

## Chapitre 3 : SQL - mise à jour des données

L'objectif de ce cours est d'apprendre à mettre à jour les informations contenues dans une base de données grâce à SQL.

### Introduction

Nous savons maintenant créer les éléments d'une base de données et effectuer des recherches sur les informations qu'elle contient. La dernière étape consiste à découvrir comment mettre à jour les données présentes dans les tables.

Les mises à jour peuvent être

- Des **ajouts** de nouvelles informations.
- Des **modifications** d'informations existantes.
- Des **suppressions** d'informations.

Il s'agit ici de mettre à jour les **données** présentes dans la base, et non de modifier la **structure** de la base elle-même (par exemple en ajoutant des tables).



Contrairement aux requêtes **SELECT**, les instructions SQL de manipulation des données vont modifier le contenu de la base. De plus, elles ne sont pas annulables une fois leur exécution terminée (sauf en utilisant le mécanisme de transaction étudié prochainement). Moralité : elles sont à manipuler avec **précaution**.

La suite de ce cours se base sur le modèle relationnel créé par les requêtes ci-dessous..

```
CREATE TABLE CLASSE (  
  ID                INTEGER                NOT NULL,  
  INTITULE          VARCHAR(10)          NOT NULL,  
  EFFECTIF          SMALLINT              NOT NULL,  
  PCT_RED           FLOAT                 DEFAULT 0,  
  CONSTRAINT PK_CLA PRIMARY KEY (ID)  
);  
  
CREATE TABLE ETUDIANT (  
  ID                INTEGER                NOT NULL PRIMARY KEY,  
  NOM               VARCHAR(100)          NOT NULL,  
  PRENOM            VARCHAR(100)          NOT NULL,  
  NUM_SECU          CHAR(15)              NOT NULL UNIQUE,  
  SEXE              CHAR(1)               NOT NULL,  
  DATE_NAISSANCE    DATE,  
  REDOUBLANT        CHAR(1)               DEFAULT 'N',  
  ID_CLA            INTEGER               NOT NULL,  
  CONSTRAINT CK_ETU_SEX CHECK (SEXE IN ('M', 'F')),  
  CONSTRAINT CK_ETU_RED CHECK (REDOUBLANT IN ( 'O', 'N' )),  
  CONSTRAINT FK_ETU_CLA FOREIGN KEY (ID_CLA) REFERENCES CLASSE(ID)  
);
```

## Ajout de données

L'ajout d'enregistrements dans une table s'effectue grâce à l'instruction SQL **INSERT**.

### Syntaxe

#### Exemple

```
INSERT INTO CLASSE (ID, INTITULE, EFFECTIF, PCT_RED)
VALUES (113, 'B1IG1', 24, 0);
```

Que fait la requête ci-dessus ?

**Elle insère une ligne dans la table CLASSE.**

La syntaxe de l'ordre **INSERT** est la suivante :

```
INSERT INTO <table> [ (<champs de la table> ) ] VALUES (<valeurs des champs>);
```

### Liste des champs

Il est possible d'omettre la liste des champs après le nom de la table, à deux conditions :

1. Donner une valeur pour tous les champs.
2. Respecter l'ordre de déclaration des champs.

**Exemple** : on veut maintenant ajouter la classe de BTS IG2. Trois requêtes sont proposées :

1. **INSERT INTO CLASSE VALUES** (114, 'B1IG2')
2. **INSERT INTO CLASSE VALUES** (114, 30, 'B1IG2', 5)
3. **INSERT INTO CLASSE VALUES** (114, 'B1IG2', 23, 5)

Quel est la requête correcte ? Pourquoi ?

**C'est la requête numéro 3. La requête 1 n'indique pas toutes les valeurs. La requête 2 inverse intitulé et effectif.**

Pour limiter les risques d'erreur, il est conseillé de toujours préciser la liste des champs de la tables dans les instructions **INSERT**.

### Liste des valeurs

L'ordre des valeurs doit évidemment correspondre à l'ordre des champs.

Il existe deux possibilités pour ne pas préciser la valeur d'un champ :

- Le mot-clé **NULL** laisse le champ vide (absence de valeur).
- L'omission du champ ou le mot-clé **DEFAULT** insère la valeur par défaut définie pour ce champ lors de la création de la table.

**Ecrire de trois manières différentes la requête qui ajoute la classe de B2IG1 (identifiant 115,**

effectif 24, aucun redoublant).

#### Solution 1

```
INSERT INTO CLASSE(ID, INTITULE, EFFECTIF, PCT_RED)  
VALUES (115, 'B2IG1', 25, 0);
```

#### Solution 2

```
INSERT INTO CLASSE(ID, INTITULE, EFFECTIF,  
PCT_RED) VALUES (115, 'B2IG1', 25, DEFAULT);
```

#### Solution 3

```
INSERT INTO CLASSE(ID, INTITULE, EFFECTIF)  
VALUES (115, 'B2IG1', 25);
```

## Ajout à partir d'une sélection

L'ordre **INSERT** peut également prendre comme valeurs d'insertion le résultat d'une instruction **SELECT**. Cette possibilité permet notamment de **recopier** tout ou partie du contenu d'une table dans une autre, à condition que leurs structures soient compatibles.

Supposons par exemple que l'administration souhaite stocker les étudiants redoublants dans une table séparée ETUDIANT\_RED ayant comme champs ID (clé primaire), NOM et PRENOM.

Ecrivez l'instruction SQL de création de cette table.

```
CREATE TABLE ETUDIANT_RED (  
    ID            INTEGER      NOT NULL PRIMARY KEY,  
    NOM           VARCHAR(100) NOT NULL,  
    PRENOM        VARCHAR(100) NOT NULL  
);
```

Complétez la requête qui permet de recopier les étudiants redoublants dans la nouvelle table.

```
INSERT INTO ETUDIANT_RED (ID, NOM, PRENOM)  
SELECT ID, NOM, PRENOM  
FROM ETUDIANT WHERE REDOUBLANT = 'O';
```

## Ajout et intégrité de la base

Dans un SGBDR, les mécanismes d'intégrité permettent de garantir la **cohérence** des données stockées. Lors de l'exécution d'un ordre **INSERT**, la ou les lignes ajoutées sont vérifiées et l'exécution échoue si l'intégrité n'est pas respectée.

#### Exemple

```
INSERT INTO CLASSE (ID, INTITULE, EFFECTIF, PCT_RED)
```

**VALUES** (113, 'B2IG2', 22, 10);

Cette requête réussit-elle ? Pourquoi ?

**Elle échoue car la clé primaire du nouveau tuple (113) est déjà attribuée à une autre classe.**

Exemple

```
INSERT INTO ETUDIANT  
(ID, NOM, PRENOM, NUM_SECU, SEXE, DATE_NAISSANCE, REDOUBLANT, ID_CLA)  
VALUES (330, 'MARQUES', 'Laurent', '1234567123458', 'M', NULL, 'O', 112);
```

Cette requête réussit-elle ? Pourquoi ?

**Non car la clé étrangère ID\_CLA ne correspond à aucune classe.**

Exemple

```
INSERT INTO ETUDIANT  
(ID, NOM, PRENOM, NUM_SECU, SEXE, DATE_NAISSANCE, REDOUBLANT, ID_CLA)  
VALUES (330, 'MARQUES', 'Laurent', NULL, 'M', NULL, 'O', 113);
```

Cette requête réussit-elle ? Pourquoi ?

**Elle échoue car le numéro de sécurité sociale doit obligatoirement être renseigné.**

L'ajout de contraintes d'intégrité lors de la création des tables permet d'augmenter la cohérence des informations qu'elle contient, en interdisant l'ajout de données corrompues .

## Modification de données

La modification d'une ou plusieurs lignes d'une table se fait grâce à l'instruction **UPDATE** .

## Syntaxe

Exemple

```
UPDATE CLASSE  
SET EFFECTIF = 25;
```

Que fait la requête ci-dessus ?

**Elle modifie l'effectif de toutes les classes.**

La syntaxe de l'instruction **UPDATE** est la suivante :

```
UPDATE <table> SET champ1 = valeur1 [, champ2 = valeur2...] [ WHERE condition ]
```

## Mise à jour de plusieurs champs

Il est possible de mettre à jour plusieurs champs de la même table en un seul **UPDATE**.

### Exemple

```
UPDATE ETUDIANT
SET NOM      = UPPER(NOM),
    PRENOM   = UPPER(PRENOM);
```

Que fait la requête ci-dessus ?

Elle met en majuscules le nom et le prénom de tous les étudiants.

## Mise à jour conditionnelle

L'instruction **UPDATE** permet de filtrer les données à mettre à jour, en ajoutant une clause **WHERE** similaire à celle d'un ordre **SELECT**.

Par exemple, on souhaite définir tous les étudiants de B1IG2 comme étant non redoublants.

Ecrivez la requête qui met à jour la base en conséquence.

```
UPDATE ETUDIANT
SET REDOUBLANT = 'N'
WHERE ID_CLA =
    (SELECT ID FROM CLASSE
     WHERE INTITULE = 'B1IG2' );
```

## Mise à jour et intégrité de la base

Tout comme un **INSERT**, l'instruction **UPDATE** peut échouer en cas de violation de l'intégrité de la base (contrainte de clé primaire, champ non nul, intégrité référentielle...).

### Exemple

```
UPDATE ETUDIANT
SET ID_CLA = 111
WHERE NOM='MARQUES' AND PRENOM='LAURENT';
```

Cette requête réussit-elle ? Pourquoi ?

Elle échoue car la clé étrangère ID\_CLA ne correspond à aucune classe existante.

## Suppression de données

La suppression d'une ou plusieurs lignes d'une table est réalisée grâce à l'ordre **DELETE**.

### Syntaxe

#### Exemple

```
DELETE FROM ETUDIANT WHERE NOM = 'BEPMAL';
```

Que fait la requête ci-dessus ?

**Elle supprime de la table ETUDIANT le ou les étudiant(s) dont le nom est BEPMAL.**

La syntaxe de l'ordre **DELETE** est la suivante:

```
DELETE FROM <table> [ WHERE <condition> ];
```

La clause **WHERE** permet de filtrer les enregistrements concernés par la suppression. Pour supprimer toutes les données d'une table, il suffit de ne pas ajouter de clause **WHERE**.

Ecrivez la requête qui supprime tous les étudiants dont le nom finit par T.

```
DELETE FROM ETUDIANT WHERE NOM LIKE '%T';
```



Une fois lancée, la suppression est irréversible. Pour limiter le risque d'erreur, il est conseillé de remplacer dans un premier temps le **DELETE** par un **'SELECT \*'**, ce qui permet de visualiser les données qui vont être effacées.

### Suppression et intégrité de la base

Une opération de suppression peut être refusée par le SGBDR si elle viole les règles d'intégrité.

#### Exemple

```
DELETE FROM CLASSE WHERE INTITULE = 'B1IG1';
```

Cette requête réussit-elle ? Pourquoi ?

**Elle échoue car il existe des étudiants appartenant à la classe B1IG1.**