INSTRUCTIONS MYSQL

LISTER LES BDD :

```
SHOW DATABASES ;
```

CREER UNE TABLE :

```
CREATE TABLE livre
(
   idLiv   INT(3) UNSIGNED NOT NULL AUTO_INCREMENT,
   libLiv  VARCHAR(128) NOT NULL,
   idGre   INT NOT NULL,
   PRIMARY KEY (idLiv),
   CONSTRAINT fk_genre
   FOREIGN KEY (idGre)
     REFERENCES genre(idGre)
);
```

MODIFIER UNE TABLE :

```
   ALTER TABLE [nom de la table]
   ADD COLUMN [nom de colonne] INT NULL AFTER [nom de
colonne];
```

Ici par exemple, on ajoute une colonne après une autre déjà existante.

```
ALTER TABLE livre
ADD COLUMN idAut INT NOT NULL AFTER libLiv;

ALTER TABLE livre
ADD INDEX(idAut);
```

SUPPRIMER UNE TABLE :

```
DROP TABLE [nom de la table];
```

MODIFIER UNE TABLE :

```
   ALTER TABLE [nom de la table]
   ADD COLUMN [nom de colonne] INT NULL AFTER [nom de
colonne];
```

```
ALTER TABLE livre
ADD COLUMN idAut INT NOT NULL AFTER libLiv;

ALTER TABLE livre
ADD INDEX(idAut);
```

ANALYSER /REPARER UNE TABLE :

Parfois le journal d'erreur peut nous remonter que le serveur a craché à cause d'un dysfonctionnement sur une table.

```
CHECK TABLE [nom de la table];
```

```
REPAIR TABLE [nom de la table];
```

//INSERTION DE DONNEE //

SYNTAXE D'INSERTION :

```
INSERT INTO[nom de la table]
(
   [champ n°1],
   [champ n°2],
   [etc.]
)
VALUES
(
   [valeur n°1],
   [valeur n°2],
   [etc.]
)
```

```
INSERT INTO livre(libLiv, idGre)
VALUES ('Dune', 2);
```

Il est possible en Mysql d'insérer **plusieurs lignes à la fois**. Pour cela, on sépare les groupes de données par une virgule :

```
INSERT INTO livre(libLiv, idGre)
VALUES
('Dune', 2),
('Les aventures de Sherlock Holmes', 3),
('La horde du contrevent', 2);
```

SUPPRIMER DES DONNEE :

```
DELETE FROM [nom de la table];
```

```
DELETE FROM Livre;
```

```
DELETE FROM [nom de la table]
WHERE [condition ciblant des données];
```

```
DELETE FROM  Livre
WHERE libLiv = 'Birth of a Theorem';
```

SUPPRIMER TOUTE LES DONNEES ET REINITIALISER UNE TABLE :

```
TRUNCATE TABLE [nom de la table];
```

```
TRUNCATE TABLE Livre;
```

MODIFIER DES DONNEES :

```
UPDATE [nom de la table]
SET [nom de la donnée] = [nouvelle valeur];
```

```
UPDATE Livre
SET libLiv = 'La horde du contrevent';
```

On peut y ajouter des critères pour cibler des données en particulier :

```
UPDATE [nom de la table]
SET [nom de la donnée] = [nouvelle valeur];
WHERE [condition ciblant des données];
```

```
UPDATE Livre
SET libLiv = 'La horde du contrevent'
WHERE libLiv = 'Birth of a Theorem';
```

//**LIRE DES DONNEE**//

SYNTAXE DE LECTURE :

Lire des données se fait via l'instruction DML **SELECT** :

```
SELECT [nom du champ 1], [nom du champ 2], etc.
FROM   [nom de la table];
```

```
SELECT idLiv, libLib
FROM   Livre;
```

LES ALIAS :

Lorsque l'on lit des données d'une table, il est important de prendre l'habitude de préfixer la table en utilisant un alias. Cette habitude sera fort utilise lorsque les requêtes vont se complexifier.

Pour cela, on ajoute après le nom de la table l'alias de notre choix, et on reprend cet alias avant chaque nom de champ remonté dans le select.

D'une manière générale, pour définir les alias on utilise des trigrammes.

```
   SELECT [alias].[nom du champ 1], [alias].[nom du champ 2],
etc.
   FROM   [nom de la table] [alias];
```

```
   SELECT liv.idLiv, liv.libLib
   FROM   Livre liv;
```

LES CLAUSES :

Les conditions peuvent être l'égalité, la différence, la supériorité, l'infériorité... et peuvent être combinées, via des relations ET, OU :

```
   SELECT [nom du champ 1], [nom du champ 2], etc.
   FROM   [nom de la table]
   WHERE  [condition(s)];
```

```
   SELECT idLiv, libLib, prix
   FROM   Livre
   WHERE  prix = 10;
```

```
   SELECT idLiv, libLib, prix
   FROM   Livre
   WHERE  prix < 10
   AND    prix > 5;
```

regrouper les critères selon notre logique :

```
   SELECT idLiv, libLib, prix
   FROM   Livre
   WHERE  (prix > 5 AND prix < 10)
   OR     (prix < 20 AND prix > 15);
```

Cette requête par exemple renvoie les livres dont le prix est soit compris entre 5 et 10, soit compris entre 15 et 20.

LES JOINTURES :

STRICTES(OPTIMISE) :

```
SELECT      [nom du champ 1], [nom du champ 2], etc.
FROM        [table A]
INNER JOIN [table B] ON [table A].[cle de jointure] = [table
B].[cle de jointure];
```

```
SELECT        liv.libLiv, aut.nom, aut.prenom
FROM          Livre liv
INNER JOIN    Auteur aut ON aut.idAut = liv.idAut;
```

OUVERTES(NON-OPTIMISE) :

```
SELECT      [nom du champ 1], [nom du champ 2], etc.
FROM        [table A]
LEFT JOIN [table B] ON [table A].[cle de jointure] = [table
B].[cle de jointure];
```

```
SELECT        liv.libLiv, aut.nom, aut.prenom
FROM          Livre liv
LEFT JOIN     Auteur aut ON aut.idAut = liv.idAut;
```

JOINTURES ET CONDITIONS :

CONDITIONS SUR TABLE DE DEPART :

Les livres dont le prix est > 10 et leurs auteurs

```
SELECT        liv.libLiv, aut.nom, aut.prenom
FROM          Livre liv
INNER JOIN    Auteur aut ON aut.idAut = liv.idAut
WHERE         liv.prix > 10;
```

CONDITIONS SUR TABLE DE JOINTURE :

Les livres dont l'auteur est Alain Damasio

```
SELECT        liv.libLiv, aut.nom, aut.prenom
FROM          Livre liv
INNER JOIN    Auteur aut ON aut.idAut = liv.idAut
                        AND aut.nom = 'Damasio';
```

FAIRE UNE JOINTURE N-N :

```
SELECT      [liste de champs]
FROM        [table A]
INNER JOIN [table Pont] ON [table A].[cle A] = [table Pont].[cle A]
INNER JOIN [table B] ON [table B].[cle B] = [table Pont].[cle B];
```

```
SELECT        liv.libLiv, aut.nom, aut.prenom
FROM          Livre liv
INNER JOIN    LivreAuteur lau ON lau.idLiv = liv.idLiv
INNER JOIN    Auteur aut ON aut.idAut = lau.idAut;
```

LE TRI :

On peut trier les données remontées par ordre croissant ou décroissant, via le mot clé **ORDER**, puis **ASC** pour croissant, ou **DESC** pour décroissant.

```
SELECT     [nom du champ 1], [nom du champ 2]
FROM       [nom de la table]
ORDER BY   [nom du champ 1] ASC;
```

```
SELECT        liv.libLiv, aut.nom, aut.prenom
FROM          Livre liv
INNER JOIN    Auteur aut ON aut.idAut = liv.idAut
ORDER BY      liv.libLiv ASC;
```

LIMITER LE NOMBRE DE LIGNES REMONTEES :

```
SELECT [nom du champ]
FROM   [nom de la table]
LIMIT  [nombre de lignes à remonter];
```

```
SELECT liv.libLib
FROM   Livre liv
LIMIT  10;
```

LE REGROUPEMENT :

Le regroupement est une opération qui consiste à agréger un certains nombres de lignes, en rassemblant les données identiques en paquets pour les associer aux différentes valeurs correspondantes.

Par exemple, si on veut connaître le nombre de livre par auteur.

Imaginons la requête suivantes :

```
SELECT       aut.nom, aut.prenom, liv.libLiv
FROM         Livre liv
INNER JOIN   LivreAuteur lau ON lau.idLiv = liv.idLiv
INNER JOIN   Auteur aut ON aut.idAut = lau.idAut;
```

En SQL, nous allons écrire cette requête de la façon suivante :

```
SELECT    [colonne de regroupement], COUNT([colonne à compter])
FROM      [table]
GROUP BY  [colonne de regroupement];
```

```
SELECT       aut.nom, aut.prenom, COUNT(liv.idLiv)
FROM         Livre liv
INNER JOIN   LivreAuteur lau ON lau.idLiv = liv.idLiv
INNER JOIN   Auteur aut ON aut.idAut = lau.idAut
GROUP BY     aut.nom, aut.prenom;
```

Remarque : on aurait pu imaginer faire plus simple. Par exemple, pourquoi ne pas faire cette requête sans aller jusqu'à la table des livres ? Voir, si l'identifiant d'auteur suffit, tout simplement compter les idLiv par idAut dans la table de jointure ?

```
SELECT       aut.nom, aut.prenom, COUNT(lau.idLiv)
FROM         Auteur aut
INNER JOIN   LivreAuteur lau ON lau.idAut = aut.idAut
GROUP BY     aut.nom, aut.prenom;
```

```
SELECT       lau.idAut, COUNT(lau.idLiv)
FROM         LivreAuteur lau
GROUP BY     lau.idAut;
```

LES CLAUSE DE REGROUPEMENT :

Pour appliquer des clauses au résultat du regroupement, on utilise une syntaxe différente, avec le mot clé **HAVING** :

Par contre, comme les fonctions de regroupement s'exécutent sur le résultat du select, les clauses WHERE ne peuvent donc pas être appliquées au résultat du regroupement...

```
SELECT    [colonne de regroupement], COUNT([colonne à compter])
FROM      [table]
WHERE     [clause sur la table]
GROUP BY  [colonne de regroupement]
HAVING    [clause sur le résultat du regroupement] ;
```

```
SELECT       aut.nom, aut.prenom, COUNT(liv.idLiv)
FROM         Livre liv
INNER JOIN   LivreAuteur lau ON lau.idLiv = liv.idLiv
INNER JOIN   Auteur aut ON aut.idAut = lau.idAut
GROUP BY     aut.nom, aut.prenom
HAVING       COUNT(liv.idLiv) > 1;
```

LES SOUS REQUETES :

Utiliser des sous requêtes est un mécanisme qui consiste à considérer le résultat d'une requête comme un ensemble de données que l'on peut lui même requêter, comme s'il s'agissait d'une table. Pour cela on va imbriquer des requêtes dans des requêtes :

```
SELECT       tab.[champ], req.[champ]
FROM         [table] tab
INNER JOIN   (
               SELECT tab2.[id], tab2.[champ]
               FROM   [table 2] tab
             ) req ON req.[id] = tab.[id];
```

```
SELECT        aut.nom, aut.prenom, nbm.nbLiv
FROM          Auteur aut
INNER JOIN    (
                 SELECT nbl.idAut auteur, MAX(nbl.nbLiv) nbLiv
                 FROM
                 (
                   SELECT        aut.idAut, COUNT(liv.idLiv) nbLiv
                   FROM          Livre liv
                   INNER JOIN    LivreAuteur lau ON lau.idLiv = liv.idLiv
                   INNER JOIN    Auteur aut ON aut.idAut = lau.idAut
                   GROUP BY      aut.idAut
                 ) nbl
              ) nbm ON nbm.auteur = aut.idAut;
```