

OFPPT

Rapport de



IDMAHAMMA Imane

01/01/2012

Formateur : Mr CHMANTI

Sommaire

1. Introduction à Microsoft SQL Server :	4
1. La sécurité en SQL :	4
2. Rôles au niveau de la base de données :	4
3. Les utilisateur :	4
4. Certificat :	5
2. Commencer à utiliser SQL SERVER :	5
3. Introduction à code SQL :	6
1. Définition des données	7
-Création des tables :	7
□ Principaux types de données	8
□ Contraintes d'intégrité	8
□ CRÉATION D'INDEX.....	10
□ MODIFICATION DU SCHÉMA	11
□ DICTIONNAIRE DE DONNÉES	12
- Manipulation des données	12
□ LA COMMANDE SELECT	13
• Expression des projections	13
• Expression des restrictions.....	14
• USER.....	15
• Valeurs nulles.....	15
• Tri du résultat d'un SELECT.....	15
• Expression des jointures.....	16
• Auto-jointure.....	16
• Jointures externes	16
• Sous-requêtes	17
• Requêtes quantifiées	18
• Le prédicat ALL ou ANY.....	18
• Expression des unions	19
• La clause GROUP BY	21
• La clause HAVING	21
• La forme générale de SELECT	22

□ La commande INSERT	23
• Insertion d'une seule ligne	23
• Insertion de plusieurs lignes	23
□ La commande UPDATE	23
□ La commande DELETE	24
4 Contrôle des données	24
□ Contrôle des accès concurrents	24
□ Une transaction est terminée	25
□ La commande COMMIT	25
□ Contrôle des droits d'accès	25
□ La commande GRANT	25
5 .Les vues	26
□ La commande CREATE VIEW	26
□ Indépendance logique	27
□ Simplification d'accès	27
□ Confidentialité des données	27

1. Introduction à Microsoft SQL Server :

Microsoft SQL Server est une application utilisée pour créer des bases de données informatiques pour la famille Microsoft Windows des systèmes d'exploitation serveur. Microsoft SQL Server fournit un environnement utilisé pour générer des bases de données qui peuvent être accessibles à partir de postes de travail, l'Internet, ou d'autres médias comme un assistant numérique personnel (PDA).

Pour suivre nos leçons, vous devez avoir accès à un ordinateur dans lequel Microsoft SQL Server 2008 est installé. Pour commencer, vous devez avoir un ordinateur qui exécute un système d'exploitation approprié. Si vous êtes seulement à apprendre, vous pouvez utiliser un ordinateur qui exécute Microsoft Windows XP Professionnel, Microsoft Windows Vista, Microsoft Windows 7 (Professionnel ou Édition Intégrale), Microsoft Windows Server 2003 ou Microsoft Windows Server 2008.

Avant d'installer Microsoft SQL Server 2008, assurez-vous d'avoir installé tous les packs de service sur votre système d'exploitation.

1.La sécurité en SQL :

La sécurité de SQL Server peut être vue comme une série d'étape ,impliquant domaines :La plateformes ,l'authentification, les objets(y compris les données) et les applications qui accèdent au système. Les rubriques suivantes vous guideront tout long des procédures de création et de mise en place de sécurité efficace.

2. Rôles au niveau de la base de données :

Pour gérer facilement les autorisations dans vos bases de données, SQL Server fournit plusieurs rôles ,qui sont des principaux de sécurité regroupement d'autres principaux. Ils sont similaires aux Groupes du système d'exploitation Microsoft Windows. Les autorisations des rôles au niveau de la base de données ont une portée à l'échelle de la base de données. Il existe deux types de rôle au niveau de la base de données dans SQL Server :Les rôles de base de données fixes qui sont prédéfinis dans la base de données et les rôles de base de donnée flexibles qui vous pouvez créer.

3. Les utilisateur :

Il existe, dans chaque base de données, des utilisateurs SQL, qui sont en fait des profils de navigation dotés de privilèges (autorisation d'exécution de commandes sur des objets).Ces utilisateurs SQL peuvent être liés à un compte de connexion.

Attention :Un utilisateur est propre a la base qui le contient. Tant est si bien que pour naviguer d'une base l'autre il faut créer autant d'utilisateur que de bases et mapper tous ces utilisateurs au même compte de connexion.

On peut créer un compte utilisateur en utilisant le code SQL :

```
CREATE USER [Raymond Kouma]  
FOR LOGIN rkouma;  
GO  
USE Exercisel;  
GO  
GRANT CREATE FUNCTION  
TO T_exemple;  
GO
```

4. Certificat :

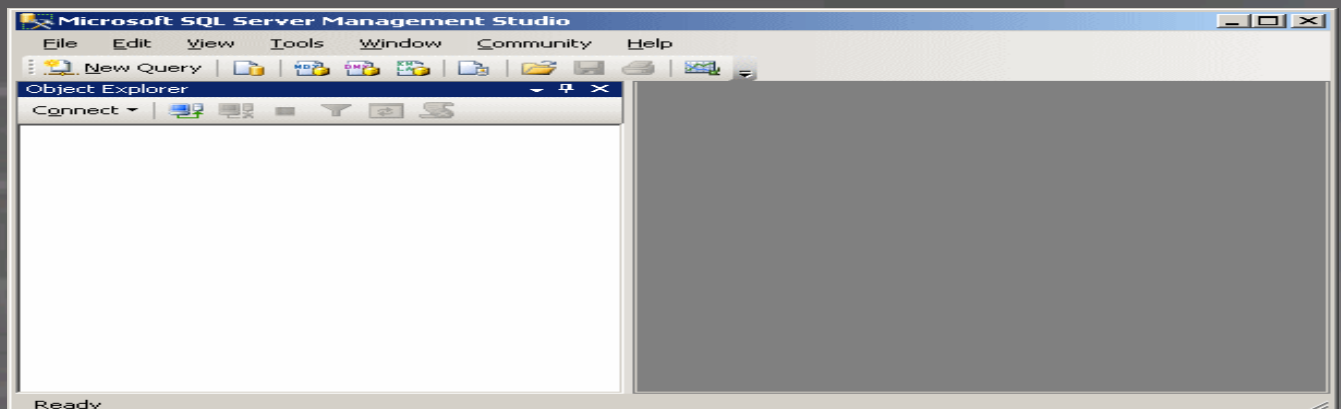
Un certificat est un objet de sécurité signé numériquement qui contient une clé publique pour SQL Server. Vous pouvez utiliser des certificats générés de manière externe ou SQL Server peut générer les certificats.

2. Commencer à utiliser SQL SERVER :

Pour lancer Microsoft SQL Server, cliquez sur Démarrer -> (Tous les) Programmes -> Microsoft SQL Server 2008 -> SQL Server Management Studio. Et nous verrons cet écran



Et automatiquement on va voir cette fenêtre :



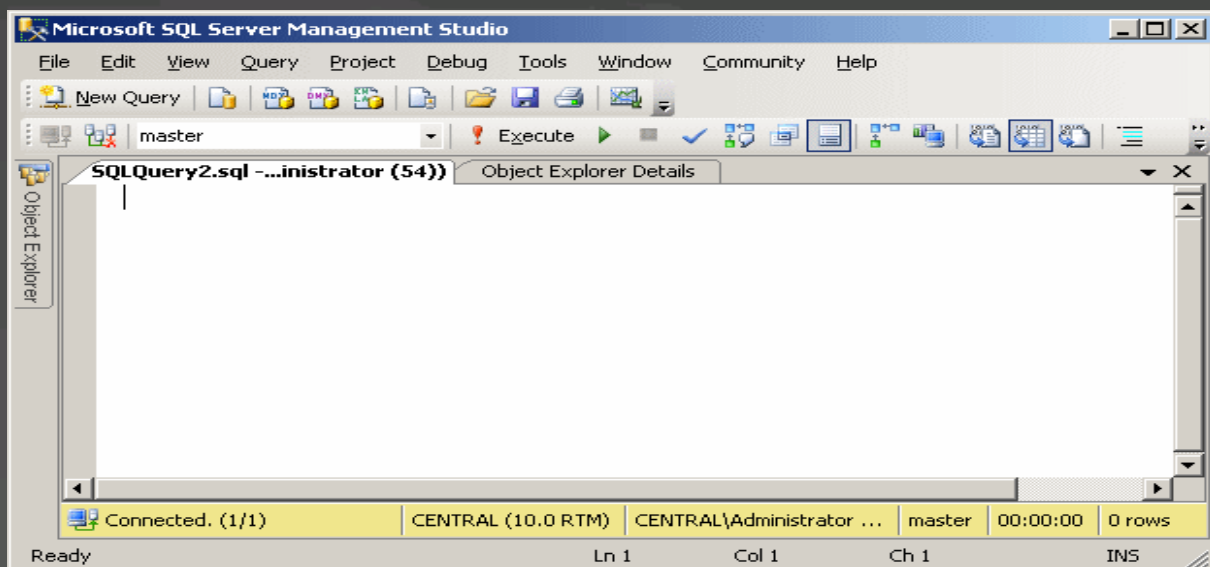
3. Introduction à code SQL :

Pour ouvrir l'éditeur:

- Sur le menu principal, vous pouvez cliquer Fichier -> Nouveau -> Requête avec la connexion actuelle
- Sur la barre d'outils Standard, cliquez sur le bouton Nouvelle requête
- Dans l'Explorateur d'objets, cliquez-droit sur le nom du serveur et cliquez sur Nouvelle requête

Cela créerait une nouvelle fenêtre et le positionner sur le côté droit de l'interface.

Et vous saisissez votre code ici :



- Introduit par IBM, évolution du langage SEQUEL, commercialisé tout d'abord par ORACLE
- SQL est devenu le langage standard pour décrire et manipuler les BDR
- Les commandes SQL :
 - De définition des données :

CREATE

DROP

ALTER

- De manipulation des données :

SELECT

INSERT

UPDATE

DELETE

- De contrôle des données :
 - Contrôle des accès concurrents

COMMIT

ROLLBACK

- Contrôle des droits d'accès

GRANT

REVOKE

- SQL peut être utilisé de 2 manières :

- en mode interactif

- **pour apprendre le langage**

SQL est un langage pour les développeurs

n'est pas destiné à un utilisateur final

Les requêtes sont envoyées à partir d'un terminal interactif

auquel les résultats sont retournés

Ex. :

SELECT C.ville

FROM client C

WHERE C.IdCli = 'c1'

- en mode intégré dans un L3G hôte

(COBOL, ADA, C, FORTRAN ...)

- **pour développer des applications**

Les constantes dans les requêtes SQL peuvent être

remplacées par des variables du programme hôte ; les

résultats doivent être transmis dans des variables

Ex. : SQL dans C

EXEC SQL SELECT C.ville INTO :laVille

FROM client C

WHERE C.IdCli = :unIdCli ;

les variables du programme sont précédées par (:) :

- La notion de curseur permet d'exploiter les résultats

d'une requête ligne à ligne

- Un programme intégrant SQL doit être précompilé par un précompilateur SQL

1. Définition des données

-Création des tables :

table

Syntaxe :

```

CREATE TABLE table
(
  -- définition des colonnes
  colonne type [ NOT NULL [UNIQUE] ]
  [ DEFAULT valeur ]
  [ PRIMARY KEY ]
  [ REFERENCES table ]
  [ CHECK condition ] ,
  ... ,
  -- contraintes de table
  [ PRIMARY KEY (liste de colonnes) ],
  [ UNIQUE (liste de colonnes) ] ,
  ... ,
  [ FOREIGN KEY (liste de colonnes) REFERENCES
  table
  [ ON DELETE (RESTRICT | CASCADE | SET NULL) ]
  [ ON UPDATE (RESTRICT | CASCADE | SET NULL) ] ,
  ... ,
  [ CHECK condition ] ,
  ...
)

```

- Principaux types de données

CHAR(n)
 SMALLINT
 INTEGER
 DECIMAL(n,m)
 DATE

- Contraintes d'intégrité

NOT NULL valeur null impossible

UNIQUE unicité d'un attribut

PRIMARY KEY clé primaire

FOREIGN KEY clé étrangère

CHECK plage ou liste de valeurs

Une contrainte qui ne fait référence qu'à une seule colonne de la table peut faire partie intégrante de la définition de colonne

IUT de Nice - Cours SGBD1 114

- Toute opération de mise à jour violant une des contraintes spécifiées sera rejetée

Le système garantit l'intégrité des données

- SQL2 permet de spécifier les actions à entreprendre

pour le maintien de l'intégrité référentielle, lors d'une suppression ou d'une modification d'un tuple référencé

CASCADE cascader les suppressions ou modifications

par ex. si on supprime un produit dans la table PRODUIT, toutes les ventes correspondantes seront supprimées dans la table VENTE

SET NULL rendre nul les attributs référençant
par ex. si on modifie la référence d'un produit dans la table PRODUIT, toutes les références correspondantes seront modifiées dans la table VENTE

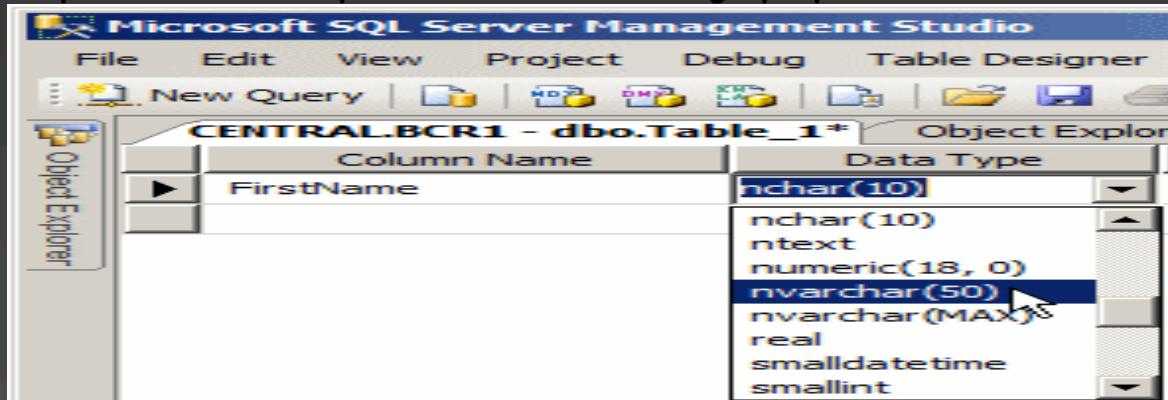
RESTRICT rejet de la mise à jour
c'est l'option par défaut

- Exemple

```
CREATE TABLE client
(
  IdCli CHAR(4) PRIMARY KEY ,
  nom CHAR(20) ,
  ville CHAR(30)
  CHECK (ville IN ('Nice', 'Paris', 'Rome')) ,
)
CREATE TABLE produit
(
  IdPro CHAR(6) PRIMARY KEY ,
  nom CHAR(30) NOT NULL UNIQUE ,
  marque CHAR(30) ,
  prix DEC(6,2) ,
  qstock SMALLINT
  CHECK (qstock BETWEEN 0 AND 100) ,
  -- contrainte de table
  CHECK (marque <> 'IBM' OR qstock < 10)
)
CREATE TABLE vente
(
  IdCli CHAR(4) NOT NULL
  REFERENCES client ,
  IdPro CHAR(6) NOT NULL ,
  date DATE NOT NULL ,
  qte SMALLINT
  CHECK (qte BETWEEN 1 AND 10) ,
  -- contrainte de table
```

PRIMARY KEY (IdCli, IdPro, date) ,
FOREIGN KEY (IdPro) REFERENCES produit
ON DELETE CASCADE ON UPDATE CASCADE
)

On peut modifier les paramètres d'une table graphiquement :



- CRÉATION D'INDEX

La commande **CREATE INDEX** permet de créer des Index multi-colonne

Syntaxe :

CREATE [UNIQUE] INDEX index
ON table (colonne [ASC|DESC], ...)

☐ L'option **UNIQUE** permet d'assurer l'unicité d'une clé

Ex.: **CREATE UNIQUE INDEX** index1 **ON** client(Nom)

☐ Les index permettent d'accélérer les recherches

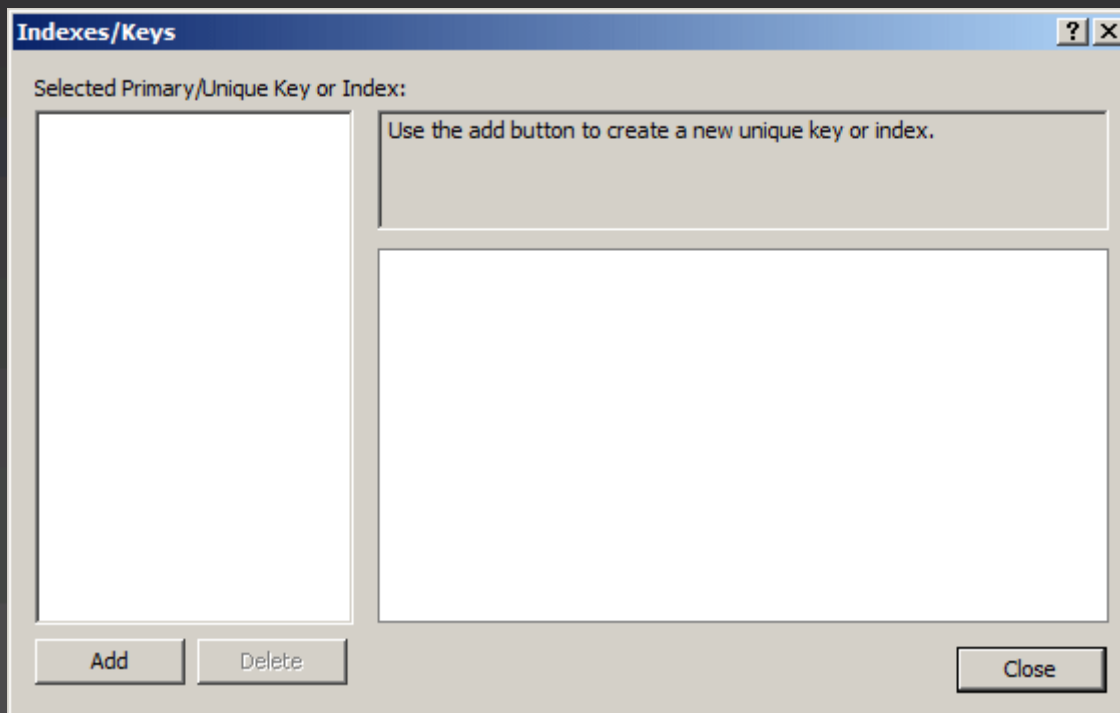
☐ Le système détermine sa stratégie d'accès en fonction des index existants

☐ Les index sont automatiquement mis à jour

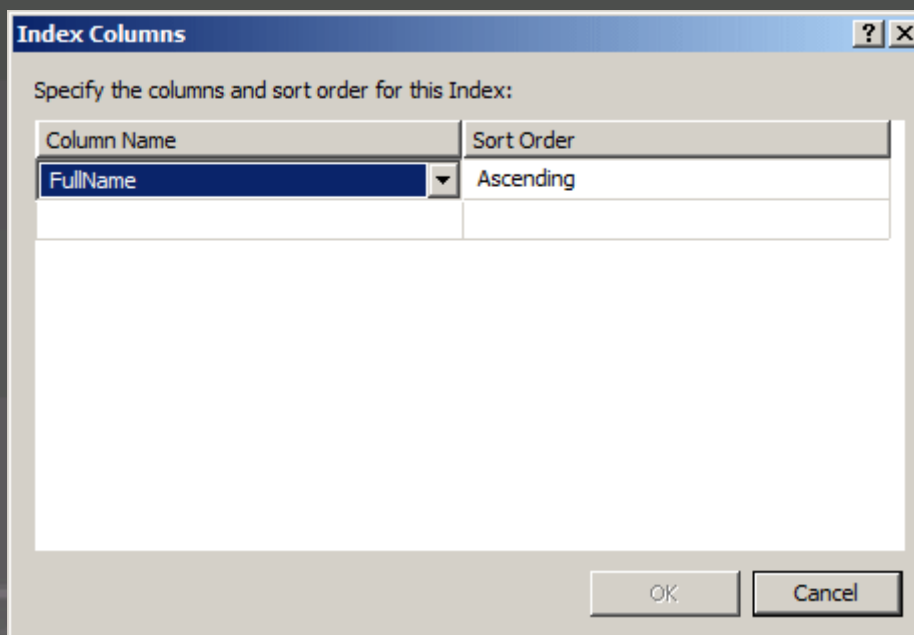
☐ Il est indispensable de créer les index appropriés pour accélérer le traitement des requêtes

☐ Il ne faut cependant pas créer des index sur n'importe quel colonne ou groupe de colonnes, car les mises à jour seraient ralenties inutilement par la maintenance de ces index

☐ Un index est supprimé par la commande **DROP INDEX**



Et on clique sur ajouter :



- **MODIFICATION DU SCHÉMA**

La modification du schéma n'est pas prévue dans SQL1 ; cependant la plupart des systèmes permettent la suppression ou la modification d'une table à l'aide des commandes :

DROP TABLE
ALTER TABLE

Ex. :

```
ALTER TABLE client  
ADD COLUMN teleph CHAR(16)
```

- DICTIONNAIRE DE DONNÉES

Le **dictionnaire de données** contient la description de tous les objets (relations, index, ...) de la BD

- Le DD est décrit sous forme de **tables systèmes**

- Par exemple, on peut citer dans DB2 :

SYSTABLES (NAME, CREATOR, COLCOUNT, ...)

description des tables

SYSCOLUMNS (NAME, TBNAME, COLTYPE, ...)

description des colonnes

- Le DD peut être consulté de la même manière que les tables de base avec le langage d'interrogation il faut toutefois connaître les noms et les schémas des tables systèmes

- Manipulation des données

SELECT, INSERT, UPDATE et DELETE sont les 4 commandes de manipulation des données en SQL

Ex. :

Recherche **SELECT**

```
SELECT P.prix  
FROM produit P  
WHERE P.IdPro = 'p1'
```

Ajout **INSERT**

```
INSERT  
INTO client (IdCli, nom, ville)  
VALUES ('c100', 'Duduche', 'Nice')
```

Mise à jour **UPDATE**

```
UPDATE produit P  
SET P.prix = P.prix * 1.20  
WHERE P.IdPro = 'p2'
```

Suppression **DELETE**

```
DELETE  
FROM produit P  
WHERE P.IdPro = 'p4'
```

- **LA COMMANDE SELECT**

La commande **SELECT** permet de rechercher des données à partir de plusieurs tables ; le résultat est présenté sous forme d'une **table réponse**

- Expression des projections

Q1 Donner les noms, marques et prix des produits

```
SELECT P.nom, P.marque, P.prix  
FROM produit P
```

Synonyme de nom de table (ou **alias)**

- On peut introduire dans la clause FROM un synonyme (*alias*) à un nom de table en le plaçant immédiatement après le nom de la table
- Les noms de table ou les synonymes peuvent être utilisés pour préfixer les noms de colonnes dans le SELECT
- Les préfixes ne sont obligatoires que dans des cas particuliers (par ex. pour une auto-jointure) ; leur emploi est cependant conseillé pour la clarté
- Un alias est utilisé par SQL comme une variable de parcours de table (dite *variable de corrélation*) désignant à tout instant une ligne de la table

Q2 Donner les différentes marques de produit

```
SELECT DISTINCT P.marque  
FROM produit P
```

☐ Contrairement à l'algèbre relationnelle, SQL n'élimine pas les doublons

☐ Pour éliminer les doublons il faut spécifier **DISTINCT**

Q3 Donner les références des produits et leurs prix majorés de 20%

```
SELECT P.IdPro, P.prix * 1.20  
FROM produit P
```

☐ Il est possible d'effectuer des opérations arithmétiques (+, -, *, /) sur les colonnes extraites

Q4 Donner tous les renseignements sur les clients

```
SELECT *  
FROM client
```

☐ Une étoile (*) permet de lister tous les attributs

- Expression des restrictions

Q5 Donner les noms des produits de marque IBM

```
SELECT P.nom
```

```
FROM produit P
```

```
WHERE P.marque = 'IBM'
```

- La condition de recherche (qualification) est spécifiée après la clause **WHERE** par un prédicat

- Un prédicat simple peut-être :

- un prédicat d'égalité ou d'inégalité (=, <>, <, >, <=, >=)

- un prédicat **LIKE**

- un prédicat **BETWEEN**

- un prédicat **IN**

- un test de valeur **NULL**

- un prédicat **EXISTS**

- un prédicat **ALL** ou **ANY**

- Un prédicat composé est construit à l'aide des connecteurs **AND**, **OR** et **NOT**

IUT de Nice - Cours SGBD1 123

- Exemples

Q6 Lister les clients dont le nom comporte la lettre A en 2^{ème} position

```
SELECT *
```

```
FROM client C
```

```
WHERE C.nom LIKE '__A%'
```

Le prédicat **LIKE** compare une chaîne avec un modèle

(_) remplace n'importe quel caractère

(%) remplace n'importe quelle suite de caractères

Q7 Lister les produits dont le prix est compris entre 5000F et 12000F

```
SELECT *
```

```
FROM produit P
```

```
WHERE P.prix BETWEEN 5000 AND 12000
```

Le prédicat **BETWEEN** teste l'appartenance à un intervalle

Q8 Lister les produits de marque IBM, Apple ou Dec

```
SELECT *
```

```
FROM produit P
```


WHERE P.marque IN ('IBM', 'Apple', 'Dec')

Le prédicat **IN** teste l'appartenance à une liste de valeurs

Q9 Lister les produits dont le prix est inconnu

SELECT *

FROM produit P

WHERE P.prix IS NULL

Q10 Lister les produits de marque IBM dont le prix est inférieur à 12000F

SELECT *

FROM produit P

WHERE P.marque = 'IBM' AND P.prix < 12000

Le connecteur **AND** relie les 2 prédicats de comparaison

• USER

Le mot réservé **USER** désigne l'utilisateur courant

• Valeurs nulles

La valeur **NULL** est une valeur particulière signifiant qu'une donnée est manquante, sa valeur est inconnue

• Dans une expression arithmétique, si l'un des termes est null, alors l'expression entière prend la valeur NULL

• Un prédicat de comparaison (=, <>, <, <=, >, >=) prend la valeur logique "inconnu" si l'un des termes de la comparaison est NULL

AND F V ? OR F V ? NOT

F F F F F F V ? F V

V F V ? V V V V V F

? F ? ? ? ? V ? ? ?

V = vrai, F = faux, ? = inconnu

● Tri du résultat d'un SELECT

La clause **ORDER BY** permet de spécifier les colonnes définissant les critères de tri

□ Le tri se fera d'abord selon la première colonne spécifiée, puis selon la deuxième colonne etc...

□ Exemple

Q11 Lister les produits en les triant par marques et à l'intérieur d'une marque par prix décroissants

SELECT *

FROM produit P

ORDER BY P.marque, P.prix DESC

□ L'ordre de tri est précisé par **ASC** (croissant) ou **DESC** (décroissant) ; par défaut **ASC**

• Expression des jointures

Le produit cartésien s'exprime simplement en incluant plusieurs tables après la clause **FROM**

□ La condition de jointure est exprimée après **WHERE**

□ Exemples :

Q12 Donner les références et les noms des produits vendus

```
SELECT P.IdPro, P.nom  
FROM produit P , vente V  
WHERE P.IdPro = V.IdPro
```

Q13 Donner les noms des clients qui ont acheté le produit de nom 'PS1'

```
SELECT C.nom  
FROM client C , produit P, vente V  
WHERE V.IdCli = C.IdCli  
AND V.IdPro = P.IdPro  
AND P.nom = 'PS1'
```

• Auto-jointure

Q14 Donner les noms des clients de la même ville que John

```
SELECT C2.nom  
FROM client C1 , client C2  
WHERE C1.ville = C2.ville  
AND C1.nom = 'John'  
AND C2.nom <> 'John'
```

· Cet exemple utilise, pour le couplage des villes, la jointure de la table Client avec elle-même (*auto-jointure*)

· Pour pouvoir distinguer les références ville dans les 2 copies, il faut introduire 2 alias différents C1 et C2 de la table client

• Jointures externes

La **jointure externe** permet de retenir lors d'une jointure les lignes d'une table qui n'ont pas de correspondant dans l'autre table, avec des valeurs nulles associées

On distingue **jointure externe gauche**, **droite** et **complète** selon que l'on retient les lignes sans correspondant des 2 tables ou seulement d'une

SQL2 offre la possibilité de spécifier les jointures externes au niveau de la clause **FROM** selon la syntaxe suivante :

```
FROM table1 [NATURAL] [(LEFT|RIGHT)] JOIN table2
```

[ON (liste de colonnes = liste de colonnes)]

NATURAL signifie jointure naturelle, c.a.d l'égalité des attributs de même nom

Q15 Lister tous les clients avec le cas échéant leurs achats

```
SELECT C.IdCli, C.nom, C.ville
```

```
V.IdPro, V.date, V.qte
```

```
FROM client C NATURAL LEFT JOIN vente V
```

• Sous-requêtes

SQL permet l'imbrication de **sous-requêtes** au niveau de la clause WHERE

d'où le terme "*structuré*" dans Structured Query Language

□ Les sous-requêtes sont utilisées :

· dans des prédicats de comparaison

(=, <>, <, <=, >, >=)

· dans des prédicats **IN**

· dans des prédicats **EXISTS**

· dans des prédicats **ALL** ou **ANY**

□ Une sous-requête dans un prédicat de comparaison doit se réduire à une seule valeur ("**singleton select**")

□ Une sous-requête dans un prédicat IN, ALL ou ANY doit représenter une table à colonne unique

□ L'utilisation de constructions du type "IN sous-requête" permet d'exprimer des jointures de manière procédurale ... *ce qui est déconseillé !!*

□ Exemple

Q16 Donner les noms des clients qui ont acheté le produit 'p1'

• Avec sous-requête

```
SELECT C.nom
```

```
FROM client C
```

```
WHERE IdCli IN
```

```
(
```

```
SELECT V.IdCli
```

```
FROM vente V
```

```
WHERE P.IdPro = 'p1'
```

```
)
```

• Avec jointure

```
SELECT C.nom
```

```
FROM client C , vente V
```

```
WHERE C.IdCli = V.IdCli
```

```
AND V.IdPro = 'p1'
```

♥ De préférence, utiliser la jointure

• Requêtes quantifiées

• Le prédicat EXISTS

Il permet de tester si le résultat d'une sous-requête est vide ou non

Q17 Donner les noms des produits qui n'ont pas été achetés

```
SELECT C.nom  
FROM produit P  
WHERE NOT EXISTS  
( SELECT *  
  FROM vente V  
  WHERE V.IdPro = P.IdPro )
```

Il permet de répondre à des questions quantifiées par

"pour tout..." : $\forall x \mid P(x) \Leftrightarrow \neg (\exists x \mid \neg P(x))$

Q18 Donner les noms des produits qui ont été achetés par tous les clients de Nice

```
SELECT P.nom  
FROM produit P  
WHERE NOT EXISTS  
(  
  SELECT *  
  FROM client C  
  WHERE C.ville = 'Nice'  
  AND NOT EXISTS  
  (  
    SELECT *  
    FROM vente V  
    WHERE C.IdCli = V.IdCli  
    AND V.IdPro = P.IdPro  
  )  
)
```

• Le prédicat ALL ou ANY

Ils permettent de tester si un prédicat de comparaison est vrai pour tous (**ALL**) ou au moins un (**ANY**) des résultats d'une sous-requête

Q19 Donner les nos des clients ayant acheté un produit en quantité supérieure à chacune des quantités de produits achetées par le client 'c1'

```
SELECT V.IdCli  
FROM vente V  
WHERE V.qte >= ALL  
(  
  SELECT W.qte  
  FROM vente W  
  WHERE W.IdCli = 'c1'  
)
```

Q20 Donner les nos des clients ayant acheté un produit en quantité supérieure à au moins l'une des quantités de produits achetées par le client 'c1'

```
SELECT V.IdCli
FROM vente V
WHERE V.qte >= ANY
(
  SELECT W.qte
  FROM vente W
  WHERE W.IdCli = 'c1'
)
```

Les prédicats ALL et ANY sont redondants, ils peuvent s'exprimer avec EXISTS

$x \Delta ANY$

```
( SELECT y
  FROM t
  WHERE p )
```

\Leftrightarrow

```
EXISTS
( SELECT *
  FROM t
  WHERE p AND  $x \Delta t.y$  )
```

$x \Delta ALL$

```
( SELECT y
  FROM t
  WHERE p )
```

\Leftrightarrow

```
NOT EXISTS
( SELECT *
  FROM t
  WHERE p AND NOT (  $x \Delta t.y$  ) )
```

Où Δ est un prédicat de comparaison (=, <>, <, <=, >, >=)

• Expression des unions

SQL1 permet d'exprimer l'opération d'union en connectant des SELECT par des UNION

Q21 Donner les nos des produits de marque IBM ou ceux achetés par le client no 'c1'

```
SELECT P.IdPro
FROM produit P
WHERE P.marque = 'IBM'
UNION
SELECT P.IdPro
FROM vente V
WHERE V.IdCli = 'c1'
```

□ L'union élimine les doublons, pour obtenir les doublons il faut spécifier ALL après UNION

- UNION est une opération binaire, on peut écrire :
(x UNION y) UNION z ou x UNION (y UNION z)
- Les parenthèses sont nécessaires dans certains cas,
par ex. :

(x UNION ALL y) UNION z
n'est pas équivalent à
x UNION ALL (y UNION z)

Fonctions de calculs

SQL fournit des fonctions de calcul opérant sur
l'ensemble des valeurs d'une colonne de table

COUNT nombre de valeurs

SUM somme des valeurs

AVG moyenne des valeurs

MAX plus grande valeur

MIN plus petite valeur

Q22 Donner le nombre total de clients

```
SELECT COUNT ( IdCli )
```

```
FROM client
```

Q23 Donner le nombre total de clients ayant acheté des
produits

```
SELECT COUNT ( DISTINCT IdCli )
```

```
FROM vente
```

- On peut faire précéder l'argument du mot clé
DISTINCT pour indiquer que les valeurs redondantes
doivent être éliminées avant application de la fonction

IUT de Nice - Cours SGBD1 137

- La fonction spéciale **COUNT (*)** compte toutes les
lignes dans une table
- Les valeurs nulles ne sont pas prises en compte, sauf
pour COUNT(*)
- Si l'argument est un ensemble vide, COUNT renvoie
la valeur 0, les autres fonctions renvoyant la valeur
NULL

- Exemples :

Q24 Donner le nombre total de 'PS1' vendus

```
SELECT SUM ( V.qte )
```

```
FROM vente V , produit P
```

```
WHERE P.IdPro = V.IdPro
```

```
AND P.nom = 'PS1'
```

Q25 Donner les noms des produits moins chers que la
moyenne des prix de tous les produits

```
SELECT P1.nom
```

```
FROM produit P1
```

```
WHERE P1.prix <
```

```
(
```



```
SELECT AVG ( P2.prix )  
FROM produit P2  
)
```

Cet exemple montre un "*singleton select*" pour calculer la moyenne des prix

IUT de Nice - Cours SGBD1 138

• La clause GROUP BY

La clause **GROUP BY** permet de partitionner une table en plusieurs groupes

- Toutes les lignes d'un même groupe ont la même valeur pour la liste des attributs de partitionnement spécifiés après GROUP BY
- Les fonctions de calcul opèrent sur chaque groupe de valeurs

□ Exemples :

Q26 Donner pour chaque référence de produit la quantité totale vendue

```
SELECT V.IdPro, SUM ( V.qte )  
FROM vente V  
GROUP BY V.IdPro
```

Q27 Donner la quantité totale achetée par chaque client (0 pour ceux qui n'ont rien acheté)

```
SELECT C.IdCli, SUM ( V.qte )  
FROM client C NATURAL LEFT JOIN vente V  
GROUP BY C.IdCli
```

• La clause HAVING

La clause **HAVING** permet de spécifier une condition de restriction des groupes

- Elle sert à éliminer certains groupes, comme WHERE sert à éliminer des lignes

□ Exemples

Q28 Donner les noms des marques dont le prix moyen des produits est < 5000F

```
SELECT P.marque, AVG ( P.prix )  
FROM produit P  
GROUP BY P.marque  
HAVING AVG ( P.prix ) < 5000
```

Q29 Donner les références des produits achetés en qte > 10 par plus de 50 clients

```
SELECT P.marque, AVG ( P.prix )  
FROM vente V  
WHERE V.qte > 10  
GROUP BY V.IdPro  
HAVING COUNT (*) > 50
```

• La forme générale de SELECT

SELECT [**DISTINCT**] liste d'attributs, expressions

FROM liste de tables ou vues

WHERE qualification

GROUP BY attributs de partitionnement

HAVING qualification de groupe

ORDER BY liste de colonnes [**ASC** | **DESC**]

□ Exemple

Q30 Donner les nos, les prix, les marques et la quantité maximum vendue de tous les produits IBM, Apple ou Dec dont la quantité totale vendue est supérieure à 500 et dont les quantités vendues sont > 10

```
SELECT P.IdPro, P.prix, P.marque,  
'Qte max vendue = ', MAX ( V.qte)  
FROM produit P , vente V  
WHERE P.IdPro = V.IdPro  
AND P.marque IN ('IBM', 'Apple', 'Dec')  
AND V.qte > 10  
GROUP BY P.IdPro, P.prix, P.marque  
HAVING SUM ( V.qte ) > 500
```

IUT de Nice - Cours SGBD1 141

Du seul point de vue logique, on peut considérer que le résultat d'un SELECT est construit suivant les étapes :

1. FROM

la clause **FROM** est évaluée de manière à produire une nouvelle table, produit cartésien des tables dont le nom figure après FROM

2. WHERE

le résultat de l'étape 1 est réduit par élimination de toutes les lignes qui ne satisfont pas à la clause **WHERE**

3. GROUP BY

le résultat de l'étape 2 est partitionné selon les valeurs des colonnes dont le nom figure dans la clause **GROUP BY**

dans l'exemple ci-dessus, les colonnes sont P.IdPro, P.prix et P.marque ; en théorie il suffirait de prendre uniquement P.IdPro comme colonne définissant les groupes (puisque le prix et la marque sont déterminés par le no de produit)

SQL oblige de faire apparaître dans la clause **GROUP BY** toutes les colonnes qui sont mentionnées dans la clause **SELECT**

4. HAVING

les groupes ne satisfaisant pas la condition **HAVING** sont éliminés du résultat de l'étape 3

5. SELECT

chacun des groupes génère une seule ligne du résultat

▣ La commande INSERT

La commande **INSERT** permet d'ajouter de nouvelles lignes à une table

INSERT

INTO table [(liste de colonnes)]

(VALUES (liste de valeurs) | requête)

- ▣ Dans le cas où la liste de colonnes n'est pas spécifiée tous les attributs de la table cible doivent être fournis dans l'ordre de déclaration
- ▣ Si seulement certaines colonnes sont spécifiées, les Autres sont insérées avec la valeur NULL
- ▣ Une insertion à partir d'une requête permet d'insérer plusieurs lignes dans la table cible à partir d'une autre table

• Insertion d'une seule ligne

Q31 Ajouter le client ('c100', 'Duduche', 'Nice') dans la table client

INSERT

INTO client (IdCli, nom, ville)

VALUES ('c100', 'Duduche', 'Nice')

• Insertion de plusieurs lignes

Q32 Ajouter dans une table « temp » de même schéma que la table Vente, toutes les ventes qui sont antérieures au 01-Jan-1994

INSERT

INTO temp (IdCli, IdPro, date, qte)

SELECT V.no_cli, V.IdPro, V.date, V.qte

FROM vente V

WHERE V.date < '01-jan-1994'

• La commande UPDATE

La commande **UPDATE** permet de changer des valeurs d'attributs de lignes existantes

UPDATE table

SET liste d'affectations

[WHERE qualification]

- ▣ L'absence de clause WHERE signifie que les changements doivent être appliqués à toutes les lignes de la table cible
- ▣ Exemples

Q33 Augmenter de 20% les prix de tous les produits

```

UPDATE produit
SET prix = prix * 1.2
Q34 Augmenter de 50% les prix des produits achetés par des
clients de Nice
UPDATE produit
SET prix = prix * 1.5
WHERE EXISTS
(
SELECT *
FROM vente V , client C
WHERE V.IdCli = C.IdCli
AND C.ville = 'Nice'
)

```

□ La commande DELETE

La commande **DELETE** permet d'enlever des lignes dans une table

```

DELETE
FROM table
[ WHERE qualification ]

```

□ L'absence de clause WHERE signifie que toutes les lignes de la table cible sont enlevées

□ Exemples

Q35 Supprimer les ventes antérieures au 01-jan-1994

```

DELETE
FROM vente
WHERE date < '01-jan-1994'

```

Q36 Supprimer les ventes des clients de Nice antérieures au 01-mar-1994

```

DELETE
FROM vente
WHERE date < '01-mar-1994'
AND IdCli IN
(
SELECT C.IdCli
FROM client C
WHERE C.ville = 'Nice'
)

```

- Contrôle des données

- Contrôle des accès concurrents

La notion de transaction

Une **transaction** est une unité logique de traitement qui est soit complètement exécutée, soit complètement abandonnée

□ Une transaction fait passer la BD d'un état cohérent à un autre état cohérent

- Une transaction est terminée

- soit par **COMMIT**
- soit par **ROLLBACK**

- La commande COMMIT

La commande **COMMIT** termine une transaction avec succès ; toutes les mises à jour de la transaction sont validées

- On dit que la transaction est **validée**
- Tous ses effets sont alors connus des autres transactions s'exécutant concurremment

- La commande ROLLBACK

La commande **ROLLBACK** termine une transaction avec échec ; toutes les mises à jour de la transaction sont annulées (tout se passe comme si la transaction n'avait jamais existé)

- On dit que la transaction est **annulée**
- Aucune des opérations effectuées par cette transaction n'est connue des autres transactions

- Contrôle des droits d'accès

Chaque créateur d'une table est **propriétaire** de cette table et obtient tous les droits d'accès à cette table (i.e. les droits d'effectuer les opérations SELECT, INSERT, UPDATE, DELETE)

- Le propriétaire d'une table peut passer ses privilèges sélectivement à d'autres utilisateurs ou à tout le monde (**PUBLIC**)

- La commande GRANT

La commande **GRANT** permet de passer des droits d'accès à un utilisateur ou un groupe d'utilisateurs
GRANT privilèges **ON** table **TO** bénéficiaire
[WITH GRANT OPTION]

- Les privilèges qui peuvent être passés sont :
 - soit **ALL** (tous les privilèges)
 - soit une liste de privilèges parmi :
 - **SELECT**
 - **INSERT**
 - **UPDATE [(liste de colonnes)]**

l'omission de la liste de colonnes signifie toutes les colonnes

- **DELETE**

- Le bénéficiaire peut être :

- soit **PUBLIC** (tous les utilisateurs)
 - soit un utilisateur ou un groupe d'utilisateurs

- L'option **WITH GRANT OPTION** permet de passer un privilège avec le droit de le transmettre

- Exemples

GRANT SELECT ON produit TO PUBLIC

GRANT INSERT, UPDATE ON produit TO toto

- Aucun utilisateur ne peut passer un privilège qu'il ne détient pas

- La commande REVOKE

La commande **REVOKE** permet de retirer des droits à un utilisateur ou groupe d'utilisateurs

REVOKE privilèges **ON** table **FROM** bénéficiaire

- Les vues

Une **vue** est une table virtuelle calculée à partir des tables de base par une requête

- Une vue apparaît à l'utilisateur comme une table réelle, cependant les lignes d'une vue ne sont pas stockées dans la BD (uniquement sa définition est enregistrée dans le DD)

- Les vues assurent **l'indépendance logique**

- Elles peuvent être utilisées pour cacher des données sensibles, ou pour montrer des données statistiques

Ex. :

CREATE VIEW prix-caché AS

SELECT P.IdPro, P.nom, P.marque

FROM produit P

CREATE VIEW stat-vente (IdPro, tot-qte)

AS SELECT V.IdPro, SUM (V.qte)

FROM vente V

GROUP BY V.IdPro

- La commande CREATE VIEW

La commande **CREATE VIEW** crée la définition d'une vue

CREATE VIEW vue [(liste de colonnes)]

AS requête [**WITH CHECK OPTION**]

- Ex. :

CREATE VIEW produitIBM (no, nom, prx)

AS SELECT P.IdPro, P.nom, P.prix

FROM produit P

WHERE P.marque = 'IBM'

- Les données des tables de bases peuvent être modifiées dans certains cas au travers d'une vue, mais cela n'est pas toujours possible
- L'option **WITH CHECK OPTION** permet de vérifier que les lignes insérées dans une table de base au-travers d'une vue vérifient les conditions exprimées dans la requête. Cela permet d'imposer des contraintes d'intégrité lors des mises à jour au travers de la vue

□ Intérêt des vues

The screenshot shows a SQL query window titled 'SQLQuery1.sql ...strateur (56))*'. The query contains the following SQL code:

```
CREATE VIEW Registration.StaffMembers
AS
SELECT Registration.GetFullName (FirstName, LastName) AS [Full Name]
FROM Registration.Teachers;
GO

SELECT * FROM Registration.StaffMembers;
GO
```

Below the query editor, the 'Results' tab is active, displaying a table with 4 rows and 1 column named 'Full Name'.

	Full Name
1	Kingston, Alex
2	Moons, Frank
3	Simms, Justin
4	Manson, Jeanne

At the bottom of the window, a status bar indicates: 'Query executed successfully. CENTRAL (10.0 SP2) Central\Administrateur... rosh 00:00:00 4 rows'.

- **Indépendance logique**

Le concept de vue permet d'assurer une indépendance des Applications vis-à-vis des modifications du schéma

- **Simplification d'accès**

Les vues simplifient l'accès aux données en permettant par exemple une pré-définition des Jointures et en masquant ainsi à l'utilisateur l'existence de plusieurs tables

Ex. :

La vue qui calcule les moyennes générales pourra être consulté par la requête :

SELECT * FROM Moyennes

- **Confidentialité des données**

Une vue permet d'éliminer des lignes sensibles et/ou des colonnes sensibles dans une table de base