Institut Supérieur d'Informatique et de Gestion de Kairouan

Cours 6 : Le langage SQL (Le DML)

Riadh ZAAFRANI
2ème année Licence Fondamentale : Sciences de l'informatique

Décembre 2009

Le langage SQL

- Si les requêtes courantes (projections, restrictions et jointures) s'expriment facilement en QBE, il faut bien reconnaître que dès que la requête se complique, l'expression en QBE devient très délicate.
- D'autres langages prennent alors toute leur puissance ; c'est le cas de **SQL**.
- SQL est un langage de définition et de manipulation de bases de données relationnelles. Son nom est une abréviation de « Structured Query Language » (langage d'interrogation structuré).
- SQL est un standard qui a été normalisé par l'organisme ANSI. Il existe pour chaque produit SQL réalisé des différences ou des compléments par rapport à la norme.

Le langage SQL

- Historiquement, après la découverte du modèle relationnel par E.F. Codd en 1970, plusieurs langages relationnels sont apparus dans les années suivantes dont SEQUEL d'IBM, puis SEQUEL/2 pour le System/R d'IBM en 1977, langage qui donna naissance à SQL.
- L'année 1981 a vu la sortie du premier SGBD relationnel implémentant SQL, le système **Oracle**. Le langage a ensuite été normalisé en 1986 pour donner SQL/86 puis légèrement modifié en 1989 pour donner SQL/89 la version la plus utilisée actuellement.
- En 1992 de nombreuses améliorations ont été apportées à la norme pour donner SQL/92 que l'on nomme généralement SQL2, beaucoup plus verbeuse que la précédente.

Le langage SQL

- SQL contient un langage de définition de données, le **DDL** (Data Definition Language), permettant de créer, modifier ou supprimer les définitions des tables de la base par l'intermédiaire des ordres **Create**, **Drop**, et **Alter**.
- Il contient aussi un langage de manipulation de données, le **DML** (Data Manipulation Language), par l'intermédiaire des ordres **Select**, **Insert**, **Update**, **Delete**.
- Il contient enfin un langage de gestion des protections d'accès aux tables en environnement multi-utilisateurs par l'intermédiaire des ordres **Grant**, **Revoke**, le **DCL** (Data Control Language).

Le langage SQL

- Une requête SQL peut être utilisée de manière interactive ou incluse dans un programme d'application (quel que soit le langage).
- Toutes les instructions SQL se terminent par un point-virgule (;).
- Un commentaire peut être introduit dans un ordre SQL par les signes /* et */ ou par le caractère % qui traite toute la fin de ligne comme commentaire.

La manipulation des données : le DML

- Tout d'abord, pourquoi commencer par le DML ? La réponse est simple. De plus en plus d'outils comme MS-ACCESS permettent de créer des tables à la souris. Pour de nombreux utilisateurs, le DML est donc le premier contact qu'ils ont avec SQL, c'est pourquoi nous commencerons par lui. Il est néanmoins évident que l'ordre chronologique d'utilisation d'une base de données nécessite l'usage du DDL avant le DML.
- On parlera tout d'abord de l'ordre **SELECT** qui est sans doute le plus utilisé de tous les ordres SQL, puis des ordres **INSERT**, **UPDATE** et **DELETE**.

L'obtention des données

- L'obtention des données se fait exclusivement par l'ordre SELECT. La syntaxe minimale de cet ordre est :
- **SELECT <liste des noms de colonnes>**
  FROM <liste des noms de tables> ;
- Elle peut être enrichie, de très nombreuses clauses permettant notamment d'exprimer les projections, les restrictions, les jointures, les tris etc ...
  - **FROM**
  - **WHERE**
  - **GROUP BY**
  - **HAVING**
  - **ORDER BY**

Expression des projections

- La projection d'une table en vue de l'obtention d'un ensemble de colonnes de cette table se fait simplement en spécifiant les noms des colonnes désirées.
- Si l'on souhaite obtenir toutes les colonnes d'une table le caractère **'*'** peut être utilisé avantageusement à la place de la liste des noms de colonnes.

A Vous

- **R1 : lister la table fournisseurs ?**

- **SELECT ***
  FROM Fournisseurs;

A Vous

- **R2 : lister les numéros et noms des fournisseurs ?**

- **SELECT numFrs, nom**
  FROM fournisseurs;

A Vous

- SQL n'élimine pas les doubles à moins que ça ne soit explicitement précisé par le mot clé **DISTINCT**.
- **R3 : Lister les désignations différentes de produits ?**

- **SELECT DISTINCT design**
  FROM produits;

design
armoire
bureau
Caisse
chaise
classeur
ecrou
fauteuil

Expression des restrictions

- La restriction s'exprime à l'aide de la clause **WHERE** que l'on ajoute à la requête juste après la liste des tables utilisées.
- **R4 : lister les données sur les produits dont le poids est supérieur à 15 ?**

- **SELECT ***
  FROM produits
  WHERE poids>15;

Expression des restrictions

- **R5 : lister les désignations de produits différentes dont le poids est supérieur à 15 ?**

```
SELECT DISTINCT design
FROM produits
WHERE poids>15;
```


Expression des restrictions

- Il est possible d'exprimer des qualifications complexes à l'aide des comparateurs arithmétiques **>,>=,<,<=,=,<>** et des opérateurs logiques **AND, OR, NOT**.
- **R6 : lister les produits dont le poids est compris entre 15 et 40 ?**

```
SELECT *
FROM produits
WHERE poids>=15 And poids<=40;
```


Expression des restrictions

- D'autres prédicats peuvent encore être utilisés. Ils servent à définir des ensembles ou des valeurs approximatives. Ce sont les prédicats **IN**,**BETWEEN**,**LIKE** qui peuvent tous trois être préfixés par **NOT** pour exprimer la négation.
- **R7 : lister les fournisseurs habitant Tunis, Sfax ou Kairouan ?**

```
SELECT *
FROM fournisseurs
WHERE ville IN ('Tunis', 'Sfax','Kairouan');
```


Expression des restrictions

- **R8 : lister les produits dont le poids n'est pas compris entre 15 et 35 ?**

```
SELECT *
FROM produits
WHERE poids Not Between 15 And 35;
```


Expression des restrictions

- Le caractère de soulignement **'_'** ou **'?'** (Sous Access) remplace n'importe quel caractère tandis que le pourcentage **'%'** ou **'*'** (Sous Access) remplace n'importe quelle séquence de n caractères (n pouvant être égal à 0)
- **R9 : lister les fournisseurs dont le nom ne commence pas par 'K'?**

```
SELECT *
FROM fournisseurs
WHERE nom Not Like 'K%';
```


Expression des restrictions

- **R10 : lister les fournisseurs dont le 1er et le 4ème caractère est un 'K'?**

```
SELECT *
FROM fournisseurs
WHERE nom Like 'K_ _K%';
```

Expression des restrictions

- Le test de valeurs manquantes est aussi possible. Une valeur manquante est représentée par une valeur spéciale notée **NULL**. Le test d'une valeur NULL ne peut pas se faire avec l'égalité classique mais doit se faire obligatoirement avec les prédicats **IS NULL** ou **IS NOT NULL**.
- **R11 : lister les fournisseurs dont la ville est renseignée ?**

- **SELECT ***
  FROM fournisseurs
  WHERE ville IS NOT NULL;

Tri et présentation des résultats

- SQL permet de trier les tuples obtenus par la requête selon différents critères grâce a la clause **order by**. Les mots clés **asc** et **desc** permettent de préciser si le tri est croissant ou décroissant.
- **R12 : Afficher les produits rouges ou verts ou bleus triés dans l'ordre croissant des noms et dans l'ordre décroissant des couleurs ?**

- **SELECT ***
  FROM produits
  WHERE couleur In ('rouge', 'vert', 'bleu')
  ORDER BY design, couleur DESC;

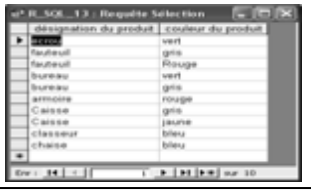
Tri et présentation des résultats

- Par défaut, les noms des colonnes de la table résultat de la requête sont les noms des colonnes spécifiées dans la liste des colonnes du SELECT.
- Il est néanmoins possible de changer le nom de la colonne à l'affichage en accolant au nom de colonne choisi le prédicat **AS** suivi du nouveau nom de colonne devant apparaître à l'affichage.

Tri et présentation des résultats

- **R13 : afficher les produits et leurs couleurs avec des noms de colonnes compréhensibles ?**

- **SELECT design AS [désignation du produit], couleur AS [couleur du produit]**
  FROM produits;



Expression des jointures

- Les jointures se font en spécifiant les tables sur lesquelles la jointure sera faite dans la liste des tables utilisées et en spécifiant la qualification de jointure dans la clause WHERE.
- On rappelle qu'une jointure sans qualification est un produit cartésien et qu'une jointure avec égalité est une équi-jointure.
- **R14 : lister le produit cartésien fournisseur * produits ?**

- **SELECT ***
  FROM fournisseurs, produits;

Expression des jointures

- Lorsque nous utilisions une seule table, il n'y a pas d'ambiguïté de nom sur les colonnes utilisées. Dès que deux tables sont utilisées, des ambiguïtés peuvent apparaître. Dans ce cas il faut préfixer le nom de colonne par le nom de la table concernée suivi d'un point.
- **R15 : lister les noms des fournisseurs avec les numéros de produits commandés ainsi que la quantité commandée ?**

- **SELECT nom, numProd, qté**
  FROM fournisseurs, commandes
  WHERE fournisseurs.numFrs=commandes.numFrs;

Expression des jointures

- Si les noms de tables qui servent de préfixes deviennent pénibles à taper, SQL offre de plus une possibilité de synonymes dans la clause FROM.
- Il suffit pour cela de préciser le synonyme juste après le nom de la table à l'aide du mot clé AS (facultatif) comme pour les synonymes de noms de colonne.

Expression des jointures

- **R16 : utilisez des synonymes de noms de tables pour lister les noms des fournisseurs avec les numéros de produits commandés ainsi que la quantité commandée?**

```
SELECT f.nom, c.numprod, c.qté
FROM fournisseurs AS f, commandes AS c
WHERE f.numFrs=c.numFrs;
```


Expression des jointures

- La requête précédente ne faisait qu'une seule jointure, mais bien sûr, plusieurs jointures peuvent être exécutées en cascade.
- **R17 : lister les couples (nom de fournisseur, nom de produit) en commande ?**

```
SELECT f.nom AS fournisseur, p.design AS produit
FROM fournisseurs AS f, produits AS p, commandes AS c
WHERE f.numFrs=c.numFrs
And p.numProd=c.numProd;
```


Expression des jointures

- Rien n'empêche bien sûr de faire une jointure d'une table sur elle même. Dans ce cas on parle alors d'auto-jointure.
- **R18 : Lister les couples de références de fournisseurs situés dans la même ville ?**

```
SELECT f1.numFrs, f2.numFrs
FROM fournisseurs AS f1, fournisseurs AS f2
WHERE f1.numFrs<f2.numFrs AND f1.ville=f2.ville;
```


Les apports de SQL 2

- SQL 2 (parfois appelé SQL/92) a apporté quelques nouvelles possibilités quant a l'expression des jointures. Avec SQL 2 il est possible de préciser les attributs de jointure dans la clause FROM du SELECT.
- Par exemple **FROM fournisseurs f JOIN commandes c ON f.numFrs=c.numFrs** permet d'exprimer une jointure entre les tables fournisseur et commandes sur l'attribut numFrs.
- Outre la jointure classique, SQL 2 offre aussi la possibilité de définir des jointures externes.

Les apports de SQL 2

- Une jointure interne ne fournit dans la table résultante que les tuples qui satisfont au critère de jonction.
- Dans une jointure externe l'une des tables utilisées dans la jointure est considérée comme directrice et aura tous ses enregistrements dans la table résultante même s'ils n'ont pas de valeurs correspondantes avec l'autre table de la requête.
- En effet, il se peut parfois que l'on souhaite aussi conserver dans la table résultante des tuples qui ne peuvent pas être complétés par l'opération de jointure. Contrairement à la jointure interne, les jointures externes permettent de les conserver en complétant les colonnes manquantes par des valeurs NULL.

Les apports de SQL 2

- Par exemple la clause **FROM fournisseurs f LEFT JOIN commandes c ON f.numFrs=c.numFrs** permet d'exprimer une jointure entre les tables fournisseur et commandes sur l'attribut numFrs et conserver dans la table résultat les fournisseurs qui n'ont pas de commande en cours.
- Pour exprimer les variantes de jointures externes, il existe notamment dans SQL 2 les ordres **LEFT JOIN** pour conserver les tuples de la table de gauche, **RIGHT JOIN** pour conserver les tuples de la table de droite. L'ordre JOIN peut être prefixé par **INNER** pour la jointure interne classique.



Les apports de SQL 2

- **R19 : afficher tous les produits de moins de 20Kg avec les quantités en cours de commande si possible ?**

- **SELECT p.numProd, design, qté, poids**
  FROM produits AS p LEFT JOIN commandes AS c
  ON c.numProd=p.numProd
  WHERE poids<20;



Les apports de SQL 2

- **R20 : afficher les produits qui ne sont pas commandés ?**

- **SELECT p.numProd**
  FROM produits AS p LEFT JOIN commandes AS c
  ON c.numProd=p.numProd
  WHERE c.numProd is NULL;



Les expressions de manipulation de données

- SQL permet bien sûr d'effectuer des calculs arithmétiques à l'affichage des colonnes. Il suffit pour cela de placer l'expression à calculer comme champ d'affichage du **SELECT**. Ces valeurs calculées peuvent être testées dans les clauses **HAVING** ou **WHERE**.
- **R21 : Calculer les prix des produits avec une TVA a 20,6% ?**

- **SELECT numProd, design, prix*1.206 AS prixTTC**
  FROM produits;



Les expressions de manipulation de données

- **R22 : Calculer le volume financier commandé pour les commandes de moins de 10 articles ?**

- **SELECT commandes.numCde, qté*prix AS volume, qté**
  FROM produits INNER JOIN commandes ON
  produits.numProd=commandes.numProd
  WHERE commandes.qté< 10;
- D'autres opérations sur chaque valeur sélectionnée peuvent être effectuées notamment sur les dates (YEAR, MONTH, DAY) et les chaînes de caractères (SUBSTRING, UPPER, LOWER, CHARACTER_LENGTH).



Les fonctions statistiques

- Les fonctions précédentes permettent de manipuler les données d'un tuple à la fois.
- Mais l'utilisateur a souvent besoin d'autres fonctions que celles présentées précédemment. Notamment, il lui faut parfois pouvoir manipuler des **fonctions de regroupement « inter tuples ».**
- SQL offre la possibilité de récupérer des données chiffrées sur les tables ou des parties de tables.



Les fonctions statistiques

- Il est par exemple possible d'obtenir le nombre de tuples répondant a un critère, la valeur moyenne d'une colonne, la valeur maximum ou minimum et la somme d'une colonne.
  - **AVG** **moyenne**
  - **COUNT** **nombre d'éléments**
  - **MAX** **maximum**
  - **MIN** **minimum**
  - **SUM** **somme**



Les fonctions statistiques

- **R23 : compter le nombre de commandes ?**
- **SELECT Count(*) AS [nombre de commandes]**
  FROM commandes;



Les fonctions statistiques

- **R24 : afficher la somme de toutes les quantités commandées ?**
- **SELECT Sum(qté)**
  FROM commandes;



Les fonctions statistiques

- **R25 : compter le nombre de livraisons du produit numéro 102 ?**
- **SELECT Count(*)**
  FROM commandes
  WHERE numProd=102;



Les fonctions statistiques

- **R26 : récupérer toutes les statistiques sur les quantités en commande ?**
- **SELECT COUNT(*), SUM(qté), MAX(qté), MIN(qté), AVG(qté)**
  FROM commandes;
- Il est important de noter que les opérations statistiques précédentes ne peuvent pas s'imbriquer. Il est par exemple interdit d'écrire quelque chose comme **AVG(SUM(montant)).** Ce type de calcul ne pourra s'obtenir qu'avec des sous-requêtes ou des vues comme nous le verrons plus tard.



Les regroupements

- Il est souvent intéressant de regrouper les données d'une table en sous-tables pour y faire des opérations par groupes. Par exemple compter les commandes par fournisseur. Ceci se fait avec la clause **GROUP BY** suivie de la colonne a partitionner.
- **R27 : afficher la somme des quantités commandées à chaque fournisseur ?**
- **SELECT numFrs, SUM(qté)**
  FROM commandes
  GROUP BY numFrs;



Les regroupements

- **R28 : lister le nombre de commandes par fournisseur ?**

- **SELECT numFrs, COUNT(*) AS Nbre**
  FROM commandes
  GROUP BY numFrs;
- Si dans un SELECT certaines colonnes sélectionnées utilisent des fonctions de calcul et d'autres non, alors toutes les colonnes sélectionnées qui n'utilisent pas de calcul doivent être spécifiées dans le **GROUP BY**.

Les regroupements

- La clause **HAVING** étroitement liée au GROUP BY permet d'exprimer une restriction sur les lignes de la table obtenue avec les fonctions de calcul.
- Les tests sur les colonnes simples se font dans la clause WHERE tandis que les tests sur les fonctions de regroupement se font dans la clause HAVING.
- **R29 : lister les fournisseurs qui ont moins de 3 commandes ?**

- **SELECT numFrs, count(*) FROM commandes**
  GROUP BY numFrs
  HAVING COUNT(*)<3;

Les regroupements

- Les fonctions arithmétiques et statistiques testées dans une clause HAVING n'ont pas à être forcément sélectionnées comme colonne résultat. Il est possible d'utiliser la fonction uniquement dans la clause HAVING sans qu'elle soit projetée.
- **R30 : Lister les références des fournisseurs qui fournissent plus d'un produit?**

- **SELECT numFrs**
  FROM commandes
  GROUP BY numFrs
  HAVING COUNT(*)>1;

Les regroupements

- **R31 : On ne s'intéresse ici qu'aux commandes dont le numéro est supérieur à 1004 et, pour ces commandes, on souhaite lister les noms des fournisseurs avec ses quantités maximales commandées, dont la moyenne est supérieure à 5. Le résultat doit être trié par ordredécroissant du nom de fournisseur.**

- **SELECT nom, max(qté)**
  FROM commandes AS c, fournisseurs AS f
  WHERE numCde>1004 And c.numFrs=f.numFrs
  GROUP BY nom
  HAVING AVG(qté)>5
  ORDER BY nom DESC;

Les sous-requêtes

- SQL donne la possibilité d'utiliser des **sous-requêtes** afin de décrire des requêtes complexes permettant d'effectuer des opérations dépendant d'autres requêtes .
- La sous-requête est toujours placée dans la clause WHERE ou HAVING en lieu et place d'une constante et doit renvoyer soit une **valeur unique** (qui est alors utilisée avec les opérateurs classiques du where comme n'importe quelle constante), soit une **colonne unique** (qui est alors utilisée avec les opérateurs **IN** et **EXISTS**, ou avec les comparateurs classiques suivis de **ALL** ou **ANY**).
- Attention : la sous-requête doit suivre obligatoirement l'opérateur de comparaison, le contraire est interdit.

Les sous-requêtes

- Les différentes formes de tests avec sous-requêtes sont donc les suivantes (dans les exemples suivants le comparateur **>** peut évidemment être remplacépar n'importe quel autre comparateur).
- **1) SELECT… WHERE numFrs > 125**
- Sélection classique, par comparaison avec une constante. Cette constante peut être remplacée par une sous-requête qui renvoie une seule valeur.
- **2) SELECT… WHERE numFrs > (SELECT numFrs from ...)**
- Sous-requête dans laquelle un champ est comparé au résultat de la requête La sous-requête doit renvoyer une seule valeur (une table d'une seule ligne et une seule colonne).

Les sous-requêtes

- **3) SELECT … WHERE numFrs IN (SELECT numFrs from ...)**
- Ici la sous-requête renvoie plusieurs valeurs (une table de plusieurs lignes et une seule colonne). Pour obtenir un succès, la valeur testée doit être présente dans le résultat de la sous-requête. La forme **NOT IN** est aussi autorisée.
- **4) SELECT … WHERE numFrs > ALL (SELECT numFrs from ...)**
- Sous-requête qui renvoie plusieurs valeurs. Pour obtenir un succès, la valeur testée doit être supérieure à toutes les valeurs obtenues par la sous-requête.

Les sous-requêtes

- **5) SELECT ... WHERE numFrs > ANY (SELECT numFrs FROM ...)**
- Sous requête qui renvoie plusieurs valeurs. Pour obtenir un succès, la valeur testée doit être supérieure à au moins une valeur obtenue par la sous-requête. La forme = ANY est équivalente à la forme IN
- **6) SELECT … WHERE EXISTS (SELECT .... WHERE numFrs = …**
- Sous-requête qui renvoie plusieurs valeurs. Pour obtenir un succès la sous-requête doit donner un succès. Ce type de sous-requête est dit "corrélée", car un champ de la requête principale est utilisé dans la sous-requête. La forme **NOT EXISTS** est aussi autorisée.

Les sous-requêtes

- **R32 : lister les fournisseurs qui habitent la même ville que le fournisseur 10 ?**

```
SELECT nom
FROM fournisseurs
WHERE ville = (SELECT ville
               FROM fournisseurs
               WHERE numFrs=10);
```


Les sous-requêtes

- **R33 : lister les différents fournisseurs qui ont commandé au moins un produit de couleur rouge ?**

```
SELECT DISTINCT NumFrs
FROM commandes
WHERE numProd IN (SELECT numProd
                  FROM produits
                  Where couleur='rouge');
```


Les sous-requêtes

- **R34 : lister les différents fournisseurs d'au moins un des produits fournis aussi par un fournisseur d'un produit rouge ?**

```
SELECT DISTINCT NumFrs
FROM commandes
WHERE numProd IN
      (SELECT numProd
       FROM commandes
       WHERE numFrs IN (SELECT numFrs
                        FROM commandes
                        WHERE numProd IN (SELECT numProd
                                          FROM produits
                                          Where couleur='rouge')));
```


Les sous-requêtes

- Pour résoudre certaines requêtes, il est parfois plus facile de traiter la forme inverse. Par exemple, pour obtenir les noms des fournisseurs ayant livré tous les produits, il est plus facile de rechercher les fournisseurs pour lesquels il n'existe aucun produit non livré.
- Cette sélection s'écrit avec une requête à deux sous requêtes corrélées qui travaillent sur 3 tables simultanément !

Les sous-requêtes

- **R35 : lister les noms des fournisseurs livrant tous les produits ?**

```
SELECT f.nom
FROM fournisseurs f
WHERE NOT EXISTS (SELECT *
      FROM produits p
            WHERE NOT EXISTS (SELECT *
                  FROM commandes c
                   WHERE f.numFrs=c.numFrs
                     AND c.numProd=p.numProd) ) ;
```


Les sous-requêtes

- Cette requête est dite "corrélative" car l'une des données du sous-select est comparée à une donnée du select principal.
- Ce type de requête prend évidemment beaucoup de temps de calcul puisque la sous-requête doit être réévaluée pour chaque tuple de la requête principale.

Les sous-requêtes

- Remarque Les prédicats **ALL** et **ANY** peuvent toujours être reformulés comme des tests d'existence. En effet, si l'on note $ le comparateur arithmétique utilise on voit immédiatement que :
- **... WHERE x $ ANY (SELECT y FROM t WHERE p)** est sémantiquement équivalent a **... WHERE EXISTS (SELECT * FROM t WHERE (p) AND x $ t.y)**
- De même, **... WHERE x $ ALL (SELECT y FROM t WHERE p)** est sémantiquement équivalent a **... WHERE NOT EXISTS (SELECT * FROM t WHERE (p) AND NOT (x $ t.y)**

Les sous-requêtes

- **R36 : Lister les références des fournisseurs livrant au moins un produit en quantité supérieure à chacun des produits livrés par le fournisseur 19 ? (sans EXISTS)**

```
SELECT DISTINCT numFrs
FROM commandes
WHERE qté > ALL (SELECT qté
                 FROM commandes
                 WHERE numFrs=19 );
```


Les sous-requêtes

- **R36 : Lister les références des fournisseurs livrant au moins un produit en quantité supérieure à chacun des produits livrés par le fournisseur 19 ? (Avec EXISTS)**

```
SELECT DISTINCT c1.numFrs
FROM commandes AS c1
WHERE NOT EXISTS
                ( SELECT *
                  FROM commandes c2
                  WHERE c2.numFrs = 19
                  AND c1.qté <= c2.qté);
```


Les opérations ensemblistes

- Comme dans l'algèbre relationnelle, les opérations ensemblistes sont au nombre de trois et s'intercalent entre deux SELECT.
- **UNION** : fournit la réunion des tuples des 2 select
- **INTERSECT**: fournit l'intersection des tuples des 2 select
- **EXCEPT** : fournit les tuples du premier select qui ne sont pas dans le second.
- Ces opérateurs ne sont valides qu'à la condition que les $i^{èmes}$ colonnes de chacune des tables aient exactement la même description (type et longueur).

Les opérations ensemblistes

- On notera néanmoins que **INTERSECT** et **EXCEPT** ne font pas partie de la norme SQL et qu'ils ne sont pas implémentés dans tous les SGBD.
- L'opérateur **UNION** supprime par défaut les doublons de l'opération. Si les doublons sont souhaités il faut utiliser la forme **UNION ALL**.



Les opérations ensemblistes

- **R37 : lister les numéros de produits dont le poids est supérieur à 20 ainsi que les produits commandés par le fournisseur 15 ?**

```
SELECT numProd FROM produits WHERE poids>20
  UNION
SELECT numProd FROM commandes WHERE numFrs=15;
```

```
SELECT DISTINCT P.NumProd, C.NumProd
FROM Produits AS P FULL JOIN Commandes AS C ON
P.NumProd=C.NumProd
WHERE P.Poids>20 Or C.NumFrs=15;
```



Les opérations ensemblistes

- **R37 Bis : lister les numéros de produits dont le poids est supérieur à 20 et qui ne sont pas commandés par le fournisseur 15 ?**

```
SELECT numProd FROM produits WHERE poids>20
  EXCEPT
SELECT numProd FROM commandes WHERE numFrs=15;
```

```
SELECT numProd FROM produits WHERE poids>20
  INTERSECT
SELECT numProd FROM commandes WHERE numFrs<>15;
```



La mise a jour d'informations

- Trois types de mise a jour sont nécessaires pour le contenu d'une table relationnelle.
- **1) L'insertion :** L'ordre **INSERT** permet d'ajouter des lignes dans une table. Dans sa forme la plus générale, SQL demande que les noms des colonnes soient explicitement citées. Les valeurs insérées sont alors en correspondance avec l'ordre dans lequel les colonnes sont citées.



La mise a jour d'informations

- Si l'ordre **INSERT** contient une clause **VALUES** alors une seule ligne est insérée dans la table ;
- si l'ordre **INSERT** contient une clause **SELECT** alors plusieurs lignes peuvent être simultanément insérées dans la table.
- L'insertion de tuples peut donc se faire soit en extension par la clause VALUE soit en intension par un ordre SELECT imbriqué.



La mise a jour d'informations

- **R38 : ajouter un nouveau produit (numéro = 8, désignation = "écrou",prix = 5, poids = 12, couleur = "vert") ?**

```
INSERT INTO produits
(numProd,design,prix,poids,couleur)
VALUES (8, 'écrou',5,12, 'vert') ;
```



La mise a jour d'informations

- **R39 : ajouter dans la table petites_commandes les numéros et quantités des produits commandés en moins de 5 exemplaires ?**

- **INSERT INTO petites_commandes(numprod,qté)**
  SELECT numprod,qté FROM commandes
  WHERE qté < 5 ;



La mise a jour d'informations

- Si une valeur est insérée dans toutes les colonnes de la table, l'énumération des colonnes est facultatif.
- La requête **R38** peut donc s'écrire de la manière suivante :
- **R38 (2ème méthode)**
- **INSERT INTO produits**
  VALUES (8, 'écrou', 5,12, 'vert') ;



La mise a jour d'informations

- SQL accepte que tous les noms de colonne de la table dans laquelle les tuples seront insérés ne soient pas précisés.
- Dans ce cas, les colonnes non précisées sont alors automatiquement remplies avec la valeur NULL.
- Il est néanmoins conseillé de toujours préciser toutes les colonnes quitte a préciser les valeurs NULL manuellement, afin d'éviter de fâcheuses erreurs de programmation.



La mise a jour d'informations

- **R40 : ajouter un nouveau produit dont on ne connaît pas le poids et la couleur (numéro=8,désignation="écrou",prix=5) ?**

- **INSERT INTO produits (numprod,design,prix)**
  VALUES (8,"écrou",5) ;
- ou aussi
- **INSERT INTO produits (numprod,design,prix,poids,couleur)**
  VALUES (8,"écrou",5,NULL,NULL) ;
- ou encore
- **INSERT INTO produits**
  VALUES (8,"écrou",5,NULL,NULL) ;



La mise a jour d'informations

- **R41 : ajouter dans la table braderie les numéros de produits dont le prix est entre 100D et 500D et les mettre à un prix de 100D ?**

- **INSERT INTO braderie (numProd, prix)**
  SELECT produits.numProd, 100
  FROM produits
  WHERE prix<=500 And prix>=100;



La mise a jour d'informations

- Une table citée dans le champ INTO de l'ordre INSERT ne peut pas être citée dans le champ FROM du sous-select de ce même INSERT.
- Il n'est donc pas possible d'insérer des éléments dans une table à partir d'une sous-sélection de cette même table.
- **Requête interdite : la duplication des éléments d'une table par un insert avec sous-select sur la même table.**
- **INSERT INTO produits**
  SELECT * FROM produits ;



La modification d'informations

- **2) Modification :** L'ordre **UPDATE** permet de modifier des lignes dans une table. L'expression caractérisant la modification à effectuer peut être une constante, une expression arithmétique ou le résultat d'un select imbriqué.
- **R42 : Augmenter le prix de tous les produits rouges de 5% ?**

```
UPDATE produits
SET prix = prix*1.05
WHERE couleur='rouge' ;
```



La suppression d'informations

- **3) Suppression :** L'ordre **DELETE** permet de supprimer des lignes dans une table selon une qualification fixée.
- **R43 : Supprimer les produits dont le prix est supérieur a 100D ?**

```
DELETE FROM produits WHERE prix>100;
```

- **L'ordre DELETE FROM <table>;** permet de vider complètement une table. Néanmoins, dans ce cas, la table existe toujours bien qu'elle soit vide.

