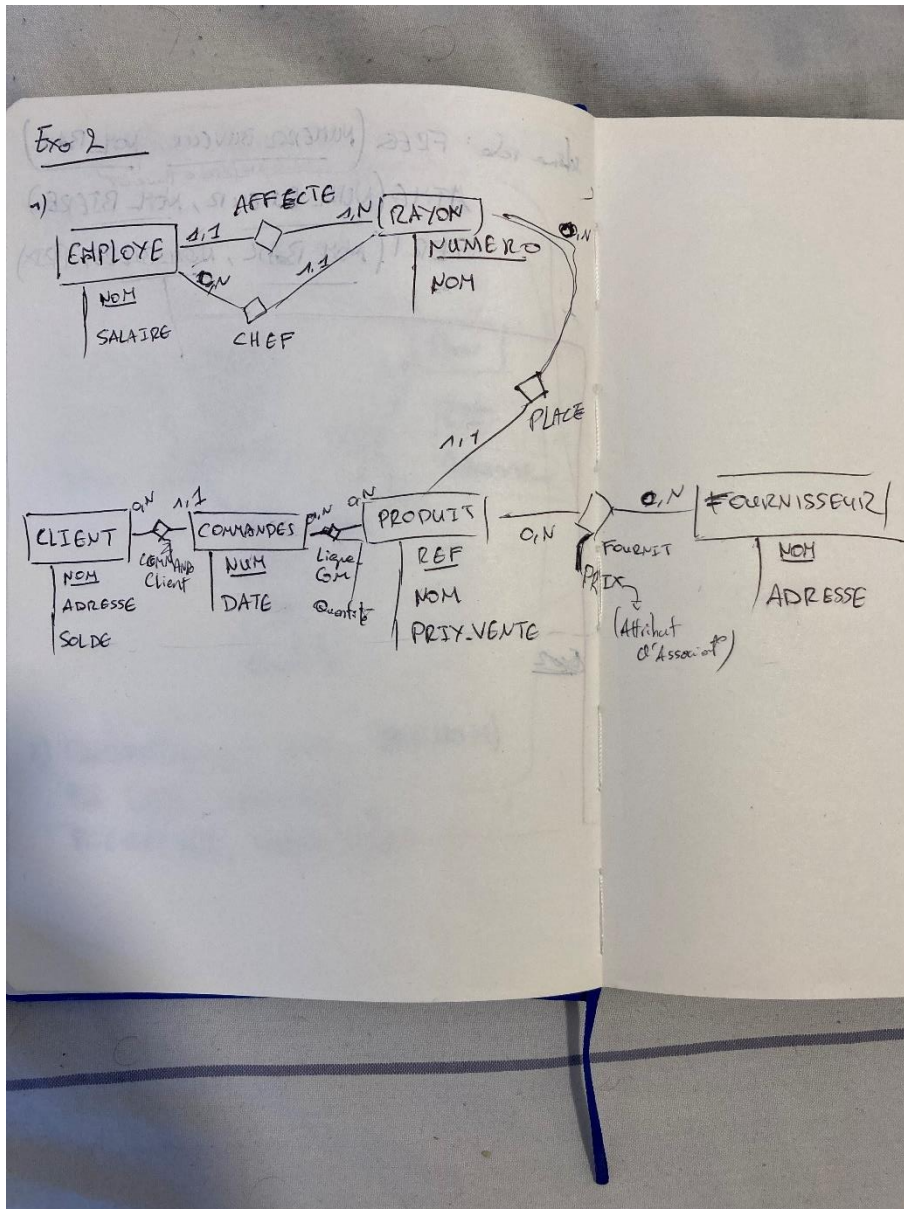
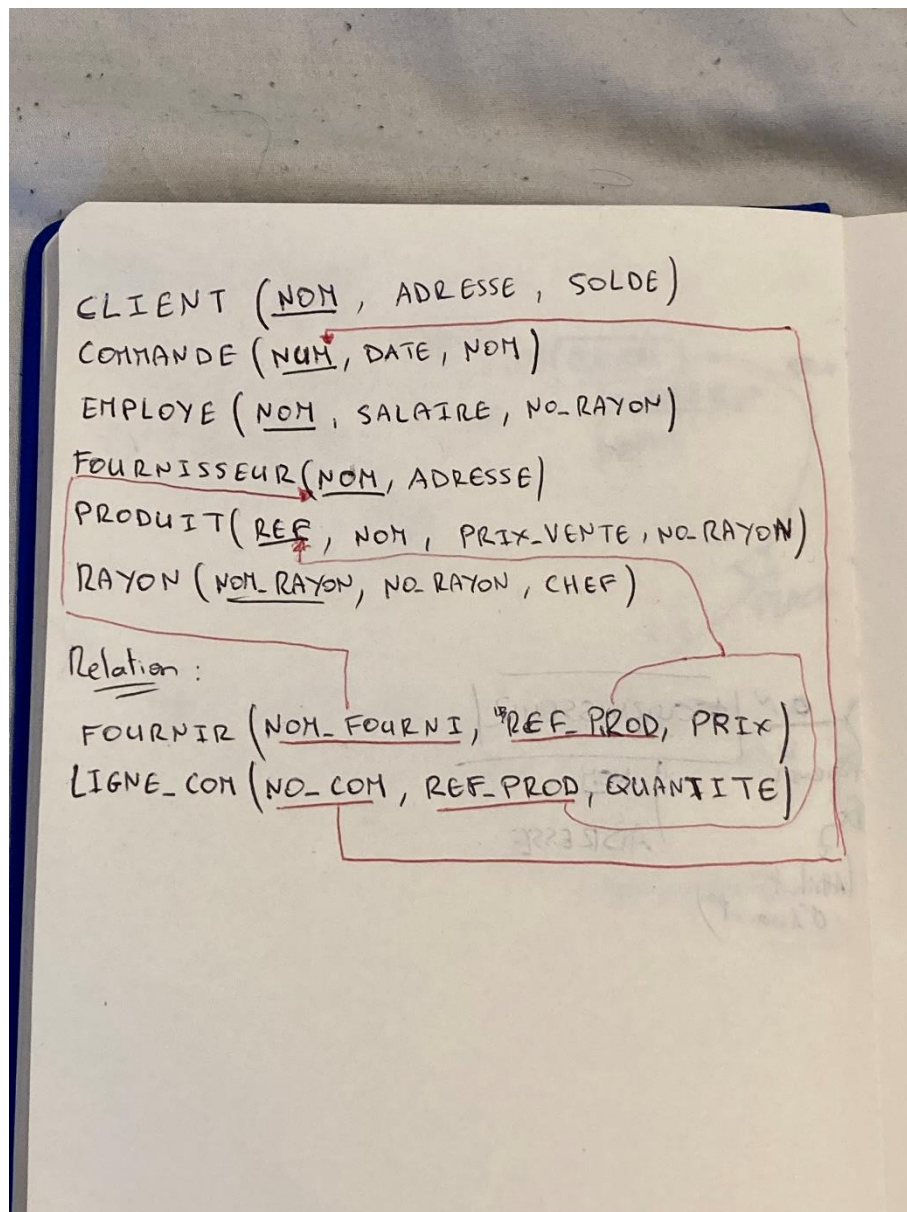


Q1)



Q2)



Q3 et Q4)

La commande sur le terminal sqlite3 pour créer la base de données :

```
.open Gestion_magasin.db
```

La commande pour créer les tables EMPLOIE (même principe pour les autres tables) et Fournir (même principe pur Ligne_com) :

```
CREATE TABLE "EMPLOIE" ("nom" TEXT, "salaire" INTEGER NOT NULL, PRIMARY KEY("nom"));
```

```
CREATE TABLE "Fournir" (
```

```
    "nom_four"    TEXT,
```

```
    "ref_prod"    INTEGER,
```

```
    "prix"        INTEGER,
```

```

        FOREIGN KEY("ref_prod") REFERENCES "PRODUIT"("ref"),

        FOREIGN KEY("nom_four") REFERENCES "FOURNISSEUR"("nom"),

        PRIMARY KEY("nom_four","ref_prod")

);

```

On vérifie chaque étape avec .schema et .tables

```

C:\Users\jrelh\Documents\SQL\sqlite3.exe
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open Gestion_magasin.db
sqlite> .databases
main: C:\Users\jrelh\Documents\SQL\Gestion_magasin.db r/w
sqlite> CREATE TABLE EMPLOYE (nom TEXT PRIMARY KEY, salaire INTEGER);
sqlite> .schema
CREATE TABLE EMPLOYE (nom TEXT PRIMARY KEY, salaire INTEGER);
sqlite> CREATE TABLE EMPLOYE (nom TEXT PRIMARY KEY, salaire INTEGER NOT NULL);
Error: table EMPLOYE already exists
sqlite> .schema
CREATE TABLE EMPLOYE (nom TEXT PRIMARY KEY, salaire INTEGER);
sqlite> CREATE TABLE COMMANDE (num INTEGER PRIMARY KEY, date NUMERIC);
sqlite> .schema
CREATE TABLE EMPLOYE (nom TEXT PRIMARY KEY, salaire INTEGER);
CREATE TABLE COMMANDE (num INTEGER PRIMARY KEY, date NUMERIC);
sqlite> CREATE TABLE CLIENT (nom TEXT PRIMARY KEY, adresse TEXT, solde INTEGER);
sqlite> CREATE TABLE FOURNISSEUR (nom TEXT PRIMARY KEY, adresse TEXT);
sqlite> CREATE TABLE PRODUIT (ref INTEGER, nom TEXT NOT NULL, prix_vente INTEGER NOT NULL);
sqlite> CREATE TABLE RAYON (numero INTEGER, nom TEXT NOT NULL)
...> ;
sqlite> .schema
CREATE TABLE EMPLOYE (nom TEXT PRIMARY KEY, salaire INTEGER);
CREATE TABLE COMMANDE (num INTEGER PRIMARY KEY, date NUMERIC);
CREATE TABLE CLIENT (nom TEXT PRIMARY KEY, adresse TEXT, solde INTEGER);
CREATE TABLE FOURNISSEUR (nom TEXT PRIMARY KEY, adresse TEXT);
CREATE TABLE PRODUIT (ref INTEGER, nom TEXT NOT NULL, prix_vente INTEGER NOT NULL);
CREATE TABLE RAYON (numero INTEGER, nom TEXT NOT NULL);
sqlite> .tables
CLIENT      COMMANDE     EMPLOYE      FOURNISSEUR  PRODUIT      RAYON
sqlite> .schema sqlite_master
CREATE TABLE sqlite_master (
  type text,
  name text,
  tbl_name text,
  rootpage integer,
  sql text
);
sqlite> .mode csv

```

Q5)

Commande pour remplir la table FOURNISSEUR (même principe pour les autres tables)

INSERT INTO FOURNISSEUR (nom,adresse)

Values ('f1', 'paris'), ('f2', 'lyon'), ('f3', 'marseille');

```

sqlite> INSERT INTO FOURNISSEUR(nom, adresse)
...> VALUES("f1", "paris"),("f2", "lyon"),("f3", "marseille");
sqlite> SELECT nom FROM FOURNISSEUR;
f1
f2
f3
sqlite>

```

Commande pour retrouver la vue avec l'année 2013

SELECT * FROM COMMANDE WHERE date LIKE '2013%';

Q6)

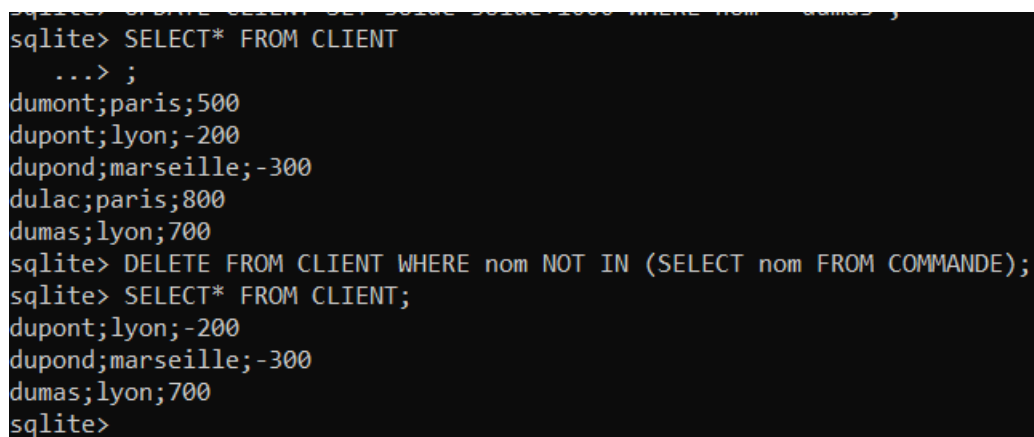
Augmenter de 1000 le solde de dumas :

```
UPDATE CLIENT SET solde = solde + 1000 WHERE nom='dumas' ;
```

Supprimer les clients n'ayant jamais passé de commande

```
DELETE FROM CLIENT WHERE nom NOT IN (SELECT nom FROM COMMANDE) ;
```

L'image ci-dessous montre le nouveau solde de dumas (700 alors qu'avant -300) et la suppression de dulac



```
sqlite> UPDATE CLIENT SET solde = solde + 1000 WHERE nom = 'dumas' ;
sqlite> SELECT* FROM CLIENT
...> ;
dumont;paris;500
dupont;lyon;-200
dupond;marseille;-300
dulac;paris;800
dumas;lyon;700
sqlite> DELETE FROM CLIENT WHERE nom NOT IN (SELECT nom FROM COMMANDE);
sqlite> SELECT* FROM CLIENT;
dupont;lyon;-200
dupond;marseille;-300
dumas;lyon;700
sqlite>
```

Les informations de dupont sont dans CLIENT, COMMANDE et Ligne_com

```
SELECT num FROM COMMANDE WHERE nom='dupont' ;
```

On obtient les num de commandes à supprimer dans LIGNE_com (1 et 3)

```
DELETE FROM CLIENT WHERE nom='dupont' ;
```

```
DELETE FROM COMMANDE WHERE nom='dupont' ;
```

```
DELETE FROM Ligne_com WHERE (no_com=1 OR no_com=3) ;
```

(capture d'écran manquante)

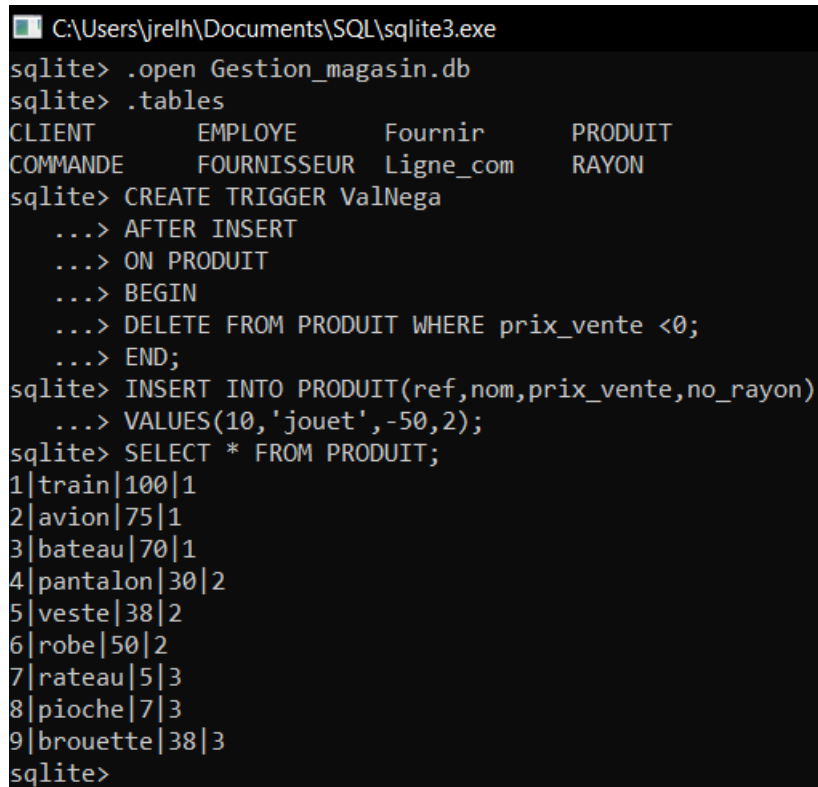
Q7)

Un trigger, également appelé déclencheur, permet d'exécuter un ensemble d'instruction SQL juste après un événement. Cela permet de faciliter et d'automatiser des actions au sein d'un Système de Gestion de Base de Données (SGBD).

Q9)

```
CREATE TRIGGER ValNega
AFTER INSERT
ON PRODUIT
BEGIN
DELETE FROM PRODUIT WHERE prix_vente <0 ;
END ;
```

Test avec un produit 'jouet' avec un pris de -50. On voit bien qu'il n'est pas dans la table PRODUIT



```
C:\Users\jrelh\Documents\SQL\sqlite3.exe
sqlite> .open Gestion_magasin.db
sqlite> .tables
CLIENT      EMPLOYE      Fournir      PRODUIT
COMMANDE    FOURNISSEUR  Ligne_com    RAYON
sqlite> CREATE TRIGGER ValNega
...> AFTER INSERT
...> ON PRODUIT
...> BEGIN
...> DELETE FROM PRODUIT WHERE prix_vente <0;
...> END;
sqlite> INSERT INTO PRODUIT(ref,nom,prix_vente,no_rayon)
...> VALUES(10,'jouet',-50,2);
sqlite> SELECT * FROM PRODUIT;
1|train|100|1
2|avion|75|1
3|bateau|70|1
4|pantalon|30|2
5|veste|38|2
6|robe|50|2
7|rateau|5|3
8|pioche|7|3
9|brouette|38|3
sqlite>
```