT ROLE. (Pour ce cas il y a qu'un seul rôle mais ce sont des bonnes manières,

tout comme utiliser **FLUSH PRIVILEGES**; à la fin du script pour reload les privilèges de tout le monde.)

### 3.7 Marche à suivre pour créer les utilisateurs et les rôles

Rendez-vous dans le terminal bash du container (comme expliqué au chapitre 3.4), puis tapez la commande **cd /var/lib/mysql-files/Scripts/Create-Roles-Users** pour pouvoir trouver le script qui permet de créer les rôles et les utilisateurs. Vous pouvez toujours taper **ls** pour bien vérifier que le fichier est présent avant de lancer la commande d'importation suivante **mysql -u root -proot < db\_pizzeria-Create-Roles-Users.sql**

```
bash-4.4# cd /var/lib/mysql-files/Scripts/Create-Roles-Users
bash-4.4# ls
db_pizzeria-Create-Roles-Users.sql
bash-4.4# mysql -u root -proot < db_pizzeria-Create-Roles-Users.sql
mysql: [Warning] Using a password on the command line interface can be insecure.
bash-4.4#
```

On peut ensuite tenter de se connecter au serveur directement avec un nouvel utilisateur créé.

```
bash-4.4# mysql -u nicolas_pizzeria -pAnalystePizzThanos!
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.0.30 MySQL Community Server - GPL
```

Puis vérifier qu'il possède bien les droits qu'ils lui ont été attribués, en entant une commande que l'utilisateur n'est pas autorisé à effectuer.

```
mysql> USE db_pizzeria
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> DROP TABLE t_client;
ERROR 1142 (42000): DROP command denied to user 'nicolas_pizzeria'@'localhost' for table 't_client'
mysql>
```

On peut également vérifier la liste de tous les utilisateurs et des rôles du serveur (en utilisant l'utilisateur root) :

```
mysql> SELECT User
-> FROM mysql.user;
+-----+
| User |
+-----+
| administrateurs_pizzeria |
| agentscaisse_pizzeria   |
| analystes_pizzeria      |
| db_user                 |
| livreurs_pizzeria       |
| managers_pizzeria       |
| pizzaiolos_pizzeria     |
| root                    |
| antonio_pizzeria        |
| jean_pizzeria           |
| laura_pizzeria          |
| luca_pizzeria           |
| marco_pizzeria          |
| mysql.infoschema        |
| mysql.session           |
| mysql.sys               |
| nicolas_pizzeria        |
| root                    |
+-----+
18 rows in set (0.00 sec)

mysql>
```

### 3.8 Requêtes SQL

On peut commencer par se connecter avec l'utilisateur root ou admin pour pouvoir effectuer ces requêtes. (On peut se connecter à l'utilisateur admin avec `mysql -u jean_pizzeria -pAdminPizzThanos!`) Cela permet de vérifier que l'utilisateur a les droits SELECT sur les tables. (Il faut pas oublier de taper la commande `USE db_pizzeria` pour utiliser la bonne DB.)

Requête n°1 :

Afficher les dix pizzas les plus vendues (sans les toppings), triés par quantités totales décroissantes. Vous devez afficher le nom et les quantités.

```
SELECT a.nom AS 'Pizza', SUM(lc.quantite) AS 'Nombres vendues'
FROM t_lignecommande lc
JOIN t_article a ON lc.article_fk = a.article_id
WHERE a.typeArticle = 'pizza'
AND lc.t_lignecommande_fk IS NULL
GROUP BY a.nom
ORDER BY SUM(lc.quantite) DESC
LIMIT 10;
```

Explication de la requête :

**(SELECT a.nom AS 'Pizza', SUM(lc.quantite) AS 'Nombres vendues')**

a.nom permet de sélectionner le nom de la pizza. SUM(lc.quantite) calcule le nombre total de pizza vendues. Les alias AS permettent de renommer les colonnes pour un meilleur affichage.

**FROM t\_lignecommande lc**

**JOIN t\_article a ON lc.article\_fk = a.article\_id**

On définit ensuite la table de départ avec FROM, puis on spécifie le nom de son alias. On peut ensuite lier la table des articles avec JOIN, puis en lui disant comment joindre la table avec **lc.article\_fk = a.article\_id**. Cela permet de relier chaque ligne de commande à l'article correspondant avec la clé étrangère.

**WHERE a.typeArticle = 'pizza'**

**AND lc.t\_lignecommande\_fk IS NULL**

On ne sélectionne que les types d'articles qui sont des pizzas, et on vérifie également de ne pas ajouter les toppings à la liste en vérifiant que la FK de la table t\_lignecommande\_fk est nulle.

**GROUP BY a.nom**

On groupe ensuite les lignes par les noms des pizzas pour pouvoir additionner les quantités vendues pour chacune.

**ORDER BY SUM(lc.quantite) DESC**

On classe ensuite les résultats par ordre décroissant du nombre de pizzas vendues, en spécifiant que le champ lc.quantite doit être trié par ordre décroissant. (DESC)

**LIMIT 10 ;**

On affiche uniquement les 10 pizzas

Résultat :

```
mysql> SELECT a.nom AS 'Pizza', SUM(lc.quantite) AS 'Nombres vendues'
-> FROM t_lignecommande lc
-> JOIN t_article a ON lc.article_fk = a.article_id
-> WHERE a.typeArticle = 'pizza'
-> AND lc.t_lignecommande_fk IS NULL
-> GROUP BY a.nom
-> ORDER BY SUM(lc.quantite) DESC
-> LIMIT 10;
```

Pizza	Nombres vendues
Margherita	176
Reine	167
Napolitaine	165
4 Fromages	164
Hawaïenne	164
Végétarienne	156
Parmigiana	150
Diavola	127

```
8 rows in set (0.00 sec)

mysql> |
```

Requête n°2 :

Afficher les toppings les plus ajoutés. Le résultat doit être ordonné par le nombre de toppings de manière décroissante. Vous devez afficher le nom et le nombre.

```
SELECT a.nom AS 'Nom du topping', COUNT(*) AS 'Nombre d ajouts'
FROM t_lignecommande lc
JOIN t_article a ON lc.article_fk = a.article_id
WHERE a.typeArticle = 'topping'
AND lc.t_lignecommande_fk IS NOT NULL
GROUP BY a.nom
ORDER BY COUNT(*) DESC;
```

Explications de la requête :

On sélectionne (comme pour la première requête), le nom de l'article. (Avec un alias Nom du topping pour pouvoir renommer la colonne) Puis on sélectionne COUNT(\*) qui va permettre de compter le nombre de fois où l'article apparaît. (Avec un alias.) On utilise également la même table de départ et la même jointure. (qui ont été expliqué plus haut.) On filtre ensuite les résultats avec WHERE a.typeArticle = 'topping' qui permet d'afficher uniquement les types d'article qui sont des toppings, puis on filtre lc.t\_lignecommande\_fk pour qu'il soit IS NOT NULL.

(Toppings uniquement). On groupe ensuite les résultats par le nom de l'article (a.nom), puis on trie les résultats par nombre d'ajout décroissant, avec ORDER BY COUNT(\*) DESC;

Résultat :

```
mysql> SELECT a.nom AS 'Nom du topping', COUNT(*) AS 'Nombre d ajouts'
-> FROM t_lignecommande lc
-> JOIN t_article a ON lc.article_fk = a.article_id
-> WHERE a.typeArticle = 'topping'
-> AND lc.t_lignecommande_fk IS NOT NULL
-> GROUP BY a.nom
-> ORDER BY COUNT(*) DESC;
+-----+-----+
| Nom du topping | Nombre d ajouts |
+-----+-----+
| Oignons       | 68              |
| Olives        | 59              |
| Poivrons      | 56              |
| Jambon cru    | 54              |
| Parmesan      | 51              |
| Chorizo       | 50              |
| Jambon        | 48              |
| Artichauts    | 47              |
| Champignons   | 46              |
| Anchois       | 45              |
| Ananas        | 41              |
| Fromage bleu  | 41              |
+-----+-----+
12 rows in set (0.00 sec)

mysql> |
```

Requête n°3 :

Afficher le chiffre d'affaires par jour (commandes livrées). Vous ne devez afficher que la date et le chiffre d'affaires (arrondi à 2 chiffres après la virgule).

```
SELECT DATE(c.dateHeure) AS 'Date', ROUND(SUM(lc.quantite * lc.prixUnitaire), 2) AS 'Chiffre d affaires'
FROM t_commande c
JOIN t_lignecommande lc ON c.commande_id = lc.commande_fk
WHERE c.statusCommande LIKE 'livr_e'
GROUP BY DATE(c.dateHeure)
ORDER BY DATE(c.dateHeure);
```

Explications de la requête :

On sélectionne la date présente dans le champ c.dateHeure avec SELECT DATE(c.dateHeure) AS 'Date', puis on sélectionne le chiffre d'affaires avec SUM(lc.quantite \* lc.prixUnitaire) qui permet de faire la somme de la multiplication du prix unitaire et de la quantité. La commande entière est ROUND(SUM(lc.quantite \* lc.prixUnitaire), 2) Ce qui permet d'arrondir la somme

avec 2 chiffres après la virgule. Puis on sélectionne la table de départ, (t\_comamnde) par exemple et on joint la table t\_lignecommande avec la bonne condition pour la jointure. (c.commande\_id = lc.commande\_fk). On filtre les résultats par le statuts de commandes qui est égal à livrée. LIKE 'livr\_e' est utilisé car il n'est pas possible d'entrer des accents dans la console. Puis finalement on groupe et on ordonne les résultat avec DATE(c.dateHeure).

Résultat :

```
mysql> SELECT DATE(c.dateHeure) AS 'Date', ROUND(SUM(lc.quantite * lc.prixUnitaire), 2) AS 'Chiffre d affaires'
-> FROM t_commande c
-> JOIN t_lignecommande lc ON c.commande_id = lc.commande_fk
-> WHERE c.statusCommande LIKE 'livr_e'
-> GROUP BY DATE(c.dateHeure)
-> ORDER BY DATE(c.dateHeure);
```

Date	Chiffre d affaires
2025-01-15	337.00
2025-01-16	402.00
2025-01-17	302.50
2025-01-18	407.00
2025-01-19	1149.00
2025-01-20	721.00
2025-01-21	307.50
2025-01-22	616.00
2025-01-23	639.00
2025-01-24	681.50
2025-01-25	557.50
2025-01-26	406.50
2025-01-27	626.00
2025-01-28	573.50
2025-01-29	493.50
2025-01-30	940.00
2025-01-31	653.00
2025-02-01	717.50
2025-02-02	564.50
2025-02-03	933.00
2025-02-04	521.00
2025-02-05	560.00
2025-02-06	786.00
2025-02-07	395.00
2025-02-08	815.50
2025-02-09	314.00
2025-02-10	481.50
2025-02-11	919.50
2025-02-12	547.00
2025-02-13	760.50
2025-02-14	609.00
2025-02-15	707.00
2025-02-16	587.50
2025-02-17	717.50
2025-02-18	655.50
2025-02-19	479.00
2025-02-20	460.00
2025-02-21	402.50
2025-02-22	588.00
2025-02-23	893.00
2025-02-24	485.50
2025-02-25	520.50
2025-02-26	468.00
2025-02-27	244.50
2025-02-28	216.00

45 rows in set (0.01 sec)

Requête n°4 :

Afficher le chiffre d'affaires par NPA (adresse de livraison).

1<sup>ère</sup> colonne : npa

2<sup>ème</sup> colonne : localité

3<sup>ème</sup> colonne : chiffre d'affaires (arrondi à 2 chiffres après la virgule)

```
SELECT a.NPA AS 'NPA', a.localite AS 'Localite', ROUND(SUM(lc.quantite * lc.prixUnitaire), 2) AS 'Chiffre d affaires'
FROM t_commande c
JOIN t_adresse a ON c.adresse_fk = a.adresse_id
JOIN t_lignecommande lc ON c.commande_id = lc.commande_fk
WHERE c.adresse_fk IS NOT NULL
GROUP BY a.NPA, a.localite
ORDER BY SUM(lc.quantite * lc.prixUnitaire) DESC;
```

Explications de la requête :

On sélectionne le NPA et la localité avec a.NPA AS 'NPA', a.localite AS 'Localite' puis on sélectionne également le chiffre d'affaire arrondi à 2 chiffres après la virgule. (Même sélection que la requête précédente) Puis on pars de la table t\_commande et on lie les tables t\_adresse et t\_lignecommande avec des JOIN. (Il faut bien vérifier que la PK est liée à la FK). On filtre ensuite les résultat avec WHERE c.adresse\_fk IS NOT NULL pour vérifier que les adresses ne sont pas NULL, puis on groupe les résultats par a.NPA et a.localite, car on doit citer tout ce qui n'est pas une fonction d'agrégation. Puis on les classe avec le chiffre d'affaire décroissant (DESC).

Résultat :

```
mysql> SELECT a.NPA AS 'NPA', a.localite AS 'Localite', ROUND(SUM(lc.quantite * lc.prixUnitaire), 2) AS 'Chiffre d affaires'
-> FROM t_commande c
-> JOIN t_adresse a ON c.adresse_fk = a.adresse_id
-> JOIN t_lignecommande lc ON c.commande_id = lc.commande_fk
-> WHERE c.adresse_fk IS NOT NULL
-> GROUP BY a.NPA, a.localite
-> ORDER BY SUM(lc.quantite * lc.prixUnitaire) DESC;
```

NPA	Localite	Chiffre d affaires
1022	Chavannes-près-Renens	2116.00
1007	Lausanne	1729.00
1003	Lausanne	1522.50
1005	Lausanne	1371.00
1004	Lausanne	1236.50
1008	Jouxteins-Mozery	1122.00
1023	Crissier	938.00
1020	Renens VD	853.50

8 rows in set (0.00 sec)

Requête n°5 :

Affiche le nombre de commandes par heure. Il s'agit par cette requête de savoir quelles sont les heures « chaudes ».

NB : les heures « chaudes » sont des heures pendant lesquelles le nombre de commandes sont les plus élevées.

```
SELECT HOUR(c.dateHeure) AS 'Heure', COUNT(*) AS 'Nombre de commandes'
FROM t_commande c
GROUP BY HOUR(c.dateHeure)
ORDER BY COUNT(*) DESC;
```

Explications de la requête :

On sélectionne cette fois les heures avec `SELECT HOUR(c.dateHeure) AS 'Heure'`  
On utilise `COUNT(*) AS 'Nombre de commandes'` pour pouvoir compter le nombre de ligne présente dans la table `t_commande`. (Il y a ensuite `FROM t_commande c`), puis on groupe les résultats par les heures, et on les classe avec le nombre de commande décroissant. (`ORDER BY COUNT(*) DESC;`)

Résultat :

```
mysql> SELECT HOUR(c.dateHeure) AS 'Heure', COUNT(*) AS 'Nombre de commandes'
-> FROM t_commande c
-> GROUP BY HOUR(c.dateHeure)
-> ORDER BY COUNT(*) DESC;
```

Heure	Nombre de commandes
12	31
17	30
4	29
23	27
13	26
7	26
10	26
2	25
8	24
6	24
1	23
5	23
18	23
22	23
16	22
11	21
21	21
0	20
15	20
3	18
14	18
19	17
9	17
20	16

```
24 rows in set (0.00 sec)

mysql> |
```

Requête n°6:

Afficher le nombre de commandes des clients les plus fidèles. Un client est fidèle si son nombre de commandes est  $\geq 5$ . Afficher le résultat par ordre décroissant du nombre de commandes, puis par ordre alphabétique du nom.

```
SELECT cl.nom AS 'Nom', cl.prenom AS 'Prenom', COUNT(c.commande_id) AS 'Nombre de commandes'
FROM t_client cl
JOIN t_commande c ON cl.client_id = c.client_fk
GROUP BY cl.client_id, cl.nom, cl.prenom
HAVING COUNT(c.commande_id) >= 5
ORDER BY COUNT(c.commande_id) DESC, cl.nom ASC;
```

Explications de la requête :

On sélectionne le prénom et le nom du client, puis on compte le nombre de commandes avec `COUNT(c.commande_id) AS 'Nombre de commandes'`. (Cela a peu d'importance d'inverser des table dans le FROM, car les conditions de jointures restent les mêmes.) On part de la table `t_client` puis on joint la table `t_commande`. On groupe ensuite les résultats, (Comme expliqué à la requête 4, on doit mettre tout ce qui n'est pas une fonction d'agrégation dans le GROUP BY. On groupe également avec `cl.client_id` pour garantir que chaque client est unique.) Finalement on met une condition sur la fonction d'agrégation, `HAVING COUNT(c.commande_id) >= 5` pour pouvoir filtrer après le GROUP BY. (On compte les commandes et on garde uniquement les clients s'il ont commandés 5 commandes ou plus). On ordonne finalement le résultat par le nombre de commandes décroissant (`COUNT(c.commande_id) DESC`) et par le nom du client, `cl.nom ASC` (ASC est optionnel, car il permet de classer par ordre alphabétique ce qui est déjà le cas par défaut.)

Résultat :

```
mysql> SELECT cl.nom AS 'Nom', cl.prenom AS 'Prenom', COUNT(c.commande_id) AS 'Nombre de commandes'
-> FROM t_client cl
-> JOIN t_commande c ON cl.client_id = c.client_fk
-> GROUP BY cl.client_id, cl.nom, cl.prenom
-> HAVING COUNT(c.commande_id) >= 5
-> ORDER BY COUNT(c.commande_id) DESC, cl.nom ASC;
```

Nom	Prenom	Nombre de commandes
Girard	Chloé	8
Berset	Luca	7
Favre	Liam	7
Girard	Leo	7
Rossi	Luca	7
Berset	Noah	6
Favrod	Jules	6
Favrod	Arthur	6
Gonzalez	Sofia	6
Morel	Nina	6
Perrin	Zoé	6
Aubert	Nina	5
Berger	Luca	5
Durand	Luca	5
Girard	Mia	5
Girard	Camille	5
Gonzalez	Lina	5
Martin	Luca	5
Meyer	Arthur	5
Meyer	Mia	5
Morel	Sofia	5
Perrin	Emma	5

```
22 rows in set (0.01 sec)

mysql> |
```

#### Requête n°7:

Afficher le total dû par commande. Afficher l'id de la commande et le montant dû (arrondi à 2 chiffres après la virgule). Ordonnez le résultat par ordre croissant des ids de commandes.

```
SELECT c.commande_id AS 'ID commande', ROUND(SUM(lc.quantite * lc.prixUnitaire), 2) AS 'Montant du'
FROM t_commande c
JOIN t_lignecommande lc ON c.commande_id = lc.commande_fk
GROUP BY c.commande_id
ORDER BY c.commande_id ASC;
```

Explications de la requête :

On sélectionne l'id des commandes, c.commande\_id AS 'ID commande' et le montant dû avec ROUND(SUM(lc.quantite \* lc.prixUnitaire), 2) comme pour le chiffre d'affaires précédemment. On part de la table t\_commande, on fait un JOIN pour joindre la table t\_lignecommande (Ce qui est la même logique à chaque fois). On groupe par les ids de commande, chaque commande correspond à une ligne. Puis on trie les résultats par ids de commande croissant (ASC).

Le résultat en entier est trop grand pour être affiché car il fait 550 lignes) :

```
mysql> SELECT c.commande_id AS 'ID commande', ROUND(SUM(lc.quantite * lc.prixUnitaire), 2) AS 'Montant du'
-> FROM t_commande c
-> JOIN t_lignecommande lc ON c.commande_id = lc.commande_fk
-> GROUP BY c.commande_id
-> ORDER BY c.commande_id ASC;
```

ID commande	Montant du
1	48.00
2	63.50
3	44.50
4	62.50
5	69.50
6	49.00
7	65.00
8	27.00
9	49.50
10	46.00
11	64.50
12	62.00
13	27.50
14	61.50
15	23.00
16	41.00
17	38.00
18	59.00
19	69.00
20	18.50
21	52.50
22	66.00
23	52.00
24	45.00
25	58.00
26	66.50
27	68.50
28	56.00
29	62.50
30	50.50

#### Requête n°8:

Afficher le total payé par commande (commande ayant au moins un paiement).  
Afficher l'id de la commande et le total payé (arrondi à 2 chiffres après la virgule).  
Ordonnez le résultat par ordre croissant des ids de commandes.

```
SELECT c.commande_id AS 'ID commande', ROUND(SUM(p.montant), 2) AS 'Total paye'
FROM t_commande c
JOIN t_paiement p ON c.commande_id = p.commande_fk
GROUP BY c.commande_id
ORDER BY c.commande_id ASC;
```

#### Explications de la requête :

Cette requête est très similaire à la précédente. On sélectionne l'id de la commande et le total payé avec `ROUND(SUM(p.montant), 2) AS 'Total paye'`. Il suffit alors de remplacer la table jointe par `t_paiement` (si l'on regarde la requête précédente) et d'adapter la condition de jointure. Le reste de la commande est la même chose.

Résultat (Trop grand pour être affiché car il y a 550 lignes) :

```
mysql> SELECT c.commande_id AS 'ID commande', ROUND(SUM(p.montant), 2) AS 'Total paye'
-> FROM t_commande c
-> JOIN t_paiement p ON c.commande_id = p.commande_fk
-> GROUP BY c.commande_id
-> ORDER BY c.commande_id ASC;
```

ID commande	Total paye
1	48.00
2	63.50
3	44.50
4	62.50
5	69.50
6	49.00
7	32.50
8	27.00
9	49.50
10	46.00
11	64.50
12	62.00
13	27.50
14	61.50
15	23.00
16	41.00
17	38.00
18	59.00
19	69.00
20	18.50
21	0.00
22	66.00
23	52.00
24	45.00
25	58.00
26	66.50
27	68.50
28	56.00
29	62.50
30	50.50

Requête n°9:

Quelle est la répartition des types de commandes.

Ordonner le résultat par le nombre de commande de chaque type, du plus grand au plus petit.

1ère colonne : type

2ème colonne : nombre de commandes de ce type

```
SELECT c.typeCommande AS 'Type', COUNT(*) AS 'Nombre de commandes'
FROM t_commande c
GROUP BY c.typeCommande
ORDER BY COUNT(*) DESC;
```

Explications de la requête :

On sélectionne le type de commande avec c.typeCommande AS 'Type' ainsi que le nombre de commandes avec COUNT(\*) AS 'Nombre de commandes' pour compter les lignes de la table t\_commande du quel on pars. (FROM t\_commande)

Puis on groupe le résultat par le type de commande pour compter le nombre de commande pour chaque type, puis on trie le résultat par nombre de commande décroissant. (ORDER BY COUNT(\*) DESC)

Résultat :

```
mysql> SELECT c.typeCommande AS 'Type', COUNT(*) AS 'Nombre de commandes'
-> FROM t_commande c
-> GROUP BY c.typeCommande
-> ORDER BY COUNT(*) DESC;
+-----+-----+
| Type      | Nombre de commandes |
+-----+-----+
| livraison |          219         |
| emporter  |          197         |
| sur_place |          134         |
+-----+-----+
3 rows in set (0.00 sec)

mysql> |
```

Requête n°10:

Quel est le délai moyen de livraison par livreur (en minutes).

Ordonner le résultat par délai moyen en minutes du plus petit au plus grand.

Aide : l'id du livreur, son nom et le délai dans le SELECT.

```
SELECT liv.livreur_id AS 'ID Livreur', liv.nom AS 'Nom',
ROUND(AVG(TIMESTAMPDIFF(MINUTE, l.depart, l.arrivee)), 0) AS 'Delai en minutes'
FROM t_livreur liv
JOIN t_livrer lv ON liv.livreur_id = lv.livreur_fk
JOIN t_livraison l ON lv.livraison_fk = l.livraison_id
GROUP BY liv.livreur_id, liv.nom
ORDER BY AVG(TIMESTAMPDIFF(MINUTE, l.depart, l.arrivee)) ASC;
```

Explications de la requête :

On sélectionne l'id et le nom du livreur, ainsi que le temps moyen de livraison par livreur en minutes avec `AVG(TIMESTAMPDIFF(MINUTE, l.depart, l.arrivee))` AVG permet de calculer une moyenne tandis que `TIMESTAMPDIFF(MINUTE, l.depart, l.arrivee)` calcule la différence de temps en minutes entre le départ et l'arrivée de

la livraison. Il y a également l'ajout de `ROUND(calcul, 0)` pour pouvoir arrondir le résultat à 0 chiffres après la virgule. (Permet d'avoir uniquement les minutes). On pars de la table `t_livreur` et on joint 2 tables `t_livrer` qui fait l'association entre `t_livreur` et `t_livraison`, puis `t_livraison`. Finalement, on regroupe avec `GROUP BY` les sélections qui ne sont pas des fonctions d'agrégation, puis on trie le résultat par la moyenne de temps (non arrondie), croissant pour que le livreur le plus rapide soie en haut de la liste.

Résultat :

```
mysql> SELECT liv.livreur_id AS 'ID Livreur', liv.nom AS 'Nom',
-> ROUND(AVG(TIMESTAMPDIFF(MINUTE, l.depart, l.arrivee)), 0) AS 'Delai en minutes'
-> FROM t_livreur liv
-> JOIN t_livrer lv ON liv.livreur_id = lv.livreur_fk
-> JOIN t_livraison l ON lv.livraison_fk = l.livraison_id
-> GROUP BY liv.livreur_id, liv.nom
-> ORDER BY AVG(TIMESTAMPDIFF(MINUTE, l.depart, l.arrivee)) ASC;
```

ID Livreur	Nom	Delai en minutes
LIV005	Lucas	12
LIV006	Nora	12
LIV001	Marco	12
LIV002	Sophie	13
LIV003	Yann	13
LIV004	Amina	13

```
6 rows in set (0.00 sec)

mysql> |
```

### 3.9 Création des index

Voici la 1<sup>ère</sup> requête à améliorer (adaptée pour correspondre au MPD)

```
SELECT c.commande_id, c.dateHeure AS 'Date creation',
c.statusCommande, cl.nom AS 'Client'
FROM t_commande AS c
JOIN t_client AS cl ON c.client_fk = cl.client_id
WHERE c.statusCommande = 'livrée'
AND c.dateHeure > '2025-01-01 00:00:00'
ORDER BY c.dateHeure DESC;
```

L'index créé pour améliorer la requête est un index composite :

```
mysql> CREATE INDEX idx_commande_statut_date
-> ON t_commande(statusCommande, dateHeure);
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Cet index est optimal pour le WHERE car il commence par statusCommande puis dateHeure (utilisée pour un range), ce qui permet d'éviter un scan complet de la table. Il y a pas besoin de créer un index supplémentaire pour autre chose. (Il y a déjà un index par défaut pour les FK)

Voici la 2<sup>ème</sup> requête (également adaptée pour correspondre au MPD)

```
SELECT a.NPA AS 'zone_npa', COUNT(c.commande_id) AS 'nb'
FROM t_commande AS c
JOIN t_adresse AS a ON c.adresse_fk = a.adresse_id
WHERE c.typeCommande = 'livraison'
AND HOUR(c.dateHeure) BETWEEN 18 AND 21
GROUP BY a.NPA
ORDER BY nb DESC;
```

L'index créé pour améliorer la requête est également un index composite :

```
mysql> CREATE INDEX idx_commande_type_hour
-> ON t_commande(typeCommande, (HOUR(dateHeure)));
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Cet index est optimal pour le WHERE, il commence par typeCommande, et MySQL peut filtrer rapidement toutes les lignes de type livraison. l'index contient également HOUR(dateHeure), comme cela MySQL peut limiter les lignes aux heures entre 18 et 21 directement grâce à l'index.

ATTENTION cependant avec l'utilisation HOUR(dateHeure), car la fonctionnalité de mettre un index avec la fonction HOUR est disponible à partir de MySQL 8. C'est donc déconseillée d'utiliser cet index si la version de MySQL n'est pas égale ou supérieur à 8. (Il est possible de voir la version avec SELECT VERSION(); )

(Les requêtes et les index sont disponibles dans /Scripts/Create-Index pour l'importation)

### 3.10 Vérifier que les requêtes utilisent les index

On peut utiliser **EXPLAIN** pour pouvoir voir si la requête utilise bien l'index créé. (Plan d'exécution.) EXPLAIN se place juste avant le SELECT de la requête. Voici une partie (le tout ne passerait pas) du EXPLAIN de la première requête avant la création de l'index :

```
mysql> EXPLAIN SELECT c.commande_id, c.dateHeure AS 'Date creation',
-> c.statusCommande, cl.nom AS 'Client'
-> FROM t_commande AS c
-> JOIN t_client AS cl ON c.client_fk = cl.client_id
-> WHERE c.statusCommande = 'livre'
-> AND c.dateHeure > '2025-01-01 00:00:00'
-> ORDER BY c.dateHeure DESC
-> ;
```

id	select_type	table	partitions	type	possible_keys	key
1	SIMPLE	c	NULL	ALL	client_fk	NULL
1	SIMPLE	cl	NULL	eq_ref	PRIMARY	PRIMARY

2 rows in set, 1 warning (0.01 sec)

On peut voir ici qu'il n'y a pas de clé pour la table c (t\_commande), ce qui signifie que MySQL va parcourir toute la table t\_commande. Les conditions sont utilisées après avoir lu toutes les lignes. Voici le EXPLAIN après la création de l'index :

```
mysql> EXPLAIN SELECT c.commande_id, c.dateHeure AS 'Date creation',
-> c.statusCommande, cl.nom AS 'Client'
-> FROM t_commande AS c
-> JOIN t_client AS cl ON c.client_fk = cl.client_id
-> WHERE c.statusCommande = 'livre'
-> AND c.dateHeure > '2025-01-01 00:00:00'
-> ORDER BY c.dateHeure DESC;
```

id	select_type	table	partitions	type	possible_keys	key
1	SIMPLE	c	NULL	range	client_fk,idx_commande_statut_date	idx_commande_statut_date
1	SIMPLE	cl	NULL	eq_ref	PRIMARY	PRIMARY

2 rows in set, 1 warning (0.00 sec)

On peut voir après que la requête utilise bien l'index idx\_commande\_statut\_date, et que la clé idx\_commande\_statut\_date est maintenant présente pour la table c. MySQL utilise maintenant l'index pour lire les lignes qui correspondent à statusCommande et dateHeure.

On peut faire de même pour la 2<sup>ème</sup> requête :

```
mysql> EXPLAIN SELECT a.NPA AS 'zone_npa', COUNT(c.commande_id) AS 'nb'
-> FROM t_commande AS c
-> JOIN t_adresse AS a ON c.adresse_fk = a.adresse_id
-> WHERE c.typeCommande = 'livraison'
-> AND HOUR(c.dateHeure) BETWEEN 18 AND 21
-> GROUP BY a.NPA
-> ORDER BY nb DESC;
```

id	select_type	table	partitions	type	possible_keys	key
1	SIMPLE	c	NULL	ALL	adresse_fk	NULL
1	SIMPLE	a	NULL	eq_ref	PRIMARY	PRIMARY

2 rows in set, 1 warning (0.01 sec)

Comme il n'y a pas de clé, MySQL parcourt toutes les lignes de `t_commande`, avant d'exécuter la condition `HOUR` sur chaque ligne. Voici le résultat après la création de l'index :

```
mysql> EXPLAIN SELECT a.NPA AS 'zone_npa', COUNT(c.commande_id) AS 'nb'
-> FROM t_commande AS c
-> JOIN t_adresse AS a ON c.adresse_fk = a.adresse_id
-> WHERE c.typeCommande = 'livraison'
-> AND HOUR(c.dateHeure) BETWEEN 18 AND 21
-> GROUP BY a.NPA
-> ORDER BY nb DESC;
```

id	select_type	table	partitions	type	possible_keys	key
1	SIMPLE	c	NULL	range	adresse_fk,idx_commande_type_hour	idx_commande_type_hour
1	SIMPLE	a	NULL	eq_ref	PRIMARY	PRIMARY

```
2 rows in set, 1 warning (0.00 sec)

mysql>
```

MySQL utilise l'index pour filtrer les lignes dès la lecture, et donc sélectionner uniquement les lignes que la requête demande.

### 3.11 Scénario de Transaction

Un client vient payer sa commande à la caisse. L'agent de caisse doit faire 2 choses en même temps. Il doit enregistrer le paiement dans `t_paiement` et changer le statut de la commande (de reçue à payée).

Le problème sans transaction est que si l'une des deux opérations échoue, peut se retrouver avec un paiement enregistré mais le statut n'est pas changé, ou alors l'inverse, un statut changé mais pas de paiement enregistré.

Le but est de faire en sorte que ces 2 opération réussissent ou échouent ensemble avec une transaction. La transaction se ferait sur les tables `t_paiement` et `t_commande`.

Voici les opérations à effectuer sur ces tables (dans l'ordre) :

- INSERT : Enregistrer le paiement (montant, mode, date, commande\_fk)
- UPDATE : Changer statusCommande (reçue en payée)

Les conditions de validations et d'annulation sont les suivantes :

Il y a un COMMIT si l'INSERT dans la table `t_paiement` réussit et que l'UPDATE du statut réussit. Il y a un ROLLBACK si l'INSERT dans `t_paiement` échoue, si l'UPDATE du statut échoue, ou s'il y a une erreur SQL.

Voici la transaction fonctionnelle :

```
SET AUTOCOMMIT = 0;

START TRANSACTION;

INSERT INTO t_paiement (modePaiement, montant, datePaiement, commande_fk)
VALUES ('carte', 48.00, NOW(), 3);

UPDATE t_commande
SET statusCommande = 'payée'
WHERE commande_id = 3;

COMMIT;
```

Explications de la transaction :

Le SET AUTOCOMMIT 0 permet d'éviter que MySQL applique automatiquement des modifications. Il est donc seulement possible de valider les modifications avec COMMIT. START TRANSACTION permet d'ouvrir une transaction, qui sera finie par COMMIT (validée) par ROLLBACK (non validée) ou par une erreur (pas de commit non plus donc non valide). Si il y a un ROLLBACK tout est annulé. On vérifie la bonne exécution des requêtes une par une, et si tout est bon et qu'il n'y a pas d'erreur on peut commit pour appliquer les modifications.

Voici le résultat de la première transaction qui est validée :

```
mysql> SET AUTOCOMMIT = 0;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> INSERT INTO t_paiement (modePaiement, montant, datePaiement, commande_fk)
-> VALUES ('carte', 48.00, NOW(), 3);
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> UPDATE t_commande
-> SET statusCommande = 'paye'
-> WHERE commande_id = 3;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1 Changed: 0 Warnings: 0

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT commande_id, statusCommande
-> FROM t_commande
-> WHERE commande_id = 3;
+-----+-----+
| commande_id | statusCommande |
+-----+-----+
|          3 | paye           |
+-----+-----+
1 row in set (0.00 sec)
```

Voici la transaction avec l'erreur provoquée d'un id inexistant.

(Le SET AUTOCOMMIT doit être égal à 0)

```
START TRANSACTION;

INSERT INTO t_paiement (modePaiement, montant, datePaiement, commande_fk)
VALUES ('especes', 50.00, NOW(), 99999);

UPDATE t_commande
SET statusCommande = 'payée'
WHERE commande_id = 7;

ROLLBACK;
```

Explications de la transaction :

(Le SET AUTOCOMMIT est pas obligatoire s'il a déjà été réglé précédemment). Dans le INSERT INTO on essaye d'ajouter des données alors que commande\_fk 99999 n'existe pas. Cela provoque donc une erreur, qui annule automatiquement la transaction. Comme il y a eu un UPDATE après l'erreur, c'est une nouvelle transaction qui commence, qui ne correspond pas à l'erreur précédente. C'est

pour cela qu'il y a un ROLLBACK à la fin qui annule l'UPDATE. (On met toutes les modifications après START TRANSACTION et on décide à la fin si on met COMMIT ou ROLLBACK pour valider ou non la transaction.)

Voici le résultat de la 2<sup>ème</sup> requête :

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> INSERT INTO t_paiement (modePaiement, montant, datePaiement, commande_fk)
-> VALUES ('especes', 50.00, NOW(), 99999);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
mysql>
mysql> UPDATE t_commande
-> SET statusCommande = 'paye'
-> WHERE commande_id = 7;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

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

mysql> SELECT commande_id, statusCommande
-> FROM t_commande
-> WHERE commande_id = 7;
+-----+-----+
| commande_id | statusCommande |
+-----+-----+
|          7 | annulé         |
+-----+-----+
1 row in set (0.00 sec)
```

La transaction fonctionne. Elle garantit que le paiement et le changement de statut se font toujours ensemble, évitant ainsi toute incohérence dans la base de données. En cas d'erreur, tout est automatiquement annulé.

(Les requêtes des transactions et les requêtes de vérifications des données sont disponibles dans /Scripts/Transactions)

## 3.12 Backup de la base de données

Ce chapitre est en cours de réalisation.

# 4 TESTS

## 4.1 Tableau des tests

Quand	Quoi	Comment	Validé
-------	------	---------	--------

10.12.25	Test de création d'une base de données	Création d'une DB avec CREATE DATABASE, suivi du MPD de looping.	<b>OK</b>
12.12.25	Validation MCD	Validation par le prof du MCD	<b>OK</b>
17.12.25	Analyse des données des fichiers TSV pour les adapter au MCD	En analysant les types de données et les données présentes	<b>OK</b>
20.12.25	Finalisation des modifications à apporter aux données pour que les données rentrent dans la DB.	En analysant les types de données et en modifiant les données et le MCD, pour que toutes les données puissent rentrer	<b>OK</b>
21.12.25	Création des scripts pour créer la DB et importer les données	En important le MPD fait par looping puis en créant des LOAD DATA	<b>OK</b>
21.12.25	Upload sur git des fichiers	En utilisant git bash	<b>OK</b>
21.12.25	Modification des volumes du docker compose pour que le LOAD DATA ne donne pas d'erreur d'importation avec --secure-file-priv.	En modifiant le docker compose et en spécifiant le chemin /var/lib/mysql-files dans le volume. (lié au dossier local)	<b>OK</b>

## 5 CONCLUSION

### 5.1 Bilan des fonctionnalités demandées

Le projet est en cours de réalisation.

### 5.2 Bilan personnel

Le projet est en cours de réalisation.

## 6 ANNEXES

[Lien du repo Git du projet](#)

