

Base de données

Séance 8

Langage de Manipulation de Données (LMD)

Plan

- INSERT
 - UPDATE
 - DELETE
 - SELECT
 - Projection
 - Restriction
 - Tri
 - Fonction
-

SQL : Langage de Manipulation des Données (LMD)

- Le LDD :
 - permet d'effectuer des opérations sur les structures
- Le LMD
 - permet d'effectuer des opérations sur les données
 - concerne le cycle de vie des données dans une application
 - implémente le modèle «CRUDE» (Create, Read, Update, Delete).

L'implémentation du CRUDE en SQL

- Le CRUDE en SQL est constitué des quatre ordres :
 - INSERT : insertion d'enregistrement(s)
 - UPDATE : modification d'enregistrement(s)
 - DELETE : suppression d'enregistrement(s)
 - SELECT : accès aux enregistrement(s)

LMD : tables pour les exemples

```
CREATE TABLE employe (  
    nom VARCHAR2(30),  
    prenom VARCHAR2(30),  
    date_embauche DATE);
```

```
CREATE TABLE etudiant (  
    nom VARCHAR2(30),  
    prenom VARCHAR2(30),  
    classe CHAR(4) DEFAULT 'ING1');
```

INSERT : ajout d'une ligne

- Syntaxe simplifiée :

```
INSERT INTO <nom_table>  
VALUES (<expr>[,<expr>...]);
```

- Exemple :

```
INSERT INTO employe  
VALUES ('DERAY', 'ODILE',  
to_date('01/09/1972', 'DD/MM/YYYY')) ;
```

- Attention à l'ordre des colonnes !

INSERT : ajout d'une ligne

□ Syntaxe INSERT explicite (plus sûr) :

```
INSERT INTO <nom_table> (<col1> [, ... ])  
VALUES (<expr1> [, ... ]);
```

□ Exemple :

```
INSERT INTO employe  
    (nom, prenom, date_embauche)  
VALUES ('DERAY', 'ODILE',  
    to_date('01/09/1972', 'DD/MM/YYYY'));
```

INSERT : insertion partielle

- ❑ N-uplet partiel complété par des valeurs NULL ou par défaut.
- ❑ Complétion explicite :

```
INSERT INTO employe  
VALUES ('DERAY', 'ODILE', NULL);  
  
CREATE TABLE etudiant (...,  
classe CHAR(4) DEFAULT 'ING1');  
INSERT INTO etudiant  
VALUES ('DUPONT', 'EMILE', DEFAULT);
```



INSERT : insertion partielle

□ Complétion implicite :

- avec la valeur NULL

```
INSERT INTO employe (nom, prenom)
VALUES ('DERAY', 'ODILE');
```

- avec la valeur par défaut

```
INSERT INTO etudiant (nom, prenom)
VALUES ('DUPONT', 'EMILE');
```

- toutes les valeurs par défaut

```
INSERT INTO ma_table DEFAULT VALUES;
```

INSERT : insertion multiple

- Insérer le résultat d'un SELECT

```
INSERT INTO personne (nom, prenom)  
SELECT nom, prenom FROM etudiant;
```

- Ajustements possibles
 - Insertion conditionnelle
 - Répartition sur plusieurs tables

INSERT : contraintes

- Attention aux respects des contraintes !
 - Clés primaires et UNIQUE (pas de doublons)
 - Clés étrangères (existence)
 - CHECK, NOT NULL

UPDATE : modification des données

- Mettre à jour les n-uplets d'une table vérifiant un certain prédicat :

```
UPDATE <nom_table>  
SET <col>=<expr>[,<col>=<expr>...]  
[WHERE <predicat>];
```

- Les expressions sont évaluées pour chaque n-uplets vérifiant le prédicat.
- Si aucune clause WHERE n'est spécifiée, **tous** les enregistrements de la table sont modifiés !

UPDATE : exemple

UPDATE

`personne`

SET

`date_naissance = DATE '1900-01-01'`

WHERE

`date_naissance IS NULL;`

DELETE : suppression des données

- ❑ Supprime d'une table les n-uplets qui vérifient un certain prédicat.
- ❑ Syntaxe générale :

```
DELETE FROM <tab> [WHERE <predicat>];
```

- ❑ Si aucune clause WHERE n'est spécifiée, **tous** les tuples sont supprimés (cf. update) !

DELETE : exemples

```
DELETE FROM personne;
```

```
DELETE FROM personne  
WHERE date_  
_naissance IS NULL;
```

SELECT : extraction de données

- ❑ Récupérer les informations stockées dans la base de manière structurée
- ❑ Syntaxe générale :

```
SELECT  
    { <col>[, <col>... ] | * }  
FROM  
    <nom_table>  
[WHERE <expr>]
```

Forme simple du SELECT

- Récupérer **tous** les enregistrements de la table 'ma_table'

```
SELECT * FROM ma_table;
```

Projection avec SELECT

- Récupérer seulement les colonnes c1 et c2 :

```
SELECT c1, c2 FROM ma_table;
```

- Garder les éventuels doublons (non ensembliste)

Unicité avec DISTINCT

- Projection en éliminant les doublons:

```
SELECT DISTINCT c1, c2
```

```
FROM ma_table;
```

- Algèbre relationnelle :

$$\Pi_{c_1, c_2}(\text{ma_table})$$

Restriction avec la clause WHERE

- Ajout d'un prédicat avec **WHERE** :

```
SELECT *  
FROM ma_table  
WHERE <predicat>;
```

- Exemple :

```
SELECT *  
FROM ma_table  
WHERE c1 = 600;
```

- Algèbre relationnelle : $\sigma_{c1=600}(ma_table)$

Les formes de la clause WHERE

- Opérateurs de comparaison :

= , <> , <= , >= , < , >

- Opérateurs logiques

AND, OR, NOT

- Exemple :

```
SELECT *  
FROM ma_table  
WHERE c1 <> 12 AND c2 = 15;
```

Le cas particulier de la nullité

- Cas particulier des champs 'NULL' :
Il sont testés avec le prédicat **IS NULL** ou **IS NOT NULL**.

- Exemple :

```
SELECT *  
FROM ma_table  
WHERE c2 IS NULL;
```

L'opérateur LIKE

- L'opérateur **LIKE** permet de tester les chaînes de caractères avec l'expression d'un motif (pattern)
- Les méta-caractères sont :
 - _** : un seul caractère
 - %** : zéro ou plusieurs caractères

L'opérateur LIKE : exemples

- Toutes les personnes dont le nom commence par 'A' :

```
SELECT *  
FROM personne  
WHERE nom LIKE 'A%' ;
```

- Tous les prénoms dont le nom contient un 'X' en deuxième position.

```
SELECT prenom  
FROM personne  
WHERE nom LIKE '_X%' ;
```

Autres opérateurs de caractères

- Concaténation : || ou CONCAT

```
SELECT first_name || ' ' || last_name  
FROM ...;
```

```
SELECT CONCAT(first_name, ' ', last_name)  
FROM ...;
```

- Sous-chaînes :

```
SUBSTR (chaîne, pos, long)
```

*ATTENTION : En SQL les chaînes de caractères sont
indiquées à partir de 1*

Conversion de caractères

- ❑ En majuscule : UPPER(chaîne)
- ❑ En minuscule : LOWER(chaîne)
- ❑ En nombre : TO_NUMBER(chaîne)
- ❑ Exemple :

```
SELECT * FROM personne  
WHERE UPPER(nom) = UPPER('Dupont');
```

Conversion nombre/texte

- Nombre => chaîne :
 TO_CHAR(nb, masque)
- **Le masque** définit des règles de transformation :
 - 9 : chiffre représenté sauf 0 non significatif
 - 0 : chiffre représenté
 - D : séparateur décimal apparent (, ou .)
 - G : séparateur de groupe (. ou ,)
 - C : symbole ISO de monnaie locale
- Exemple : TO_CHAR(1555444.2, '999G999G999D99C')
 => 1.555.444,20EUR

Conversion de date

- ❑ chaîne => date : TO_DATE(chaine, format)
- ❑ date => chaîne : TO_CHAR(date, format)
- ❑ Le **format** définit les parties utilisées et leur ordre
- ❑ Exemple : lundi 17 novembre 2008
 - Année : YYYY, YY, YEAR
 - ❑ 2008, 08, TWO THOUSAND EIGHT (non traduit)
 - Mois : MM, Mon, Month
 - ❑ 11, Nov, Novembre
 - Semaine : WW (année), W (mois)
 - ❑ 46, 3
 - Jour : DDD (année), DD (mois), D (semaine), DAY, DY
 - ❑ 322, 17, 1, Lundi, LU

Conversion de date

□ Exemple : 14H53'45

- Heures : HH, HH12, AM, PM

- 14, 2, pm, pm

- Minutes : MM

- 53

- Secondes : SS

- 45

□ Séparateurs

- - / . , ; : espace

- "a" : entre quotes pour les autres caractères

- Exemple : 'HH"h"MM:SS' => 14h53:45

Tri avec ORDER BY

- Syntaxe :

```
SELECT ...
```

```
FROM ma_table
```

```
[WHERE ...]
```

```
ORDER BY COL1 [ASC|DESC]
```

```
        [, COL2 [ASC|DESC] *
```

Tri avec ORDER BY : exemple

- Tri de personne par année décroissante puis par nom dans l'ordre alphabétique :

```
SELECT *  
FROM PERSONNE  
ORDER BY ANNEE DESC, NOM
```

Fonction sur une ligne

□ Fonctions disponibles

- mathématiques : +, -, *, /
- chaînes de caractères, dates

□ Exemple :

- Montant total d'une ligne de commande

```
SELECT nom, quantite, prix_unitaire, quantite*prix_unitaire  
FROM ligne_commande;
```

- Renommage avec le mot-clé AS :

```
SELECT nom, quantite*prix_unitaire AS prix_total  
FROM ligne_commande;
```

Fonction sur une colonne

- Calcul sur toutes les occurrences d'une colonne: AVG, SUM, MIN, MAX, COUNT, VARIANCE, STDDEV

```
SELECT fonction(un_attribut) FROM ma_table;
```

Fonction sur une colonne : exemples

```
SELECT COUNT(*) FROM personne;
```

personne	id	nom	prenom
	1	DERAY	Odile
	2	DUPONT	Emile
	3	DURAND	Norbert



R	count
	3

```
SELECT AVG(prix) FROM produit;
```

produit	libelle	prix
	chaise	25
	table	225
	lampe	20



R	avg
	90