POLYTECH MARSEILLE
Aix*Marseille Université

PeiP
1ère année

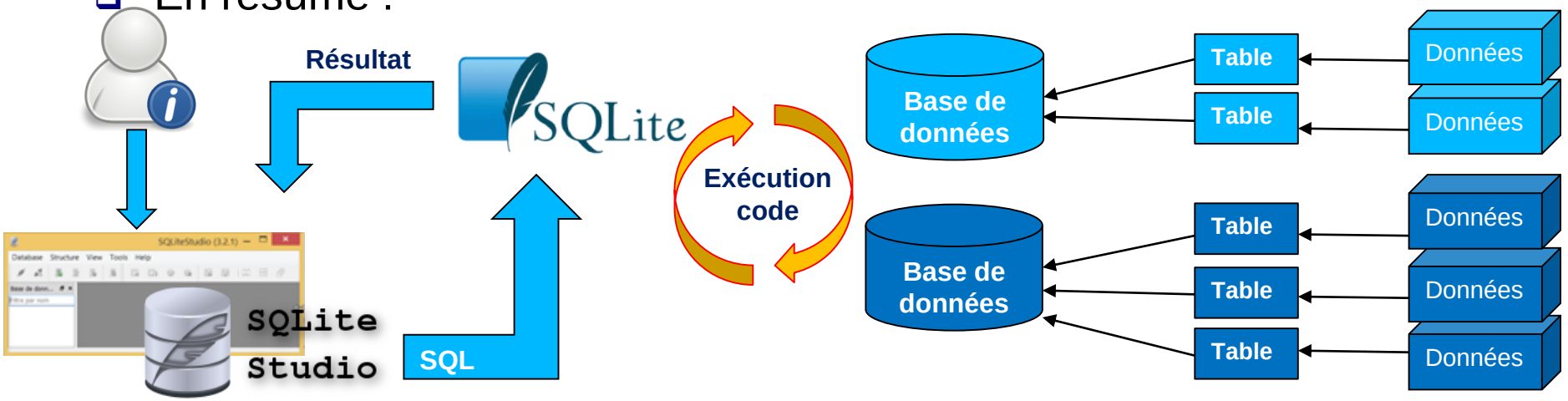
# UE Informatique
## Conception et Exploitation de Bases de Données

**Session 3 :** SQL

# 1. Généralités

# Histoire de comprendre ce qu'on fait …

- SQLite Studio c'est :
  - Un SGBD (SQlite) capable de comprendre des instructions (requêtes) SQL … +
  - Un « front end », une interface graphique qui permet de dialoguer avec SQLite via des interfaces graphiques …
- Bref, le côté Studio masque le travail du SGBD…
- … pas très grave s'agissant d'une initiation à la conception et l'exploitation de BD … mais à garder en tête…
- En résumé :



# Retour sur les fonctions d'un SGBD

- Dans sa mission de mémorisation et d'exploitation des données, le SGBD doit permettre d'/e:
  - *Ajouter*
  - *Modifier (mise à jour)*
  - *Supprimer*
  - *Rechercher*

**des données**

Langage De Manipulation des Données

- Ce qui suppose d'avoir auparavant :
  - *Défini les données*

Langage De Description des Données

- => *SQL est un langage permettant de spécifier cela de manière <u>générique</u>*

# Structured Query Language : SQL

- Standard de langage de manipulation de donnée : tous les SGBD respectent cette norme (ou du moins la majorité des spécifications du langage : par ex. MySQL Lite utilise un sous ensemble de SQL)
- Pourquoi un standard ?

- *A noter : dans le cadre de l'objectif de ce cours, nous n'exploreront pas l'ensemble des possibilités du langage.*

# SQL : Illustration

- Retour sur l'exemple du 1ere session


- Cette entité décrite au niveau conceptuelle voire logique doit être traduite au niveau du SGBD pour pouvoir contenir des données comme les suivantes :

| *ISBN* | *NomAuteur* | *PrénomAuteur* | *Titre* |
|---|---|---|---|
| 209178527X | NEY | Henry | Automatique et informatique industrielle |
| 2851102869 | FAURE | Jacques | Almanach Vermot 2010 |
| 2070628035 | ABOUET… | Marguerite | Aya de Yopougon, Tome 5 |

# BD & SQL : vocabulaire

- *Colonne*: correspond à l'attribut/propriété déclaré/e dans le MCD
- *Enregistrement* ou *ligne* : une occurrence d'une entité enregistrable dans la table correspondante dans la base de donnée.
- Requête : « commande » envoyée au SGBD concernant un traitement (création table, recherche d'information ….)

# SQL : Illustration

- Code SQL de création de la table

```
CREATE TABLE Livre (
   ISBN              CHAR(10)NOT NULL PRIMARY KEY,
   Nom_Auteur        CHAR(10)NULL,
   Prenom_Auteur     CHAR(10)NULL,
   Titre             CHAR(20)NOT NULL
);
```

= identifiant => SGBD vérifiera à chaque ajout d'enregistrement que la valeur est bien unique

Type de données … as in VB

- Insertion de données :

```
INSERT INTO     Livre
VALUES
   ('209178527X', 'NEY', NULL ,'Automatique …   ')
;
```

# SQL : Illustration

- "Affichage" des données contenu dans une table :

```
SELECT ISBN, Nom_Auteur, Prenom_Auteur, Titre
FROM Livre
;
```

Où se trouve l'information (ie table)

Informations qui m'intéressent

- Modification de données :

```
UPDATE Livre
   SET Prenom_Auteur='Henry'
WHERE
   Nom_Auteur='NAY'
;
```

= Modifier (remplacer) la valeur stockée dans la propriété PrenomAuteur par Henry

Modification à apporter à tous les enregistrements vérifiant cette contrainte : valeur de la propriété Nom_Auteur = NAY

# 2. Syntaxe SQL
# Les bases

*Types de données, Création tables, Population table, requêtes monotable.*

# Types de donnée SQL

| Type de donnée | Syntaxe | Definition |
|---|---|---|
| Integer | Integer | entier |
| Small Int | Smallint | |
| Numeric | numeric(p,s) | ***p*** : precision globale ie nb chiffres, ***s*** précision après la virgule. Par ex. numeric(4,2) peut contenir 1234  123,4 12,34 1,234 -> converti en 1,23  12,349 -> converti en 12,35. |
| Decimal | decimal(p,s) | idem |
| Real | real | Nombre décimal à simple précision |
| double precision | double precision | Nombre décimal de double précision |
| Real (Float) | float(p) | ***p*** précision |
| Chaîne de caractères | char(n) | ***n*** nombre de caractères à stocker. S'il y a moins de n caractère, la chaîne est remplie d'espaces. |
| Chaine de caractères variables | varchar2(x) | Where ***x*** is the number of characters to store. This data type does NOT space pad. |
| bit | bit(n) | ***n*** nb de bits à stocker. |
| bit variable | bit varying(n) | ***n*** nombre de bits à stocker. La longueur peut aller jusqu'. |

# Types de donnée SQL

| Type de donnée | Syntaxe | Definition |
|---|---|---|
| date | date | Stocke des dates (année, mois et jour) |
| time | time | Stores the hour, minute, and second values. |
| Timestamp | timestamp | Stocke année, mois, jour, heure, minute et secondes |
| time with time zone | time with time zone | Pareil que time avec en plus l'information de décalage par rapport UTC de l'heure stockee. |
| timestamp with time zone | timestamp with time zone | Idem que timestamp … |
| | … | … |

Remarque: bien que censé être générique, chaque SGBD tend à vouloir définir ses propres types de données adaptées à des usages spécifiques ou répondant à des objectifs d'optimisation.

# Type de données *as understood by SQLite*

- **Types de donnée réduits par simplification**
  - **NULL : valeur est « nulle » i.e rien …** (ex. NULL comme valeur de note signifie .. qu'il n'y a pas de note, alors que 0 est une valeur …)
  - **INTEGER :** SQLite gère l'espace mémoire en fonction de la valeur ( 1,2.. 8 octets selon besoins 1o = 0..255 , 8=2^64 )
  - **REAL** : valeur réelle (stocké sur 8o)
  - **TEXT** : …
  - **BLOB** : forget about it
- **Le reste des types habituels de données est converti selon besoin :**
  - **Boolean** : stocké comme INTEGER avec valeur 0 ou 1
  - **Date/Heure** : stockée comme
    - un texte : "YYYY-MM-DD HH:MM:SS.SSS«
    - un nombre (REAL) de jours depuis le 24 novembre -4714 (calendrier Julien).
    - un entier (Unix Time) : nb de secondes depuis 1er janvier 1970

# Création d'une base de données

- C'est là que seront créées, gérées les tables et les données qu'elles contiennent.
- Syntaxe SQL :

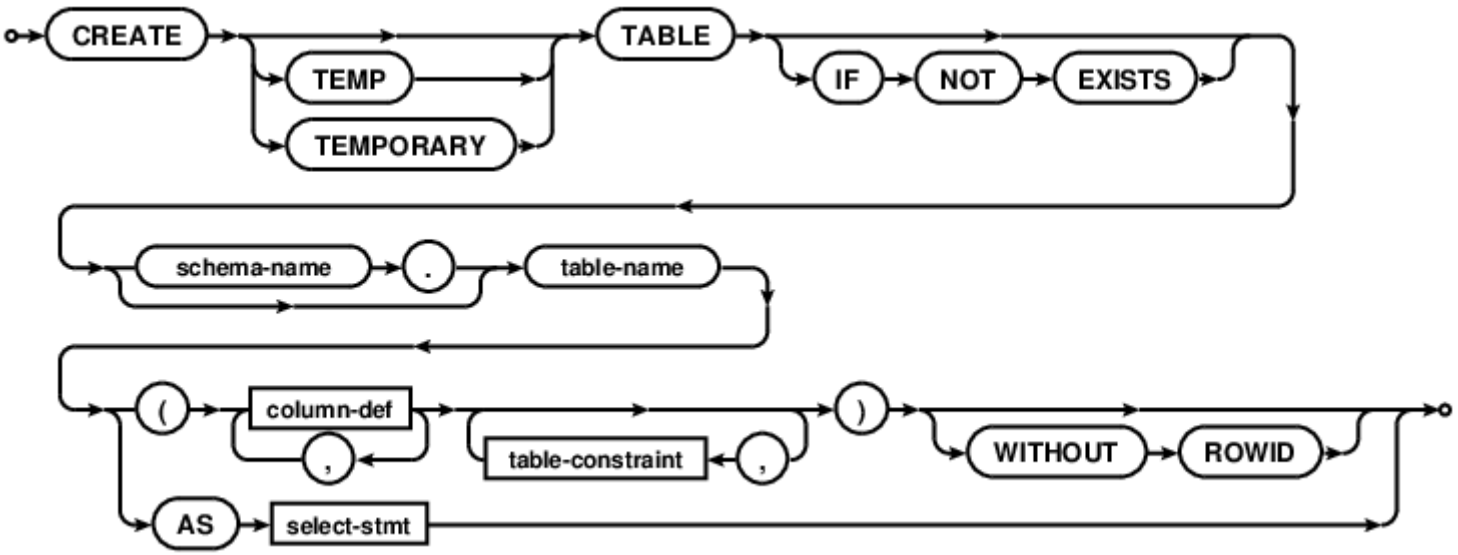
```
CREATE DATABASE Nom_Base_de_donnee ;
```

- Cette instruction « ordonne » le SGBD de créer cette **Base de Données**… charge à lui de se « débrouiller » pour gérer ça au niveau du disque dur/système de fichiers.
  - Ex: SQLite met tout dans 1 seul fichier…

# CREATE TABLE : Création d'une table

- Syntaxe générale

```
CREATE TABLE [Base de donnée . ] nom_table (
                nom_colonne1 type [contraintes] ,
                nom_colonne2 type [contraintes] ,
                 …,
   );
```



# CREATE TABLE : Création d'une table

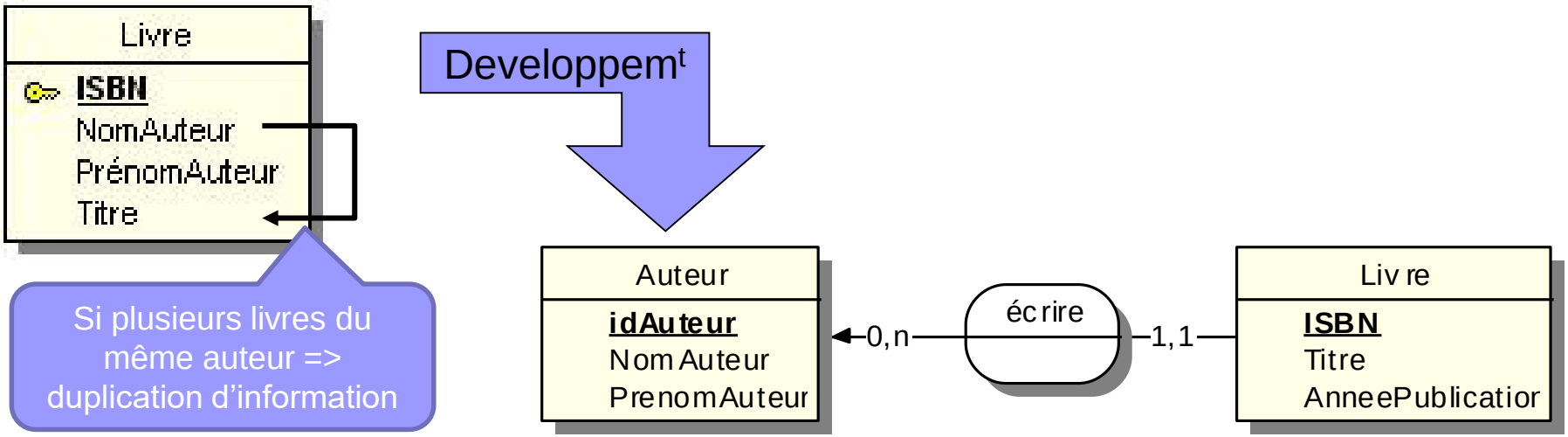
```
CREATE TABLE
    [Base de donnée . ] nom_table (
   col1 type [contraintes] ,
   col2 type [contraintes] ,
   PRIMARY KEY (col1, col2) );
```

- Contraintes usuelles :
  - Clef primaire :
    - **PRIMARY KEY** [AutoIncrement]
    - **Autoincrement** laisse le SGBD gérer les clefs numériques (incrémente à chaque ajout d'enregistrement/ligne => ne pas remplir soit même la colonne)
  - Clef étrangère :
    - **REFERENCES** table_etrangere (nom_colonne)
  - « remplissage » de la colonne obligatoire (pas de valeur vide) :
    - **NOT NULL**
  - Contraintes sur les valeurs autorisées :
    - **CHECK** (condition sur la valeur) (cf. section 3.)
  - Unique (pas de doublon) :
    - **UNIQUE**
    - Une fois saisie, la valeur ne peut plus être utilisée …
- Il est possible de rajouter la contrainte en fin de déclaration. Cela est indispensable en cas de clefs composées. *(Cf encadré plus haut)*

# Création d'une table : exemples

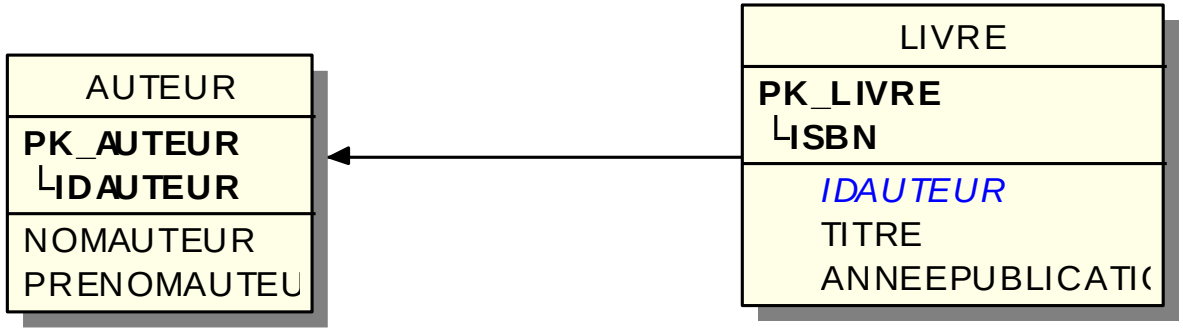
- Exemple initial

```
CREATE TABLE Livre (
   PK_ISBN       CHAR(10)    NOT NULL PRIMARY KEY,
   NomAuteur     CHAR(10),
   PrenomAuteur  CHAR(10),
   Titre         CHAR(20)    NOT NULL
);
```



# Création d'une table : exemples

- Exemple revu & corrigé :




- Code SQL correspondant :

```
CREATE TABLE Auteur (
    PK_IdAuteur INTEGER      PRIMARY KEY,
    Nom_Auteur     CHAR(10),
    Prenom_Auteur  CHAR(10)
    );
```

```
CREATE TABLE Livre (
    PK_ISBN        CHAR(10)   PRIMARY KEY,
    Titre          CHAR(10)   NOT NULL,
    AnneePubli     DATE,
    FK_IdAuteur    INTEGER
            REFERENCES Auteur (PK_IdAuteur)
    );
```

# Insérer des données dans une table

- Syntaxe SQL (version simple) :

```
INSERT INTO Nom_Table
   ('colonne1', 'colonne2', …,'colonnen')
VALUES
   ('val1', 'val2',…, 'valn')
;
```

- La ligne colonne est optionnelle si, pour chaque enregistrement, la donnée de chaque colonne est fournie dans l'ordre de sa déclaration (càd celui du CREATE TABLE).
- Si l'enregistrement ne respecte pas les contraintes, il n'est pas ajouté dans la base de données.
  - => SQL Error … (le SGBD refuse l'insertion)

# Insertion de données : exemple

- Syntaxe SQL (version simple) :

```
INSERT INTO       "Livre"
("ISBN", "Nom_Auteur", "Prenom_Auteur", "titre")
VALUES
    ('209178527X', 'NEY', NULL, 'Automatique … ')
;
```

- Syntaxe SQL (version avec clef etrangere) :

```
INSERT INTO       "Auteur"
    ("PK_IdAuteur", "Nom_Auteur","Prenom_Auteur")
VALUES
    (1, 'FAURE', 'Jacques')
);

INSERT INTO       "Livre"
    ("PK_ISBN", "titre", "FK_IdAuteur")
VALUES
    ('209178527X', 'Almanach Vermot 2010' , 1)
;
```

# INSERT INTO & « REFERENCES »

- Rappel :
  - Déclaration de la contrainte REFERENCES lors de la création d'une table définit une contrainte d'intégrité référentielle
  - soit en français, la valeur de clef qu'on ajoute doit exister dans la table référencée … sinon, ajout de données incohérentes … une hérésie pour un SGBD…
- Conséquence Ajout :
  - Si on ajoute un livre avec un identifiant d'un auteur qui n'existe pas dans la table auteur => SGBD refuse l'ajout du livre…

```
INSERT INTO Livre
VALUES
(NULL, "Livre inexistant", NULL, 10) ;
```

```
CREATE TABLE Livre (
    PK_ISBN  CHAR(10)  PRIMARY KEY,
    Titre    CHAR(10)            NOT NULL,
    AnneePubli       DATE,
    FK_IdAuteur      INTEGER
             REFERENCES Auteur (PK_IdAuteur)
    );
```

Erreur pendant l'exécution de la requête sur la base de données « LivreAuteur » : FOREIGN KEY constraint failed

# UPDATE TABLE

- Modifier la valeur d'une ou plusieurs colonnes:

```
UPDATE Nom_Table1 SET
   col1 = val1
   [, col1=val2]
WHERE
   condition ;
```

- Exemple

```
UPDATE Auteur SET
   Nom_Auteur = 'Abouet',
   Prenom_Auteur = 'Marguerite'
WHERE
      Nom_auteur LIKE 'abouet%'
  AND Prenom_Auteur LIKE 'marguerite%';
```

# « Lister » les données d'une table

- SELECT

```
SELECT
   Nom_Col1 [As nom1], …, Nom_Coln [As nomn]
FROM
   TABLE1[, TABLE2…]
WHERE
   condition
[ORDER BY COL1 [ASC|DESC] ];
```

- Condition est une expression booléenne :
  - Comparant 2 valeurs (‘=‘, ‘<>’, ‘<‘, ‘>’,’>=‘, ‘<=‘):
    - Ex. prix < 10.50 où prix est la valeur d'une colonne pour un enregistrement
  - Combinant plusieurs expressions avec des opérateurs conjonctifs (AND, OR, NOT)
    - Prix > 10 AND Prix < 20

# Lister les données d'une table : SELECT suite

- Condition est une expression booléenne utilisant des opérateurs logiques (‘`IS NULL`’, ‘`BETWEEN`’, ‘`IN`’, ‘`LIKE`’…)
  - **`IS NULL`** (et son dual `IS NOT NULL`) teste la présence d'une valeur dans la colonne.
    - Par ex. **`FK_IdAuteur IS NULL`** => le livre **n**'a **pas** d' auteur renseigné
    - et inversement **`FK_IdAuteur IS NOT NULL`** => il a un auteur (mais on ne s'intéresse pas à son identité)
  - **`BETWEEN`** : `cout BETWEEN 10 AND 20` est équivalent* à `cout > 10 AND cout < 20`
  - **`IN`** : la valeur de la colonne vaut l'une des valeurs listées par IN. Evite d'avoir de multiple égalités séparées par des AND… mais aussi utile pour les requêtes imbriquées pouvant « renvoyer » plusieurs valeurs…
    - couleur IN (‘rouge’, ‘bleu’, ‘vert’ )
  - **`LIKE :`** Ex. **`NOM LIKE H_G%`** : nom commençant par H suivit de n'importe quel caractère (mais il en faut 1) puis d'un G… la fin n'étant pas importante => HUG, HUGO, HUGOT, HIGELLIN valident cette expression.
- ORDER BY permet de trier les résultats selon une colonne dans l'ordre croissant (ASC) ou décroissant (DESC).

* Selon le SGBD 10 et 20 seront inclus … ou non :/

# SELECT Exemple

- Afficher toutes les informations sur toutes les colonnes

```
SELECT * FROM Livre;
```

- Afficher les noms des auteurs :

```
SELECT Nom_Auteur FROM Auteur;
```

- Afficher les livres publiés apres une date

```
SELECT Titre, annee_publi
FROM Livre
WHERE annee_publi > '2000:11:15';
```

- Afficher les livres et le nom de leurs auteurs

```
SELECT Nom_Auteur, Prenom_Auteur, Titre
FROM Auteur, Livre
WHERE Pk_IdAuteur=FK_idAuteur;
```

# DELETE : suppression d'enregistrements

- Comme son nom l'indique…

```
DELETE FROM nom_table
WHERE Condition;
```

- Exemple :

```
DELETE FROM Auteur
WHERE Pk_IdAuteur = 10;
```

```
DELETE FROM Auteur
WHERE Nom_Auteur = 'MOUAIS'
      AND Prenom_Auteur = 'Bof';
```

- Most of the time suppression définitive ! (sauf si sauvegarde de la BD ☺, Rollback si transaction …)
- Mais suppression non garantie si violation contraintes….

# DELETE & « REFERENCES »

- contrainte d'intégrité référentielle définie lors de la création de la table
- Conséquence (paramétrable via propriété CASCADE) Suppression d'un propriétaire ayant un chien :
  - **Propagation de la suppression** => on supprime les chiens de ce propriétaire pour ne pas avoir d'enregistrements corrompus.
  - **Refus de la suppression** : le SGBD considère que ce n'est pas normal, on aurait du supprimer les chiens d'abord => refuse
  - **Remplacement de la valeur de clef étrangère** chez les chiens concernés **par** une valeur **NULL** (sans propriétaire) si la contrainte le permet ou une valeur par défaut.

```
DELETE FROM Auteur
WHERE Pk_IdAuteur = 6;
```

Auteur existe & il y a des livres ayant 6 comme valeur de clef étrangère

Comportement par défaut de SQLiteStudio

Erreur pendant l'exécution de la requête sur la base de données « LivreAuteur » : FOREIGN KEY constraint failed

# Opérateurs d'agrégation

- Permet d'effectuer des calculs sur les tables. Dans le cadre d'un SELECT, on peut calculer
  - une moyenne : **`AVG(x)`**
  - un nombre de valeurs : **`COUNT(x)`**
  - Max/min : **`MAX(x)`**,
  - une somme : **`SUM(x)`**
  - Où x est le nom de la colonne sur laquelle porte l'opération

- Nécessite de grouper les enregistrements par valeur identique puisque c'est sur ces groupes de valeurs que se fera le calcul
- => indiquer
  - La colonne sur laquelle doit se faire le calcul
  - Répéter le nom de la colonne afin que l'on sache à quelle colonne correspond le calcul (sinon, on ne voit que les totaux :/ )
- Pour restreindre le calcul on utilise la clause HAVING (restreindra l'affichage des résultats des calcul) de la même manière que WHERE (ne concerne que les données à "afficher" sur lesquelles se feront les calculs). Ex. n'afficher que les auteurs d'un seul livre.

# Opérateurs d'agrégation : illustration avec Count

- Calculer le nombre de livres dans la table Livre : '*' = tous … donc on compte les enregistrements

```
SELECT Count(*) FROM Livre ;
```

- **Count**() compte le nombre de valeur qu'on lui présente : si le paramètre est une colonne, compte le nb d'enregistrements ayant une valeur dans cette colonne… ainsi, un enregistrement ayant NULL comme valeur sur cette colonne, ne sera pas compté… ex. nombre livre avec auteur…

```
SELECT Count(FK_IdAuteur) FROM Livre ;
```

- La requête précédente compte le nombre de valeurs… donc un même auteur sera compté autant de fois qu'il apparait… Si on veut le nombre d'auteurs ayant publié au moins 1 livre … même s'il a publié 10 livres, l'auteur ne sera compté qu'1 fois… donc on ne s'intéresse qu'au nombre de valeurs de clefs étrangères … distinctes donc ☺

```
SELECT Count(DISTINCT FK_IdAuteur)
FROM Livre ;
```

# Opérateurs d'agrégation : still counting

- **Nombre de livre écrits par auteur** (mono table) : considérons une approche « manuelle »… sur les données suivantes :

| | PK_ISBN | Titre | AnneePubli | FK_IdAuteur |
|---|---|---|---|---|
| 1 | 2070628035 | Aya de Yopo | 2010-01-01 | 3 |
| 2 | 209178527X | Automatique et | 2008-01-01 | 1 |
| 3 | 209X0239 | Roman de Renart | 1300:01:01 | NULL |
| 4 | 209XEese | 100 H | 2002:01:01 | NULL |
| 5 | 2851102869 | Almanach Vermot 2010 | 2010-01-01 | 2 |
| 6 | dddddd | L automatique pour les nuls | NULL | 2 |
| 7 | dfff | Dune | NULL | 6 |
| 8 | fddg | Messie de Dune | NULL | 6 |
| 9 | fdfgdf | Les Rats | NULL | 7 |
| 10 | kdjdjddsfsd | Le chat qui pleure | NULL | 1 |
| 11 | kdjdjdjdj | Le chien qui pleure | NULL | 1 |

- S'il fallait traiter cette demande *a mano*, je devrais
  1. **Regrouper** les livre par « paquet » d'auteurs
  2. Compter les livres dans chaque paquet
  3. Faire un bilan auteur / nb de lignes…

# Opérateurs d'agrégation : still counting

- **Nombre de livre écrits par auteur** (mono table) : considérons une approche « manuelle »… sur les données suivantes :

| | PK_ISBN | Titre | AnneePubli | FK_IdAuteur |
|---|---|---|---|---|
| 1 | 2070628035 | Aya de Yopo | 2010-01-01 | 3 |
| 2 | 209178527X | Automatique et | 2008-01-01 | 1 |
| 3 | 209X0239 | Roman de Renart | 1300:01:01 | NULL |
| 4 | 209XEese | 100 H | 2002:01:01 | NULL |
| 5 | 2851102869 | Almanach Vermot 2010 | 2010-01-01 | 2 |
| 6 | dddddd | L automatique pour les nuls | NULL | 2 |
| 7 | dfff | Dune | NULL | 6 |
| 8 | fddg | Messie de Dune | NULL | 6 |
| 9 | fdfgdf | Les Rats | NULL | 7 |
| 10 | kdjdjddsfsd | Le chat qui pleure | NULL | 1 |
| 11 | kdjdjdjdj | Le chien qui pleure | NULL | 1 |

| PK_ISBN | Titre | AnneePubli | FK_IdAuteur | |
|---|---|---|---|---|
| 209178527X | Automatique et | 01/01/2008 | 1 | |
| kdjdjddsfsd | Le chat qui pleure | | 1 | **=> Auteur 1 : 3** |
| kdjdjdjdj | Le chien qui pleure | | 1 | |
| | | | | |
| 2851102869 | Almanach Vermot 2010 | 01/01/2010 | 2 | **=> Auteur 2 : 2** |
| dddddd | L automatique pour les nuls | | 2 | |
| | | | | |
| 2070628035 | Aya de Yopo | 01/01/2010 | 3 | **=> Auteur 3 : 1** |
| | | | | |
| dfff | Dune | | 6 | **=> Auteur 6 : 2** |
| fddg | Messie de Dune | | 6 | |
| | | | | |
| fdfgdf | Les Rats | | 7 | **=> Auteur 7 : 1** |
| | | | | |
| 209X0239 | Roman de Renart | 1300:01:01 | | => NULL : 2 |
| 209XEese | 100 H | 2002:01:01 | | |

- C'est la même démarche en SQL :

```
SELECT FK_IdAuteur, Count(*) FROM Livre
GROUP BY FK_Id_Auteur ;
```

# Opérateurs d'agrégation : still counting

- A noter, la lecture de https://www.sqlite.org/lang_select.html montre qu'en réalité, un SELECT n'est pas forcément suivi de « FROM TABLE… » … surtout lorsqu'on s'en sert pour faire des calculs :

```
SELECT 'AAA-' || '1234' ;  -- || = operateur concatenation
SELECT 3/5 ;  -- => =0 car division entière
SELECT 3*1.0 / 5 -- => 0,6 car 3*1.0 convertit entier en réel
```

- **Nombre de livre moyen par auteur** : *in fine*, = nb livres / nb auteurs, dc :

```
SELECT (SELECT Count(*) FROM Livre)
                / (SELECT Count(*) FROM Auteur ) ;
```

- Mais cela inclut les auteurs sans livre (sic) et les livres sans auteur …

```
SELECT (SELECT Count(FK_IdAuteur) FROM Livre )
       / SELECT Count(DISTINCT FK_IdAuteur) FROM Livre;
```

- S'il y avait une colonne Prix on aurait pu calculer le prix moyen d'un livre :

```
SELECT AVG ( Prix ) FROM Livre ;  -- voir idem par auteur :)
SELECT FK_IdAuteur, AVG ( Prix ) FROM Livre GROUP BY FK_IdAuteur ;
```

# 3. Syntaxe SQL

## *Un peu plus loin dans SQL*

*Requêtes multi-tables, requêtes imbriquées, vues*

# Jointures … théorie…

- Lié à l'approche relationnelle dans la conception des bases de données.
- Mise en relations de 2 (ou plusieurs) tables puis affichage du résultat :
  - Produit cartésien = toute les combinaisons possibles (**a**, **b**) avec **a** provenant de la 1ere table et **b** de la 2eme…
  - => peu utile sans critère de jointure

```
sqlite:> SELECT * FROM Auteur, Livre;

PK_IdAuteur  Nom_Auteur  Prenom_Auteur  PK_ISBN     Titre                AnneePubli  FK_IdAuteur
-----------  ----------  -------------  ----------  -------------------  ----------  -----------
1            Faure       Jacques        209178527X  Almanach Vermot 2010             1
1            Faure       Jacques        2070628035  Aya de Yopougon      3           3
1            Faure       Jacques        209178527X  Automatique et infor 2           2
1            Faure       Jacques        XXXXSQSS    Aya de youpougon 2               3
2            NEY         Henry          209178527X  Almanach Vermot 2010             1
2            NEY         Henry          2070628035  Aya de Yopougon      3           3
2            NEY         Henry          209178527X  Automatique et infor 2           2
2            NEY         Henry          XXXXSQSS    Aya de youpougon 2               3
3            Abouet      Marguerite     209178527X  Almanach Vermot 2010             1
3            Abouet      Marguerite     2070628035  Aya de Yopougon      3           3
3            Abouet      Marguerite     209178527X  Automatique et infor 2           2
3            Abouet      Marguerite     XXXXSQSS    Aya de youpougon 2               3
```

Equivalent à

```
SELECT *
FROM Auteur JOIN Livre ;
```

# Jointures … théorie…

- Mais ce n'est pas ça qui m'intéresse…
  - Je voudrais les auteurs avec « en face » le titre des livres qu'ils ont écrit …
  - Donc les livres dont la valeur de clef étrangère (FK_IdAuteur) est la même que celle de son auteur !
  - En d'autres termes : PK_idAuteur = FK_IdAuteur…

```
sqlite:> SELECT * FROM Auteur, Livre;

PK_IdAuteur  Nom_Auteur  Prenom_Auteur  PK_ISBN     Titre                AnneePubli  FK_IdAuteur
-----------  ----------  -------------  ----------  -------------------  ----------  -----------
1            Faure       Jacques        209178527X  Almanach Vermot 2010             1
1            Faure       Jacques        2070628035  Aya de Yopougon      3           3
1            Faure       Jacques        209178527X  Automatique et infor 2           2
1            Faure       Jacques        XXXXSQSS    Aya de youpougon 2               3
2            NEY         Henry          209178527X  Almanach Vermot 2010             1
2            NEY         Henry          2070628035  Aya de Yopougon      3           3
2            NEY         Henry          209178527X  Automatique et infor 2           2
2            NEY         Henry          XXXXSQSS    Aya de youpougon 2               3
3            Abouet      Marguerite     209178527X  Almanach Vermot 2010             1
3            Abouet      Marguerite     2070628035  Aya de Yopougon      3           3
3            Abouet      Marguerite     209178527X  Automatique et infor 2           2
3            Abouet      Marguerite     XXXXSQSS    Aya de youpougon 2               3
```

# Jointures Interne

- Permet de joindre des données de tables différentes :
  - Exemple : afficher les auteurs et les livres qu'ils ont écrit…
- => indiquer
  - les tables à joindre
  - Le critère de jointure : ex. l'id de l'auteur est le même que la valeur de la clef étrangère dans l'autre table

```
SELECT PK_ISBN as ISBN, Titre, Nom_Auteur as Nom,
                               Prenom_Auteur as "Prénom"
FROM Livre, Auteur
WHERE
   Auteur.Pk_IDAuteur = Livre.FK_IdAuteur
```

  - Appelée aussi jointure interne : la syntaxe générale est :

```
SELECT Col1, Col2, ..., Col3
FROM Table1 INNER JOIN Table2 ON
   Table1.Pkxxxx = Table2.Fkxxxx
```

# Illustration Jointure

**TABLE Auteur**

| PK_IdAuteur | Nom_Auteur | Prenom_Auteur |
|---|---|---|
| 1 | NEY | Henry |
| 2 | FAURE | Jacques |
| 3 | ABOUET | Marguerite |
| 4 | NONIM | Anne |
| 5 | CONUT | Alain |

2 auteurs sans lvire

**TABLE Livre**

| PK_ISBN | Titre | AnneePubli | FK_IdAuteur |
|---|---|---|---|
| 209178527X | Automatique et | 01/01/2008 | 1 |
| 2851102869 | Almanach Vermot | 01/01/2010 | 2 |
| 2070628035 | Aya de Yopo | 01/01/2010 | 3 |
| kdjdjdjdj | Le chien qui pl | | 1 |
| kdjdjddsfs | Le chat qui ple | | 1 |
| dddddd | L'automatique p | | 2 |
| 209X0239 | Roman de Renart | 1300:01:01 | |
| 209XEese | 100 H | 2002:01:01 | |

2 livres sans auteurs

- Jointure « manuelle »
  - `SELECT Nom_Auteur, Prenom_Auteur, Titre`
    `FROM Auteur, Livre`
    `WHERE PK_IdAuteur = FK_IdAuteur ;`

| Nom_Auteur | Prenom_Auteur | Titre |
|---|---|---|
| NEY | Henry | Automatique et |
| FAURE | Jacques | Almanach Vermot |
| ABOUET | Marguerite | Aya de Yopo |
| NEY | Henry | Le chien qui pl |
| NEY | Henry | Le chat qui ple |
| FAURE | Jacques | L'automatique p |

Livre avec auteur
(PK = FK) est VRAI
=> Jointure Interne = Intersection


Auteurs sans livre
(PK = FK) est FAUX car
pas de Livre avec leur PK

Livre sans auteur
(PK = FK) est FAUX car
FK est NULL

# Illustration Jointure

**TABLE Auteur**

| PK_IdAuteur | Nom_Auteur | Prenom_Auteur |
|---|---|---|
| 1 | NEY | Henry |
| 2 | FAURE | Jacques |
| 3 | ABOUET | Marguerite |
| 4 | NONIM | Anne |
| 5 | CONUT | Alain |

**TABLE Livre**

| PK_ISBN | Titre | AnneePubli | FK_IdAuteur |
|---|---|---|---|
| 209178527X | Automatique et | 01/01/2008 | 1 |
| 2851102869 | Almanach Vermot | 01/01/2010 | 2 |
| 2070628035 | Aya de Yopo | 01/01/2010 | 3 |
| kdjdjdjdj | Le chien qui pl | | 1 |
| kdjdjddsfs | Le chat qui ple | | 1 |
| dddddd | L'automatique p | | 2 |
| 209X0239 | Roman de Renart | 1300:01:01 | |
| 209XEese | 100 H | 2002:01:01 | |

- Jointure explicite
  - **SELECT** Nom_Auteur, Prenom_Auteur, Titre
    **FROM** Auteur **INNER JOIN** Livre **ON** PK_IdAuteur = FK_IdAuteur ;

Livre avec auteur (PK = FK) est VRAI

On joint les 2 tables en une table « virtuelle »

Critère de Jointure

- Jointure naturelle
  - Possible lorsqu'on veut faire la jointure sur des colonnes ayant le même nom (ex. renommer PK_IdAuteur et FK_IdAuteur en IdAuteur dans leurs tables respectives)

# Jointures Interne Naturelle

- Manière de simplifier la jointure en utilisant le même libellé pour la clef primaire et la clef étrangère. La jointure s'écrit alors :

```
SELECT Col1, Col2, Col3
FROM Table1 NATURAL JOIN Table2 ;
```

  - Exemple :
    - Imaginons que les colonnes Pk_IdAuteur et FK_IdAuteur soient renommées IdAuteur …(dans les 2 tables Auteur et Livre donc)
  - La requête est alors simplifiée en :

```
SELECT Nom_Auteur, Titre
FROM Livre NATURAL JOIN Auteur
;
```

# Jointures Externe

- Joint des tables dans lesquelles on n'a pas pour chaque enregistrement de lien avec la 2ème (cas des valeurs nulles)
  - Exemple : afficher les auteurs et les livres y compris quand les auteurs n'ont pas encore publié de livre (donc leur id n'apparaît nulle part dans la colonne Clef étrangère FK_IdAuteur de la table livre )
- Syntaxe :
  - les tables à joindre
  - Le critère de jointure : ex. l'id de l'auteur est le même que la valeur de la clef étrangère dans l'autre table

```
SELECT *
FROM Auteur LEFT OUTER JOIN Livre
     ON PK_IdAuteur = FK_IdAuteur;
```

  - Cad : on fait la jointure sur la clef, on affiche les valeurs qui vont bien et on ajoute les valeurs de la table de gauche (LEFT)
  - RIGHT fait la meme chose avec la table de droite et FULL le fait sur les 2 tables
  - => On voit apparaître des enregistrement avec par endroit aucune valeur en face de la colonne.

# Illustration Jointure Externe

**TABLE Auteur**

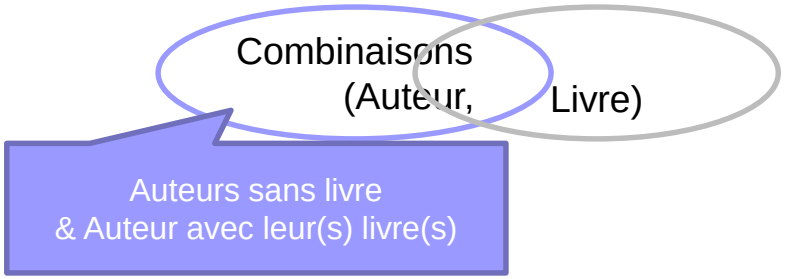
| PK_IdAuteur | Nom_Auteur | Prenom_Auteur |
|---|---|---|
| 1 | NEY | Henry |
| 2 | FAURE | Jacques |
| 3 | ABOUET | Marguerite |
| 4 | NONIM | Anne |
| 5 | CONUT | Alain |

**TABLE Livre**

| PK_ISBN | Titre | AnneePubli | FK_IdAuteur |
|---|---|---|---|
| 209178527X | Automatique et | 01/01/2008 | 1 |
| 2851102869 | Almanach Vermot | 01/01/2010 | 2 |
| 2070628035 | Aya de Yopo | 01/01/2010 | 3 |
| kdjdjdjdj | Le chien qui pl | | 1 |
| kdjdjddsfs | Le chat qui ple | | 1 |
| dddddd | L'automatique p | | 2 |
| 209X0239 | Roman de Renart | 1300:01:01 | |
| 209XEese | 100 H | 2002:01:01 | |

- Jointure externe sur la table de gauche
  - => on veut tous les enregistrements de la table Auteur et lorsqu'ils ont écrit des livres, on veut leur titre.
  - ```
    SELECT Nom_Auteur, Prenom_Auteur, Titre
    FROM Auteur LEFT OUTER JOINT Livre
    ON PK_IdAuteur = FK_IdAuteur ;
    ```
  - Remarque : SQLITE ne gère pas les jointures à droite (ex. liste de livres avec ou sans auteur `(FK_IdAuteur = NULL)`. Mais aisément faisable …

| Nom_Auteur | Prenom_Auteur | Titre |
|---|---|---|
| NEY | Henry | Automatique et |
| FAURE | Jacques | Almanach Vermot |
| ABOUET | Marguerite | Aya de Yopo |
| NEY | Henry | Le chien qui pl |
| NEY | Henry | Le chat qui ple |
| FAURE | Jacques | L'automatique p |
| NONIM | Anne | |
| CONUT | Alain | |


    ```

# Vues … *and what are they used for*

- Qu'est ce ?
  - Sorte de requête SELECT enregistrée dans la base de données,
  - Peut également se manipuler comme un table dans une autre requête !
  - Sorte de table virtuelle donc
- Intérêt ?
  - Les requêtes sur la vues masquent la structure des tables sur laquelle elle est basée
  - Réutilisable
  - Simplifie des requêtes complexes
- Limitations
  - Accès en **lecture seule** : donc pas d'ajout, modification, suppression d'enregistrements.
- Syntaxe : simple as that ;

```
CREATE VIEW nom_vue AS
Clause SELECT ;
```

# Vues … *Exemple*

- Reprenons l'exemple « compliqué » :

```
SELECT Nom_Auteur, COUNT(PK_ISBN)
FROM  Livre INNER JOIN Auteur
ON Pk_IdAuteur = Fk_IdAuteur
GROUP BY Fk_IdAuteur ;
```

- Vue qui reprend la jointure (car peut être utilisée par de nombreuses autres requêtes)

```
CREATE VIEW JointLivreAuteur AS
   SELECT Pk_IdAuteur, Nom_Auteur, Prenom_Auteur, PK_ISBN,
                                                  Titre, AnneePubli
   FROM  Livre INNER JOIN Auteur
   ON Pk_IdAuteur = Fk_IdAuteur;
```

- Utilisation de la vue …

Of course, la colonne doit être présente dans la vue

```
SELECT * FROM JointLivreAuteur;
```

la 2e requête ressemble à une requête monotable

```
SELECT Nom_Auteur , COUNT(PK_ISBN)
FROM JointLivreAuteur
GROUP BY Nom_Auteur;
```

# Requêtes imbriquées : principes

- Un sélect permet de filtrer l'affichage des données d'une table… mais son résultat peut intervenir dans une autre notamment.
- Peut être utilisée pour trouver une valeur qui servira de contrainte pour l'autre requête :
  - *pour simplifier une requête* : par ex. au lieu d'avoir une condition longue, on peut la découper en 2 requêtes imbriquées => plus facile a tester (évite le big bang/all in).
  - *pour effectuer des calculs* qui serviront dans la requête : moyenne, constitution de liste…
- Of course, on peut aussi utiliser une vue ☺

# Requêtes imbriquées : exemple

- Liste des livres dont le nom de l'auteur commence par NE :
  - Version jointure

```
SELECT Titre
FROM Auteur INNER JOIN Livre
ON PK_IdAuteur = FK_IdAuteur
WHERE Nom_Auteur LIKE "NE%";
```

  - Version requêtes imbriquées

```
SELECT Titre
FROM Livre
WHERE FK_IdAuteur IN (
  SELECT PK_IdAuteur
  FROM Auteur
  WHERE Nom_Auteur LIKE "NE%"
);
```

Si plusieurs auteurs sinon **=**

Requête « renvoie » comme résultat « 1 » càd identifiant de l'auteur NEY

# Retour Opérateurs d'agrégation

- Exemple
  - Calculer le nombre de livres dans la table Livre

```
SELECT Count(id) FROM Livre ;
```

  - Nombre de livre par auteur :

```
SELECT Fk_IdAuteur , COUNT(PK_ISBN)
FROM Livre GROUP BY Fk_IdAuteur;
```

  - Mais si on veut rendre le résultat un peu plus clair, il faut ajouter le nom de l'auteur => nécessité de faire une jointure

```
SELECT Nom_Auteur, COUNT(PK_ISBN)
FROM  Livre INNER JOIN Auteur
ON Pk_IdAuteur = Fk_IdAuteur
GROUP BY Nom_Auteur ;
;
```

  - **Attention** : les colonnes mentionnées dans le Select doivent apparaître dans le group By…

# More on GROUP BY…

- Rappel de la contrainte colonne dans SELECT => dans Group By.
  - Si on a 2 auteurs ayant le même nom… la requête précédente les confondra … et cumulera leur nombre de livre (logique si critère = Nom_Auteur…)
  - Alors ajoutons le prénom de l'auteur ?

```
SELECT Nom_Auteur, Prenom_auteur, COUNT(PK_ISBN)
FROM  Livre INNER JOIN Auteur
ON Pk_IdAuteur = Fk_IdAuteur
GROUP BY Nom_Auteur
;
```

| | Nom_Auteur | Prenom_Auteur | COUNT(PK_ISBN) |
|---|---|---|---|
| 1 | ABOUET | Marguerite | 1 |
| 2 | FAURE | Jacques | 2 |
| 3 | HERBERT | James | 3 |
| 4 | NEY | Henry | 3 |

  - Wrong… la requête sera :
    - soit refusée, car Prenom_auteur absent du Group By
    - soit acceptée mais avec résultats indésirés. Par exemple ici, il y avait 2 auteurs avec le même nom de famille… SQlite a donc regroupé les livres relevant de ce nom et à calculé le nombre de livres (des 2 donc).
  - SQLite très permissif cf tr. suivant

# More on GROUP BY…

- La règle générale des requêtes avec fonctions d'agrégation est que les colonnes du SELECT (output) doivent être soit dans une fonction d'agrégation soit mentionnées dans la clause Group By
- Certains SGBD peuvent le tolérer (SQLite, MySQL) ce qui peut poser problème si les valeurs de  colonne n'est pas unique (cf ex. pcdent avec 2 auteurs ayant le même nom).

```
SELECT Nom_Auteur, Prenom_auteur, COUNT(PK_ISBN)
FROM  Livre INNER JOIN Auteur
ON Pk_IdAuteur = Fk_IdAuteur
GROUP BY Nom_Auteur, Prenom_Auteur
;
```

| | Nom_Auteur | Prenom_Auteur | COUNT(PK_ISBN) |
|---|---|---|---|
| 1 | ABOUET | Marguerite | 1 |
| 2 | FAURE | Jacques | 2 |
| 3 | HERBERT | Franck | 2 |
| 4 | HERBERT | James | 1 |
| 5 | NEY | Henry | 3 |

- Autre solution, utilisation de la concaténation Nom prénom (but **) … par ex avec CONCAT( Nom_Auteur, Prenom_Auteur) MAIS SQLite ne le supporte pas (il est lite).

** Alexandre Dumas