Bases de données	NSI T^ale
Le Langage de Manipulation des Données	Cours/TD

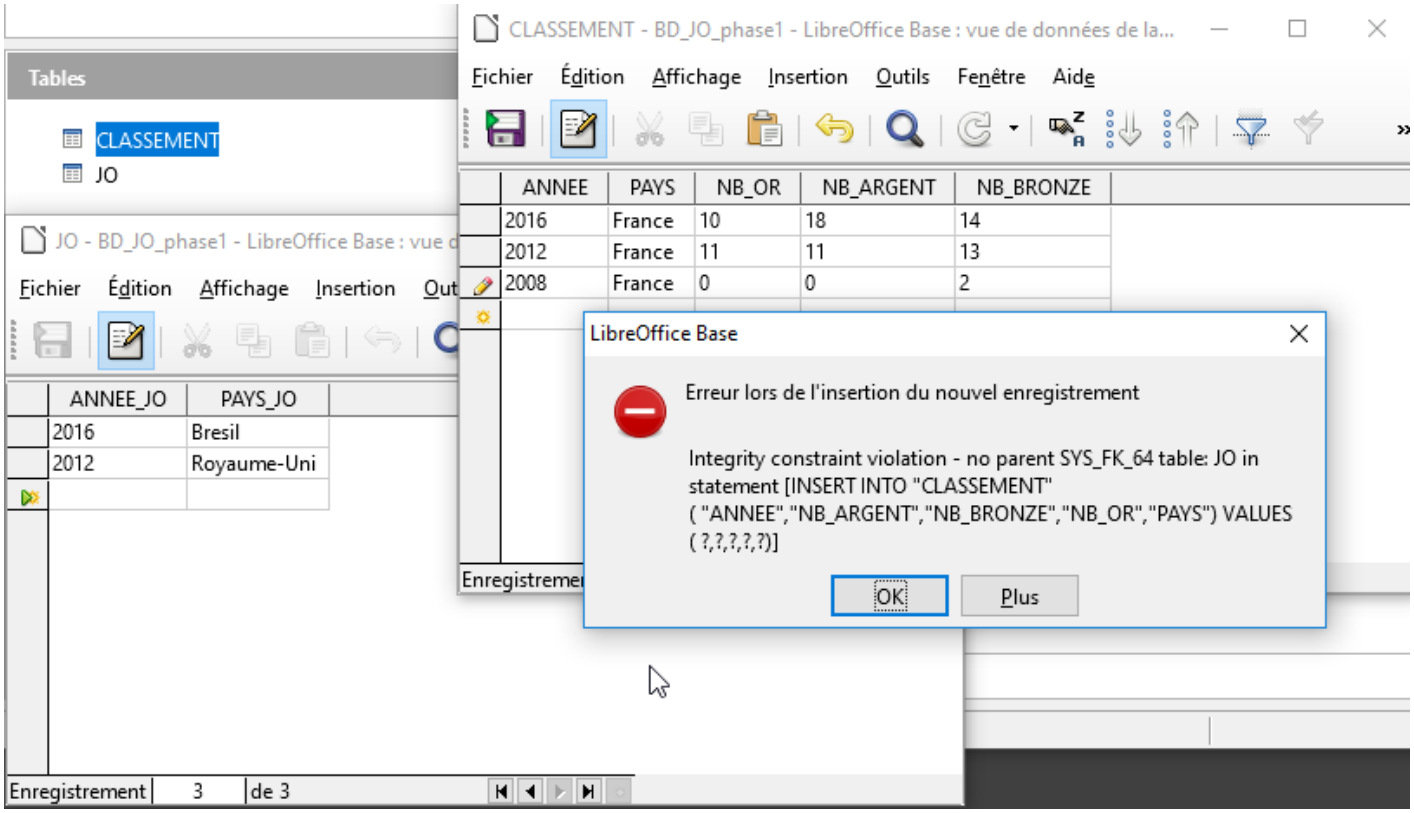
*

1. Introduction

Le LID fait partie d'un ensemble plus large d'instructions : le Langage de Manipulation de Données (LMD ou DML en anglais : Data Manipulation Language).
Nous verrons qu'il existe 3 autres instructions en plus de *SELECT* : *INSERT* pour ajouter des données à la Base, *DELETE* pour en supprimer et *UPDATE* pour effectuer des mises à jour.

2. Manipuler des données dans Base

Le but de cette étape est de montrer simplement l'ajout, la suppression et la modification de données via l'interface graphique.
En ouvrant les tables (icône à gauche Table > au milieu en bas, double clic sur chaque table ou clic droit et choisir Ouvrir...) ajouter les données suivantes (attention à l'ordre de saisie dans les tables) et constater l'erreur (une valeur dans une colonne clé étrangère doit pré-exister dans la colonne clé primaire correspondante) :



Insertion de données et erreur provoquée lorsqu'il n'y a pas de « clé parente » (la valeur d'une colonne clé étrangère n'existe pas dans la colonne clé primaire correspondante).

Mettre à jour une ligne se fait par simple clic sur une cellule à modifier. Supprimer une ligne se fait par clic droit sur les carrés gris à gauche de chaque ligne (plusieurs lignes peuvent être sélectionnées avec la touche SHIFT/Majuscule).

Créer et modifier une base de données via une interface graphique est pratique et ludique, mais peu utile quand il s'agit de manipuler un certain nombre de tables ainsi que de multiples lignes. Le langage SQL, avec les instructions du LDD (Langage de Définition des Données) et LMD, est alors utilisé.

3. Insertion de données

→ L'instruction INSERT.

Syntaxe

Format 1: insertion par saisie

```
insert into <nom_table> [(<colonne1>, <colonne>,...)]
values (<valeur1>, <valeur2> ,...);
```

Format 2: insertion à partir d'une autre table

```
insert into <nom_table> [(<colonne1>, <colonne> ,...) ]
select ... from ...;
```

où *<valeur1>* est la valeur à donner à la colonne1, donnée soit explicitement, soit par une variable paramétrée.

Deux contraintes principales doivent être respectées :

- les valeurs données doivent respecter les contraintes d'intégrité et, en particulier, le type de la colonne (numérique, chaîne, etc.),
- les chaînes de caractères sont écrites entre ' '.

Les instructions entre crochets [ ] sont facultatives. La barre verticale | exprime un choix (ou)

Exemple

```
-- 1. Ajouter un nouveau client en saisissant
      directement les valeurs .
insert into Client
values ( 'C012', 'Frederic', 'Toulouse');

-- 2. Ajouter dans Produit les produits de la table NouvProd valant
      plus de 100 euros .
insert into Produit
select * from NouvProd
where Npu > 100;
```

4. Suppression de données

→ L'instruction DELETE.

Syntaxe

```
delete from <nom_table>
[ where <condition>];
```

Exemples

```
-- 1. Supprimer tous les produits dont le prix de vente est
      inférieur a 10 euros .
delete from Produit
where Pu < 10;

-- 2. Supprimer toutes les factures antérieures au 01/01/15.
delete from Facture
where datefact < '01-JAN-15';
```

5. Modification de valeurs

→ L'instruction UPDATE.

Syntaxe

Format 1: modification par valeur donnée ou calculée

```
update <nom_table>
set(<colonne1> = <expression1>
[,<colonne2 > = <expression2> ,...]
[ where <condition >];
```

Format 2: modification à partir d'une autre table

```
update <nom_table>
set(<colonne1>, <colonne2>,...) = <sous-requête>
[ where <condition>];
```

<expression> est une valeur donnée directement ou calculée, et *<sous-requête>* est une requête *select* donnant autant de valeurs que de colonnes à mettre à jour.
Attention, les valeurs fournies doivent être compatibles en type et en nombre avec les colonnes à mettre à jour.

Exemples

```
-- 1. Tous les prix ont augmentés de 10%.
update Produit
set Pu = Pu * 1.1;

-- 2. Le nouveau prix unitaire du produit 'P03' est 35 euros .
update Produit
set Pu = 35
where Copro = 'P03';
```

```
09 |
10 | -- 3. Tous les prix de moins de 10 euros sont majorés de 0,50 euros.
11 | update Produit
12 | set Pu = Pu + 0.5
13 | where Pu < 10;
14 |
15 | -- 4. Mettre le produit P10 au même prix que le produit P11.
16 | update Produit
17 | set Pu = (select Pu from Produit
18 |           where Copro = 'P11')
19 | where Copro = 'P10';
```

6. Prise en compte des modifications

→ Les instructions COMMIT et ROLLBACK.

Syntaxe

```
01 | commit;
02 | rollback
```

Principe

Avec l'ordre *Commit*, les mises à jour sont validées et leur effet est effectivement matérialisé dans la base. Les valeurs fournies doivent être compatibles en type et en nombre avec les colonnes à mettre à jour.

Avec l'ordre *Rollback*, toutes les transactions effectuées depuis le dernier Commit sont annulées (on retourne donc dans un état cohérent de la base).

Remarque : Toute fin de session ou de programme génère toujours un *Commit*. On peut imposer/suspendre un Commit automatique après chaque mise à jour par la commande : *set autocommit on/off;*.