Les Bases de données

SQL

Un langage d'interrogation de la base

Introduction

SQL signifie Structured Query Language c'est-à-dire Langage d'interrogation structuré.

En fait SQL est un langage complet de gestion de bases de données relationnelles.

Il a été conçu par IBM dans les années 70. Il est devenu le langage standard des systèmes de gestion de bases de données (SGBD) relationnelles (SGBDR).

C'est à la fois :

- un langage d'interrogation de la base (ordre SELECT)
- un langage de manipulation des données (LMD; ordres UPDATE, INSERT, DELETE)
- un langage de définition des données (LDD ; ordres CREATE, ALTER, DROP),
- un langage de contrôle de l'accès aux données (LCD ; ordres GRANT, REVOKE).

Le langage SQL est utilisé par les principaux SGBDR : DB2, Oracle, Informix, Ingres, RDB,... Chacun de ces SGBDR a cependant sa propre variante du langage. Ce support de cours présente un noyau de commandes disponibles sur l'ensemble de ces SGBDR.

Les opérations de base

- La projection

Formalisme : R = **PROJECTION (**R1, liste des attributs**)**

Exemples :

CHAMPIGNONS

Espèce	Catégorie	Conditionnement
Rosé des prés	Conserve	Bocal
Rosé des prés	Sec	Verrine
Coulemelle	Frais	Boîte
Rosé des prés	Sec	Sachet plastique

Les opérations de base

- La projection

R1 = PROJECTION (CHAMPIGNONS, Espèce)

Espèce
Rosé des prés
Coulemelle

R2 = PROJECTION (CHAMPIGNONS, Espèce, Catégorie)

Espèce	Catégorie
Rosés des prés	Conserve
Rosé des prés	Sec
Coulemelle	Frais

•Cet opérateur ne porte que sur 1 relation.

•Il permet de ne retenir que certains attributs spécifiés d'une relation.

•On obtient tous les n-uplets de la relation ***à l'exception des doublons***.

Les opérations de base

- La projection

Le langage SQL

SELECT DISTINCT liste d'attributs **FROM** table **;**
SELECT liste d'attributs **FROM** table **;**

Exemples :

SELECT DISTINCT Espèce FROM Champignons ;
SELECT DISTINCT Espèce, Catégorie FROM Champignons ;

La clause DISTINCT permet d'éliminer les doublons.

Les opérations de base

- La sélection

Formalisme : R = **SELECTION (**R1, condition**)**

Exemple :

CHAMPIGNONS

Espèce	Catégorie	Conditionnement
Rosé des prés	Conserve	Bocal
Rosé des prés	Sec	Verrine
Coulemelle	Frais	Boîte
Rosé des prés	Sec	Sachet plastique

Les opérations de base

- La sélection

R3 = SELECTION (CHAMPIGNONS, Catégorie = "Sec")

Espèce	Catégorie	Conditionnement
Rosé des prés	Sec	Verrine
Rosé des prés	Sec	Sachet plastique

•Cet opérateur porte sur 1 relation.

•Il permet de ne retenir que les n-uplets répondant à une condition exprimée à l'aide des opérateurs arithmétiques (=, >, <, >=, <=, <>) ou logiques de base (ET, OU, NON).

•Tous les attributs de la relation sont conservés.

•Un attribut peut ne pas avoir été renseigné pour certains n-uplets. Si une condition de sélection doit en tenir compte, on indiquera simplement : nomattribut "non renseigné".

Les opérations de base

- La sélection

Le langage SQL

SELECT * FROM table **WHERE** condition **;**

Exemple :

SELECT * FROM Champignons WHERE Catégorie="Sec" ;

La condition de sélection exprimée derrière la clause WHERE peut être spécifiée à l'aide :

•des opérateurs de comparaison : =, >, <, <=, >=, <>

•des opérateurs logiques : AND, OR, NOT

•des opérateurs : IN, BETWEEN, LIKE, IS, ALL

Les opérations de base

- La sélection

Le langage SQL

Autres exemples :

Soit la table ETUDIANT(N°Etudiant, Nom, Age, CodePostal, Ville)

```
SELECT *
FROM ETUDIANT
WHERE Age IN (19, 20, 21, 22, 23) ;

SELECT *
FROM ETUDIANT
WHERE Age BETWEEN 19 AND 23 ;

SELECT *
FROM ETUDIANT
WHERE CodePostal LIKE '42%' ;  // sous Access : LIKE "42*"
```

Les opérations de base

- La sélection

Le langage SQL

Soit la table ETUDIANT(N°Etudiant, Nom, Age, CodePostal, Ville)

```
SELECT *
FROM ETUDIANT
WHERE CodePostal LIKE '42___' ;    // sous Access : LIKE "42???«

SELECT *
FROM ETUDIANT
WHERE CodePostal LIKE ‘%42%' ;    // sous Access : LIKE " *42* "

SELECT *
FROM ETUDIANT
WHERE Ville IS NULL ;     // Etudiants pour lesquels la ville n'est pas renseignée

SELECT *
FROM ETUDIANT
WHERE Ville IS NOT NULL ;     // Etudiants pour lesquels la ville est renseignée

SELECT *
FROM ETUDIANT
WHERE Age >= ALL (SELECT Age FROM ETUDIANT) ; // Etudiant(s) le(s) plus âgé(s)
```

Les opérations de base

- La jointure **(équi-jointure)**

Formalisme : R = **JOINTURE (**R1, R2, condition d'égalité entre attributs**)**

Exemple :

PRODUIT

CodePrd	Libellé	Prix unitaire
590A	HD 1,6 Go	1615
588J	Scanner HP	1700
515J	LBP 660	1820

DETAIL_COMMANDE

N°cde	CodePrd	quantité
97001	590A	2
97002	515J	1
97003	515J	3

Les opérations de base

- La jointure **(équi-jointure)**

R = JOINTURE (PRODUIT, DETAIL_COMMANDE, Produit.CodePrd=Détail_Commande.CodePrd)

A.CodePrd	Libellé	Prix unitaire	N°cde	B.CodePrd	quantité
590A	HD 1,6 Go	1615	97001	590A	2
515J	LBP 660	1820	97002	515J	1
515J	LBP 660	1820	97003	515J	3

Remarque : Des jointures plus complexes que l'équi-jointure peuvent être réalisées en généralisant l'usage de la condition de jointure à d'autres critères de comparaison que l'égalité (<,>, <=,>=, <>).

Les opérations de base

- La jointure **(équi-jointure)**

Le langage SQL

En SQL de base

SELECT * FROM table1**,** table2**,** table3**, ...**
WHERE table1.attribut1**=**table2.attribut1 **AND** table2.attribut2**=**table3.attribut2 **AND** ...**;**

Exemple :
SELECT * FROM Produit, Détail_Commande
WHERE Produit.CodePrd=Détail_Commande.CodePrd ;

ou en utilisant des alias pour les noms des tables :
SELECT * FROM Produit A, Détail_Commande B
WHERE A.CodePrd=B.CodePrd ;

Les opérations de base

- La jointure **(équi-jointure)**

Le langage SQL

Avec la clause **INNER JOIN** (jointure dite interne) à partir du SQL2

SELECT *
FROM table1 **INNER JOIN** table2 **ON** table1.attribut1=table2.attribut1
INNER JOIN table3 **ON** table2.attribut2=table3.attribut3... ;

Le mot clé INNER est facultatif sur la plupart des SGBDR (sauf MS Access).
Cette notation rend plus lisible la requête en distinguant clairement les conditions de jointures, derrière ON, et les éventuelles conditions de sélection ou restriction, derrière WHERE.
De plus, l'oubli d'un ON (et donc de la condition de jointure) empêchera l'exécution de la requête, alors qu'avec l'ancienne notation, l'oubli d'une condition de jointure derrière WHERE, n'empêche pas l'exécution de la requête, produisant alors un bien coûteux produit cartésien entre les tables !

Les opérations de base

- La jointure **(équi-jointure)**

Le langage SQL

Le même exemple que précédemment en utilisant aussi les alias :

```
SELECT *
FROM Produit A INNER JOIN Détail_Commande B ON A.CodePrd=B.CodePrd ;
```

Les opérations de base

- La jointure **(équi-jointure)**

Le langage SQL

Avec les **jointures externes** à partir du SQL2

On retiendra notamment les jointures externes Gauche (**LEFT OUTER JOIN**) et Droite (**RIGHT OUTER JOIN**).
Dans le cas d'une jointure externe gauche A->B, toute les lignes de la table A sont incluses même s'il ne leur correspond pas de ligne dans la table B.

Les opérations de base

- La jointure **(équi-jointure)**

Le langage SQL

Avec les **jointures externes** à partir du SQL2

Sur l'exemple précédent :

```
SELECT *
FROM Produit A LEFT OUTER JOIN Détail_Commande B ON A.CodePrd=B.CodePrd ;
```

Le résultat renvoyé est le suivant :

A.CodePrd	Libellé	Prix unitaire	N°cde	B.CodePrd	quantité
590A	HD 1,6 Go	1615	97001	590A	2
588J	Scanner HP	1700	NULL	NULL	NULL
515J	LBP 660	1820	97002	515J	1
515J	LBP 660	1820	97003	515J	3

Tous les produits apparaissent même si certains n'ont pas fait l'objet de commande (exemple : 588J). Les colonnes manquantes sont alors complétées par des valeurs NULL.

Les opérations de base

EXRCICES

Les opérations ensemblistes

- Opération UNION

Formalisme : R = **UNION (**R1**,** R2**)**

Exemple :

E1 : Enseignants élus au CA

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

Les opérations ensemblistes

- Opération UNION

Exemple :

On désire obtenir l'ensemble des enseignants élus au CA ou représentants syndicaux.

R1 = UNION (E1, E2)

n°enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND
6	MICHEL

• Cet opérateur porte sur deux relations qui doivent avoir le même nombre d'attributs définis dans le même domaine (ensemble des valeurs permises pour un attribut). On parle de relations ayant le même schéma.

• La relation résultat possède les attributs des relations d'origine et les n-uplets de chacune, avec élimination des doublons éventuels.

Les opérations ensemblistes

- Opération UNION

Le langage SQL

SELECT liste d'attributs **FROM** table1
UNION
SELECT liste d'attributs **FROM** table 2 **;**

Exemple :
SELECT n°enseignant, NomEnseignant FROM E1
UNION
SELECT n°enseignant, NomEnseignant FROM E2 ;

Les opérations ensemblistes

- Opération INTERSECTION

R = **INTERSECTION (**R1**,** R2**)**

Exemple :

E1 : Enseignants

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

On désire obtenir l'ensemble des enseignants ou représentants syndicaux.

Les opérations ensemblistes

- Opération INTERSECTION

On désire connaître les enseignants du CA qui sont des représentants syndicaux.

R2 = INTERSECTION (E1, E2)

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN

•Cet opérateur porte sur deux relations de même schéma.
•La relation résultat possède les attributs des relations d'origine et les n-uplets communs à chacune.

Les opérations ensemblistes

- Opération INTERSECTION

Le langage SQL

En SQL de base, plusieurs possibilités :

SELECT attribut1, attribut2, ... **FROM** table1
WHERE attribut1 **IN (SELECT** attribut1 **FROM** table2**) ;**

SELECT attribut1, attribut2, ... **FROM** table1
WHERE EXISTS (SELECT * **FROM** table2 **WHERE** table1.attribut1=table2.attribut1**) ;**

SELECT attribut1, attribut2, ... **FROM** table1
WHERE attribut1 **= ANY (SELECT** attribut1 **FROM** table2**) ;**

Les opérations ensemblistes

- Opération INTERSECTION

Le langage SQL

ou avec l'opérateur INTERSECT (SQL2) :

```
SELECT attribut1, attribut2, ... FROM table1
INTERSECT
SELECT attribut1, attribut2, ... FROM table2 ;
```

Les opérations ensemblistes

- Opération INTERSECTION

Le langage SQL

ou avec une équi-jointure :

```
SELECT attribut1, attribut2, ...
FROM table1 INNER JOIN table2 ON table1.attribut1 = table2.attribut1 ;
```

Les opérations ensemblistes

- Opération INTERSECTION

Le langage SQL

Exemple :

```
SELECT n°enseignant, NomEnseignant FROM E1
WHERE n°enseignant IN (SELECT n°enseignant FROM E2) ;
```

Ou

```
SELECT n°enseignant, NomEnseignant FROM E1
INTERSECT
SELECT n°enseignant, NomEnseignant FROM E2 ;
```

Les opérations ensemblistes

- Opération DIFFERENCE

Formalisme : R = **DIFFERENCE (**R1**,** R2**)**

Exemple :

E1 : Enseignants

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

On désire obtenir la liste des enseignants qui ne sont pas des représentants syndicaux.

Les opérations ensemblistes

- Opération DIFFERENCE

R3 = DIFFERENCE (E1, E2)

n°enseignant	nom_enseignant
3	DURAND
5	BERTRAND

•Cet opérateur porte sur deux relations de même schéma.
•La relation résultat possède les attributs des relations d'origine et les n-uplets de la première relation qui n'appartiennent pas à la deuxième.
•Attention ! DIFFERENCE (R1, R2) ne donne pas le même résultat que DIFFERENCE (R2, R1)

Les opérations ensemblistes

- Opération DIFFERENCE

Le langage SQL

En SQL de base, plusieurs possibilités :

SELECT attribut1**,** attribut2**,** ... **FROM** table1
WHERE attribut1 **NOT IN (SELECT** attribut1 **FROM** table2**) ;**

SELECT attribut1**,** attribut2**,** ... **FROM** table1
WHERE NOT EXISTS (SELECT * **FROM** table2 **WHERE** table1.attribut1=table2.attribut1**) ;**

SELECT attribut1**,** attribut2**,** ... **FROM** table1
WHERE attribut1 **<> ALL (SELECT** attribut1 **FROM** table2**) ;**

Les opérations ensemblistes

- Opération DIFFERENCE

Le langage SQL

ou avec l'opérateur EXCEPT (SQL2) :

```
SELECT attribut1, attribut2, ... FROM table1
EXCEPT
SELECT attribut1, attribut2, ... FROM table2 ;
```

Les opérations ensemblistes

- Opération DIFFERENCE

Le langage SQL

ou encore, avec la jointure externe (SQL2),

si par exemple vous utilisez d'un version SGBDR qui ne dispose ni du EXCEPT, ni de la possibilité de SELECT imbriqués :

```
SELECT table1.attribut1, table1.attribut2,...
FROM table1 LEFT JOIN table2 ON table1.attribut1 = table2.attribut1
WHERE table2.attribut1 IS NULL ;
```

Les opérations ensemblistes

- Opération DIFFERENCE

Le langage SQL

Exemple :

```
SELECT n°enseignant, NomEnseignant FROM E1
WHERE n°enseignant NOT IN (SELECT n°enseignant FROM E2) ;
```

ou

```
SELECT n°enseignant, NomEnseignant FROM E1
EXCEPT
SELECT n°enseignant, NomEnseignant FROM E2 ;
```

ou encore

```
SELECT E1.n°enseignant, E1.NomEnseignant
FROM E1 LEFT JOIN E2 ON E1.n°enseignant = E2.n°enseignant
WHERE E2.n°enseignant IS NULL ;
```

Les opérations ensemblistes

- Opération DIFFERENCE

Le langage SQL

Pour mieux comprendre cette dernière version, voici le résultat renvoyé par la jointure externe gauche entre E1 et E2 :

E1.n°enseignant	E1.NomEnseignant	E2.n°enseignant	E2.NomEnseignant
1	DUPONT	1	DUPONT
3	DURAND	NULL	NULL
4	MARTIN	4	MARTIN
5	BERTRAND	NULL	NULL

Les opérations ensemblistes

- Opération PRODUIT CARTESIEN

Formalisme : R = **PRODUIT (**R1**,** R2**)**

Exemple :

Étudiants

n°étudiant	**nom**
101	DUPONT
102	MARTIN

Épreuves

libellé épreuve	**coefficient**
Informatique	2
Mathématiques	3
Gestion financière	5

Les opérations ensemblistes

- Opération PRODUIT CARTESIEN

Examen = PRODUIT (Étudiants, Épreuves)

n°étudiant	nom	libellé épreuve	coefficient
101	DUPONT	Informatique	2
101	DUPONT	Mathématiques	3
101	DUPONT	Gestion financière	5
102	MARTIN	Informatique	2
102	MARTIN	Mathématiques	3
102	MARTIN	Gestion financière	5

•Cet opérateur porte sur deux relations.
•La relation résultat possède les attributs de chacune des relations d'origine et ses n-uplets sont formés par la concaténation de chaque n-uplet de la première relation avec l'ensemble des n-uplets de la deuxième.

Les opérations ensemblistes

- Opération PRODUIT CARTESIEN

Le langage SQL

SELECT * FROM table1**,** table2 **;**

Exemple :

SELECT * FROM Etudiants, Epreuves ;

Les opérations ensemblistes

EXRCICES

ECRITURE D'UNE REQUETE

La plupart des requêtes (ou interrogations) portant sur une base de données relationnelle ne peuvent pas être réalisées à partir d'une seule opération mais en enchaînant successivement plusieurs opérations.

Exemple

Soient les deux tables (ou relations) suivantes :

CLIENT(CodeClient, NomClient, AdrClient, TélClient)

COMMANDE(N°Commande, Date, CodeClient#)

On désire obtenir le code et le nom des clients ayant commandé le 10/06/2008 :

ECRITURE D'UNE REQUETE

R1=SELECTION(COMMANDE, Date=10/06/2008)
R2=JOINTURE(R1, CLIENT, R1.CodeClient=CLIENT.CodeClient)
R3=PROJECTION(R2, CodeClient, NomClient)

ECRITURE D'UNE REQUETE

Le langage SQL

Une même instruction SELECT permet de combiner Sélections, Projections, Jointures.

```
SELECT DISTINCT CLIENT.CodeClient, NomClient
FROM CLIENT, COMMANDE
WHERE CLIENT.CodeClient=COMMANDE.CodeClient AND Date='10/06/2008';
```

ou avec la clause INNER JOIN :

```
SELECT DISTINCT CLIENT.CodeClient, NomClient
FROM CLIENT INNER JOIN COMMANDE ON
CLIENT.CodeClient=COMMANDE.CodeClient
WHERE Date='10/06/2008';
```

Remarque : Si l'on ne précisait pas CLIENT.CodeClient au niveau du SELECT, la commande SQL ne pourrait pas s'exécuter à cause de l'ambiguïté sur le CodeClient

ECRITURE D'UNE REQUETE

Le langage SQL

Autres solutions si le SGBDR les supportent

```
SELECT DISTINCT CodeClient, NomClient
FROM CLIENT INNER JOIN COMMANDE USING(CodeClient)
WHERE Date='10/06/97';
```

```
SELECT DISTINCT CodeClient, NomClient
FROM CLIENT NATURAL JOIN COMMANDE
WHERE Date='10/06/97';
```

Remarque :

Dans ces 2 dernières solutions, l'attribut CodeClient n'étant pas dupliqué, il n'est pas nécessaire de le préfixer avec le nom d'une table.

ECRITURE D’UNE REQUETE

EXRCICES

Les opérations d'agrégation

Elles sont utilisées dans les opérateurs **CALCULER** et **REGROUPER_ET_CALCULER**.

Les fonctions statistiques de base

Elles portent sur un ou plusieurs groupes de n-uplets et évidemment sur un attribut de type numérique.

Somme(attribut) : total des valeurs d'un attribut
Moyenne(attribut) : moyenne des valeurs d'un attribut
Minimum(attribut) : plus petite valeur d'un attribut
Maximum(attribut) : plus grande valeur d'un attribut

Remarque : les valeurs "non renseignées" de l'attribut sont ignorées.

La fonction de comptage : Comptage()

La fonction de comptage donne le nombre de n-uplets d'un ou de plusieurs groupes de n-uplets. Il n'est donc pas nécessaire de préciser d'attribut.

Les opérations d'agrégation

Opération calculer

R=**CALCULER(**R0, fonction_agreg1, fonction_agreg2, ...**)**
ou
N=**CALCULER(**R0, fonction_agreg**)**

Exemple
Soit le relation: LIGNE_COMMANDE

N°BonCommande	CodeProduit	Quantité	PuHt
96008	A10	10	83
96008	B20	35	32
96009	A10	20	83
96010	A15	4	110
96010	B20	55	32

On désire obtenir le chiffre d'affaires total Ht, ainsi que le nombre total de produits commandés

Les opérations d'agrégation

Opération calculer

On désire obtenir le chiffre d'affaires total Ht, ainsi que le nombre total de produits commandés

R1=CALCULER(LIGNE_COMMANDE, Somme(Quantité*PuHt), Somme(Quantité))

Somme(Quantité*PuHt)	Somme(Quantité)
5810	124

- Les calculs et/ou comptage portent sur la relation R0.
- La relation résultat ne comportera qu'une ligne avec autant de colonnes que de résultats demandés ou pourra simplement être considérée comme un nombre N utilisable ultérieurement en tant que tel dans le cas où un seul résultat est attendu.

Les opérations d'agrégation

Opération calculer

Le langage SQL

SELECT fonction_agreg1(attribut1), fonction_agreg2(attribut2), ...
FROM table **;**

Exemple :
SELECT SUM(Quantité*PuHt), SUM(Quantité)
FROM LIGNE_COMMANDE;

Les opérations d'agrégation

Opération regrouper et calculer

R=REGROUPER_ET_CALCULER(R0, att1, att2, ..., fonction_agreg1, fonction_agreg2, ...**)**

Exemple
Soit la relation LIGNE_COMMANDE:

N°BonCommande	CodeProduit	Quantité	PuHt
96008	A10	10	83
96008	B20	35	32
96009	A10	20	83
96010	A15	4	110
96010	B20	55	32

On désire obtenir le montant total Ht de chaque bon de commande :

Les opérations d'agrégation

Opération regrouper et calculer

On désire obtenir le montant total Ht de chaque bon de commande :

R2=REGROUPER_ET_CALCULER(LIGNE_COMMANDE, N°BonCommande, MontantHt : Somme(Quantité*PuHt))

N°BonCommande	MontantHt
96008	1950
96009	1660
96010	2200

•Le regroupement s'effectue sur un sous ensemble des attributs de la relation R0.
•La relation résultat comportera autant de lignes que de groupes de n-uplets, les fonctions s'appliquant à chacun des groupes séparément.

Les opérations d'agrégation

Opération regrouper et calculer

Le langage SQL

SELECT attribut1, attribut2, ..., fonction_agreg1(attribut3), fonction_agreg2(attribut4), ...
FROM table
GROUP BY attribut1, attribut2, ... **;**
Exemple :

```
SELECT N°BonCommande, SUM(Quantité*PuHt)
    FROM LIGNE_COMMANDE
    GROUP BY N°BonCommande ;
```

•La clause GROUP BY est obligatoire dès qu'il y a à la fois des attributs et des fonctions d'agrégation derrière la clause SELECT.
•Sur la plupart des SGBDR, tous les attributs placés derrière la clause SELECT doivent être présents derrière la clause GROUP BY. La proposition inverse n'est pas vraie.

Les opérations d'agrégation

Opération regrouper et calculer

Le langage SQL

Il est possible de sélectionner des lignes issues d'un regroupement (grâce à la clause HAVING) et même de les trier.

Exemple : on souhaite, parmi l'ensemble des commandes, ne retenir que celles dont la montant total hors taxes est supérieur à 10000. De plus on souhaite les voir apparaître par ordre décroissant de leurs montants respectifs.

```
SELECT N°BonCommande, SUM(Quantité*PuHt)
FROM LIGNE_COMMANDE
GROUP BY N°BonCommande HAVING SUM(Quantité*PuHt)>10000
ORDER BY 2 DESC ;
```

Il n'est pas toujours possible de placer une fonction derrière la clause ORDER BY. Mais les colonnes projetées derrière la clause SELECT étant implicitement numérotées de 1 à n, il est facile d'y faire référence. Ceci explique la présence du chiffre 2 derrière le ORDER BY de l'exemple : il fait référence à SUM(Quantité*PuHt).

Les opérations d'agrégation

EXRCICES

L'opération de TRI

Formalisme : R = **TRI(**R0, att1↑, att2↓, ...**)**

Exemple :

CHAMPIGNONS

Espèce	Catégorie	Conditionnement
Rosé des prés	Conserve	Bocal
Rosé des prés	Sec	Verrine
Coulemelle	Frais	Boîte
Rosé des prés	Sec	Sachet plastique

Trier la relation en ordre croissant sur les espèces et on ordre décroissant sur le catégories

L’opération de TRI

R1 = TRI (CHAMPIGNONS, Espèce↓,Catégorie↑,Conditionnement↑)

Espèce	Catégorie	Conditionnement
Rosé des prés	Conserve	Bocal
Rosé des prés	Sec	Sachet plastique
Rosé des prés	Sec	Verrine
Coulemelle	Frais	Boîte

- Le tri s'effectue sur un ou plusieurs attributs, dans l'ordre croissant ou décroissant.
- La relation résultat a la même structure et le même contenu que la relation de départ.

L’opération de TRI

Le langage SQL

SELECT attribut1**,** attribut2**,** attribut3**,** ...
FROM table
ORDER BY attribut1 **ASC,** attribut2 **DESC, ... ;**

ASC : par ordre croissant (Ascending)
DESC : par ordre décroissant (Descending)

Exemple :
SELECT Espèce, Catégorie, Conditionnement
FROM Champignons
ORDER BY Espèce DESC, Catégorie ASC, Conditionnement ASC ;
Remarque : par défaut le tri se fait par ordre croissant si l'on ne précise pas ASC ou DESC.

Attributs calculés et renommés

Attributs calculés
Un attribut calculé est un attribut dont les valeurs sont obtenues par des opérations arithmétiques portant sur des attributs de la même relation.
Le calcul est spécifié lors d'une projection ou lors de l'utilisation d'une fonction.
R=PROJECTION(R0, att1, att2, att3, att4, att1*att2, att3/att2)

Attributs renommés
Il est possible de renommer n'importe quel attribut en le faisant précéder de son nouveau nom suivi de ":".
R=PROJECTION(R0, att1, att2, att3, att4, newatt1:att1*att2, newatt2:att3/att2)

Attributs calculés et renommés

Exemple

Soit la relation: LIGNE_COMMANDE

N°BonCommande	CodeProduit	Quantité	PuHt
96008	A10	10	83
96008	B20	35	32
96009	A10	20	83
96010	A15	4	110
96010	B20	55	32

On désire afficher pour chaque commande le montant à payer

Attributs calculés et renommés

R=PROJECTION(LIGNE_COMMANDE, N°BonCommande, CodeProduit, Montant:Quantité*PuHt)

N°BonCommande	CodeProduit	Montant
96008	A10	830
96008	B20	1120
96009	A10	1660
96010	A15	440
96010	B20	1760

Attributs calculés et renommés

Le langage SQL

Attributs calculés

```
SELECT N°BonCommande, CodeProduit, Quantité*PuHt
FROM LIGNE_COMMANDE ;
```

Attributs renommés

```
SELECT N°BonCommande, CodeProduit, Quantité*PuHt AS Montant
FROM LIGNE_COMMANDE ;
```

Opération DIVISION

Formalisme: R = **DIVISION (**R1, R2**)**

Exemple :

PARTICIPER

Athlète	Epreuve
Dupont	200 m
Durand	400 m
Dupont	400 m
Martin	110 m H
Dupont	110 m H
Martin	200 m

EPREUVE

Epreuve
200 m
400 m
110 m H

DIVISION (PARTICIPER, EPREUVE)

Athlète
Dupont

L'athlète Dupont participe à toutes les épreuves

Opération DIVISION

Cet opérateur porte sur 2 relations qui doivent avoir au moins un attribut défini dans le même domaine.

Tous les attributs du diviseur (ici EPREUVE) doivent être des attributs du dividende (ici PARTICIPER).

La relation dividende doit avoir au moins une colonne de plus que la relation diviseur.

La relation résultat, le quotient, possède les attributs non communs aux deux relations initiales et est formée de tous les n-uplets qui, concaténés à chacun des n-uplets du diviseur (ici EPREUVE) donne toujours un n-uplet du dividende (ici PARTICIPER).

Opération DIVISION

Il n'existe pas en SQL d'équivalent direct à la division.
Cependant il est toujours possible de trouver une autre solution, notamment par l'intermédiaire des opérations de calcul et de regroupement.

Dans l'exemple présenté, on souhaite trouver les athlètes qui participent à toutes les épreuves.

```
SELECT Athlète FROM PARTICIPER
GROUP BY Athlète
HAVING COUNT(*) = (SELECT COUNT(*) FROM EPREUVE) ;
```

```
SELECT Athlète FROM PARTICIPER
WHERE Epreuve IN (SELECT Epreuve FROM EPREUVE)
GROUP BY Athlète
HAVING COUNT(*) = (SELECT COUNT(DISTINCT Epreuve) FROM
EPREUVE) ;
```

Opération DIVISION

Autres solutions

```
SELECT Athlète FROM PARTICIPER
WHERE Epreuve IN (SELECT Epreuve FROM EPREUVE)
GROUP BY Athlète
HAVING COUNT(*) = (SELECT COUNT(DISTINCT Epreuve) FROM
EPREUVE) ;
```

```
SELECT Athlète
FROM PARTICIPER A LEFT JOIN (SELECT DISTINCT Epreuve FROM
EPREUVE) B ON A.Epreuve = B.Epreuve
GROUP BY Athlète
HAVING COUNT(*) = (SELECT COUNT(DISTINCT Epreuve) FROM
EPREUVE) AND COUNT(B.Epreuve) = (SELECT COUNT(DISTINCT
Epreuve) FROM EPREUVE) ;
```

Syntaxe simplifiée de l'instruction SELECT

SELECT [**DISTINCT**] att1 [, att2, att3, ...]
FROM Table1 [t1 **JOIN** Table2 t2 **ON** t1.cle1=t2.cle1
[**JOIN** Table3 t3 **ON** t2.cle2=t3.cle2 ...]]
[**WHERE** conditions de sélection]
[**GROUP BY** att1 [, att2, ...] [**HAVING** conditions de sélection sur lignes regroupées]]
[**ORDER BY** att1 [**ASC** | **DESC**] [, att2 [**ASC** | **DESC**], ...] ;

JOIN : uniquement les enregistrements en correspondance
LEFT JOIN : tous les enregistrements de Table1 et ceux de Table2 en correspondance
[ ] : optionnel
| : ou

Les Bases de données

SQL

Un langage de définition des données LDD
Création

LES TABLES sql 2

Syntaxes

CREATE [{ GLOBAL | LOCAL } TEMPORARY] TABLE nom_table
(colonne | contrainte_de_table [{ , colonne | contrainte_de_table }...])

---

colonne ::
nom_colonne { type | domaine }
[DEFAULT valeur_default]
[contrainte_de_colonne...]
[COLLATE collation]

---

contrainte_de_colonne ::
[CONSTRAINT nom_contrainte]
[NOT] NULL
| UNIQUE | PRIMARY KEY
| CHECK (prédicat_de_colonne)
| FOREIGN KEY [colone] REFERENCES table (colonne) spécification_référence

LES TABLES

Syntaxes

contrainte_de_table ::
CONSTRAINT nom_contrainte
{ UNIQUE | PRIMARY KEY (liste_colonne)
| CHECK (prédicat_de_table)
| FOREIGN KEY liste colonne REFERENCES nom_table (liste_colonne)
spécification_référence }

1. une table peut être créée de manière durable (par défaut) ou temporaire et dans ce dernier cas uniquement pour l'utilisateur et la connexion qui l'a créé ou bien pour l'ensemble des utilisateurs de la base
2. une table comporte des colonnes et des contraintes de table
3. une colonne peut être spécifiée d'après un type SQL ou un domaine créé par l'utilisateur
4. une colonne définie peut être dotée de contraintes de colonnes telles que : obligatoire, clef, unicité, intégrité référentielle et validation
5. une contrainte de table porte sur une ou plusieurs colonnes et permet l'unicité, la validation et l'intégrité référentielle

RAPPEL : un nom de colonne doit être unique au sein de la table

LES TABLES

Exemples

```
CREATE TABLE T_CLIENT
(CLI_NOM CHAR(32),
CLI_PRENOM VARCHAR(32))
```

Une table de clients dotée de deux colonnes avec les nom et prénom des clients.

```
CREATE TABLE T_CLIENT
(CLI_ID INTEGER NOT NULL PRIMARY KEY,
CLI_NOM CHAR(32) NOT NULL,
CLI_PRENOM VARCHAR(32))
```

Une table de clients dotée de trois colonnes avec la clef (numéro du client) les nom et prénom des clients.

```
CREATE DOMAIN D_NUM_ID INTEGER
CONSTRAINT C_CLEF CHECK (VALUE > 0)
CREATE DOMAIN D_ALFA_FIX_32 CHAR(32)
CREATE DOMAIN D_ALFA_VAR_32 VARCHAR(32)
CREATE TABLE T_CLIENT
(CLI_ID D_NUM_ID NOT NULL PRIMARY KEY,
CLI_NOM D_ALFA_FIX_32 NOT NULL,
CLI_PRENOM D_ALFA_VAR_32)
```

Une table de clients similaire à l'exemple 1 à base de domaines mais la clef ne peut être négative.

LES TABLES

Exemples

```
CREATE TABLE T_CLIENT
(CLI_ID INTEGER NOT NULL PRIMARY KEY,
CLI_NOM CHAR(32) NOT NULL CHECK (SUBSTRING(VALUE, 1, 1) <> ' ' AND
UPPER(VALUE) = VALUE),
CLI_PRENOM VARCHAR(32) REFERENCES TR_PRENOM (PRN_PRENOM))
```

Une table de clients similaire à l'exemple 1 dont le nom ne peut commencer par un blanc, doit être en majuscule et dont le prénom doit figurer dans la table de référence TR_PRENOM à la colonne PRN_PRENOM.

```
CREATE TABLE T_VOITURE
(VTR_ID INTEGER NOT NULL PRIMARY KEY,
VTR_MARQUE CHAR(32) NOT NULL, VTR_MODELE VARCHAR(16),
VTR_IMMATRICULATION CHAR(10) NOT NULL UNIQUE,
VTR_COULEUR CHAR(16) CHECK (VALUE IN ('BLANC', 'NOIR', 'ROUGE', 'VERT', 'BLEU')))
```

Une table de voiture avec immatriculation unique et couleur limitée à 'BLANC', 'NOIR', 'ROUGE', 'VERT', 'BLEU'.

LES TABLES

Exemples

```
CREATE TABLE T_PERSONNE9
(PRS_NOM VARCHAR(32) NOT NULL,
PRS_PRENOM VARCHAR(32) NOT NULL,
PRS_TELEPHONE CHAR(14),
CONSTRAINT PK_PRS PRIMARY KEY (PRS_NOM, PRS_PRENOM))
```

clef primaire sur PRS_NOM / PRS_PRENOM

```
CREATE TABLE T_PERSONNE
(PRS_ID INTEGER, PRS_NOM VARCHAR(32),
PRS_PRENOM VARCHAR(32),
CONSTRAINT UNI_NOM_PRENOM UNIQUE (PRS_NOM, PRS_PRENOM))
```

Un contrainte d'unicité peut être portée sur plusieurs colonnes. Dans ce cas chaque n-uplets de valeurs explicite doit être différents.

LES TABLES

Exemples

```
CREATE TABLE T_FACTURE
(FTC_ID INTEGER,
PRS_ID INTEGER FOREIGN KEY REFERENCES T_PERSONNE5 (PRS_ID) ,
FCT_DATE DATE,
FCT_MONTANT DECIMAL(16,2))
```

La table T_FACTURE est liée à la table T_PERSONNE5 et ce lien se fait entre la clef étrangère PRS_ID de la table T_FACTURE1 et la clef de la table T_PERSONNE5 qui s'intitule aussi PRS_ID.

LES TABLES

Les contraintes de table

Une table peut être pourvue des contraintes de ligne suivante :

PRIMARY KEY : précise que la ou les colonnes composent la clef de la table.
ATTENTION : nécessite que chaque colonne concourrant à la clef soit NOT NULL.
UNIQUE : les valeurs de la ou les colonnes doivent être unique ou NULL, c'est à dire qu'à l'exception du marqueur NULL, il ne doit jamais y avoir plus d'une fois la même valeur (pas de doublon) au sein de l'ensemble de données formé par les valeurs des différentes colonnes composant la contrainte.
CHECK : permet de préciser un prédicat validant différentes colonnes de la table et qui accepterons les valeurs s'il est évalué à vrai.
FOREIGN KEY : permet, pour les valeurs de la ou les colonnes, de faire référence à des valeurs préexistantes dans une ou plusieurs colonnes d'une autre table. Ce mécanisme s'appelle intégrité référentielle.

Les vues

Les vues de la norme SQL 2 ne sont autre que des requêtes instanciées.

Syntaxe

```
CREATE VIEW nom_vue [ ( nom_col1, [, nom_col2 ... ] ) ]
AS
requête_select [WITH CHECK OPTIONS]
```

Les vues

Exemple

soit la table suivante :

```
CREATE TABLE T_EMPLOYE
(EMP_ID INTEGER PRIMARY KEY,
EMP_MATRICULE CHAR(8),
EMP_TITRE VARCHAR(4),
EMP_NOM VARCHAR(32),
EMP_PRENOM VARCHAR(32),
EMP_DATE_NAIS DATE,
EMP_SALAIRE FLOAT,
EMP_STATUT CHAR(8),
EMP_MAIL VARCHAR(128),
EMP_TEL CHAR(16))
```

pour le service comptable, on pourra définir la vue suivante :

```
CREATE VIEW V_EMP_SYNDICAT
AS
SELECT
EMP_ID,
EMP_PRENOM,
EMP_NOM,
EMP_SALAIRE
FROM T_EMPLOYE
WHERE
STATUT = ‘Titulaire'
WITH CHECK OPTIONS
```

Elle pourra être mis à jour uniquement pour les salariés de type ‘Titulaire '

La clause WITH CHECK OPTION implique que si la vue peut être mise à jour, alors les valeurs modifiées insérées ou supprimées doivent répondre à la validation de la clause WHERE comme s'il s'agissait d'une contrainte.

Les Bases de données

SQL

Un langage de définition des données LDD
Alter et Drop

LES TABLES

Les ordres ALTER et DROP sont les ordres de modification (ALTER pour altération) et (DROP pour suppression).
L'ordre ALTER peut porter sur un domaine, une assertion, une table, une vue, etc...
L'ordre **ALTER** sur une table permet de :

1. supprimer une colonne
2. supprimer une contrainte
3. ajouter une colonne
4. ajouter une contrainte
5. ajouter une contrainte de ligne DEFAULT

Il ne permet pas de :

1. changer le nom d'une colonne
2. changer le type d'une colonne
3. ajouter une contrainte de ligne NULL / NOT NULL

LES TABLES

Syntaxe de l'ordre ALTER sur table :

ALTER TABLE nom_table
{ ADD definition_colonne
| ALTER nom_colonne { SET DEFAULT valeur_défaut
| DROP DEFAULT } | DROP nom_colonne [CASCADE | RESTRICT]
| ADD définition_contrainte_ligne
| DROP CONSTRAINT nom_contrainte [CASCADE | RESTRICT] }

L'option CASCADE / RESTRICT permet de gérer l'intégrité de référence de la colonne ou la contrainte.
Si RESTRICT, alors tout objet dépendant de cette colonne ou de cette contrainte provoquera l'annulation de l'opération de suppression.
Si CASCADE, alors tous les objets dépendant de cette colonne ou contrainte seront supprimés.

LES TABLES

Exemple

```
ALTER TABLE T_CLIENT
ADD CLI_PRENOM VARCHAR(25)
ALTER TABLE T_CLIENT
ADD CLI_DATE_NAISSANCE DATE,
ALTER TABLE T_CLIENT
ADD CONSTRAINT CHK_DATE_NAISSANCE CHECK (CLI_DATE_NAISSANCE
BETWEEN '1880-01-01' AND '2020-01-01')
```

LES TABLES

DROP est l'ordre de suppression. Sa syntaxe est :

Syntaxe

DROP {TABLE | DOMAIN | ASSERTION | VIEW } nom_objet

Les assertions

Les assertions sont des expressions devant être satisfaites lors de la modifications de données pour que celles-ci puissent être réalisées. Ainsi, elles permettent de garantir l'intégrité des données. Leur syntaxe est la suivante:

CREATE ASSERTION Nom_de_la_contrainte CHECK (expression_conditionnelle)

La condition à remplir peut (et est généralement) être effectuée grâce à une clause *SELECT*. Les assertions ne sont pas implémentées dans l'ensemble des SGBDR...

Les assertions Exemple

Par exemple vous pouvez définir une règle de gestion qui indique que le montant des commandes non réglées ne doit pas dépasser 20% du montant du chiffre d'affaire déjà réalisé par le client.
Pour réaliser notre exemple nous avons besoin des tables T_CLIENT, T_FACTURE et T_COMPTE, définies de manière simpliste comme suit :

LES TABLES

Les assertions Exemple

```
CREATE TABLE T_CLIENT
      (CLI_ID       INTEGER       NOT NULL PRIMARY KEY,
       CLI_NOM      CHAR(32)      NOT_NULL)

CREATE TABLE T_FACTURE
      (FCT_ID       INTEGER       NOT NULL PRIMARY KEY,
       CLI_ID       INTEGER       NOT NULL REFERENCES T_CLIENT (CLI_ID),
       FCT_DATE     DATE          NOT NULL DEFAULT CURRENT_DATE,
       FCT_MONTANT  DECIMAL (16,2) NOT NULL,
       FCT_PAYE     BIT(1)        NOT NULL DEFAULT 0)

CREATE TABLE T_COMPTE
      (CPT_ID       INTEGER       NOT NULL PRIMARY KEY,
       CLI_ID       INTEGER       NOT NULL REFERENCES T_CLIENT (CLI_ID),
       CPT_CREDIT   DECIMAL (16,2)
       CPT_DEBIT    DECIMAL (16,2)
       CPT_DATE     DATE          NOT NULL DEFAULT CURRENT_DATE)
```

Les assertions Exemple

Dans ce cas, l'assertion prendra la forme :

```
CREATE ASSERTION AST_VERIFACTURE
    CHECK (SELECT SUM(FCT_MONTANT)
           FROM   T_FACTURE F
           WHERE  FCT_PAYE = 0
           GROUP  BY CLI_ID, FCT_PAYE) < (SELECT 0.2 * (SUM(CPT_DEBIT)
    - SUM(CPT_CREDIT)) FROM   T_COMPTE  WHERE  CLI_ID = F.CLI_ID)
```

REMARQUE : certains SGBDR n'utilise pas les assertions mais propose des mécanismes similaires généralement nommés RULE (règle)..

Les Bases de données

SQL

Un langage de manipulation des données LMD

INSERTION « INSERT INTO »

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 }
```

La liste des colonnes visées peut être omise 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 SGBDR à l'accepter (Oracle est l'un des rares SGBDR à accepter cette syntaxe).

INSERTION « INSERT INTO »

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.

Exemple 1

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

Exemple 2

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

INSERTION « INSERT INTO »

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

Exemple 1

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

Exemple 2

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

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

INSERTION « INSERT INTO »

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)
```

Exemple

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

Ou encore

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

INSERTION « INSERT INTO »

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 :

Exemple

```
INSERT INTO T_SUIVI_CONNEXION DEFAULT VALUES
```

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

INSERTION « INSERT INTO »

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 identique 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

INSERTION « INSERT INTO »

Insertion multiple à base de sous requête SELECT

Exemple 1

```
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
```

Exemple 2

```
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
```

SUPPRESSION « 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'il existe des intégrités référentielles de suppression en cascade.

SUPPRESSION « DELETE »

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 :

Exemple

```
DELETE FROM T_PROSPECT
```

SUPPRESSION « DELETE »

Suppression conditionnelle

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

Exemple

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

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

SUPPRESSION « DELETE »

Suppression avec sous requête conditionnelle

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

Exemple

```
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
```

Supprime les prospects dont le couple de valeurs nom/prénom se trouve dans la table des clients

MODIFICATION « 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]
```

MODIFICATION « UPDATE »

Mise à jour d'une colonne unique sans condition

C'est la forme la plus simple de l'ordre UPDATE.

Exemple

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
```

MODIFICATION « UPDATE »

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).

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
```

MODIFICATION « UPDATE »

Mise à jour d'une colonne unique avec filtrage

On peut ajouter une clause de filtrage WHERE dans une requête de mise à jour

Exemple

Par exemple nous pouvons décider de n'augmenter de 15% que les tarifs des petits déjeuners des périodes postérieures à 2007.

```
UPDATE T_TARIF
SET TRF_PETIT_DEJEUNE = TRF_PETIT_DEJEUNE * 1.15
WHERE EXTRACT(YEAR FROM TRF_DATE_DEBUT) > 2007
```

MODIFICATION « UPDATE »

Mise à jour de plusieurs colonnes simultanément

Pour mettre à jour simultanément plusieurs colonnes, il suffit de répéter autant 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.

Exemple

```
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.

UPPER () Transforme les caractères d'une expression en minuscules en caractères majuscules.

MODIFICATION « UPDATE »

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)
```

MODIFICATION « UPDATE »

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

l 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 précèdent, 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
```

MODIFICATION « UPDATE »

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 :

1. violation de clef (index primaire)
2. violation de contrainte d'index secondaire unique
3. violation de contrainte de données (colonne not null)
4. violation d'intégrité référentielle
5. violation de contrainte de contrôle de validité (min, max, ...)

VALEURS AMBIGÜES

Il arrive lors des insertions et des mise à jour que la valeur passée en argument soit ambiguë car son format ou son type ne peut être exprimé que par l'intermédiaire du jeu de caractère ordinaire. Comment donc savoir si la chaîne "AF12" est une chaîne de caractères ou un code hexadécimal représentant 4 octets soit 2 caractères ?

Pour lever cette ambiguïté, on doit utiliser une lettre de préfixage :

N	Unicode (National)
B	Bit
X	Hexadécimal

Exemple

```
INSERT INTO T_NOMECLATURE
(NMC_COMPOSANT, NMC_ADRESSE_MEMOIRE)
VALUES (B'0', X'AF12')
```

REFERENCES

http://webtic.free.fr/sql/
http://sqlpro.developpez.com/cours/sqlaz/ddl/