LIF4 - Initiation aux Bases de données : SQL - 3

E.Coquery

emmanuel.coquery@liris.cnrs.fr

http://liris.cnrs.fr/~ecoquery

Modification des données stockées dans une base

La modification s'effectue par ajout, suppression ou modification de n-uplets (lignes) dans l'instance de la base.

- SQL sert ici de langage de manipulation de données.
- Trois instructions SQL permettent ces modifications :
  - INSERT
  - DELETE
  - UPDATE
- Ces instructions de mise à jour peuvent utiliser des (morceaux de) requête afin d'effectuer des calculs pour sélectionner et/ou générer des données.

Insertion

Instruction INSERT

- INSERT INTO *nom_table*(att_1, ..., att_n) VALUES(val_1, ..., val_n)
- Ajoute le n-uplet (val_1, ..., val_n) à la relation *nom_table*.
- val_i correspond à l'attribut att_i.
- Si un attribut de la relation *nom_table* n'apparaît pas dans $att_1, \ldots, att_n$, alors la valeur du n-uplet pour cet attribut est NULL.
- La spécification des attributs att_1, ..., att_n est *optionnelle*
- Si on précise pas les attributs, il faut donner une valeur à tous les attributs.
  - L'ordre sur des valeurs (val_1, ..., val_n) est celui des attributs dans la définition de la relation *nom_table*.
  - C'est un des cas où cet ordre est important.

Exemple

Schéma : Batiment(Num_bat, Nom_bat, Ent_princ, Ent_sec)

Num_bat	Nom_bat	Ent_princ	Ent_sec
1	Turing	Nord	Ouest
2	Einstein	Ouest	NULL
3	Newton	Sud	Nord
4	Pointcarre	Est	NULL
5	Curie	Nord	NULL
6	Bohr	Sud	Est

```
INSERT INTO Batiment(Nom_bat,Num_bat,Ent_princ)
VALUES ('Curie',5,'Nord');

INSERT INTO Batiment VALUES (6,'Bohr','Sud','Est');
```


Insertion utilisant une requête

INSERT INTO *nom_table*($att_1, \ldots, att_n$)
SELECT $e_1, \ldots, e_n$
FROM ...

- Insertion dans *nom_table* des n-uplets calculés par la requête SELECT ... FROM ...
- La requête ne peut pas contenir de ORDER BY
  - De toute façon, c'est le SGBD qui détermine l'ordre dans lequel les n-uplets sont stockés.
- Le nom des colonnes dans le résultat de la requête n'est pas important : c'est l'ordre des expressions qui compte.

Exemple

Schéma :
Departement(Num_dept, Nom_dept, Num_bat, Num_chef)
Batiment(Num_bat, Nom_bat, Ent_princ, Ent_sec)
Dept_important(Nom,Bat)

Ajouter à la table Dept_important les départements qui sont dans des batiments ayant une entrée secondaire :

```
INSERT INTO Dept_important(Bat,Nom)
SELECT Nom_bat, Nom_dept
FROM Batiment, Departement
WHERE Departement.Num_bat = Batiment.Num_bat
AND Ent_sec IS NOT NULL;
```

Supression

DELETE FROM *nom_table*
WHERE *condition*

- Supprime les n-uplets de la relation *nom_table* qui vérifient *condition*.
- *condition* peut être aussi complexe qu'une condition exprimée dans le WHERE d'un SELECT.
  - En particulier, *condition* peut contenir des requêtes imbriquées.
  - Les requêtes imbriquées ne peuvent pas faire référence à *nom_table*, car elle est en cours de modification.
- WHERE *condition* est optionnel.
  - Si le WHERE est omis, tous les n-uplets sont supprimés (cela revient à utiliser a condition TRUE).

Exemple

Num_bat	Nom_bat	Ent_princ	Ent_sec
1	Turing	Nord	Ouest
2	Einstein	Ouest	NULL
3	Newton	Sud	Nord
4	Pointcarre	Est	NULL
5	Curie	Nord	NULL
6	Bohr	Sud	Est

Supprimer le bâtiment numéro 5 :

```
DELETE FROM Batiment
WHERE Num_bat = 5 ;
```


Exemple - 2

Schéma :
Batiment(Num_bat, Nom_bat, Ent_princ, Ent_sec)
Departement(Num_dept, Nom_dept, Num_bat, Num_chef)

Supprimer les bâtiments qui ne correspondent à aucun département :

```
DELETE FROM Batiment
WHERE Num_bat NOT IN
   (SELECT Departement.Num_bat
    FROM Departement) ;
```


Modification de n-uplets

UPDATE *nom_table*
SET $att_1 = e_1$,
$att_2 = e_2$,
...
WHERE *condition*

- *condition* indique les lignes à modifier.
- att_i prend la valeur calculée par l'expression e_i.
- e_i peut utiliser $att_1, att_2, \ldots$, y compris att_i.
  - Ce sont les anciennes valeurs de $att_1, att_2, \ldots$ qui seront utilisées pour le calcul.
- Les e_i peuvent être des requêtes à condition qu'elles renvoient un unique résultat et que *nom_table* n'apparaisse pas dans un FROM.
- Similairement au DELETE, le WHERE est optionnel.
  - Si le WHERE est omis, tous les n-uplets sont modifiés.

Exemple

Schéma : Batiment(Num_bat, Nom_bat, Ent_princ, Ent_sec)

Changer le nom du bâtiment numéro 3 en 'Copernic' :

```
UPDATE Batiment SET Nom_bat = 'Copernic' ;
```

Num_bat	Nom_bat	Ent_princ	Ent_sec
1	Turing	Nord	Ouest
2	Einstein	Ouest	NULL
3	Copernic	Sud	Nord

Exemple - 2

Schéma : Employe(Nom, Num, Fonction, Salaire, Num_Dept)

Augmenter de 10% le salaire des ingénieurs :

```
UPDATE Employe
SET Salaire = Salaire * 1.1
WHERE Fonction = 'ingenieur' ;
```

Exemple - 3

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche,
Salaire, Commission, Num_Dept)
Departement(Num_dept, Nom_dept, Num_bat, Num_chef)

Pour chaque département dont le chef n'est pas connu, spécifier que son chef est le plus ancien employé de ce département occupant la fonction de directeur.

Exemple - 3 - suite

```
UPDATE Departement
SET Num_chef =
   (SELECT Num
    FROM Employe
    WHERE Fonction = 'directeur'
    AND Employe.Num_dept = Departement.Num_dept
    AND Embauche <= ALL
          (SELECT Embauche
           FROM Employe E
           WHERE Fonction = 'directeur'
           AND E.Num_dept = Departement.Num_dept))
WHERE Num_chef IS NULL ;
```


Transactions

- Une transaction est un ensemble de modifications de la base qui forme un tout indivisible.
- Ces modifications doivent être effectuée entièrement ou pas du tout, sous peine de laisser la base dans un état incohérent.
- Au cours d'une transaction, seul l'utilisateur ayant démarré cette transaction voit les modifications effectuées.

Transactions en SQL

Gestion des transactions en SQL :

- COMMIT ;
  - Valide les modifications effectuées.
  - Les modifications sont alors définitives et visibles par tous.
- ROLLBACK ;
  - Annule les modifications effectuées depuis le début de la transaction.

Différences de traitement des transactions entre SGBDs

Dans Oracle :

- Une nouvelle transaction est implicitement démarrée au début de la connection et après chaque COMMIT.
- Le système assure la cohérence des données en cas de mise à jour simultanée par deux utilisateurs, en utilisant un système de verrouillage automatique.

Dans MySQL :

- Les tables doivent être stockées en utilisant le moteur de stockage InnoDB ou BDB pour que les transactions soient gérées.
- Par défaut, chaque mise à jour est immédiatement validée (COMMIT automatique).
- Pour démarrer explicitement une transaction, on utilise l'instruction BEGIN ;
  - Dans ce cas le COMMIT automatique est désactivé.

Gestion du schéma d'une base

SQL est également un langage de définition de données :

- Permet de créer ou supprimer des tables.
- Permet de modifier la structure d'une table.
- Permet de spécifier certaines contraintes d'intégrité sur le schéma.

DESC *nom_table* ;

Permet d'obtenir des informations sur le schéma d'une table.

- Les attributs et leur type.
- Des informations sur certaines contraintes d'intégrité.

Création de table

Lors de la création d'une table on indique :

- Le nom des attributs.
- Le type de chaque attribut.

De manière optionnelle :

- Certaines contraintes d'intégrité.
- Des caractéristiques de stockage.
- Des données provenant d'une requête.

Création simple

CREATE TABLE *nom_table*(att_1 $type_1$, att_2 $type_2$, ...) ;

- Crée une table *nom_table* ;
- ayant pour attributs $att_1, att_2, \ldots$;
- att_i ayant le type $type_i$.

Exemple :

```
CREATE TABLE Departement
        (Num_dept integer, Nom_dept varchar(30),
         Num_bat integer, Num_chef integer) ;
```


Création avec insertion de données

CREATE TABLE *nom_table*(att_1 $type_1$, att_2 $type_2$, ...)
AS SELECT ... ;

- Crée la table comme précédemment
- Ajoute les données à la table comme si on avait exécuté :
  INSERT INTO *nom_table* SELECT ... ;
- La spécification des attributs est optionnelle. Si les attributs sont omis :
  - Le nom des attributs est donné par le SELECT.
    - Implique un renommage obligatoire des expressions du SELECT.
  - Le type des attributs est déduit à partir du SELECT.
    - On peut utiliser les fonctions de conversion de type dans le SELECT.
- Pas de ORDER BY dans le SELECT.

Exemple

Créer une table dans laquelle on indique pour chaque département son nom et le numéro de son chef, ce dernier étant l'employé du département ayant le salaire le plus élevé.

```
CREATE TABLE Chef_dept
AS
SELECT Nom_dept, Num Chef
FROM Employe, Departement
WHERE Employe.Num_dept = Departement.Num_dept
AND Employe.Salaire >=
    (SELECT MAX(Salaire)
     FROM Employe E
     WHERE E.Num_dept = Departement.Num_dept) ;
```


Vues

Une vue est une requête à laquelle on donne un nom.

- Utilisable comme une table dans un SELECT.
- La vue est recalculée à chaque utilisation.
- Pas d'opération de mise à jour directement sur une vue.

Création :

```
CREATE VIEW nom_vue
AS
SELECT ...
```


Quelques types SQL Numériques

Type DECIMAL(*precision*,*echelle*)

- Représente un nombre codé sur *precision* chiffres, avec *echelle* chiffres après la virgule.
- *echelle* est optionnel et vaut 0 par défaut.
- *precision* est optionnel si *echelle* n'est pas indiqué.
  - Oracle → valeur par défaut : 38
  - MySQL → valeur par défaut : 10

Type FLOAT(*precision*)

- Représente un nombre à virgule flottante.
- *precision* est optionnel.
  - Oracle → *precision* en binaire, par défaut : 126 (soit 38 en décimal)
  - MySQL → *precision* en décimal, par défaut : 10

Les types INTEGER, INT, DOUBLE, ... sont des raccourcis pour des formes particulières de DECIMAL ou FLOAT

Quelques types SQL sur les caractères

Type CHAR(*longueur*)

- Chaîne de caractères de taille **fixe** *longueur*.

Type VARCHAR(*longueur*)

- Chaîne de caractères de taille **variable** *longueur*.

Objets de grande taille

Oracle

- Types BLOB et CLOB : jusqu'à 8 To de données binaires (BLOB) ou de caractères (CLOB).

MySQL

- Types BLOB et TEXT : jusqu'à 64 Ko de données binaires ou de caractères.
- Types MEDIUMBLOB et MEDIUMTEXT : jusqu'à 16 Mo.
- Types LONGBLOB et LONGTEXT : jusqu'à 4 Go.

Dates

Oracle

- Type DATE : date, y compris l'heure à la seconde près.
- Type TIMESTAMP : plus précis.

MySQL

- Type DATE : date au jour près.
- Type DATETIME : date + heure à la seconde près
- Type TIMESTAMP : nombre de secondes écoulées depuis le 1er janvier 1970, affichage similaire à DATETIME
- Type TIME : un nombre d'heures:minutes:secondes

Types énumérés

MySQL :

- Type ENUM('val_1', 'val_2', ...) :
  - Les valeurs autorisées pour l'attribut sont val_1, val_2, ...

Oracle :

- Type VARCHAR(n) CHECK (att IN ('val_1', 'val_2', ...)) :
  - att est l'attribut dont on définit le type.
  - Les valeurs autorisées pour l'attribut sont val_1, val_2, ...
  - n doit être supérieur à la plus grande longueur de valeur val_i.

Références

Oracle :
`http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14220/datatype.htm`

MySQL :
`http://dev.mysql.com/doc/refman/5.0/fr/column-types.html`

Contrainte NOT NULL

- Il est possible d'ajouter NULL ou NOT NULL après un type dans une définition de table pour indiquer si la valeur NULL est acceptée pour l'attribut.
- Par défaut, la valeur NULL est acceptée.

Exemple :

```
CREATE TABLE Bureau(Num_emp INTEGER,
                    Num_bat INTEGER NOT NULL,
                    Emplacement VARCHAR(20) NOT NULL);
```

Crée une table Bureau avec un attribut Nom_emp entier pouvant être NULL, Num_bat contenant un entier qui ne peut pas être NULL et enfin Emplacement contenant une chaîne de caractère et qui ne peut pas être NULL.

Contraintes d'intégrité en général

CREATE TABLE *nom_table* (
att$_1$ *type*$_1$, *att*$_2$ *type*$_2$, ...,
CONSTRAINT *nom*$_1$ *contrainte*$_1$,
CONSTRAINT *nom*$_2$ *contrainte*$_2$, ...
);

- CONSTRAINT *nom*$_i$ permet de nommer une contrainte.
- Le nom est optionnel.

Contraintes UNIQUE et PRIMARY KEY

..., CONSTRAINT *nom*$_c$ UNIQUE (*att*$_i$, *att*$_j$, ...), ...

- Impose que chaque n-uplet aie une combinaison de valeurs différente pour les attributs *att*$_i$, *att*$_j$, ...
  - Il est par contre possible d'avoir deux fois la même valeur pour un attribut *att*$_i$
  - Si une des valeurs pour *att*$_i$, *att*$_j$, ... est NULL, la contrainte ne s'applique pas sur le n-uplet concerné.

..., CONSTRAINT *nom*$_c$ PRIMARY KEY (*att*$_i$, *att*$_j$, ...), ...

- Indique que l'ensemble d'attribut (*att*$_i$, *att*$_j$, ...) sert d'identifiant principal (également appelé clé primaire) pour les n-uplets de la table.
- Implique NOT NULL sur chacun des (*att*$_i$, *att*$_j$, ...) et UNIQUE(*att*$_i$, *att*$_j$, ...)
- Il y a au maximum une contrainte PRIMARY KEY par table.

Clés étrangères

..., CONSTRAINT *nom*$_c$ FOREIGN KEY (*att*$_1$, ..., *att*$_k$)
REFERENCES *table_cible*(*att*$'_1$,...,*att*$'_k$), ...

Une clé étrangère est une *référence* vers la clé primaire d'une table.

- Tout comme les clé primaires, elles peuvent être constituées de plusieurs attributs.
- Les valeurs pour (*att*$_1$, ..., *att*$_k$) doivent correspondre aux valeurs d'un des n-uplets de *table_cible* pour les attributs (*att*$'_1$,...,*att*$'_k$) ;

Rmq : dans MySQL, seul le moteur de stockage InnoDB gère correctement les clés étrangères.

Contrainte CHECK

..., CONSTRAINT *nom*$_c$ CHECK (*condition*), ...

- *condition* doit être vérifiée par chaque n-uplet stocké dans la table.
- La forme (*att* IN ('*val*$_1$', '*val*$_2$', ...)) utilisée pour les types énumérés est un cas particulier de cette contrainte.

!! Contrainte non vérifiée dans MySQL

MySQL : moteurs de tables

Dans MySQL, il existe plusieurs moteurs de stockage pour les tables, parmi lesquels :

- MyISAM : le moteur de stockage par défaut.
- InnoDB : permet de gérer les clés étrangères et les transactions.

CREATE TABLE *nom_table* (...) ENGINE = InnoDB ;

- Permet de créer une table utilisant le moteur InnoDB.

ALTER TABLE *nom_table* ENGINE = InnoDB ;

- Permet de changer le moteur de stockage d'une table en InnoDB.

Suppression et renommage de tables

DROP TABLE *nom_table* ;

- Supprime la table *nom_table*.
- Il ne faut pas qu'une clé étrangère d'une autre table référence la table à supprimer.
- En Oracle, on peut ajouter à la fin le mot clé CASCADE pour déclencher la suppression des clés étrangères qui référencent la table à supprimer.

RENAME *ancien_nom* TO *nouveau_nom* ;

- Renomme la table *ancien_nom* en *nouveau_nom*.

Ajout, modification ou suppression d'attribut

ALTER TABLE *nom_table* ADD *att type* NOT NULL ;

- Ajoute à la table *nom_table* un attribut *att* contenant des données correspondant à *type*.
- On peut optionnellement spécifier NOT NULL lorsque l'on souhaite interdire la valeur NULL.

ALTER TABLE *nom_table* MODIFY *att nouveau_type* NOT NULL ;

- Change le type de l'attribut *att*, en spécifiant optionnellement NOT NULL.

ALTER TABLE *nom_table* RENAME COLUMN *att* TO *nouvel_att* ;

- Change le nom de *att* en *nouvel_att*.

ALTER TABLE *nom_table* DROP COLUMN *att* ;

- Supprime l'attribut *att* de la table *nom_table*.

Ajouter ou supprimer une contrainte d'intégrité

ALTER TABLE *nom_table* ADD CONSTRAINT nom_c *contrainte* ;

- Ajoute la contrainte *contrainte* sur la table *nom_table*.
- CONSTRAINT nom_c spécifie le nom optionnel de la contrainte.

ALTER TABLE *nom_table* DROP PRIMARY KEY ;

- Supprime la clé primaire.

ALTER TABLE *nom_table* DROP FOREIGN KEY *nom_cle* ;

- Supprime la clé étrangère nommée *nom_cle*.

Exemple : Création du schéma Entreprise - 1

```
CREATE TABLE Employe (
        Nom VARCHAR(30) NOT NULL,
        Num INTEGER,
        Fonction VARCHAR(30) NOT NULL,
        Num_sup INTEGER,
        Embauche DATE NOT NULL,
        Salaire FLOAT NOT NULL,
        Num_dept INTEGER NOT NULL,
        Commission FLOAT,
        PRIMARY KEY (Num)
) ENGINE=InnoDB ;
```


Exemple : Schéma Entreprise - 2

```
CREATE TABLE Batiment (
        Num_bat INTEGER NOT NULL,
        Nom_bat VARCHAR(30) NOT NULL,
        Ent_princ VARCHAR(10) NOT NULL,
        Ent_sec VARCHAR(10),
        PRIMARY KEY (Num_bat)
) ENGINE=InnoDB ;
```


Exemple : Schéma Entreprise - 3

```
CREATE TABLE Departement (
        Num_dept INTEGER NOT NULL,
        Nom_dept VARCHAR(30) NOT NULL,
        Num_bat INTEGER,
        Num_chef INTEGER,
        PRIMARY KEY (Num_dept)
) ENGINE=InnoDB ;
```


Exemple : Schéma Entreprise - 4

```
ALTER TABLE Employe
ADD CONSTRAINT fk_emp_dept
FOREIGN KEY (Num_dept)
REFERENCES Departement(Num_dept) ;
```

```
ALTER TABLE Employe
ADD CONSTRAINT fk_emp_sup
FOREIGN KEY (Num_sup)
REFERENCES Employe(Num) ;
```

Exemple : Schéma Entreprise - 5

```
ALTER TABLE Departement
ADD CONSTRAINT fk_dept_bat
FOREIGN KEY (Num_bat)
REFERENCES Batiment(Num_bat);
```

```
ALTER TABLE Departement
ADD CONSTRAINT fk_dept_chef
FOREIGN KEY (Num_chef)
REFERENCES Employe(Num);
```


Exemple : Remplissage des tables - 1

Des clés étrangères ont été définies :

- Les insertions ne peuvent pas se faire dans n'importe quel ordre.

Comme Batiment ne possède pas de clé étrangère on peut la remplir sans problème :

```
INSERT INTO Batiment VALUES (1,'Turing','Nord','Ouest');
INSERT INTO Batiment VALUES (2,'Einstein','Ouest',NULL);
...
```


Exemple : Remplissage des tables - 2

Si on ne spécifie pas les chefs (*i.e.* valeur NULL), on peut à présent remplir la table Departement :

```
INSERT INTO departement VALUES (10,'Marketing',1,NULL);
INSERT INTO departement VALUES (20,'Developpement',2,NULL);
INSERT INTO departement VALUES (30,'Direction',3,NULL);
```

Bien sûr, il faudra mettre à jour la table une fois les employés saisis.

Exemple : Remplissage des tables - 3

Il faut à présent saisir les employés en respectant l'ordre hiérarchique afin de ne pas violer la clé étrangère fk_emp_sup.

```
INSERT INTO Employe VALUES
('Martin',16712,'directeur',NULL,'1990-05-23',40000,30,NULL);
INSERT INTO Employe VALUES
('Julius',12569,'directeur',16712,'2001-02-25',32000,20,NULL);
INSERT INTO Employe VALUES
('Lambert',25012,'directeur',16712,'1998-09-20',30000,20,NULL);
INSERT INTO Employe VALUES
('Bellot',13021,'ingenieur',25012,'1996-05-18',25000,20,NULL);
INSERT INTO Employe VALUES
('Soule',28963,'directeur',16712,'1996-10-21',25000,10,10000);
...
```


Exemple : Remplissage des tables - 4

Enfin, on désigne le chef de chaque département comme étant le directeur gagnant le plus pros salaire :

```
UPDATE Departement
SET Num_chef =
    (SELECT Num
     FROM Employe
     WHERE Fonction = 'directeur'
         AND Employe.Num_dept = Departement.Num_dept
         AND Salaire >=
             (SELECT MAX(Salaire)
              FROM Employe e
              WHERE e.Num_dept = Departement.Num_dept
                  AND Fonction = 'directeur')
);
```