CENTRE DE RÉADAPTATION MULHOUSE
Rééducation et Formation Professionnelle

Langage de Manipulation de Données

SQL-SERVER

CONTENU

INTRODUCTION

Codd a posé les principes du futur langage SQL dès la création du modèle relationnel en 1970. Ce sont les 2 langages SQUARE et SEQUEL (rebaptisé SQL en 1976) qui sont à l'origine du langage SQL acuel.

L'ANSI parvient à harmoniser les développements en 1986, même si les constructeurs offrent toujours des extensions à la norme.

Le langage SQL s'appuie sur l'algèbre relationnelle, basée sur la théorie des ensembles.

Une bonne maîtrise de l'algèbre relationnelle permet de concevoir n'importe quelle requête aussi complexe soit elle avant de la mettre en œuvre à l'aide du langage SQL.

Parmi les opérations de l'algèbre relationnelle, on dispose **d'opérations classiques sur les ensembles** (union, intersection, différence, produit cartésien) et d'**opérations propres** (projection, sélection, jointure, division).

Le langage SQL intègre également des opérations de **calcul,** de **regroupement**, de **comptage** et de **tri**, non définies à l'origine par Codd mais très utiles.

LES OPERATIONS MISES EN ŒUVRE DANS SQL

LES OPERATIONS PROPRES A L'ALGEBRE RELATIONNELLE

LES OPERATIONS DE BASE

OPERATION PROJECTION

Formalisme : RelationResultat = **PROJECTION (**RelationInitiale, liste des attributs**)**

Exemple : CHAMPIGNONS

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

Rosé des prés	Sec	Sachet plastique

R1 = PROJECTION (CHAMPIGNONS, Espèce)

Espèce
Rosé des prés
Coulemelle

R2 = PROJECTION (CHAMPIGNONS, Espèce, Catégorie)

Espèce	**Catégorie**
Rosé 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***.

OPERATION SELECTION

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

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.

OPERATION JOINTURE (EQUIJOINTURE)

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

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

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

- Cet opérateur porte sur 2 relations qui doivent avoir au moins un attribut défini dans le même domaine (ensemble des valeurs permises pour un attribut).
- La condition de jointure peut porter sur l'égalité d'un ou de plusieurs attributs définis dans le même domaine (mais n'ayant pas forcément le même nom).
- Les n-uplets de la relation résultat sont formés par la concaténation des n-uplets des relations d'origine qui vérifient la condition de jointure.

OPERATION 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"

- 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).

LES OPERATIONS ENSEMBLISTES

OPERATION 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

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.

OPERATION INTERSECTION

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

Exemple :

E1 : Enseignants élus au CA E2 : Enseignants représentants syndicaux

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

n°enseignant	**nom_enseignant**
1	DUPONT
4	MARTIN
6	MICHEL

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.

OPERATION DIFFERENCE

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

Exemple :

E1 : Enseignants élus au CA E2 : Enseignants représentants syndicaux

n° enseignant	**nom_enseignant**
1	DUPONT
3	DURAND
4	MARTIN

n°enseignant	**nom_enseignant**
1	DUPONT
4	MARTIN
6	MICHEL

5	BERTRAND

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

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)

OPERATION PRODUIT CARTESIEN

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

Exemple :

Etudiants

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

Epreuves

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

Examen = PRODUIT (Etudiants, Epreuves)

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.

OPERATIONS COMPLEMENTAIRES

OPERATION TRI

R = **TRI(**R0, att1 ascendant, att2 descendant, ...**)**

Exemple :

CHAMPIGNONS

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

R1 = TRI (CHAMPIGNONS, Espèce descendant, Catégorie ascendant, Conditionnement ascendant)

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.

ATTRIBUTS CALCULES ET RENOMMES

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)

Exemple

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

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

OPERATION CALCULER

R=**CALCULER(**R0, fonction1, fonction2, ...**)** ou N=**CALCULER(**R0, fonction**)**

Exemple

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 :

R1=CALCULER(LIGNE_COMMANDE, 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.

OPERATION REGROUPER_ET_CALCULER

R=REGROUPER_ET_CALCULER(R0, att1, att2, ..., fonction1, fonction2, ...**)**

Exemple

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 : 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 FONCTIONS

Les fonctions sont utilisées dans les opérateurs CALCULER et REGROUPER_ET_CALCULER..

Les fonctions de calcul

Les fonctions de calculs 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

La fonction de comptage : Comptage()

La fonction de comptage donne le nombre de n-uplets d'un ou de plusieurs groupes de n-uplets

Les fonctions sont utilisées avec les opérateurs de calcul et de regroupement.

LES REQUETES D'ACCES AUX DONNEES (SELECT)

PRINCIPE D'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*)

Remarque : les clés primaires sont soulignées et les clés étrangères sont marquées par *

On désire obtenir le code et le nom des clients ayant commandé le 10/06/97 :

R1=SELECTION(COMMANDE, Date=10/06/97)

R2=JOINTURE(R1, CLIENT, R1.CodeClient=CLIENT.CodeClient)

R3=PROJECTION(R2, CodeClient, NomClient)

REQUETES SIMPLES

L'opération la plus importante d'un SGBD consiste à rechercher des données au moyen de critères de sélection plus ou moins complexes : c'est le rôle de SELECT qui va sélectionner des données (ou des tuples) parmi toutes les autres. Toute requête SELECT comporte au moins deux indications complémentaires appelées **CLAUSES** : nom(s) de la(les) table(s) où rechercher les données : clause FROM et le(s) attribut(s) à extraire (ou visualiser) : clause SELECT.

REALISATION D'UNE PROJECTION

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

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

La clause DISTINCT permet d'éliminer les doublons.

Exemples :

SELECT Espèce, Catégorie FROM Champignons ;

SELECT DISTINCT Espèce FROM Champignons ;

La visualisation globale d'une table , tous les attributs, sera réalisée très simplement par :

```
SELECT *
FROM table;
```

Les attributs sont affichés dans l'ordre de leur définition.

Remarques :

1. Dans une clause FROM il est possible d'associer au nom d'une table un alias (ou pseudonyme) qui pourra être utilisé en lieu et place du nom de la table et qui autorisera une auto-jointure (voir plus loin). Pour ce faire il faut placer l'alias après le nom de la table dans la clause FROM, les deux étant séparés par un espace.
2. Dans la plupart des SGBD, une requête SQL se termine par le caractère délimiteur ';' ce qui permet d'écrire la requête sur plusieurs lignes consécutives.

REALISATION D'UNE SELECTION

La clause WHERE permet de réaliser une sélection de tuples par une condition à remplir par la valeur d'un ou de plusieurs attributs. Le résultat de la sélection est une table sous ensemble de la(les) table(s) de départ.

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

CRITERES DE SELECTION DE TUPLES.

Comparaison à une valeur.

La condition de sélection exprimée derrière la clause WHERE peut être spécifiée à l'aide des opérateurs de comparaison : =, >, <, <=, >=, <>

Exemple :

SELECT * FROM Champignons WHERE Catégorie="Sec" ;

Comparaison à une plage de valeurs.

L'opérateur BETWEEN permet d'indiquer une plage de valeurs de bornes comprises à laquelle doit appartenir l'attribut des tuples à sélectionner ; il est couplé au mot clé AND qui sépare les 2 valeurs extrêmes de la plage. L'opérateur BETWEEN peut être associé à NOT pour former l'opérateur NOT BETWEEN.

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

```
SELECT *
FROM ETUDIANT
WHERE Age BETWEEN 19 AND 23 ;
```

Comparaison à une liste de valeurs.

L'opérateur IN permet de donner entre parenthèses la liste exhaustive des valeurs données explicitement ou fournies en tant que résultat d'une requête imbriquée à laquelle doit appartenir l'attribut des tuples à sélectionner. L'opérateur IN peut être associé à NOT pour former l'opérateur NOT IN.

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

Comparaison par rapport à un filtre.

La recherche de chaînes de caractères peut être rendue très performante au moyen de l'opérateur LIKE (associable avec NOT) qui ne spécifie plus rigoureusement (comme pour '=' et 'IN') la valeur de l'attribut mais seule(s) une(des) portions de la chaîne au moyen des deux caractères spéciaux :

1. '%' qui spécifie n'importe quelle chaîne de zéro ou plusieurs caractères (* sous Access),
2. '_' qui représente un et un seul caractère quelconque (? Sous Access).

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

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

Conditions multiples

L'expression logique pourra faire appel aux opérateurs logiques classiques NOT (négation), AND (ET logique) et OR (OU logique).

La priorité des opérateurs logiques S.Q.L. est la suivante (dans l'ordre décroissant): comparaisons, NOT, AND, OR. Dans tous les cas les parenthèses permettent de regrouper des expressions et d'en modifier l'interprétation.

Test d'indétermination d'une valeur.

Toute sélection spécifiant des valeurs d'attributs ne fournira jamais les tuples dont la valeur de cet attribut est non initialisée. Si l'on souhaite obtenir ces tuples malgré tout, il faut utiliser l'opérateur IS NULL (ou son inverse IS NOT NULL) dans la clause WHERE.

TRI DES RESULTATS.

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 CALCULES

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

Les opérateurs arithmétiques de base peuvent être utilisés pour calculer des attributs de la table résultat.

ATTRIBUTS RENOMMES

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

REQUETES SUR LES GROUPES

LES EXPRESSIONS ET LES FONCTIONS.

Les 4 opérateurs arithmétiques de base peuvent être utilisés dans les clauses où il est possible d'utiliser une valeur d'attribut. Suivant les systèmes la gamme des opérateurs et des fonctions utilisables peut être très évoluée : elle comporte toujours des fonctions de traitement de chaînes de caractères, des opérateurs sur les dates, etc…

Il existe également un certain nombre de fonctions, dites **fonctions de groupe** effectuant un calcul sur l'ensemble des valeurs d'un attribut d'un groupe de tuples. Les fonctions de groupe les plus couramment utilisées sont :

- **COUNT(*)**, afin de compter les occurrences,
- **COUNT(**attribut**)**, afin de compter les occurrences pour un attribut
- **COUNT(DISTINCT** attribut**)**, afin de compter les occurrences distinctes pour un attribut,
- **SUM(**attribut**)**, pour sommer les valeurs de l'attribut (de type numérique),
- **AVG(**attribut**)**, pour obtenir la moyenne (Average) des valeurs de l'attribut,
- **MAX(**attribut**)**, pour obtenir la valeur maxi de l'attribut.
- **MIN(**attribut**)**, pour obtenir la valeur mini de l'attribut.

Les valeurs indéterminées ne sont pas prises en compte.

SELECT fonction1(attribut1), fonction2(attribut2), ...
FROM table **;**

Exemple :

```
SELECT SUM(Quantité*PuHt), SUM(Quantité), AVG(PuHt)
FROM LIGNE_COMMANDE;
```

OPERATION DE REGROUPEMENT.

Clause GROUP BY

Un groupe est un sous-ensemble des tuples d'une table tel que la valeur d'un attribut ou d'un groupe d'attributs y reste constante ; un groupe est spécifié au moyen de la clause **GROUP BY** suivi du nom de l'attribut ou des attributs à l'origine du groupement. En l'absence de cette clause tous les tuples sélectionnés forment le groupe.

Les fonctions de groupe sont évaluées pour chacun des groupes ce qui permet de réarranger les résultats en fonction de l'attribut commun.

SELECT attribut1, attribut2, ..., fonction1(attribut3), fonction2(attribut4), ...
FROM table
GROUP BY attribut1, attribut2, ... **;**

Exemple :

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

- Tous les attributs placés derrière la clause GROUP BY doivent être présents derrière la clause SELECT. La proposition inverse n'est pas vraie.
- La clause GROUP BY est obligatoire dès lors qu'il y a à la fois des attributs et des fonctions de calcul derrière la clause SELECT.

Il est possible de trier des lignes issues d'un regroupement.

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

Clause HAVING

Cette clause est l'équivalent du WHERE mais elle se rapporte aux groupes. Elle porte, en général, sur la valeur d'une fonction de groupe. Seuls les groupes répondant au critère spécifié par HAVING feront partie du résultat.

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 est interdit 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).

Il est possible de combiner une clause WHERE qui s'applique à tous les tuples sélectionnés, et une clause HAVING qui concerne les groupes.

```
SELECT N°BonCommande, SUM(Quantité*PuHt)
WHERE DateCommande > 01/01/2000
FROM LIGNE_COMMANDE
```

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

Le principe de fonctionnement du moteur est le suivant :

1. sélection par la clause WHERE,
2. regroupement par la clause GROUP BY,
3. sélection des groupes par HAVING,
4. fourniture des résultats selon la clause SELECT avec une ligne par groupe.

Dans l'étape 4 seuls des attributs caractéristiques du groupe ou des fonctions de groupe peuvent être utilisés.

REQUETES IMBRIQUEES.

Puisqu'une requête SELECT fournit en résultat une table (satisfaisant les clauses associées) il est possible d'utiliser cette table pour une nouvelle requête par imbrication directe : l'évaluation du résultat se faisant depuis la requête de niveau le plus interne en remontant vers la requête de niveau supérieur, les requêtes de niveaux inférieur étant simplement écrites entre parenthèses. A ce stade deux cas peuvent se présenter suivant que la requête interne retourne **une** ou **plusieurs valeurs**.

La requête interne retourne un seul résultat:

Le résultat peut servir de comparaison dans une clause WHERE ce qui permet de spécifier une clause dont le critère dépend de la donnée associée à un tuple de la même table ou d'une autre table.

Exemple :

Nom des pilotes gagnant plus que la moyenne

```
SELECT nompil
FROM pilote
WHERE sal >
(SELECT AVG(sal) FROM Pilote);
```

Remarques :

- On notera que ce résultat pourrait être obtenu par à une auto jointure (étudiée plus loin) mais la lisibilité en serait moins bonne.
- Il est tout à fait possible d'utiliser plusieurs requêtes imbriquées dans la clause WHERE du moment que chacune ne retourne qu'un seul résultat.

La requête interne retourne plusieurs valeurs:

Elle ne pourra être imbriquée que dans une clause WHERE.

En fonction des conditions que l'on veut exprimer au niveau des valeurs renvoyées par la requête, on peut utiliser les prédicats IN, ANY, ALL ou EXISTS.

Prédicat	Utilisation
IN	L'opérateur de comparaison est l'égalité et l'opération logique entre les valeurs est OU.
ANY	Permet de vérifier si *au moins une valeur* de la liste satisfait la condition.
ALL	Permet de vérifier si la condition est réalisée pour *toutes les valeurs* de la liste.
EXISTS	Si la sous-interrogation renvoie un résultat, la valeur retournée est Vrai sinon la valeur Faux est retournée.

Exemple 1 :

- Afficher les notes de l'étudiant numéro 1 qui sont égales aux notes de l'étudiant numéro 2.

SELECT Note
FROM NOTES
WHERE Numetu=1 **AND** Note **IN**
(**SELECT** Note **FROM** NOTES **WHERE** Numetu=2);

Exemple 2 :

Afficher les notes de l'étudiant numéro 1 qui sont supérieures aux notes de l'étudiant numéro 2.

SELECT Note
FROM NOTES
WHERE Numetu=1 **AND** Note > **ANY**
(**SELECT** Note **FROM** NOTES **WHERE** Numetu=2);

Exemple 3 :

- Afficher les notes de l'étudiant numéro 1 qui sont inférieures à toutes les notes de l'étudiant numéro 2.

SELECT Note
FROM NOTES
WHERE Numetu=1 **AND** Note < **ALL**
(**SELECT** Note **FROM** NOTES **WHERE** Numetu=2);

Exemple 4 :

- Afficher toutes les informations sur les étudiants qui n'ont aucune note.

Cet exemple utilise une requête corrélée

```
SELECT *
FROM ETUDIANT E
WHERE NOT EXISTS
(SELECT Numetu FROM NOTES WHERE Numetu=E.Numetu);
```

Requêtes dans une projection

Exemple:

- Afficher le pourcentage du nombre de notes de chaque étudiant
-

```
SELECT Numetu, count(*)*100/(select count(*) from NOTES) as '% nb notes'
FROM NOTES
GROUP BY Numetu
```

Réalisation d'une 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. En algèbre relationnelle une autre solution que la division pourrait être :

```
N=CALCULER(EPREUVE, Comptage())
R1=REGROUPER_ET_CALCULER(PARTICIPER, Athlète, Nb:Comptage())
R2=SELECTION(R1, Nb=N)
R3=PROJECTION(R2, Athlète)
```

et en SQL :

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

Remarque : L'imbrication de requête peut tout aussi bien être utilisée dans une clause HAVING.

REQUETES MULTI-TABLES.

Etant donné que dans une BD il existe toujours plusieurs tables il faut pouvoir accéder à des données au travers éventuellement de plusieurs tables : ce sont les opérations ensemblistes d'une part (Union, Intersection, Différence) et l'opération fondamentale de l'algèbre relationnelle : la jointure ou jonction *(join)*.

JOINTURE DE TABLES.

La jointure utilise la notion de clé étrangère. On rappelle qu'une clé étrangère est un attribut qui fait référence à une autre table où il est clé primaire. Le moteur (ou noyau) du SGBDR produit la jointure en réalisant **le produit cartésien** des tables concernées. Il existe en fait plusieurs types de jointures :

EQUIJOINTURE.

L'**équijointure** consiste en une jointure par égalité de valeurs d'attributs de tables différentes : il suffira de préciser la liste des noms des tables concernées dans la clause SELECT et la liste de(s) égalité(s) d'attributs à satisfaire dans la clause WHERE.

SELECT * FROM table1**,** table2
WHERE table1.attribut1=table2.attribut1**;**

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 ;

Autre syntaxe:

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

En SQL, il est possible d'enchaîner plusieurs jointures dans la même instruction SELECT :

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

AUTO-JOINTURE.

Au moyen d'un **alias** (au nom de la table) il est possible de générer une requête avec un critère portant sur la valeur d'un attribut par rapport à la valeur de ce même attribut dans un autre tuple de la même table. L'alias est le seul moyen d'obtenir le produit cartésien d'une table par elle même, c'est à dire une auto-jointure.

Exemple :

Nom des pilotes gagnant le même salaire que dupont:

```
SELECT nompil FROM Pilote alias, Pilote autres
WHERE alias.nompil="Durant" AND alias.salaire = autres.salaire
        AND autres.nompil <> "Durant";
```

JOINTURE EXTERNE (OUTER JOIN).

Une jointure externe permet de fournir en résultat une liste de tuples (résultant d'une jointure avec une autre table) mais en incluant également les tuples pour lesquels la condition de jointure n'est pas réalisée en l'absence d'un des champs utilisé dans la jointure, tuples qui ne seraient pas apparus sinon.

SELECT * FROM table1 [LEFT/RIGHT] JOIN table2 ON table1.attribut1 = table2.attribut2**;**

Utilisez une opération LEFT JOIN pour créer une jointure externe gauche. Avec deux tables, les jointures externes gauches comprennent tous les enregistrements de la première table (celle de gauche) même s'il n'existe aucune valeur correspondante aux enregistrements dans la deuxième table (celle de droite).

Utilisez une opération RIGHT JOIN pour créer une jointure externe droite. Avec deux tables, les jointures externes droites comprennent tous les enregistrements de la deuxième table (celle de droite) même s'il n'existe aucune valeur correspondante aux enregistrements dans la première table (celle de gauche).

OPERATEURS ENSEMBLISTES

UNION

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 ;

INTERSECTION

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

Ou

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

Exemple :

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

Ou

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

DIFFERENCE

SELECT attribut1**,** attribut2**,** ... **FROM** table1
MINUS
SELECT attribut1**,** attribut2**,** ... **FROM** table2 **;**

ou

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

Exemple :

SELECT n°enseignant, NomEnseignant FROM E1
MINUS
SELECT n°enseignant, NomEnseignant FROM E2 ;

ou

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

QUELQUES PRINCIPES POUR L'ECRITURE D'UNE REQUETE SELECT.

	En SQL
☞ Déterminer les tables utilisées En déduire les jointures éventuelles	Dans la clause FROM Critères de jointures dans la clause WHERE
☞ Réaliser la projection: lister les attributs	Dans la clause SELECT
☞ Réaliser les opérations REGROUPER-CALCULER	Attributs de regroupement et fonctions dans la clause SELECT Clause GROUP BY reprenant tous les attributs cités dans le SELECT, sauf les fonctions de groupe
☞ Réaliser les sélections.	Présenter les critères de sélection dans la clause HAVING sur les groupes et dans la clause WHERE sur des attributs individuels (présents au niveau des tuples)
☞ Pour utiliser, dans une sélection individuelle, une fonction sur les groupes ☞ Pour utiliser une valeur d'attribut d'un autre tuple de la même table	Requête imbriquée
☞ Pour "rapprocher" des relations comportant les mêmes attributs	Opérateurs ensemblistes: UNION, INTERSECTION et DIFFERENCE
☞ Pour préciser l'ordre de rangement des tuples	Clause ORDER BY

MISE A JOUR DES DONNEES

AJOUT DE TUPLES

La commande INSERT est utilisée pour insérer de nouveaux tuples dans une table. Son utilisation nécessite de spécifier : le nom de la table, le nom des attributs à initialiser et les valeurs à affecter à chacun d'eux (ces valeurs devant être encadrées par des parenthèses). En l'absence de spécification du nom des attributs ils sont tous concernés et doivent être précisés selon l'ordre de déclaration utilisé lors de la création de la table.

Exemples :

```
INSERT INTO défauts (def_num, def_lib, def_freq)
VALUES (1, 'niveau d''huile MIN', 0) ;

INSERT INTO défauts
VALUES (1, 'niveau d''huile MIN', 0) ;
```

Tout attribut que l'on ne souhaite pas initialiser (dans la mesure où il n'a pas été déclaré NOT NULL !) doit être, soit initialisé explicitement à NULL, soit non spécifié dans la liste des attibuts à renseigner.

Exemple :

```
INSERT INTO défauts (def_num, def_lib)
VALUES (2, 'niveau d'huile MAX' );
```

ou

```
INSERT INTO défauts VALUES (2, 'niveau d'huile MAX', NULL) ;
```

Il est également possible d'insérer des valeurs issues d'une requête de sélection

La clause **INSERT INTO** précise le nom de la table à remplir suivie soit d'une clause **VALUES** spécifiant les valeurs des attributs du tuple à insérer, soit d'une clause SELECT dont le résultat constitue l'ensemble des valeurs à insérer.

Exemple :

```
INSERT INTO défauts_fréquents
SELECT def_num, def_lib, def_freq
FROM défauts
WHERE défauts.def_freq > 50;
```

Sur certains SGBDR il est possible de créer et d'insérer dans une table en une seule opération au moyen de **CREATE TABLE** associé à la clause **AS**.

MODIFICATION DE TUPLES

La commande **UPDATE** permet la modification de tuple(s) d'une table spécifiée au moyen des clauses **WHERE** (quels tuples à modifier ?) et **SET** (quelle nouvelle valeur ?). La nouvelle valeur peut comporter des constantes, des noms d'attributs et même des requêtes imbriquées.

Exemples :

```
UPDATE défauts
SET def_freq = def_freq + 1
WHERE def_num = 19;

UPDATE défauts
SET def_freq =
(SELECT MAX (def_freq)
FROM défauts
WHERE def_num < 10
GROUP BY def_machine);
```

SUPPRESSION DE TUPLES

Pour la destruction de tuple(s) la commande **DELETE** demande la spécification d'une clause FROM pour le nom de la table et d'une clause WHERE précisant le(s) tuple(s) à détruire. La destruction de tous les tuples de la table ne demande pas de clause WHERE ; d'autre part la clause WHERE peut contenir une requête imbriquée.

Exemple :

```
DELETE FROM défauts
WHERE def_machine =
      (SELECT def_machine
      FROM défauts
      WHERE def_lib LIKE '%HUILE%' );
```

--- FIN DU DOCUMENT ---