DICJ

Département d'informatique

Les bases de données DML – Data Manipulation Language

420-JQA-JQ

Insertion et requête de base

DML Data Manipulation Langage

- Le langage DML de SQL permet de consulter le contenu des tables et de les modifier. Il comporte 4 verbes:
  - **SELECT** extrait des données des tables
  - **INSERT** insère de nouvelles lignes dans une table
  - **DELETE** supprime des lignes d'une table
  - **UPDATE** modifie les valeurs de colonnes de lignes existantes

Ajout de données

- La commande permettant d'ajouter une ligne dans une table est **INSERT.**

```
INSERT INTO tblNomTable(liste des champs)
        VALUES(liste des valeurs);
```

```
INSERT INTO tblPersonne(nom, prenom, age)
        VALUES('Gagnon', 'Jean', 22);
```

- La liste des champs contient les champs de la table pour lesquels on souhaite fournir des valeurs.
- Les caractères et les dates doivent être entre apostrophes mais pas les valeurs numériques.

Deux formats

- On effectue une commande **INSERT** complète pour chaque ligne à ajouter.

```
INSERT INTO tblPersonne(nom, prenom, age)
        VALUES('Gagnon', 'Jean', 22);
INSERT INTO tblPersonne(nom, prenom, age)
        VALUES('Perron', 'Julie', 25);
```

- On peut utiliser la version abrégée en séparant les groupes de valeurs par des virgules

```
INSERT INTO tblPersonne(nom, prenom, age) VALUES
    ('Gagnon', 'Jean', 22),
    ('Perron', 'Julie', 25);
```

Règles pour l'ajout

- Si certains champs sont omis dans la liste des champs, alors les valeurs par défaut définies dans le script de création seront utilisées (DEFAULT).
- S'il n'y a pas de valeurs par défaut et que NULL est permis, la donnée contiendra NULL.
- S'il n'y a pas de valeurs par défaut et que le champ est NOT NULL, la valeur suivra les règles suivantes:

Valeur	Type donnée
0	pour un nombre
' '	pour une chaîne
0000-00-00	pour une date
00:00:00	pour une heure

Chaînes de caractères et dates

- Lorsqu'un champ contient une chaîne de caractères on doit utiliser des apostrophes.
- Si cette chaîne contient elle-même un apostrophe, on le double.

```
INSERT INTO tblCateg(CodeCateg, DescCateg, dateInscr)
VALUES('Da', 'Demande d''admission', '2019-03-06');
```

- **Attention**: il s'agit bien de 2 apostrophes de suite et non de guillemets.
- Ou le mettre entre guillemets

```
INSERT INTO tblCateg(CodeCateg, DescCateg, dateInscr)
VALUES('Da', "Demande d'admission", "2019-03-06");
```

Ajout sans noms de colonnes

- Il existe une syntaxe plus courte mais avec laquelle des valeurs doivent être fournies pour tous les champs de la table.

INSERT INTO tblNomTable **VALUES**(*valeurs dans l'ordre des champs*);

- Soit la définition de table suivante:

```
CREATE TABLE tblBallon
(       idBallon        TINYINT         NOT NULL,
        taille          TINYINT         NOT NULL,
        couleur         VARCHAR(20)     NULL,
        PRIMARY KEY(idBallon)
);
```

```
INSERT INTO tblBallon VALUES(1, 20, 'rouge');          bon
INSERT INTO tblBallon VALUES(1, 'rouge', 20);          mauvais
INSERT INTO tblBallon VALUES('rouge' );                mauvais
INSERT INTO tblBallon VALUES(1, 20, NULL);             bon
```

Ajout avec une clé auto incrémentée

- Lorsque la clé primaire est auto incrémentée, deux méthodes peuvent être utilisées pour les chargement initial de la base de données:
  - On peut laisser le système générer automatiquement la clé en utilisant un INSERT sans fournir de valeurs pour la clé
  - On peut forcer le numéro de la clé en le fournissant dans les instructions INSERT

Des exemples suivent!

Sans fournir le nom du champ de la clé

- La liste des champs de l'instruction INSERT ne contient pas la clé
- Un numéro séquentiel sera généré automatiquement

```
CREATE TABLE tblMembre
(           noMembre        tinyint     AUTO_INCREMENT,
            nomMembre       varchar(25),
            prenMembre      varchar(25),
            PRIMARY KEY(noMembre)
);

INSERT INTO tblMembre(nomMembre, prenMembre)
            VALUES('Bouchard', 'Julie');

INSERT INTO tblMembre(nomMembre, prenMembre) VALUES
            ('Fortin', Lucie'), ('Tremblay', 'Jonathan'), ('Bouchard', 'Annie');
```

noMembre n'est pas ici

ni là

Sans fournir le nom des colonnes

- L'instruction INSERT ne contient pas les noms des colonnes.
- Vous devez utiliser le mot-clé DEFAULT à la place de la valeur de la clé.

```
CREATE TABLE tblMembre
(           noMembre        tinyint     AUTO_INCREMENT,
            nomMembre       varchar(25),
            prenMembre      varchar(25),
            PRIMARY KEY(noMembre)
);
INSERT INTO tblMembre
VALUES(DEFAULT, 'Bouchard', 'Julie');
```

En fournissant la valeur de la clé

- La liste des champs de l'instruction INSERT contient la clé
- La clé prendra la valeur fournie, à condition qu'elle n'existe pas déjà dans la table, mais, par la suite, si on ajoute des données via un programme, le numéro augmentera automatiquement

```
CREATE TABLE tblMembre
(           noMembre        tinyint     AUTO_INCREMENT,
            nomMembre       varchar(25),
            prenMembre      varchar(25),
            PRIMARY KEY(noMembre)
);
INSERT INTO tblMembre(noMembre, nomMembre, prenMembre)
            VALUES(2, 'Bouchard', 'Julie');
```

noMembre est ici

et là

Attention aux FOREIGN KEY

- Vous devez vous même lier les données en inscrivant des valeurs appropriées dans les champs des FOREIGN KEY
- Soit un membre relié à une seule succursale

```
CREATE TABLE tblMembre
(       noMembre        smallint        NOT NULL AUTO_INCREMENT,
        nom             varchar(25)     NOT NULL,
        prenom          varchar(25)     NOT NULL,
        noSucc          tinyint         NOT NULL,
        PRIMARY KEY(noMembre),
        FOREIGN KEY(noSucc) REFERENCES tblSuccursale(noSucc)
);

INSERT INTO tblMembre(nom, prenom, noSucc)
        VALUES( 'Bouchard', 'Julie', 107);
```

La succursale 107 doit déjà exister dans tblSuccursale

Les tables d'intersection

- Si un membre peut être relié à plusieurs succursales, nous aurons une table d'intersection donc deux FK à associer

```
CREATE TABLE tblMembreSuccursale
(       noMembre        smallint            NOT NULL,
        noSucc          tinyint             NOT NULL,
        PRIMARY KEY(noMembre, noSucc),
        FOREIGN KEY(noMembre) REFERENCES tblMembre(noMembre),
        FOREIGN KEY(noSucc) REFERENCES tblSuccursale(noSucc)
);

INSERT INTO tblMembreSuccursale VALUES
            ( 1, 107),
            ( 3, 79);
```

Le membre 1 sera associé à la succursale 107 et le membre 3 à la succursale 79

Extraction de données

- La commande permettant d'extraire une ou plusieurs lignes d'une table se nomme **SELECT**.

- Nous pouvons faire de l'extraction simple, c'est –à-dire de l'extraction sur une seule table ou de l'extraction complexe avec des opérateurs, des jointures permettant de faire des requêtes sur plusieurs tables et des regroupements de données.

- Nous nous concentrerons sur des requêtes simples pour commencer.

Extraction simple

- Une extraction simple utilise une seule table
- La requête **SELECT** renvoie un résultat sous la forme d'une table
- La syntaxe:

```
SELECT {liste_champs}
FROM {TABLE}
WHERE {conditions};
```

Les champs de la table qui seront présents dans le résultat de la requête

Spécifie dans quelle table la recherche doit être effectuée

Clause optionnelle où sont posées les conditions auxquelles les enregistrements doivent satisfaire pour être inclus dans le résultat.

Un exemple d'extraction Simple

```
SELECT noCli, nomCli, villeCli
FROM   tblClient;
```

noCli	nomCli	villeCli
B062	BERGERON	Québec
B112	HANSENNE	Gatineau
B332	MONTI	Alma
B512	GILLET	Montréal
C003	AVRON	Montréal
C123	MERCIER	Québec
C400	FERARD	Gatineau
D063	MERCIER	Montréal
F010	TOUSSAINT	Gatineau
F011	PONCELET	Montréal
F400	JACOB	Toronto
K111	VANBIST	Laval
K729	NEUMAN	Montréal
L422	FRANCK	Québec
S127	VANDERKA	Québec
S712	GUILLAUME	Lévis

```
SELECT *
FROM tblClient;
```

* liste tous champs de la table

Un exemple avec une condition

```
SELECT noCli, nomCli
FROM   tblClient
WHERE  villeCli = 'Montréal';
```

WHERE permet d'ajouter des conditions

noCli	nomCli
B512	GILLET
C003	AVRON
D063	MERCIER
F011	PONCELET
K729	NEUMAN

La liste des clients est plus courte

DISTINCT - Éliminer les doublons

```
SELECT villeCli
FROM   tblClient
WHERE  cat = 'C1';
```

villeCli
Gatineau
Québec
Gatineau
Québec
Québec

Certaines lignes sont en double

```
SELECT DISTINCT villeCli
FROM   tblClient
WHERE  cat = 'C1';
```

villeCli
Québec
Gatineau

DISTINCT permet d'enlever les doublons

ORDER BY - Mettre en ordre

- Ordre croissant ou **ASC**endant (par défaut)

```
SELECT DISTINCT villeCli
FROM tblClient
ORDER BY villeCli ASC
```

villeCli
Alma
Gatineau
Laval
Lévis
Montréal
Toronto
Québec

ASC est facultatif donc
`ORDER BY villeCli` donne le même résultat

- Ordre décroissant ou **DESC**endant

```
SELECT DISTINCT villeCli
FROM tblClient
ORDER BY villeCli DESC
```

villeCli
Québec
Toronto
Montréal
Lévis
Laval
Gatineau
Alma

Valeur Null

- L'absence de valeur est indiquée par un marqueur spécial, dit *valeur NULL*.
- Plusieurs situations sont possibles pour l'instant nous nous limiterons au cas où un champ est vide.
- Le champ a été prévu dans la table mais aucune valeur n'a été entrée pour un enregistrement donné.

Conditions utilisant des valeurs NULL

```
SELECT noCli
FROM   tblClient
WHERE  cat = NULL;
```

noCli

NULL ne peut être comparé à rien, même pas à lui-même ! C'est une valeur vide

```
SELECT noCli
FROM   tblClient
WHERE  cat IS NULL;
```

noCli
D063
K729

```
SELECT noCli
FROM   tblClient
WHERE  cat IS NOT NULL;
```

La bonne syntaxe

Conditions avec LIKE

```
SELECT noCli
FROM   tblClient
WHERE  cat LIKE 'B_';
```

'_' = un caractère quelconque

```
SELECT noProd
FROM   tblProduit
WHERE  nomProd LIKE '%SAPIN%';
```

'%' = une chaîne quelconque

'B_' ⟶ 'B1'
'Bd'
'B '

'%SAPIN%' ⟶ 'PL. SAPIN 200x20x2'
'Boite en SAPIN'
'SAPIN VERNI'

'B_' ↛ 'xB'
'B'
'B12'

'%SAPIN%' ↛ 'Boite en SA PIN'
'Achetez S A P I N !'