Insertion, modification et suppression

1. Insérer à l'aide d'INSERT

La syntaxe de base de l'ordre SQL d'insertion de données dans une table est la suivante :

INSERT [INTO] *nom_de_la_table_cible* [(*liste_des_colonnes_visées*)]
{VALUES (*liste_des_valeurs*) | *requête_select* | DEFAULT VALUES }

NOTA :

- la liste des colonnes visées peut être ommise à condition que l'ordre d'insertion concerne toutes les colonnes de la table.
- la liste des valeurs peut être remplacée par un constructeur de lignes valuées pour une insertion de plusieurs lignes en un seul ordre, mais rares sont les SGBD à l'accepter (Oracle est l'un des rares SGBD à accepter cette syntaxe).

ATTENTION : l'ordre des valeurs de la liste doit être le même que l'ordre des colonnes visées et cela, même si la liste des colonnes visées est ommise.

REMARQUE :
Du fait de sa syntaxe, l'ordre INSERT se décompose en trois ordres assez différents :

- Insertion simple explicite
- Insertion multiple explicite (à l'aide du constructeur de lignes valuées)
- Insertion à base de sous requête SELECT
- Insertion des valeurs par defaut

1.1. Insertion simple explicite

Cette syntaxe porte sur l'insertion d'une ligne unique au sein de la table. Il s'agit de préciser les valeurs à insérer explicitement.

```
INSERT INTO T_MODE_PAIEMENT (PMT_CODE, PMT_LIBELLE)
    VALUES          ('CB'   , 'Carte bancaire') ;
```

Cet exemple propose d'insérer dans la table T_MODE_PAIEMENT une ligne comportant les valeurs "CB" et "Carte bancaire" dans les colonnes respectives PMT_CODE et PMT_LIBELLE.

Compte tenu que cette table ne possède que deux colonnes, on aurait pu omettre de préciser les colonnes. Dans ce cas, la requête devient :

```
INSERT INTO T_MODE_PAIEMENT
    VALUES          ('CB'   , 'Carte bancaire') ;
```

Qui, bien évidemment donne le même résultat.

NOTA : lorsqu'une valeur n'est pas connue, il est possible de préciser le mot clef NULL (marqueur) qui laisse la colonne vide.

```
INSERT INTO T_MODE_PAIEMENT
    VALUES          ('X'    , NULL) ;
```

Qui insère un mode de paiement de code "X" et dont le libellé n'est pas renseigné.

1.2. Insertion multiple explicite à l'aide du constructeur de lignes valuées

Cette syntaxe porte sur l'insertion de multiples ligne au sein de la table cible. Il faut préciser les lignes de valeurs à insérer explicitement.

```
INSERT T_TITRE (TIT_CODE, TIT_LIBELLE)
    VALUES  ('M.'    , 'Monsieur',
             'Mlle.' , 'Mademoiselle'
             'Mme.'  , 'Madame') ;
```

Cet exemple propose d'insérer dans la table T_TITRE trois lignes de valeurs dans les colonnes TIT_CODE et TIT_LIBELLE.

De même que dans notre précédent exemple, cette table ne possèdant que deux colonnes, on aurait pu omettre de préciser les colonnes. Dans ce cas, la requête devient :

```
INSERT T_TITRE
    VALUES  ('M.'    , 'Monsieur',
             'Mlle.' , 'Mademoiselle'
             'Mme.'  , 'Madame') ;
```

Le constructeur de lignes valuées consiste à donner une liste de ligne en argument.

NOTA : le constructeur de lignes valuées est rarement implémenté dans les SGBDR. Oracle est l'un de seuls à accepter une telle syntaxe.

1.3. Insertion partiellement explicite avec le mot clef DEFAULT

Si la définition de la table possède une ou plusieurs valeurs par défaut, alors il est possible de les y insérer en utilisant le mot clef DEFAUT en lieu et place de la valeur.
Supposons que nous créons une table permettant de "pister" les connexions à la base de données, de la manière suivante :

```
CREATE TABLE T_SUIVI_CONNEXION
(CNX_USER       VARCHAR(128)  NOT NULL  DEFAULT 'Administrateur',
 CNX_DATE_HEURE  TIMESTAMP     NOT NULL  DEFAULT CURRENT_TIMESTAMP) ;
```

Alors nous pouvons insérer l'heure et la date par defaut sans en préciser la valeur, à l'aide de la requête :

```
INSERT INTO T_SUIVI_CONNEXION (CNX_USER, CNX_DATE_HEURE)
    VALUES              ('Dupont', DEFAULT) ;
```

Qui insère automatiquement la valeur par défaut dans la colonne CNX_DATE_HEURE.

Bien entendu on peut omettre la liste des noms de colonnes puisque, à nouveau, toutes les colonnes sont concernées par l'ordre d'insertion :

```
INSERT INTO T_SUIVI_CONNEXION
    VALUES              ('Dupont', DEFAULT) ;
```

Qui donne le même résultat si l'ordre est exécuté au même moment !

On peut aussi ne donner qu'une liste partielle des colonnes visées, les autres dotées d'une valeur par défaut seront automatiquement alimentées :

```
INSERT INTO T_SUIVI_CONNEXION (CNX_USER)
    VALUES              ('Dupont')
```

Qui donne encore le même résultat si l'ordre est exécuté au même moment !

1.4. Insertion totalement implicite avec l'expression DEFAULT VALUES

Si chacune des colonnes de la définition de la table possède des valeurs par défaut, on peut demander l'insertion de toutes les valeurs par défaut en utilisant l'expression clef DEFAULT VALUES. Dans ce cas il ne faut pas préciser les noms des colonnes :

```
INSERT INTO T_SUIVI_CONNEXION
    DEFAULT VALUES ;
```

Qui insérera l'utilisateur 'Administrateur' avec la date et l'heure courante.

1.5. Insertion multiple à base de sous requête SELECT

On peut insérer une ou plusieurs lignes dans une table en utilisant une sous requête de type SELECT. Dans ce cas les colonnes retournées par l'ordre SELECT doivent avoir les contraintes suivantes :

- être en nombre didentique aux colonnes précisées dans la liste ou en l'absence de précision de cette liste le même nombre de colonnes que la table
- avoir le même ordre que l'ordre des noms de colonnes de la liste ou bien le même ordre que les colonnes de la table si l'on omet cette liste
- avoir des types correspondant
- répondre à toutes les contraintes et dans le cas ou au moins une seule valeur viole une contrainte aucune ligne n'est insérée

Nous allons décrire différents cas et le comportement du SGBDR correspondant. Pour ce faire, je vous propose de créer de toutes pièces une nouvelle table de notre base de données exemple, la table des prospects et d'y insérer explicitement quelques données :

```
CREATE TABLE T_PROSPECT
(PRP_ID             INTEGER      NOT NULL,
 PRP_CODE_TITRE     CHAR(4)      NULL    ,
 PRP_NOM            CHAR(25)     NOT NULL,
 PRP_PRENOM         VARCHAR(16)  NULL    ,
 PRP_ENSEIGNE       VARCHAR(60)  NULL    ,
 PRP_DATE_SAISIE    TIMESTAMP    NOT NULL  DEFAULT CURRENT_DATETIME,
 CONSTRAINT PK_T_PROSPECT PRIMARY KEY (PRP_ID));
```

```
INSERT INTO T_PROSPECT (PRP_ID, PRP_CODE_TITRE, PRP_NOM, PRP_PRENOM, PRP_ENSEIGNE,
PRP_DATE_SAISIE)
    VALUES       (1,    'M.',       'Dupont','Alain',   NULL,       DEFAULT      );
INSERT INTO T_PROSPECT
    VALUES       (2,    'Mme.',     'Durand','Aline',   'SNCF',     '2000-12-25'  );
INSERT INTO T_PROSPECT (PRP_ID, PRP_CODE_TITRE, PRP_NOM )
    VALUES       (3,    'M.',       'Dubois') ;
```

La table T_PROSPECT contient donc :

```
PRP_ID  PRP_CODE_TITRE PRP_NOM    PRP_PRENOM   PRP_ENSEIGNE   PRP_DATE_SAISIE
------- -------------- ---------- ------------ -------------- -----------------------
1       M.             Dupont     Alain        NULL           2002-03-16 00:00:00.000
2       Mme.           Durand     Aline        SNCF           2000-12-25 00:00:00.000
3       M.             Dubois     NULL         NULL           2002-03-16 00:00:00.000
```

Un premier exemple, que nous allons décortiquer, va nous permettre de comprendre comment fonctionne l'insertion avec une sous-requête SELECT :

```
INSERT INTO   T_CLIENT (CLI_ID, TIT_CODE,       CLI_NOM, CLI_PRENOM)
      SELECT          PRP_ID, PRP_CODE_TITRE, PRP_NOM, PRP_PRENOM
      FROM   T_PROSPECT ;
```

Décomposons le travail du SGBD...

Première étape, exécution de la sous requête SELECT :

```
SELECT PRP_ID, PRP_CODE_TITRE, PRP_NOM,
PRP_PRENOM
FROM   T_PROSPECT ;
```

```
PRP_ID  PRP_CODE_TITRE PRP_NOM
PRP_PRENOM
------- -------------- ---------- ----------
1       M.             Dupont     Alain
2       Mme.             Durand     Aline
3       M.             Dubois     NULL
```

Seconde étape, Il faut maintenant insérer ces données dans les colonnes CLI_ID, TIT_CODE, CLI_NOM, CLI_PRENOM correspondantes de la table T_CLIENT. Celle-ci contient :

```
SELECT CLI_ID, TIT_CODE, CLI_NOM,
CLI_PRENOM
FROM T_CLIENT ;
```

```
CLI_ID  TIT_CODE CLI_NOM
CLI_PRENOM
------- -------- --------- ----------
1       M.       DUPONT    Alain
2       M.       MARTIN    Marc
3       M.       BOUVIER   Alain
4       M.       DUBOIS    Paul
5       M.       DREYFUS   Jean
6       M.       FAURE     Alain
7       M.       LACOMBE   Paul
8       Melle.   DUHAMEL   Evelyne
9       Mme.     BOYER     Martine
10      M.       MARTIN    Martin
...
```

L'exécution de la requête donne :

```
Serveur: Msg 2627, Niveau 14, État 1, Ligne 1
Violation de la contrainte PRIMARY KEY 'PK_T_CLIENT'.
Impossible d'insérer une clé en double dans l'objet 'T_CLIENT'.
L'instruction a été arrêtée.
```

En effet, on ne peut insérer des clefs en double, or nous tentons d'insérer un deuxième client portant la clef 1, un autre portant le clef 2, etc...
En l'occurence aucune ligne n'est donc insérée car toute requête de mise à jour est une transaction et fonctionne en "tout ou rien", c'est à dire qu'aucune ligne n'est insérée si au moins une contrainte n'est pas vérifié. En l'occurence la violation de la contrainte de clef primaire étant ici évidente rien n'est inséré dans la table T_CLIENT.

Nous savons, à la lecture des données de la table des clients, que la clef la plus haute possède la valeur 100. Il est donc possible d'insérer des clients dont la clef est supérieure à cette valeur. D'ou l'idée de rajouter cette valeur à la valeur de clef de la table T_PROSPECT récupérée à l'aide de la requête SELECT, par une simple addition :

Et là nos trois prospects ont été insérés :

```
CLI_ID  TIT_CODE CLI_NOM  CLI_PRENOM  CLI_ENSEIGNE
------- -------- -------- ----------- ------------
101     M.       Dupont   Alain       NULL
102     Mme.     Durand   Aline       NULL
103     M.       Dubois   NULL        NULL
```

Mais comme nous sommes déjà en train de traiter une sous requête SELECT, il semble facile de remplacer la valeur 100 par une autre sous requête renvoyant la valeur maximale de la clef de la table T_CLIENT afin de l'ajouter à la valeur de la clef de la table T_PROSPECT :

```
INSERT INTO   T_CLIENT (CLI_ID, TIT_CODE, CLI_NOM, CLI_PRENOM)
      (SELECT PRP_ID + (SELECT MAX(CLI_ID)
      FROM T_CLIENT), PRP_CODE_TITRE, PRP_NOM, PRP_PRENOM
      FROM   T_PROSPECT
```

Ce qui s'exécute parfaitement et donne :

```
CLI_ID  TIT_CODE CLI_NOM  CLI_PRENOM  CLI_ENSEIGNE
------- -------- -------- ----------- ------------
104     M.       Dupont   Alain       NULL
105     Mme.     Durand   Aline       NULL
106     M.       Dubois   NULL        NULL
```

L'explication est simple : la sous requête (SELECT MAX...) n'est exécutée qu'une seule fois puisque qu'elle n'est pas corrélée avec la table cible de l'insertion.

1.7. Insertion en auto référence

L'insertion en auto référence consiste à ajouter à une table une ou plusieurs nouvelles lignes calculées d'après les lignes existantes de la table cible.

Par exemple nous voulons insérer dans la tables des tarifs une nouvelle ligne avec comme date d'application du tarif, le premier janvier 2002, le même taux de taxe que le tarif précédent et un prix de petit déjeuner de 10% de plus que le précédent tarif.

Obtenir le précédent tarif consiste à trouver la ligne de la table dont la valeur de la date est maximale. Cela peut s'effectuer à l'aide de la requête suivante :

```
SELECT *
FROM   T_TARIF
WHERE  TRF_DATE_DEBUT = (SELECT MAX(TRF_DATE_DEBUT)
                  FROM T_TARIF)

TRF_DATE_DEBUT   TRF_TAUX_TAXES    TRF_PETIT_DEJEUNE
--------------------------  --------------------------  ------------------------------
2002-01-01                        20.6000                 69.5750
```

Dès lors on peut utiliser cet ordre SELECT en le modifiant un peu de manière à lui faire insérer la nouvelle ligne tarifaire :

```
INSERT INTO T_TARIF (TRF_DATE_DEBUT, TRF_TAUX_TAXES, TRF_PETIT_DEJEUNE)
     SELECT       '2002-01-01',   TRF_TAUX_TAXES, TRF_PETIT_DEJEUNE * 1.1
     FROM   T_TARIF
     WHERE  TRF_DATE_DEBUT = (SELECT MAX(TRF_DATE_DEBUT)
                       FROM T_TARIF)
```

REMARQUE :
Voici les principaux cas pour lesquels un ordre d'insertion ne peut aboutir :

- violation de clef (index primaire)
- violation de contrainte d'index secondaire unique
- violation de contrainte de données (colonne not null)
- violation d'intégrité référentielle
- violation de contrainte de contrôle de validité (min, max, étendue, domaine...)

2. Suppression à l'aide de DELETE

La syntaxe de base de l'ordre SQL de suppression de données dans une table est la suivante :

DELETE [FROM] *nom_table_cible*
[WHERE *condition*]

Le seul cas pour lequel cet ordre peut ne pas aboutir est lorsque la suppression viole la contrainte d'intégrité référentielle. Il est en effet absurde de vouloir supprimer un client si les factures relatives à ce client n'ont pas été préalablement supprimées.

NOTA : Dans certains cas, il se peut que la suppression d'une ligne entraîne la suppressions d'autres lignes dans d'autres tables lorsqu'existe des intégritées référentielles de suppression en cascade.

2.1. Suppression de toutes les lignes d'une table

C'est la forme la plus simple de l'ordre DELETE puisqu'il suffit d'omettre la clause WHERE :

```
DELETE FROM T_PROSPECT
```

Supprime tous les prospects.

2.2. Suppression conditionnelle

Il suffit de rajouter la clause WHERE dotée d'un prédicat.

```
DELETE FROM T_PROSPECT
WHERE  PRP_PRENOM LIK'%d'
```

Supprime tous les prospects dont le nom se termine par la lettre 'd'.

2.3. Suppression avec sous requête conditionnelle

Il est possible d'utiliser une sous requête conditionnelle dans la clause WHERE d'un ordre DELETE.

Supprimons les prospects dont le couple de valeurs nom/prénom se trouve dans la table des clients. Procédons pour cela par étape. Pour obtenir la liste des prospects qui figurent en tant que client, nous pouvons faire la requête suivante :

```
SELECT PRP_ID, PRP_NOM, PRP_PRENOM
FROM   T_PROSPECT P
     JOIN T_CLIENT C
        ON C.CLI_NOM = P.PRP_NOM
          AND C.CLI_PRENOM = P.PRP_PRENOM
```

```
PRP_ID      PRP_NOM        PRP_PRENOM
----------- -------------- ----------
1           Dupont         Alain
```

Dès lors il suffit de supprimer les prospects dont l'identifiant est récupéré par la sous requête :

```
DELETE FROM T_PROSPECT
WHERE  PRP_ID = (SELECT PRP_ID
            FROM   T_PROSPECT P
                 JOIN T_CLIENT C
                    ON C.CLI_NOM = P.PRP_NOM
                      AND C.CLI_PRENOM = P.PRP_PRENOM);
```

On peut procéder aussi à l'aide du constructeur de lignes valuées, si votre SGBDR le supporte, ce qui simplifie l'écriture de la requête :

```
DELETE FROM T_PROSPECT
WHERE  (CLI_NOM, CLI_PRENOM) = (SELECT PRP_NOM, PRP_PRENOM
                        FROM   T_CLIENT) ;
```

3. Modification à l'aide d'UPDATE

La syntaxe de base de l'ordre SQL de modification de données dans une table est la suivante :

UPDATE *nom_table_cible*
SET *colonne = valeur* [, *colonne2 = valeur2 ...*]
[WHERE *condition*]

3.1. Mise à jour d'une colonne unique sans condition

C'est la forme la plus simple de l'ordre UPDATE. Nous voulons par exemple fixer à 55 F les tarifs de nos petits déjeuners dans la table T_TARIF :

```
UPDATE T_TARIF
SET    TRF_PETIT_DEJEUNE = 55 ;
```

3.2. Mise à jour d'une colonne unique avec reprise de valeur (auto référence)

On peut aussi reprendre la valeur de la colonne (ou d'une autre colonne de la table cible). Par exemple nous pouvons demander une augmentation de 15% des tarifs des petits déjeuners :

```
UPDATE T_TARIF
SET    TRF_PETIT_DEJEUNE = TRF_PETIT_DEJEUNE * 1.15 ;
```

3.3. Mise à jour d'une colonne unique avec filtrage

On peut ajouter une clause de filtrage WHERE dans une requête de mise à jour. Par exemple nous pouvons décider de n'augementer de 15% que les tarifs des petits déjeuners des périodes postérieures à 1999.

```
UPDATE T_TARIF
SET    TRF_PETIT_DEJEUNE = TRF_PETIT_DEJEUNE * 1.15
WHERE  EXTRACT(YEAR FROM TRF_DATE_DEBUT) > 1999;
```

3.4. Mise à jour de plusieurs colonnes simultanément

Pour mettre à jour simultanément plusieurs colonnes, il suffit de répéter aitant de fois que nécessaire le contenu de la clause SET, à raison d'un couple colonne/valeur par colonne visées par la mise à jour.

```
UPDATE T_CLIENT
SET    CLI_NOM      = UPPER(CLI_NOM),
       CLI_PRENOM   = UPPER(CLI_PRENOM)
       CLI_ENSEIGNE = UPPER(CLI_ENSEIGNE) ;
```

NOTA : dans ce cas, la nullité de l'exécution de modification d'une valeur dans une colonne possédant le marqueur NULL, n'entraîne pas la nullité de l'exécution des mises à jour des autres colonnes, chaque modification de colonne étant évaluées séparément.

3.5. Mise à jour avec sous requête

Comme dans les ordres INSERT et DELETE, il est possible d'utiliser une sous requête dans la clause WHERE de l'ordre UPDATE afin de filtrer de manière plus complète.

Par exemple, afin d'éviter de confondre des prospects qui ont le même nom et prénom que certains clients, on désire ajouter le mot "bis" aux prospects homonymes :

```
UPDATE T_PROSPECT
SET    PRP_NOM = TRIM(RIGHT, PRP_NOM) || ' bis'
WHERE  PRP_ID = (SELECT  PRP_ID
                 FROM   T_PROSPECT P
                      JOIN T_CLIENT C
                         ON C.CLI_NOM = P.PRP_NOM
                           AND C.CLI_PRENOM = P.PRP_PRENOM);
```

3.6. Mise à jour de valeurs particulières (défaut et marqueur NULL)

Il est possible de mettre à jour une colonne à sa valeur par défaut si elle possède une telle spécificité élaborée dans la création de la table :

En reprenant la définition des tables de connexion vu au paragraphe 1.3, donnons à la colonne CNX_USER sa valeur par défaut pour toutes les lignes de la table :

```
UPDATE  T_SUIVI_CONNEXION
SET     CNX_USER = DEFAULT ;
```

Il est aussi possible de supprimer le contenu d'une colonne (ou de plusieurs) en y plaçant le marqueur NULL :

```
UPDATE  T_CLIENT
SET     CLI_ENSEIGNE = NULL ;
```

Cette requête vide la colonne CLI_ENSEIGNE de la table des clients en y plaçant NULL.

NOTA : c'est le seul cas ou l'on trouvera le mot clef NULL associé au signe égal, car dans ce cas le signe egal est un opérateur d'affectation. De manière syntaxique il aurait mieux valut une construction du genre :

```
UPDATE  T_CLIENT
SET     CLI_ENSEIGNE AS NULL ;
```

ATTENTION :
Une mise à jour peut échouer si elle viole les contraintes.
Voici les principaux cas pour lesquels un ordre de modification ne peut aboutir :

- violation de clef (index primaire)
- violation de contrainte d'index secondaire unique
- violation de contrainte de données (colonne not null)
- violation d'intégrité référentielle
- violation de contrainte de contrôle de validité (min, max, étendue, domaine...)