

# **Apprendre à manipuler les données d'une base SQL**

# Table des matières

|  |           |
|--|-----------|
| <b>I. Contexte</b>                         | <b>3</b>  |
| <b>II. Ajouter des données</b>             | <b>3</b>  |
| <b>III. Exercice : Appliquez la notion</b> | <b>6</b>  |
| <b>IV. Lire des données</b>                | <b>6</b>  |
| <b>V. Exercice : Appliquez la notion</b>   | <b>12</b> |
| <b>VI. Modifier des données</b>            | <b>12</b> |
| <b>VII. Exercice : Appliquez la notion</b> | <b>13</b> |
| <b>VIII. Supprimer des données</b>         | <b>14</b> |
| <b>IX. Exercice : Appliquez la notion</b>  | <b>16</b> |
| <b>X. Essentiel</b>                        | <b>16</b> |
| <b>XI. Auto-évaluation</b>                 | <b>16</b> |
| A. Exercice final .....                    | 16        |
| B. Exercice : Défi .....                   | 18        |
| <b>Solutions des exercices</b>             | <b>18</b> |

## I. Contexte

**Durée :** 1 h

**Environnement de travail :** MariaDB, DataGrip

**Pré-requis :** Connaître les principes d'une base de données relationnelle

### Contexte

Les fonctions de manipulation de données sont regroupées en quatre catégories. Le terme usuellement employé est **CRUD**, pour :

- **Create** : création de données (ce cours emploiera le terme d'« insertion »)
- **Read** : lecture de données
- **Update** : mise à jour de données
- **Delete** : suppression de données

Ce sont les quatre opérations standard d'une base de données. Ces opérations sont disponibles sur tout SGBD.

Afin de pouvoir réaliser les exercices de ce cours, il est nécessaire de disposer d'une base de données utilisable et de savoir y exécuter des requêtes. Dans le cadre de ce cours, le SGBD MariaDB est recommandé. Il est simple d'installation, disponible sur tous les systèmes d'exploitation usuels.

Pour pouvoir accéder à la base de données et exécuter nos requêtes SQL, nous utiliserons l'EDI de base de données DataGrip<sup>1</sup> développé par JetBrains, auquel vous avez accès dans le cadre de l'offre GitHub Student Developer Pack<sup>2</sup>.

## II. Ajouter des données

### Objectif

- Insérer des données dans une table

### Mise en situation

La première chose à faire après avoir créé notre base de données est de pouvoir insérer des lignes de données dans les tables. L'instruction permettant de le faire est **INSERT**.

### Méthode La requête INSERT

La requête **INSERT** utilise le mot-clé **INSERT INTO**, suivi du nom de la table et des colonnes qui vont recevoir une valeur. Le mot-clé **VALUES** permet ensuite de préciser leurs valeurs.

```
1 INSERT INTO Table
2   (colonne1, colonne2, colonne3)
3 VALUES
4   (valeurColonne1, valeurColonne2, valeurColonne3)
```

La liste des valeurs doit être dans le même ordre que la liste des colonnes.

<sup>1</sup> <https://www.jetbrains.com/fr-fr/datagrip/>

<sup>2</sup> <https://education.github.com/pack>

### Exemple

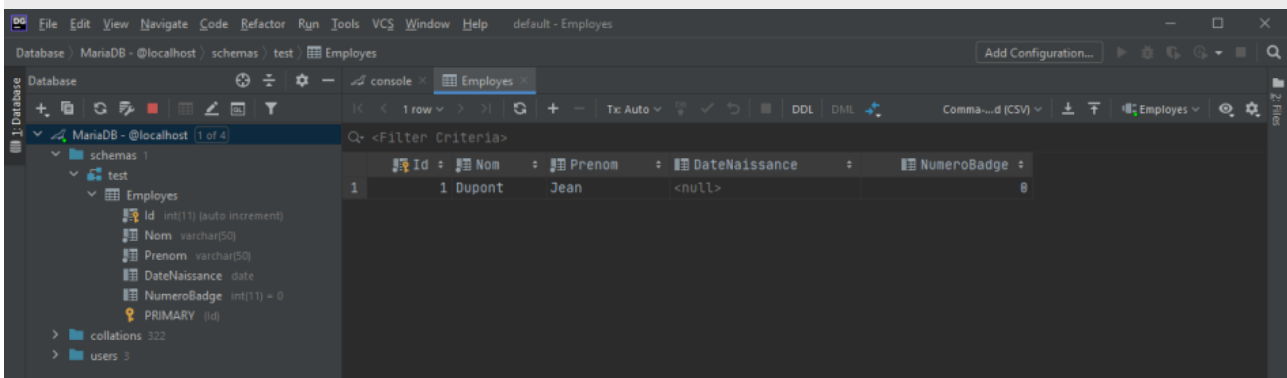
Admettons que nous ayons une base de données permettant de gérer des employés :

```
1 CREATE TABLE Employes
2 (
3     Id int NOT NULL AUTO_INCREMENT, -- Identifiant unique, calculé automatiquement
4     Nom varchar(50) NOT NULL, -- Chaîne de caractères, obligatoirement renseignée
5     Prenom varchar(50) NOT NULL, -- Chaîne de caractères, obligatoirement renseignée
6     DateNaissance date, -- Date, facultative. Vide si non renseignée.
7     NumeroBadge int DEFAULT 0, -- Nombre entier, facultatif avec une valeur par défaut si non
    renseigné
8     PRIMARY KEY (Id)
9 )
```

On remarque que, parmi ces données, seules deux sont obligatoires : le nom et le prénom. Les autres sont soit facultatives, soit automatiquement calculées. Notre requête d'insertion doit donc comporter au moins ces deux champs :

```
1 INSERT INTO Employes
2     (Nom, Prenom)
3 VALUES
4     ('Dupont', 'Jean')
```

Les autres valeurs de Jean Dupont recevront la valeur attribuée par le SGBD (celle calculée pour l'ID, la valeur par défaut, 0, pour le numéro de badge et une valeur nulle pour la date) :



Pour leur donner une valeur spécifique, il suffit de les rajouter dans la liste des colonnes et de leur attribuer une valeur. Ainsi, toutes ces requêtes sont des insertions valides dans cette table :

```
1 INSERT INTO Employes (Prenom, Nom) VALUES ('Jean', 'Dupont');
2 INSERT INTO Employes (Prenom, Nom, DateNaissance) VALUES ('Jean', 'Dupont', '2000-01-01');
3 INSERT INTO Employes (Prenom, Nom, DateNaissance, NumeroBadge) VALUES ('Jean', 'Dupont', NULL,
    5);
```

### Attention Valeur par défaut != valeur nulle

NULL signifie une absence de valeur : il n'y a rien dans la colonne à cette ligne-là, c'est une case vide. Il est important de distinguer l'absence de valeur d'une valeur par défaut : une valeur par défaut est une valeur attribuée automatiquement si elle n'est pas renseignée.

Par exemple, la colonne NumeroBadge a 0 comme valeur par défaut. La valeur 0 n'est pas vide, mais représente une valeur : cela signifie qu'il existe un numéro de badge égal à 0 que l'on peut utiliser. Si on voulait exprimer le fait que notre employé n'avait pas de badge, alors NULL serait un meilleur choix.

**Remarque** Insérer sans liste de colonnes

La liste des colonnes est facultative. En cas d'absence, le SGBD considérera que les valeurs sont dans l'ordre des colonnes dans la table. Pour la table `Employes`, il faudra écrire :

```
1 INSERT INTO
2   Employes
3 VALUES
4   (3, 'Dupont', 'Jean', NULL, 0)
```

Dans ce cas, il est nécessaire de renseigner un certain nombre de valeurs supplémentaires.

Le plus problématique est la colonne `Id`. Dans les cas précédents, elle est renseignée automatiquement par la base de données en tant que clé primaire auto-incrémentée. Or, si l'on ne précise pas les colonnes, il est indispensable de lui donner une valeur qui ne doit pas être déjà existante dans la base de données, puisque c'est une clé primaire.

**Il est fortement recommandé de citer la liste des colonnes utilisées dans une instruction `INSERT` pour éviter toute ambiguïté, et en particulier lorsqu'une table contient une clé primaire auto-incrémentée.**

**Les types de données**

Il faut faire attention aux données que l'on souhaite insérer : les valeurs données doivent correspondre au type de données attendues par la table et avoir la bonne longueur.

Si une donnée n'est pas du bon type, le SGBD renverra une erreur et n'insérera pas la ligne.

Si une donnée est trop longue (par exemple un texte de 10 caractères dans une colonne de 5), le comportement dépendra du SGBD, de sa version et de son paramétrage : certains renverront une erreur, d'autres inséreront une valeur tronquée sans lever d'avertissement.

Les types de données les plus utilisés sont :

- Les nombres. Si c'est un nombre décimal, alors le séparateur doit être un point : `36,12.5`.
- Les chaînes de caractères, entourées de guillemets simples : `'du texte'`. Si le texte doit contenir un guillemet simple, il faut le doubler : `'D' 'Artagnan'`.
- Les dates, écrites sous forme de chaînes de caractères. Il est recommandé de l'écrire au format `AAAA-MM-JJ HH:MM:SS` : `'2020-01-01 14:30:57'`. Un SGBD peut interpréter d'autres formats, mais cela dépend de son paramétrage.
- Les mots-clés comme `NULL`. Ils doivent être écrits sans guillemets, sinon ils seront interprétés comme du texte.

**Syntaxe** À retenir

- L'instruction pour ajouter des données dans une table est `INSERT INTO`.
- Il faut lui donner la liste des colonnes pour lesquelles des valeurs vont être insérées, puis leurs valeurs dans le même ordre.
- La liste des colonnes est facultative, auquel cas il faut respecter l'ordre et le nombre des colonnes de la table, mais l'omettre n'est pas recommandé.

**Complément**

[https://fr.wikipedia.org/wiki/Insert\\_\(SQL\)](https://fr.wikipedia.org/wiki/Insert_(SQL))

### III. Exercice : Appliquez la notion

#### Question

[solution n°1 p.19]

Écrivez une instruction permettant d'alimenter la table Employes pour Mme Jeanne Dupont, née le 10 septembre 1998 et ayant le badge n° 1002. Cette instruction doit laisser la base de données renseigner l'identifiant de ligne.

Pour rappel, la table Employes a été créée via la requête suivante :

```
1 CREATE TABLE Employes
2 (
3     Id int NOT NULL AUTO_INCREMENT, -- Identifiant unique, calculé automatiquement
4     Nom varchar(50) NOT NULL, -- Chaîne de caractères, obligatoirement renseignée
5     Prenom varchar(50) NOT NULL, -- Chaîne de caractères, obligatoirement renseignée
6     DateNaissance date, -- Date, facultative. Vide si non renseignée.
7     NumeroBadge int DEFAULT 0, -- Nombre entier, facultatif avec une valeur par défaut si non
      renseigné
8     PRIMARY KEY (Id)
9 )
```

#### Indice :

Dans ce cas précis, il est nécessaire de fournir la liste des colonnes à renseigner. N'oubliez pas les guillemets simples ' autour du texte et de la date.

#### Indice :

N'oubliez pas les guillemets simples ' autour du texte et de la date.

### IV. Lire des données

#### Objectif

- Lire des données dans une table

#### Mise en situation

Une fois que des données ont été insérées dans une table, il faut pouvoir les récupérer. Pour cela, nous allons utiliser l'instruction `SELECT`.

#### Syntaxe L'instruction SELECT

L'instruction permettant de lire des données dans une table se nomme `SELECT`. Elle va permettre de sélectionner les lignes et les colonnes que nous souhaitons récupérer :

```
1 SELECT
2     colonne1,
3     colonne2
4 FROM
5     Table
```

Il est également possible de récupérer toutes les colonnes grâce au caractère `*`. Dans certains cas, il peut être nécessaire de préciser le nom de la table en écrivant `NomDeTable.*`, notamment lorsque l'on veut récupérer des colonnes précises en plus du `*`.

```
1 SELECT * FROM Table;
2 SELECT Colonne1, Colonne2, Table.* FROM Table;
```

**Exemple**

```
1 SELECT * FROM employes ;  
2 SELECT Nom, Prenom, Employes.* FROM Employes
```

Cette requête aura pour effet de retourner d'abord la colonne Nom et Prenom puis viendra **toutes** les colonnes, en commençant par l'ID et ainsi de suite dans l'ordre dans lequel les colonnes s'affichent.

**Exemple**

Pour obtenir tous les noms et prénoms contenus dans la table Employes, on peut écrire :

```
1 SELECT  
2     Nom,  
3     Prenom  
4 FROM  
5     Employes
```

**Attention**

Toutes les colonnes listées doivent être suivies d'une virgule, à part la dernière. Cette syntaxe est stricte et tout manquement pourra causer un message d'erreur du SGBD.

**Retirer les doublons**

Il est possible que l'instruction `SELECT` renvoie, en apparence, plusieurs fois la même ligne. En effet, s'il y a plusieurs enregistrements « Jean Dupond », `SELECT` les renverra tous. Il s'agit en réalité de plusieurs lignes différentes, ayant notamment un Id différent, mais puisque cette colonne n'a pas été demandée, `SELECT` ne la renvoie pas.

Il est possible de demander à `SELECT` de ne pas renvoyer les doublons : cela se fait au moyen du mot-clé `DISTINCT`, qui doit être utilisé juste après le `SELECT`, avant toute liste de champs.

```
1 SELECT DISTINCT  
2     Nom,  
3     Prenom  
4 FROM  
5     Employes
```

Grâce à `DISTINCT`, s'il y a plusieurs lignes « Jean Dupond », une seule sera renvoyée.

**Exemple**

Si la table Employes contient (notez la ligne « Jean Dupond » en double) :

|   | Nom    | Prenom   |
|---|--------|----------|
| 1 | Dupond | Jean     |
| 2 | Martin | Jeanne   |
| 3 | Dupond | Jean     |
| 4 | Dupond | Frédéric |

Le DISTINCT renverra :

|   | Nom    | Prenom   |
|---|--------|----------|
| 1 | Dupond | Jean     |
| 2 | Martin | Jeanne   |
| 3 | Dupond | Frédéric |

#### Remarque

Si le mot-clé `DISTINCT` est très pratique, il est trop souvent utilisé pour dissimuler un problème de doublons, plutôt que d'essayer de le résoudre. Attention à ne pas en abuser !

#### Syntaxe

#### La clause WHERE

`SELECT` renvoie toutes les lignes de la table, mais, très souvent, on recherche une ligne ou un ensemble de lignes bien précis. Pour cela, il faut ajouter une clause `WHERE` suivie d'une condition qui permettra de filtrer les résultats. Une condition est composée d'une colonne, d'un opérateur et d'une valeur : le `SELECT` retournera ainsi toutes les lignes pour lesquelles la condition est vraie.

```

1 SELECT
2     colonne1,
3     colonne2
4 FROM
5     Table
6 WHERE
7     colonne1 = 'valeur'
    
```

Il est possible de séparer les conditions avec des `AND` (« et ») si l'on veut que toutes les conditions soient remplies, ou des `OR` (« ou ») si seules certaines conditions doivent l'être. Il est possible d'utiliser des parenthèses pour préciser la hiérarchie des conditions.

```

1 SELECT
2     colonne1,
3     colonne2
4 FROM
5     Table
6 WHERE
7     Condition1
8     AND Condition2
9     AND (Condition3 OR Condition4)
    
```



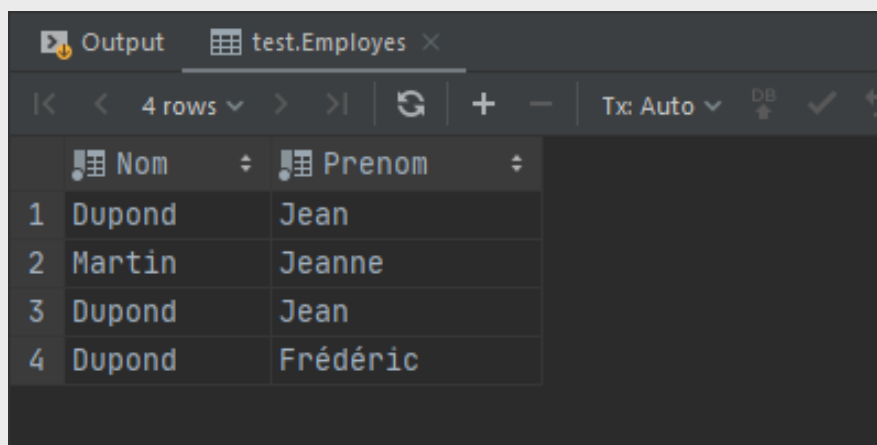
**Exemple**

Par exemple, pour trouver tous les employés ayant comme nom « Dupond » et ayant le badge n° 0 :

```
1 SELECT
2     Nom,
3     Prenom
4 FROM
5     Employes
6 WHERE
7     Nom = 'Dupond'
8     AND NumeroBadge = 0
```

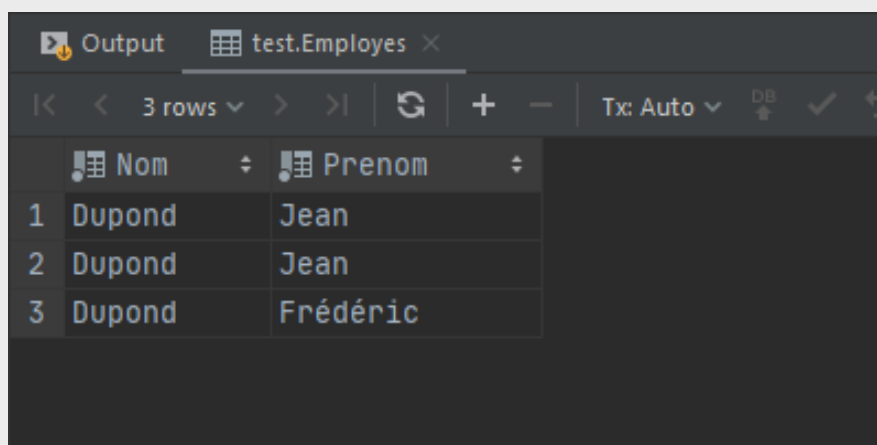
**Exemple**

Autre exemple, admettons que notre table contient ces données :



|   | Nom    | Prenom   |
|---|--------|----------|
| 1 | Dupond | Jean     |
| 2 | Martin | Jeanne   |
| 3 | Dupond | Jean     |
| 4 | Dupond | Frédéric |

Pour récupérer tous les employés ayant le nom de famille « Dupond », il faudra utiliser une clause `WHERE Nom = 'Dupond'`, ce qui retournera :



|   | Nom    | Prenom   |
|---|--------|----------|
| 1 | Dupond | Jean     |
| 2 | Dupond | Jean     |
| 3 | Dupond | Frédéric |

**Fondamental** Liste des opérateurs de comparaison

- = égal
- < inférieur
- <= inférieur ou égal
- > supérieur

- `>=` supérieur ou égal
- `<>` ou `!=` différent

#### Attention

`NULL` n'est pas une valeur, mais une absence de valeur. Ainsi, la condition `WHERE DateNaissance = NULL` sera toujours fausse, car `NULL` ne peut pas être égal à quoi que ce soit (même pas à un autre `NULL`).

Pour trouver une ligne n'ayant pas de valeur renseignée pour une colonne donnée, il faut utiliser l'opérateur `IS` : `WHERE DateNaissance IS NULL`. À l'inverse, si l'on veut que cette valeur soit renseignée : `WHERE DateNaissance IS NOT NULL`.

#### Question d'ordre

Dans quel ordre est-ce que le `SELECT` renvoie les lignes ? On pourrait imaginer qu'elles sont retournées dans leur ordre d'insertion, mais la vraie réponse est que, si rien n'est précisé, il n'y a pas d'ordre.

Le SGBD tente de répondre à une demande le plus efficacement possible : si l'utilisateur lui demande un résultat sans préciser l'ordre, le SGBD les renverra le plus vite possible. Généralement, cela veut dire dans l'ordre dans lequel les lignes sont écrites, mais ce n'est pas une certitude. Si l'ordre des lignes est important, il faut le préciser.

#### Syntaxe Clause ORDER BY

Pour préciser un ordre de restitution des données, il faut utiliser la clause `ORDER BY`, qui vient après la clause `WHERE`.

```
1 SELECT
2     colonne1,
3     colonne2
4 FROM
5     Table
6 WHERE
7     colonne1 = 'valeur'
8 ORDER BY
9     colonne1 ASC ou DESC,
10    colonne2 ASC ou DESC ...
```

Le tri se fait dans l'ordre des colonnes mentionnées dans la clause `ORDER BY`. Si rien n'est précisé, le tri se fait de façon ascendante (0 à 9, A à Z).

Il est possible de préciser `ASC` pour ascendant, ou `DESC` pour descendant (Z à A, 9 à 0).

#### Exemple

Pour obtenir la liste des employés du plus jeune au plus âgé, si la date de naissance est renseignée et, en cas d'égalité, par ordre alphabétique, il faudra écrire :

```
1 SELECT
2     Nom,
3     Prenom,
4     DateNaissance
5 FROM
6     Employes
7 WHERE
8     DateNaissance IS NOT NULL
9 ORDER BY
10    DateNaissance DESC,
11    Nom,
```

```
12      Prenom
1  SELECT
2  Nom,
3  Prenom
4  FROM
5  Employes
6  ORDER BY
7  Nom DESC
```

**Remarque**    **Tri sur des valeurs nulles**

Ce n'est pas par hasard si les valeurs nulles ont été exclues de la requête ci-dessus.

Un `ORDER BY` sur une colonne ayant des valeurs nulles rendra les lignes dans un ordre différent selon le SGBD utilisé.

Il est préférable de filtrer les valeurs nulles, ou bien d'écrire une condition de tri pour gérer les valeurs nulles (car il est possible de mettre une condition dans un `ORDER BY`) :

```
1  SELECT
2      Nom,
3      Prenom,
4      DateNaissance
5  FROM
6      Employes
7  WHERE
8      DateNaissance IS NULL
9  ORDER BY
10     DateNaissance DESC
11     Nom,
12     Prenom
```

La requête ci-dessus va venir récupérer tous les employés dont la date de naissance est nulle pour les trier dans l'ordre décroissant. Or, celle-ci étant nulle, il est impossible de les trier. Pour éviter toute confusion avec SQL et pour éviter que la requête ne renvoie des données dans le désordre, on ajoute d'autres options de tri comme le Nom et le Prénom.

**Syntaxe**    **À retenir**

- L'instruction pour lire les données d'une table est `SELECT . . . FROM`.
- Il est possible de supprimer les doublons grâce au `DISTINCT`.
- La clause `WHERE` permet de filtrer des données selon des conditions.
- La clause `ORDER BY` permet d'ordonner les lignes.

**Complément**

[https://fr.wikipedia.org/wiki/Select\\_\(SQL\)](https://fr.wikipedia.org/wiki/Select_(SQL))

## V. Exercice : Appliquez la notion

### Question

[solution n°2 p.19]

Écrivez une requête fournissant la liste du personnel (Nom, Prénom et Numéro de badge) par ordre alphabétique, puis par badge, pour tous les employés ayant un numéro de badge différent de 0.

Pour rappel, la table Employes possède la structure suivante :

```
1 CREATE TABLE Employes
2 (
3     Id int NOT NULL AUTO_INCREMENT, -- Identifiant unique, calculé automatiquement
4     Nom varchar(50) NOT NULL, -- Chaîne de caractères, obligatoirement renseignée
5     Prenom varchar(50) NOT NULL, -- Chaîne de caractères, obligatoirement renseignée
6     DateNaissance date, -- Date, facultative. Vide si non renseignée.
7     NumeroBadge int DEFAULT 0, -- Nombre entier, facultatif avec une valeur par défaut si non
      renseigné
8     PRIMARY KEY (Id)
9 )
```

### Indice :

Pour écrire une requête, il est plus simple de travailler petit bout par petit bout.

Faites une première requête qui renvoie la liste des colonnes attendues sans aucun filtre ni ordre, rajoutez ensuite le filtre et, enfin, rajoutez l'ordre.

## VI. Modifier des données

### Objectif

- Modifier des données déjà existantes dans une table

### Mise en situation

Une fois que des données ont été insérées dans une table, il faut pouvoir les modifier. Il serait possible de les supprimer puis de les recréer, mais il existe une méthode plus élégante.

#### Syntaxe

#### L'instruction UPDATE

L'instruction permettant de modifier des données dans une table se nomme `UPDATE`. Sa syntaxe est la suivante :

```
1 UPDATE
2     Table
3 SET
4     Colonne1 = Valeur1,
5     Colonne2 = Valeur2,
6     ...
7 WHERE
8     Liste de conditions
```

L'instruction `UPDATE` va modifier une ou plusieurs colonnes sur toutes les lignes correspondant à la condition demandée par la clause `WHERE`.

**Exemple**

Pour remplacer toutes les occurrences du nom « Dupond » par « Dupont » dans toute la table Employes, on peut écrire :

```
1 UPDATE
2   Employes
3 SET
4   Nom = 'Dupont'
5 WHERE
6   Nom = 'Dupond'
```

La requête suivante permet de voir les données à modifier. Il est préférable de la lancer avant, puis après l'UPDATE, afin de vérifier les modifications effectuées.

À noter que la clause WHERE est strictement équivalente dans le SELECT et dans l'UPDATE.

```
1 SELECT *
2 FROM Employes
3 WHERE
4   Nom = 'Dupont'
```

**Attention**

- De même que pour un SELECT, la clause WHERE est facultative dans un UPDATE. Cependant, les conséquences ne sont pas les mêmes, car l'instruction UPDATE modifie les données, tandis que le SELECT ne fait que les lire. Pour cette raison, quand on utilisera update, pour éviter toute mauvaise surprise, on va préférer l'utiliser sur un id ou une donnée précise.
- S'il n'y a pas de clause WHERE spécifiée, l'instruction UPDATE agira sur l'ensemble de la table. Ceci est rarement voulu dans un environnement professionnel en raison de la perte de données que cela représente.

Il est préférable de toujours écrire une clause WHERE et de la tester au préalable via un SELECT afin de s'assurer que les lignes sélectionnées sont les bonnes avant de les modifier.

**Syntaxe** **À retenir**

- L'instruction pour modifier des données dans une table est UPDATE ... SET ....
- La clause WHERE permet de filtrer des données selon des conditions. Elle est vivement recommandée.

**Complément**

[https://fr.wikipedia.org/wiki/Update\\_\(SQL\)](https://fr.wikipedia.org/wiki/Update_(SQL))

**VII. Exercice : Appliquez la notion****Question 1**

[solution n°3 p.19]

Le directeur du personnel a décidé qu'à présent, lorsqu'un employé n'a pas de badge personnel attribué, il reçoit un badge numéroté 9999. Pour s'assurer que la mise à jour va bien impacter les bons employés, écrivez dans un premier temps une requête SELECT renvoyant la liste de tous les employés ayant pour badge 0.

Pour rappel, la structure de la table Employes est la suivante :

```
1 CREATE TABLE Employes
2 (
3   Id int NOT NULL AUTO_INCREMENT, -- Identifiant unique, calculé automatiquement
4   Nom varchar(50) NOT NULL, -- Chaîne de caractères, obligatoirement renseignée
5   Prenom varchar(50) NOT NULL, -- Chaîne de caractères, obligatoirement renseignée
```

```

6   DateNaissance date, -- Date, facultative. Vide si non renseignée.
7   NumeroBadge int DEFAULT 0, -- Nombre entier, facultatif avec une valeur par défaut si non
    renseigné
8   PRIMARY KEY (Id)
9 )

```

### Indice :

Un `SELECT` possède la syntaxe suivante : `SELECT ... FROM ... WHERE ...`

### Question 2

[solution n°4 p.19]

En vous basant sur la requête précédente, écrivez une requête permettant de donner le numéro de badge 9999 à tous les employés ayant le badge 0.

### Indice :

Pour l'instruction `UPDATE ... SET ... WHERE ...`, la clause `WHERE` est strictement la même que pour le `SELECT` : il suffit de la copier.

## VIII. Supprimer des données

### Objectif

- Supprimer des données existantes dans une table

### Mise en situation

La dernière opération majeure que l'on peut réaliser sur le contenu de nos tables est de supprimer leur contenu. C'est la dernière lettre de **CRUD** : D pour *Delete*.

#### Syntaxe L'instruction DELETE

L'instruction permettant de supprimer des données dans une table se nomme `DELETE`. Contrairement aux instructions précédentes, le `DELETE` ne demande pas de liste de colonnes : il supprime la ligne entière. Sa syntaxe est la suivante :

```

1 DELETE FROM
2   Table
3 WHERE
4   Conditions

```

L'instruction `DELETE` supprimera toutes les lignes de la table mentionnée à la suite de la clause `FROM` et qui remplissent les conditions demandées dans la clause `WHERE`.

#### Attention

Contrairement à beaucoup de logiciels où un simple `ctrl+z` permet d'annuler une suppression, la suppression en base de données est irréversible. Il n'y a pas de corbeille ni de retour en arrière.

Tout comme pour l'instruction `UPDATE`, la clause `WHERE` d'un `DELETE` est facultative, mais très fortement conseillée. Encore une fois, pour cette raison, on va préférer l'utiliser sur un id ou une donnée précise.

Et de même, il est préférable de tester au préalable en effectuant un `SELECT`, de façon à s'assurer que les lignes retournées sont bien les lignes à supprimer.

**Remarque**    **Supprimer la valeur d'une colonne**

Si l'on souhaite simplement supprimer la valeur d'une colonne plutôt que supprimer toute une ligne, il suffit de faire un `UPDATE` en filtrant via une clause `WHERE` et en attribuant `NULL` à cette valeur. Ce n'est possible que si la colonne correspondante l'autorise.

**Syntaxe**    **L'instruction TRUNCATE**

Il arrive qu'il soit nécessaire de vider une table entière. Cela se fait souvent pour des tables dites « de travail », qui ne contiennent que des données temporaires.

Il est bien entendu possible d'écrire `DELETE FROM MaTable`, mais il existe une autre instruction qui est souvent employée à la place : le `TRUNCATE`. La syntaxe est la suivante :

```
1 TRUNCATE TABLE MaTable
```

**Remarque**    **Différence entre DELETE FROM et TRUNCATE**

Même si ces deux opérations vont supprimer toutes les lignes de la table en question, il y a toutefois une grande différence entre les deux.

`TRUNCATE TABLE` est une opération de structure, et non pas une opération de données : son opération est semblable au fait de supprimer puis de recréer une table vide. Cela signifie qu'elle ne supprime pas seulement les lignes : elle remet aussi les compteurs à zéro. Ainsi, le compteur d'une clé primaire auto-incrémentée revient à sa valeur de base avec un `TRUNCATE`, alors que `DELETE FROM` garde la valeur actuelle du compteur.

**Exemple**

Si notre table `Employes` contient 10 lignes (donc avec des employés dont les `Id` sont numérotés de 1 à 10) et que l'on joue les requêtes suivantes :

```
1 DELETE FROM Employes;
2 INSERT INTO Employes (Nom, Prenom) VALUES ('Dupont', 'Jean');
```

Alors Jean Dupont aura pour `Id` 11 : le compteur d'auto-incrémentation n'a pas été réinitialisé. En revanche, avec les requêtes suivantes :

```
1 TRUNCATE TABLE Employes;
2 INSERT INTO Employes (Nom, Prenom) VALUES ('Dupont', 'Jean');
```

Jean Dupont aura maintenant pour `Id` 1.

**Syntaxe**    **À retenir**

- L'instruction pour supprimer des données dans une table est `DELETE FROM`. La clause `WHERE` permet de filtrer des données selon des conditions, ce qui est vivement recommandé.
- L'instruction `TRUNCATE TABLE` permet de vider complètement une table et de réinitialiser les compteurs.

**Complément**

[https://fr.wikipedia.org/wiki/Delete\\_\(SQL\)](https://fr.wikipedia.org/wiki/Delete_(SQL))

## IX. Exercice : Appliquez la notion

### Question

[solution n°5 p.19]

Écrivez une requête permettant de sélectionner tous les employés n'ayant pas de date de naissance renseignée.

On peut utiliser l'instruction `SELECT *` dans ce but. Pour rappel, tester une valeur nulle se fait avec la condition `IS NULL`.

Ensuite, écrivez une requête permettant de supprimer toutes ces lignes.

### Indice :

La clause `WHERE` est identique entre le `SELECT` de la première requête et le `DELETE` de la seconde. Il suffit de la copier.

N'oubliez pas que deux requêtes SQL doivent être séparées par un `;`.

## X. Essentiel

## XI. Auto-évaluation

### A. Exercice final

#### Exercice 1

[solution n°6 p.20]

Exercice

Pour ajouter des données dans une table, il faut utiliser l'instruction...

- ☐ CREATE
- ☐ INSERT
- ☐ ADD

Exercice

Pour ajouter des données dans une table, il est nécessaire de spécifier la liste des colonnes...

- ☐ Toujours
- ☐ Jamais
- ☐ Parfois

Exercice

Le mot-clé `NULL` indique...

- ☐ Une absence de valeur
- ☐ Une valeur par défaut
- ☐ Une valeur interdite

Exercice

Pour lire des données dans une table, il faut utiliser l'instruction...

- ☐ SELECT
- ☐ READ
- ☐ LOCATE



## Exercice

Pour ne pas récupérer de doublons dans un `SELECT`, on peut utiliser...

- ☐ `SELECT MaColonne FROM DISTINCT MaTable`
- ☐ `SELECT MaColonne DISTINCT FROM MaTable`
- ☐ `SELECT DISTINCT MaColonne FROM MaTable`

## Exercice

Pour récupérer une ligne précise, il faut utiliser la clause...

- ☐ `WHERE`
- ☐ `LOCATE`
- ☐ `FIND`

## Exercice

Pour identifier les valeurs vides dans une colonne appelée `MaColonne`, il faut écrire...

- ☐ `WHERE MaColonne = NULL`
- ☐ `WHERE MaColonne IS NULL`
- ☐ `WHERE MaColonne IS NOT NULL`

## Exercice

Pour trier des valeurs dans un `SELECT`, il faut utiliser la clause...

- ☐ `ORDER`
- ☐ `ORDER BY`
- ☐ `SORT BY`

## Exercice

Pour modifier « Dupond » en « Dupont » dans une colonne `Nom` dans la table `Personnel`, il faut écrire...

- ☐ `UPDATE Personnel SET Nom = 'Dupont' WHERE Nom = 'Dupond'`
- ☐ `UPDATE Personnel (Nom = 'Dupont') WHERE Nom = 'Dupond'`
- ☐ `FIND Nom = 'Dupond' SET Nom = 'Dupont' IN Personnel`

## Exercice

Pour supprimer quelques lignes bien précises dans une table `Personnel`, il faut utiliser...

- ☐ `DELETE FROM Personnel`
- ☐ `TRUNCATE TABLE Personnel`
- ☐ `DELETE FROM Personnel WHERE (Condition)`
- ☐ `DROP TABLE Personnel`

## B. Exercice : Défi

Pour pouvoir gérer son stock, un magasin liste ses produits dans une base de données. Grâce à vos nouvelles connaissances, vous allez pouvoir gérer les données dans leurs tables.

### Question

[solution n°7 p.22]

Pour sa gestion des stocks, un magasin possède une base de données contenant une table Produits avec les informations suivantes :

- Id : ceci sera l'identifiant unique auto-incrémenté géré par la base
- Nom : le nom commercial du produit
- Code : le code de série utilisé en interne par le produit. Il s'agit d'une série de chiffres, mais celui-ci sera considéré comme un texte dans la base de données
- Prix : le prix de vente du produit
- Quantite : la quantité présente en stock
- DateCreation : la date de création du produit. Si elle n'est pas renseignée, la base de données mettra automatiquement la date du jour

La table sera créée par l'instruction suivante :

```
1 CREATE TABLE Produits
2 (
3     Id Int NOT NULL AUTO_INCREMENT,
4     Nom Varchar(50) NOT NULL,
5     Code Varchar(10) NOT NULL,
6     Prix Decimal(10,2) NOT NULL,
7     Quantite Int,
8     DateCreation DateTime DEFAULT NOW(),
9     PRIMARY KEY (Id)
10 )
```

Écrivez les quatre instructions nécessaires au fonctionnement de la table :

- Une instruction `INSERT INTO` pour alimenter la table. Cette instruction doit laisser la base alimenter les champs Id et DateCreation.
- Une instruction `SELECT` pour récupérer les informations de nom, prix et quantité en fonction du code du produit. S'il y a plusieurs lignes, elles doivent être triées par ordre de date de création décroissante.
- Une instruction `UPDATE` pour modifier la quantité de produits disponibles en fonction du code fourni.
- Une instruction `DELETE` pour supprimer tous les produits dont la quantité est à 0.

### Indice :

L'instruction `INSERT INTO` doit spécifier la liste des colonnes.

### Indice :

N'oubliez pas que le code est un texte, même si c'est une suite de chiffres. Il doit être mis entre guillemets simples.

### Indice :

Le tri en ordre décroissant se fait via le mot-clé `DESC`.

### Indice :

Les valeurs non fournies, par exemple dans le `INSERT INTO`, sont laissées à l'imagination de chacun. Un exemple de produit : Chaise, code 0000123, prix 19.9, quantité 10.

## Solutions des exercices

**p. 6 Solution n°1**

```
1 INSERT INTO Employes
2     (Nom, Prenom, DateNaissance, NumeroBadge)
3 VALUES
4     ('Dupont', 'Jeanne', '1998-9-10', 1002)
5
```

**p. 12 Solution n°2**

```
1 SELECT
2     Nom,
3     Prenom,
4     NumeroBadge
5 FROM
6     Employes
7 WHERE
8     NumeroBadge <> 0
9 ORDER BY
10    Nom,
11    Prenom,
12    NumeroBadge
```

**p. 13 Solution n°3**

```
1 SELECT
2     *
3 FROM
4     Employes
5 WHERE
6     NumeroBadge = 0
7 ;
```

**p. 14 Solution n°4**

```
1 UPDATE
2     Employes
3 SET
4     NumeroBadge = 9999
5 WHERE
6     NumeroBadge = 0
7 ;
```

**p. 16 Solution n°5**

```
1 SELECT
2     *
3 FROM
4     Employes
5 WHERE
6     DateNaissance IS NULL
```

```

7 ;
8 DELETE
9 FROM
10     Employes
11 WHERE
12     DateNaissance IS NULL
13
14
15

```

## Exercice p. 16 Solution n°6

### Exercice

Pour ajouter des données dans une table, il faut utiliser l'instruction...

- ☐ CREATE  
*C'est une instruction de structure pour créer une table, pas une instruction de données.*
- ☒ INSERT  
*Pour être plus précis, INSERT INTO.*
- ☐ ADD  
*C'est une instruction de structure qui permet de rajouter des colonnes à une table.*

### Exercice

Pour ajouter des données dans une table, il est nécessaire de spécifier la liste des colonnes...

- ☐ Toujours  
*Non, ce n'est pas toujours obligatoire, mais c'est recommandé.*
- ☐ Jamais  
*Spécifier la liste des colonnes est obligatoire si l'on veut utiliser le compteur d'une clé primaire.*
- ☒ Parfois  
*Certaines situations imposent de lister des colonnes. C'est parfois possible de s'en passer, mais toujours conseillé de le faire !*

### Exercice

Le mot-clé NULL indique...

- ☒ Une absence de valeur
- ☐ Une valeur par défaut  
*La valeur par défaut est spécifiée dans la définition de la colonne. Cependant, si on force une valeur à NULL, cela ignorera la valeur par défaut.*
- ☐ Une valeur interdite  
*Un attribut à NULL peut être interdit dans la définition d'une colonne via la définition NOT NULL. Si ce n'est pas le cas, NULL est parfaitement valable.*

### Exercice

Pour lire des données dans une table, il faut utiliser l'instruction...

☒ SELECT

☐ READ

*Cette instruction n'existe pas en SQL.*

☐ LOCATE

*Il s'agit d'une fonction de texte.*

### Exercice


---

Pour ne pas récupérer de doublons dans un `SELECT`, on peut utiliser...

☐ `SELECT MaColonne FROM DISTINCT MaTable`

☐ `SELECT MaColonne DISTINCT FROM MaTable`

☒ `SELECT DISTINCT MaColonne FROM MaTable`

 Le mot-clé `DISTINCT` doit être placé immédiatement après le `SELECT`.

### Exercice

---

Pour récupérer une ligne précise, il faut utiliser la clause...

☒ WHERE

☐ LOCATE

*LOCATE est une fonction permettant de trouver du texte dans un champ texte, pas une ligne dans une table.*

☐ FIND

*Cette commande n'existe pas en SQL.*

### Exercice

---

Pour identifier les valeurs vides dans une colonne appelée `MaColonne`, il faut écrire...

☐ `WHERE MaColonne = NULL`

*NULL n'est pas une valeur. L'opérateur `=` (égal) ne fonctionne que pour les valeurs. Cette condition ne fonctionnera jamais.*

☒ `WHERE MaColonne IS NULL`

*Il faut toujours utiliser `IS` ou `IS NOT` avec un `NULL`.*

☐ `WHERE MaColonne IS NOT NULL`

*Cette condition filtre au contraire les lignes pour lesquelles `MaColonne` n'est pas vide.*

### Exercice

---

Pour trier des valeurs dans un `SELECT`, il faut utiliser la clause...

☐ ORDER

*La clause s'écrit `ORDER BY`, pas `ORDER`.*

☒ ORDER BY

*Et rajouter ensuite la liste des ordres de tri.*

☐ SORT BY

*Cette clause n'existe pas en SQL.*

### Exercice

Pour modifier « Dupond » en « Dupont » dans une colonne Nom dans la table Personnel, il faut écrire...

- ☒ UPDATE Personnel SET Nom = 'Dupont' WHERE Nom = 'Dupond'
- ☐ UPDATE Personnel (Nom = 'Dupont') WHERE Nom = 'Dupond'
- ☐ FIND Nom = 'Dupond' SET Nom = 'Dupont' IN Personnel

### Exercice

Pour supprimer quelques lignes bien précises dans une table Personnel, il faut utiliser...

- ☐ DELETE FROM Personnel  
*Sans clause WHERE, ceci videra toute la table !*
- ☐ TRUNCATE TABLE Personnel  
*Cette instruction vide toute la table !*
- ☒ DELETE FROM Personnel WHERE (Condition)  
*Il ne faut jamais oublier le WHERE, sans quoi l'instruction videra toute la table.*
- ☐ DROP TABLE Personnel  
*Cette instruction ne va pas seulement vider la table, elle va la supprimer entièrement !*

### p. 18 Solution n°7

```

1 INSERT INTO Produits
2     (Nom, Code, Prix, Quantite)
3 VALUES
4     ('Chaise', '0000123', 19.9, 10)
5 ;
6 SELECT
7     Nom,
8     Prix,
9     Quantite
10 FROM
11     Produits
12 WHERE
13     Code = '0000123'
14 ORDER BY
15     DateCreation DESC
16 ;
17 UPDATE
18     Produits
19 SET
20     Quantite = 9
21 WHERE
22     Code = '0000123'
23 ;
24 DELETE
25 FROM
26     Produits
27 WHERE
28     Quantite = 0
    
```