```sql
-- Supprimer table si elle existe
DROP TABLE IF EXISTS `aliment`;

-- Créer table
CREATE TABLE
    `aliment` (
        `id` int NOT NULL AUTO_INCREMENT,
        `nom` varchar(100) NOT NULL,
        `marque` varchar(100) DEFAULT NULL,
        `sucre` float DEFAULT NULL,
        `calories` int NOT NULL,
        `graisses` float DEFAULT NULL,
        `proteines` float DEFAULT NULL,
        `bio` tinyint(1) DEFAULT '0',
        PRIMARY KEY (`id`)
    ) ENGINE = InnoDB AUTO_INCREMENT = 21 DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4_general_ci;

-- C R U D --
-- CREATE READ UPDATE DELETE --

-- Insérer une entrée dans la table "aliment"
INSERT INTO `aliment`VALUES ( 1,  'pomme',  'sans marque',    19.1, 72, 0.2, 0.4,  0 );
--
INSERT INTO aliment ( `nom`,  `marque`, `calories`,  `sucre`, `graisses`, `proteines`, `bio` ) VALUES ('haricots verts',  'monoprix',  25, 3, 0, 1.7,  false);
-- Récuperer toutes les données avec SELECT * de la table utilisateur
SELECT * FROM utilisateur;

-- Lire seulement quelques colonnes avec SELECT
SELECT `nom`, `prenom`, `email` FROM utilisateur ;

-- Lire seulement le nom et les calories pour chaque aliments
SELECT `nom`, `calories` FROM aliment;

-- Mettre a jour un email utilisateur
-- UPDATE utilisateur (mettre a jour cette table)
-- SET `email` = 'valeur de l'email' (nouvelle adresse mail)
-- WHERE `id`='' (filtrer pour chercher quelle entrée changer)
-- UPDATE table Signifie à SQL que vous souhaitez mettre à jour de la donnée dans votre BDD.
-- Vous indiquez aussi la table dans laquelle se trouve(nt) le ou les objets que vous souhaitez modifier.
--SET colonne = valeur Sert à indiquer à SQL quelles sont la ou les colonnes à modifier, et quelles sont la ou les valeurs
-- qu'elles doivent désormais prendre.
--WHERE colonne = valeur C'est ce qu'on appelle un filtre.
--Vous allez les voir plus en détail dans la partie 3, mais sachez qu'ils servent à restreindre la commande en cours
--à un ou des objets répondant à des conditions précises. Ici, on va mettre à jour uniquement l'objet dont l'id est 1,
--soit le premier utilisateur !
```

```sql
UPDATE `utilisateur`
SET`email` = 'inconnu@outlook.fr'
WHERE `id` = '12';
UPDATE `aliment`
SET`nom` = 'pomme golden',`marque` = 'intermarche' WHERE `id` = '1';

-- Supprimer une entrée
DELETE FROM `utilisateur` WHERE `id` = '12';
DELETE FROM `aliment` WHERE `id` = '1';

-- Afficher les aliments avec les calories qui n'exede pas 90kCal
SELECT * FROM aliment WHERE calories < 90;

-- Afficher les mail "gmail.com"
-- %gmail.com Texte se terminant par "gmail.com"
-- gmail.com% Texte commancant par "gmail.com"
-- %gmail.com% Texte contenant "gmail.com" au debut ou a la fin
SELECT * FROM utilisateur WHERE email LIKE "%gmail.com";

-- Afficher aliments par orde croissant des "calories" ORDER BY...ASC
SELECT * FROM aliment ORDER BY calories ASC;

-- Afficher aliments par orde decroissant des "calories" ORDER BY...DESC
SELECT * FROM aliment ORDER BY calories DESC;

-- afficher que les aliments dont les calories ne dépassent pas 90 kcal, mais de manière décroissante
SELECT * FROM aliment WHERE calories < 90 ORDER BY calories DESC;

-- Afficher les users avec prenom par ordre alphabetique
SELECT * FROM utilisateur ORDER BY prenom ASC;

SELECT * FROM aliment WHERE bio = 'false' ORDER BY proteines DESC;

SELECT COUNT(*) FROM utilisateur WHERE email LIKE "%gmail.com";

-- Par exemple,SELECT COUNT(*)FROM utilisateur; compte tous les utilisateurs,
-- tandis que  SELECT COUNT(nom) FROM utilisateur;  compte tous les noms de famille.
-- Par exemple, SELECT COUNT(DISTINCT nom) FROM utilisateur;
-- comptera uniquement les noms de familles différents, vous permettant de voir si certains utilisateurs ont le même nom !
----------------------------------------------------------------
-- AVG : nous donne la moyenne de la colonne sur la sélection ;
-- SUM : nous donne la somme de la colonne sur la sélection ;
-- MAX : nous donne le maximum de la colonne sur la sélection ;
-- MIN : nous donne le minimum de la colonne sur la sélection.
----------------------------------------------------------------
```

```sql
-- Le max de sucre
SELECT MAX(sucre) FROM aliment;
-- teneur moyenne en calories des aliment de 30kcal ou plus
SELECT AVG(calories) FROM aliment WHERE calories >=30;
-- Compter les aliments qui ne sont pas bio
SELECT COUNT(*) FROM aliment WHERE bio = false;
-- Contenance maximal de proteines des elements pas bio
SELECT MAX(proteines) FROM aliment WHERE bio = false;
-- Contenance minimal de proteines des elements pas bio
SELECT MIN(proteines) FROM aliment WHERE bio = false;
-- Moyenne de proteines des elements pas bio
SELECT AVG(proteines) FROM aliment WHERE bio = false;

--sauvegarder dans une vue la commande suivante : les utilisateurs dont l'adresse mail est une adresse Gmail.
-- Je créer la vue "utilisateurs_gmail_vw". Cette dernière s'utilise désormais comme une table.
-- Je filtre le type de données à extraire avec WHERE et je filtre le format avec LIKE
CREATE VIEW UTILISATEURS_GMAIL_VW AS SELECT * FROM utilisateur WHERE email LIKE "%GMAIL.COM";

--Ainsi, pour récupérer les utilisateurs avec une adresse Gmail, plus besoin d'écrire ma requête compliquée !
-- recuperer toutes les données avec SELECT depuis la vue utilisteurs_gmail_vw  et le vw pour faire la distinction avec les vrais tables
SELECT * FROM utilisateurs_gmail_vw;

--La convention chez les utilisateurs de SQL est de toujours préfixer le nom d'une vue avec "_vw", pour la distinguer des "vraies" tables.
--afficher les utilisateurs dont l'adresse e-mail est une adresse Gmail ET dont le prénom contient la lettre "m"
SELECT * FROM utilisateurs_gmail_vw WHERE prenom LIKE "%m%";

-- Créez une vue reprenant notre liste des aliments non bio, classés par contenance en protéines (de manière décroissante).
CREATE VIEW ALIMENTS_NON_BIO_VW AS SELECT * FROM aliment WHERE bio = "false" ORDER BY proteines DESC;
SELECT * from aliments_non_bio_vw;

--Afficher les aliments non bio par ordre croissant et proteines superieur a 10
SELECT * from aliments_non_bio_vw WHERE proteines > 10;
--
-- Relation 1 à plusieurs utilisateur et langue (one-to-many, en anglais)
-- Chaque utilisateur est relié à une langue.Et chaque langue peut être reliée à plusieurs utilisateurs.
-- c'est l'objet qui se trouve du côté "plusieurs" de la relation qui va être modifié,
INSERT INTO `langue` VALUES ('francais');

-- joindre deux tables selon un identifiant qu'elles ont en commun.
SELECT FROM `utilisateur`JOIN `langue` ON `utilisateur`.`langue_id` =`langue`.`id`;

-- Nous avons demandé à MySQL de sélectionner tous les utilisateurs. => SELECT * FROM `utilisateur`
-- Auxquels nous voulons joindre les langues. => JOIN `langue`
-- En précisant à MySQL de les relier, en considérant que l'id de la langue est stockée dans chaque utilisateur dans le champ langue_id.
-- ON `utilisateur`.`langue_id` = `langue`.`id`
```

```
--
-- donner tous les noms de famille des utilisateurs ayant sélectionné le français.
SELECT *FROM `utilisateur`JOIN `langue` ON `utilisateur`.`langue_id` = `langue`.`id` WHERE langue.nom = 'francais';
--
-- Relation plusieurs à plusieurs utilisateur et langue (many-to-many, en anglais)
-- Chaque objet d'une table pouvant être relié à plusieurs objets de l'autre table, et vice versa.
-- utilisateurs_id est une table de liaison Par convention, elle prend le nom {table1}_{table2} , et sert à relier les tables 1 et 2
-- qui y sont stockées, en sauvegardant l'id d'un objet de la table 1, à l'id de l'objet de la table 2 correspondant.
-- Voici la commande pour relier tous les utilisateurs aux aliments qu'ils ont scannés :
SELECT * FROM utilisateur JOIN utilisateur_alimen tON (utilisateur.id = utilisateur_aliment.utilisateur_id) JOIN aliment ON (aliment.id =
utilisateur_aliment.aliment_id);
--  Nous avons demandé à MySQL de sélectionner tous les utilisateurs.
-- SELECT * FROM `utilisateur`
-- Auxquels nous voulons joindre la table utilisateur_aliment.
-- JOIN `utilisateur_aliment`
-- En précisant à MySQL de les relier en considérant que l'id de l'utilisateur est stocké en tant que utilisateur_id
-- dans la table utilisateur_aliment.
-- ON (utilisateur.id = utilisateur_aliment.utilisateur_id)
-- À ce JOIN , on veut à nouveau lier de la donnée de la table aliment, soit un nouveau JOIN .
-- JOIN `aliment`
-- Pour ce faire, on précise à MySQL que l'id de l'aliment est stocké dans utilisateur_aliment en tant que aliment_id.
-- ON (aliment.id = utilisateur_aliment.utilisateur_id)
--
-- voir tous les aliments sélectionnés par les utilisateurs dont l'adresse e-mail et une adresse Gmail.
SELECT * FROM utilisateur JOIN utilisateur_aliment ON (utilisateur.id = utilisateur_aliment.utilisateur_id) JOIN aliment ON ( aliment.id =
utilisateur_aliment.aliment_id) WHERE email LIKE '%gmail.com%';
-- OU
SELECT * FROM utilisateurs_gmail_vw JOIN utilisateur_aliment ON (utilisateurs_gmail_vw.id = utilisateur_aliment.utilisateur_id) JOIN aliment ON (
aliment.id = utilisateur_aliment.aliment_id)
-- many to many
SELECT *FROM film JOIN film_pays_de_sortie ON (film.id = film_pays_de_sortie.film_id)
-- clé primaire : film.id
-- clé etrangère : film_pays_de_sortie.film_id
    JOIN pays_de_sortie ON (
        pays_de_sortie.id = film_pays_de_sortie.pays_de_sortie_id
    );
-- clé primaire : pays_de_sortie.id
-- clé etrangère : film_pays_de_sortie.pays_de_sortie_id
-- table de liaison c'est film_pays_de_sortie
-- ATTENTION il n'y a pas de cle primaire dans une table de liaison
--
-- one to many exemple
SELECT * FROM film JOIN note ON (film.note_id = note.id );
-- clé primaire : note.id
-- clé etrangère : film.note_id
```