Résumé
S Q L

Auteur: Alexandre PATIN
Edition: 22 février 2000
E-mail : alexandre.patin@free.fr
URL : http://alexandre.patin.free.fr/

Sommaire

I. INTRODUCTION

Objectifs

Cet ouvrage recueille et développe la plupart des éléments du langage SQL.
Dans les premiers chapitres, les concepts, commandes et fonctions du langage sont énumérés de la manière la plus complète possible et souvent illustrés par des exemples.
Ensuite, une liste d'exemples de requêtes de complexité progressive aborde les principaux aspects du langage.
Ainsi, que vous soyez un utilisateur débutant ou confirmé, vous disposez de tous les éléments pour créer et adapter vos requêtes spécifiquement à vos besoins.

Conventions syntaxiques

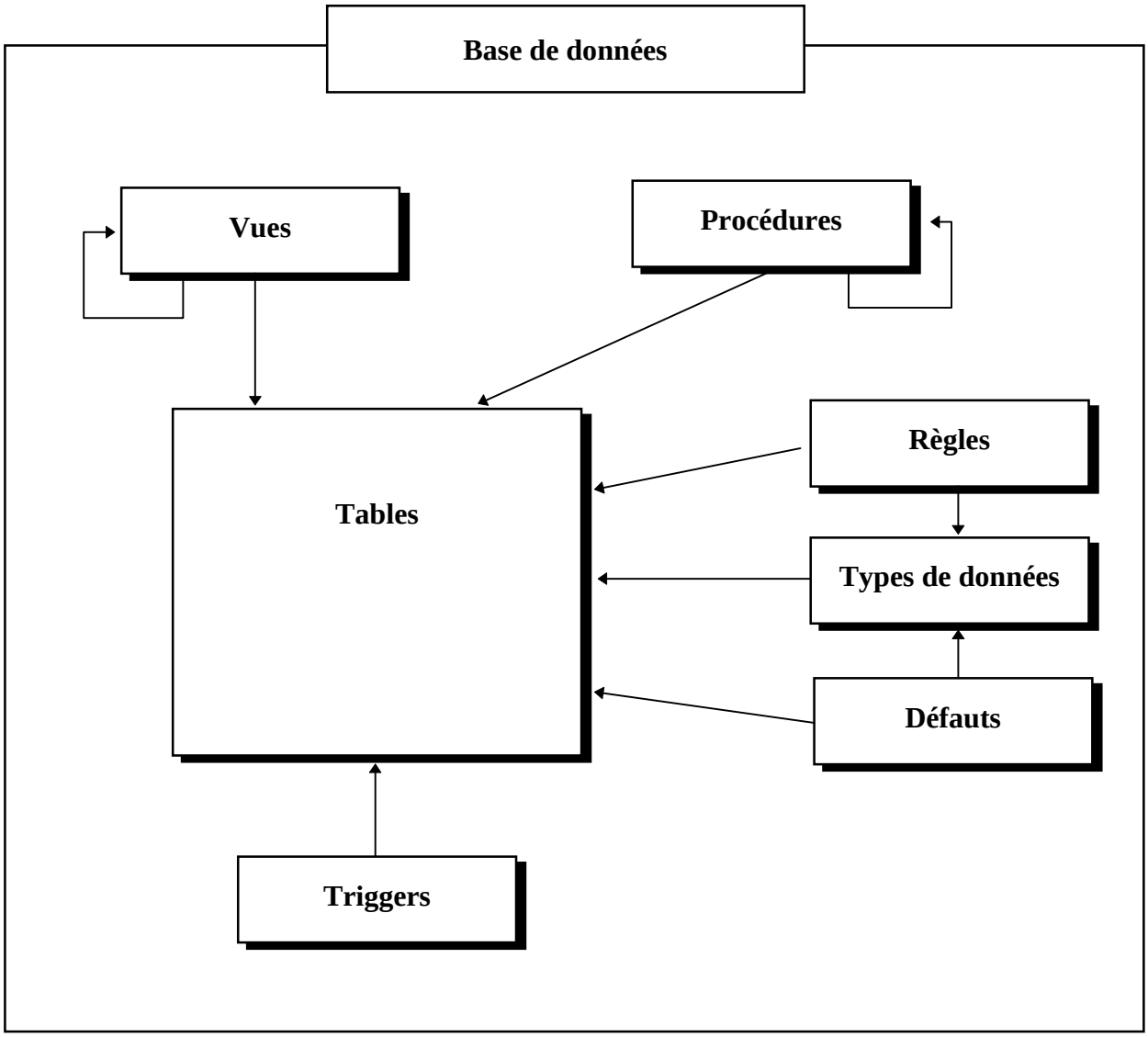
Elément	Description	Exemple
{ }	Les accolades indiquent que vous devez choisir au moins une des options proposées.	**{** option1, option2, option3 **}**
|	Les barres verticales signifient que vous ne pouvez sélectionner qu'une des options proposées.	{ option1 | option2 | option3 }
[ ]	Les crochets indiquent que leur contenu est optionnel. *Remarque : Il ne faut pas saisir les crochets.*	**[** option **]**
...	Les « ... » indiquent que vous pouvez répéter le dernier bloc autant de fois que vous le désirez.	

II. CONCEPTS

Contenu d'une base de données

Une base de données est une collection de tables contenant des informations corrélées.

Schéma du contenu d'une base de données:



Types de données

Données numériques exactes entières

Type de données	Synonymes	Plage de valeurs	Octets de stockage
tinyint		0 à 255	1
smallint		-2^{15} à $2^{15}-1$ (-32 768 à 32 767)	2
int	integer	-2^{31} à $2^{31}-1$ (-2 147 483 648 à 2 147 483 647)	4

Données numériques exactes décimales

Type de données	Synonymes	Plage de valeurs	Octets de stockage
numeric (p,s)		-10^{38} à $10^{38}-1$	2 à 17
decimal (p,s)		-10^{38} à $10^{38}-1$	2 à 17

Données numériques arrondies

Type de données	Synonymes	Plage de valeurs	Octets de stockage
float (precision)		en fonction du matériel	4 ou 8
double precision		en fonction du matériel	8
real		en fonction du matériel	4

Données monétaires

Type de données	Synonymes	Plage de valeurs	Octets de stockage
smallmoney		-214 748.3648 à 214 748.3647	4
money		-922 337 203 685 477.5808 à 922 337 203 685 477.5807	8

Données horaires

Type de données	Synonymes	Plage de valeurs	Octets de stockage
smalldatetime		du 01/01/1900 au 06/06/2079	4
datetime		du 01/01/1753 à 00h00 au 31/12/9999 à 23h59	8

Données alphanumériques

Type de données	Synonymes	Plage de valeurs	Octets de stockage
char(n)	character	255 caractères ou moins	n
varchar(n)	character varying, char varying	255 caractères ou moins	taille réelle de la chaine
nchar(n)	national charcter, national char	255 caractères ou moins	n x@@ncharsize
nvarchar(n)	nchar varying, national char varying, national character varying	255 caractères ou moins	nombre de caractères x @@ncharsize
text(n)		2^{31}-1 caractères ou moins	0 ou un multiple de 2k

Données binaires

Type de données	Synonymes	Plage de valeurs	Octets de stockage
binary (n)		255 octets ou moins	n
varbinary(n)		255 octets ou moins	taille réelle de la donnée
image		2^{31}-1 octets ou moins	0 ou un multiple de 2k

Données booléennes

Type de données	Synonymes	Plage de valeurs	Octets de stockage
bit		0 ou 1	8

Données sécurisées (uniquement sur serveur SQL sécurisé)

Type de données	Synonymes	Plage de valeurs	Octets de stockage
sensitivity			4
sensitivity_boundary			4

Valeur Nulle

La valeur nulle : NULL, marque les colonnes ayant une valeur inconnue (par opposition à celles qui ont la valeur zéro ou une chaîne vide). NULL permet la distinction entre une saisie délibérée de zéro (pour les colonnes numériques) ou vide (pour les colonnes de type caractère) et non-saisie.

Une valeur NULL ne peut jamais vérifier une égalité, même avec une autre valeur NULL.

Une règle liée à une colonne doit inclure le NULL dans sa définition pour permettre l'insertion de valeurs NULL. Quand une ligne est créée, si aucune valeur n'est spécifiée dans la colonne, aucun défaut n'est défini, les NULL sont permis dans la colonne, alors le serveur assignera automatiquement la valeur NULL.

Exemples d'utilisation du mot clé NULL:

Dans une commande CREATE TABLE :
nom_colonne type_de donnée **[NULL | not NULL]**
Dans une commande SELECT :
where *nom_colonne* **is [not] NULL**
Dans une commande UPDATE :
set *nom_colonne* **= {** *expression* **| NULL }**
Dans une commande INSERT :
values ({ *contante* **| NULL } [, {** *constante* **| NULL }] ...)**

Opérateurs

Opérateurs arithmétiques

Symbole	Signification
+	addition
-	soustraction
*	multiplication
/	division
%	modulo (extension TRANSACT-SQL)

Opérateurs binaires

Symbole	Signification
&	ET binaire (2 opérandes)
|	OU binaire (2 opérandes)
^	OU Exclusif binaire (2 opérandes)
~	NON binaire (1 opérande)

Opérateurs de comparaison

Symbole	Signification
=	égal à
<	Supérieur à
>	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à
<>	différent de
!=	non égal à (extension TRANSACT-SQL)
!>	non supérieur à (extension TRANSACT-SQL)
!<	non inférieur à (extension TRANSACT-SQL)

Opérateurs logiques

Opérateur	Fonction
NOT	Permet de sélectionner des lignes qui ne répondent pas à des critères de recherche.
AND	Opérateur logique ET.
OR	Opérateur logique OU.

Priorité des opérateurs logiques

Si le nombre d'expressions logiques est important, le résultat dépend de l'ordre dans lequel les opérations élémentaires s'effectuent. Cet ordre est défini par la priorité des opérateurs.

SQL effectue d'abord les comparaisons, puis les NOT, puis les AND et en dernier les OR.

Pour plus de sûreté, il est conseillé d'employer des parenthèses afin de grouper les expressions.

Opérateur UNION

Retourne un résultat unique combinant les résultats de deux requêtes ou plus.

L'union de plusieurs tables aboutit à une table logique qui unit les lignes renvoyées par chacune des sélections.

Syntaxe

select liste_de_selection **[** clause **into]**
[clause **from] [** clause **where]**
[clause **group by] [** clause **having]**
[union [all]
select liste_de_selection
[clause **from] [** clause **where]**
[clause **group by] [** clause **having]] ...**
[clause **order by]**
[clause **compute]**

Exemple:

Enoncé:
« *Sélectionner les auteurs du 19ème siècle et les auteurs des livres dont le titre commence par un 'E'.* »

Requête:

```
SELECT nom_auteur FROM AUTEURS WHERE siecle = 19
UNION
SELECT nom_auteur FROM LIVRES WHERE titre LIKE 'E%'
```

Retour:

nom_auteur
BALZAC
DUMAS

Remarques: Les lignes en double sont éliminées du résultat tant que le mot clé ALL n'est pas spécifié.
Les tables renvoyées par les sélections successives doivent contenir le même nombre de colonnes; ces colonnes doivent avoir des caractéristiques identiques.

Tables

Dans une base de données relationnelle, les données sont organisées en tables.

Une table contient les données relatives à une classe particulière d'objets.

Les tables sont constituées de lignes (ou enregistrements) et de colonnes (ou attributs).

Chaque colonne a un nom.

Chaque colonne contient une propriété de l'objet concerné par la table.

Chaque colonne contient des données d'un type unique.

Chaque ligne contient les données relatives à une occurrence de l'objet concerné par la table.

Un nom de table, un nom de colonne et une ligne déterminent un élément unique.

Les commandes associées à la définition de tables sont:

- CREATE TABLE (création)
- ALTER TABLE (modification de la structure)
- DROP TABLE (suppression)

Les commandes principales associées à la modification du contenu de tables sont:

- INSERT (ajout)
- UPDATE (modification)
- DELETE (effacement)

Jointures

Les jointures comparent deux tables (ou vues) ou plus en spécifiant une colonne pour chaque table, comparant les valeurs de ces colonnes ligne par ligne et concaténant les lignes ayant des valeurs correspondantes.

Une jointure peut être incluse dans une commande SELECT, UPDATE, INSERT, DELETE, ou une sous-requête. Les clauses et autres conditions de recherche peuvent accompagner une jointure.

Utilisation des jointures

Toutes les tables doivent être listées dans la clause FROM.
La clause WHERE spécifie la condition de jointure.
Les noms de colonnes ambiguës doivent être préfixés par le nom de la table.

Exemple:

Enoncé:

« *Sélectionner les livres (titre, numéro d'édition, distributeur, prix) dont le prix est inférieur à 30 francs.* »

Requête:

```
SELECT   titre, num_edition, distributeur, prix
FROM   LIVRES, TARIFS
WHERE   LIVRES.reference = TARIFS.reference
AND   prix < 30
```

Retour:

Titre	num_edition	distributeur	prix
Les chouans	1	X	25
Les justes	1	X	20
The case book of Sherlock Holmes	1	Y	14
The case book of Sherlock Holmes	1	X	17

Jointures externes

Une jointure externe permet d'afficher les lignes d'une table même si elles n'ont aucune correspondance avec une autre table.

Les jointures externes sont spécifiées en introduisant le caractère ***** du coté du signe **=** de la table pour laquelle on veux afficher toutes les lignes.

Exemple:

Enoncé:

« *Sélectionner tous les livres (titre, référence, numéro d'édition, date d'édition) avec leur date de deuxième édition s'il y en a.*

Requête:

```
SELECT   titre, LIVRES.reference, num_edition, date_edition
FROM   LIVRES, EDITIONS
WHERE   LIVRES.reference *= EDITIONS.reference
AND   num_edition = 2
```

Retour:

titre	reference	num_edition	date_edition
Eugénie Grandet	L01		
L'amour	L02		
La peste	L03	2	08/07/96
Les justes	L04		
Les chouans	L05	2	28/08/82
Les trois mousquetaires	L06		
The case book of Sherlock Holmes	L07		

Restriction: La table interne (la plus éloignée du signe *) ne doit pas apparaître dans une autre condition de jointure de la clause WHERE.

Procédures stockées

Une procédure stockée est un suite d'ordres SQL stockée dans la base, pouvant être exécutée par l'appel de son nom.

Les procédures peuvent recevoir et renvoyer des paramètres, retourner des valeurs et appeler d'autres procédures.

Les procédures s'exécutent généralement plus rapidement que les mêmes instructions lancées de manière interactive ou à partir d'un batch.

L'utilisation de procédures réduit le trafic réseau.

Gestion des procédures

Les procédures sont créées, supprimées et exécutées respectivement par les commandes suivantes: CREATE PROC, DROP PROC, EXEC

Syntaxe de création (*simplifiée*):

create proc *nom_procédure*
as *requête_SQL*
return

Syntaxe de suppression:

drop proc *nom_procédure*

Syntaxe d'exécution:

[exec] *nom_procédure*

Utilisation de paramètres

Les paramètre améliorent la flexibilité des procédures.

Les noms, les types et les valeurs par défaut des paramètres sont définis lors de la création de la procédure.

Les valeurs des paramètres sont spécifiées par l'appelant lorsque la procédure s'exécute.

Syntaxe de création (*complète*):

CREATE PROC - Syntaxe de création

create proc *nom_procédure*
[(**@***nom_paramètre1* *type_paramètre1* **[** = *valeur_par_défaut* **]**
[, **@***nom_paramètre2* *type_paramètre2* **[** = *valeur_par_défaut* **] ...])]**
as *requête_SQL*
return

Syntaxe d'appel:

EXEC - syntaxe de d'appel

[exec] *nom_procédure* **[** *valeur_paramètre1* **[,** *valeur_paramètre2* **] ...]**

Remarques: Une procédure peut avoir jusqu'à 255 paramètres.
Une valeur passée en paramètre peut contenir des caractères génériques.
Pour passer une variable locale en paramètre, il faut la spécifier par son nom à l'appel de la procédure.
Il est possible de spécifier les paramètres par leur nom (Si l'on commence à les spécifier par leurs noms, il faut continuer)

Exemple:

Enoncé:

« Appeler la procédure proc1 définie ci dessous »

Définition de la procédure :

```
CREATE PROC  PROC1  ( @a int, @b int, @c int, @d int )
AS  ...
RETURN
```

Appels:

```
EXEC   PROC1   24, 12, 18, 87
EXEC   PROC1   24, 12, @c=18, @d=87
EXEC   PROC1   24, 12, @d=87, @c=18
```

Utilisation de paramètres pour retourner des valeurs depuis une procédure.

Les paramètres retournés sont définis en utilisant le mot clé OUTPUT.
OUTPUT - Utilisation de paramètres pour retourner des valeurs depuis une procédure

La déclaration des paramètres en tant que « OUTPUT » doit être effectuée à la fois par la procédure appelée et par l'appelant.

Exemple:

Enoncé:

« Créer puis appeler une procédure proc2 qui compte de nombre de livres (les couples référence - numéro d'édition identifient les livres) proposés pour un distributeur donné. Le nom du distributeur et le retour de la procédure seront placés dans deux paramètres. »

Script de création de la procédure:

```
CREATE   PROC   PROC2
        ( @distributeur   varchar(50),   @livres   int   OUTPUT  )
AS
        SELECT   @livres = count(*)
        FROM     TARIFS
        WHERE    distributeur = @distributeur
RETURN
```

Appel de la procédure:

```
DECLARE   @X_livres   int
EXEC   PROC2   'X',  @X_livres   OUTPUT
```

Utilisation du code status retourné par une procédure

Chaque procédure retourne automatiquement un code status.

C'est le mot clé RETURN qui permet la transmission du code.
RETURN - Utilisation du code status retourné par une procédure

Le code status est du type int.

Les valeurs comprises entre -99 et 0 sont réservés par le serveur (spécifique SYBASE).

La valeur 0 correspond à un succès de la procédure.

Exemple:

Enoncé:

« Créer puis appeler une procédure proc3, semblable à proc2 (qui compte de nombre de livres (les couples référence - numéro d'édition identifient les livres) proposés pour un distributeur donné. Le nom du distributeur et le retour de la procédure étant placés dans deux paramètres.) et qui place la valeur 1 dans le code status si le nombre de livres est supérieur à zéro. »

Script de création de la procédure:

```
CREATE   PROC   PROC3
        ( @distributeur   varchar(50),   @livres   int   OUTPUT  )
AS
        SELECT   @livres = count(*)
        FROM     TARIFS
        WHERE    distributeur = @distributeur
        If @livres > 0
                RETURN  1
RETURN
```

Appel de la procédure:

```
DECLARE   @X_livres   int,   @status   int
EXEC   @status = PROC3   'X',  @X_livres   OUTPUT
```

Triggers

Un trigger est un type particulier de procédure stockée.

Les triggers sont automatiquement et obligatoirement déclenchés par le serveur quand des données d'un table spécifiée sont insérées, modifiées ou supprimées.

Ils ne peuvent pas être invoqués directement et ils ne prennent aucun paramètre.

Les triggers permettent de placer les contraintes d'intégrité dans la base plutôt que dans chaque application.

Gestion des triggers

Les triggers sont créés et supprimés respectivement par les commandes suivantes: CREATE TRIGGER, DROP TRIGGER
- Gestion des triggers

Syntaxe de création (*simplifiée*):

create trigger *nom_trigger*
on *non_table*
for { insert | update | delete }
[, {insert | update | delete }] ...
as *requête_SQL*

Syntaxe de suppression:

drop trigger *nom_trigger*

Fonctionnement d'un trigger

Un trigger peut reconnaître l'événement qui l'a déclenché en consultant les tables **inserted** et **deleted**.

Les tables **inserted** et **deleted** ont la même structure que la table modifiée.

inserted contient les lignes ajoutées à la table. Ces lignes sont le résultat d'un ordre INSERT ou UPDATE.
deleted contient les lignes supprimées dans la table. Ces lignes sont le résultat d'un ordre DELETE ou UPDATE.

Ces tables peuvent être référencées dans une jointure.

Un trigger peut savoir quelle sont les colonnes qui ont été modifiées en utilisant « IF UPDATE *test* ».

Syntaxe:

if update (*nom_colonne*)
[{ AND | OR } update (*non_colonne*)]...

La condition « **update** (*nom_colonne*) » est vrai si la colonne a été incluse dans la clause SET d'un ordre UPDATE, ou si une valeur non nulle a été insérée dans la colonne par un ordre INSERT.

Remarques: Lorsque plus d'un trigger est défini pour une action donnée sur une table donnée, le plus récemment défini est exécuté.
Lorsqu'une table est détruite, ses triggers sont supprimés.
Les ordres INSERT, UPDATE et DELETE à l'intérieur d'un trigger n'affectent pas le contenu des tables **inserted** ou **deleted** pour ce trigger.
Les triggers ne sont pas activés par les ordres TRUNCATE TABLE ou BULK COPY.

III. PRINCIPALES COMMANDES

ALTER TABLE

Cette commande ajoute de nouvelles colonnes à une table existante, ou ajoute, change, ou enlève des contraintes.

Syntaxe:

alter table [*base_de_données***. [** *propriétaire***].]** *nom_table*
{ add *nom_colonne type_de_donnée*
[default {*expression_constante* **|** *utilistaeur* **| null }]**
{ [{ identity | null }]
| [[constraint *nom_contrainte* **]**
{ { unique | primary key }
[clustered | nonclustered]
[with fillfactor = x] [on *nom_segment* **]**
| references [[*base_de_données***.]** *propriètaire***.]***ref_table*
[(*ref_colonne***)]**
| check (*condition_de_recherche***) }] } ...**
{ [, colonne_suivante] } ...

| add { [constraint *nom_contrainte***]**
{ unique | primary key }
[clustered | nonclustered]
(*nom_colonne* **[{,***nom_colonne* **} ...])**
[with fillfactor = x] [on *nom_segment* **]**
| foreign key (*nom_colonne* **[{,** *nom_colonne* **} ...])**
references [[*base_de_données***.]** *propriètaire***.]***ref_table*
[(*ref_colonne* **[{,** *ref_colonne* **} ...])]**
| check (*condition_de_recherche* **) }**

| drop constraint *nom_contrainte*

| replace *nom_colonne*
default **{** *expression_constante* **|** *utilisateur* **| null } }**

Uniquement sue des serveurs SQL sécurisés :
| set { maxhold [=] « label1 » , minhold [=] « label2 » }]

CREATE TABLE

La commande CREATE TABLE crée de nouvelles tables et, de manière optionnelle, des contraintes d'intégrité.

Syntaxe:

- *Syntaxe complète:*

create table [*base_de_données***. [** *propriètaire***].]** *nom_table*
(*nom_colonne***)** *type_de_donnée*
[default {*expression_constante* **|** *utilistaeur* **| null}]**
{[{identity | null | not null}]
| [[constraint *nom_contrainte***]**
{{unique | primary key}
[clustered | nonclustered]
[with fillfactor = x] [on *nom_segment***]**
| references [[*base_de_données***.]** *propriètaire***.]***ref_table*
[(*ref_colonne***)]**
| check (*condition_de_recherche***)}]}...**
| [constraint *nom_contrainte***]**
{{unique | primary key}
[clustered | nonclustered]
(*nom_colonne***[{,***nom_colonne* **}...])**
[with fillfactor = x] [on *nom_segment***]**
| foreign key (*nom_colonne***[{,***nom_colonne* **}...])**
references [[*base_de_données***.]** *propriètaire***.]** *ref_table*
[(*ref_colonne* **[{,** *ref_colonne* **}...])]**
| check (*condition_de_recherche***)}**
[{, {*colonne_suivante* **|** *contrainte_suivante***}}...])**
[on *nom_segment***]**

Uniquement sue des serveurs SQL sécurisés :

[with { maxhold [=] « label1 » , minhold [=] « label2 » }]

- *Syntaxe simplifiée:*

create table [*base_de_données***. [** *propriètaire***].]** *nom_table*
(*nom_colonne type_de_donnée* **[null | not null]**
[, *nom_colonne type_de_donnée* **[null | not null]]...)**
[on *nom_segment* **]**

Exemple:

Enoncé:

« *Créer la table COMANDES avec les champs ref_client, nom_client, reference, num_edition, quantité, date, ref_commande ayant pour styles respectivement varchar(8), varchar(25), varcher(3), tinyint, smallint, smalldatetime et varchar(12).* »

Requête:

```
CREATE   TABLE    COMMANDES
        (  ref_client          VARCHAR(8),
           nom_client          VARCHAR(25),
           reference           VARCHAR(3),
           num_edition         TINYINT,
           quantite            smallint,
           date                smalldatetime,
           ref_commande        VARCHAR(12)    )
```

DECLARE

La commande DECLARE crée une ou plusieurs de variables locales.

Les variables sont créées avec un nom et un type.

Les variables locales sont créées dans les batchs, les triggers ou les procédures.
Elles disparaissent quand leur batch, trigger ou procédure se termine.

Le nom des variables doit débuter par le caractère « @ ».

Les variables sont scalaires (elle ne contiennent qu'une seule valeur).

Elles sont automatiquement initialisées à NULL.

Syntaxe:

declare **@***non_variable* *type_de_donnée*
[, @*non_variable* *type_de_donnée* **] ...**

Exemple:

```
DECLARE   @quantite    int,
          @tva         float,
          @prix_ttc    smallmoney
```

Remarques: L'affectation des variables locales se fait avec la commande SELECT.
Une variable peut être le résultat d'une commande SELECT.
Il est recommandé de grouper les déclarations de variables dans la même instruction pour des raisons de performance.

DELETE

La commande DELETE efface une ou plusieurs lignes (enregistrements) d'une table ou d'une vue.

Syntaxe:

delete [[*base_de_données.* **]** *propriétaire***.] { ***nom_table* | *nom_vue* **}**
[from [[*base_de_données.* **]** *propriétaire***.] { ***nom_table* | *nom_vue* **}**
[, [[*base_de_données.* **]** *propriétaire***.] { ***nom_table* | *nom_vue* **}] ...]**
[where *conditions_de_recherche* **]**

Exemple:

Enoncé:

« *Effacer de la table AUTEURS, les auteurs du 19ème siècle.*»

Requête:

```
DELETE   FROM   AUTEURS
WHERE   siecle = 19
```

DROP TABLE

Enlève de la base la définition de la table et toutes ses données, ses index et triggers, et les permissions associées.

Syntaxe:

drop table [[*base_de_données***. [** *propriètaire***.]]** *nom_table*
[, [*base_de_données***. [** *propriètaire***.]]** *nom_table* **] ...]**

Exemple:

Enoncé:

« *Enlever la table TARIFS (et tous les index et triggers associés) de la base.* »

Requête:

```
DROP   TABLE   TARIFS
```

GRANT

Cette commande assigne une permission aux utilisateurs.

On distingue deux types de permissions : les accès aux objets et les permissions sur les commandes.

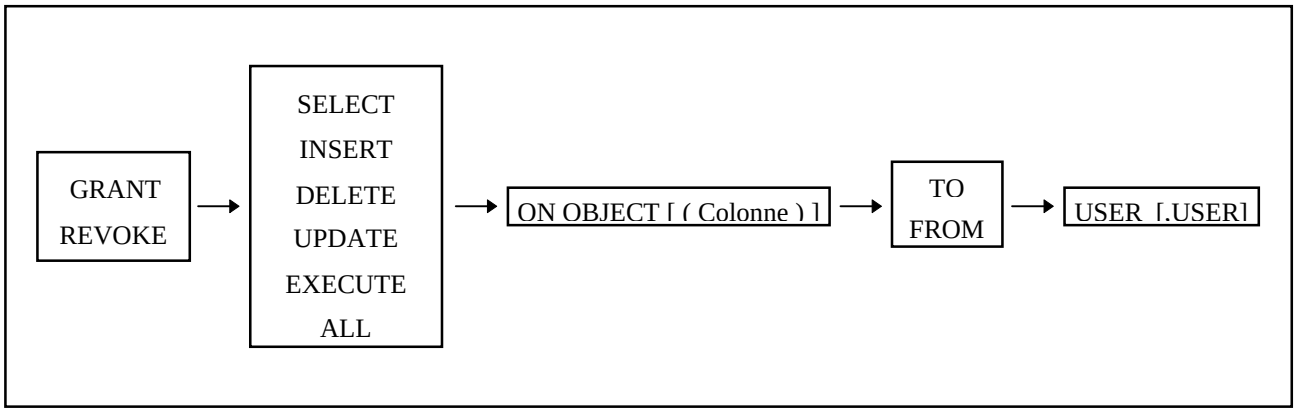
Permissions d'accès aux objets:

Le propriétaire d'un objet a les droits exclusifs pour modifier la structure de cet objet (CREATE INDEX, ALTER TABLE, DROP objet).

Le propriétaire d'un objet contrôle les accès des autres utilisateurs aux données de cet objet.

Aucun utilisateur ne peut accéder à un objet s'il n'a pas reçu les permissions sur cet objet.

Schéma de contrôle d'accès aux objets:



L'ordre chronologique détermine la permission résultante.

Pour exclure un petit nombre d'utilisateurs, donner les permissions à tout le monde, puis en exclure quelques uns.
Pour exclure un petit nombre de colonnes, donner les droits sur la table, puis exclure les colonnes concernées.

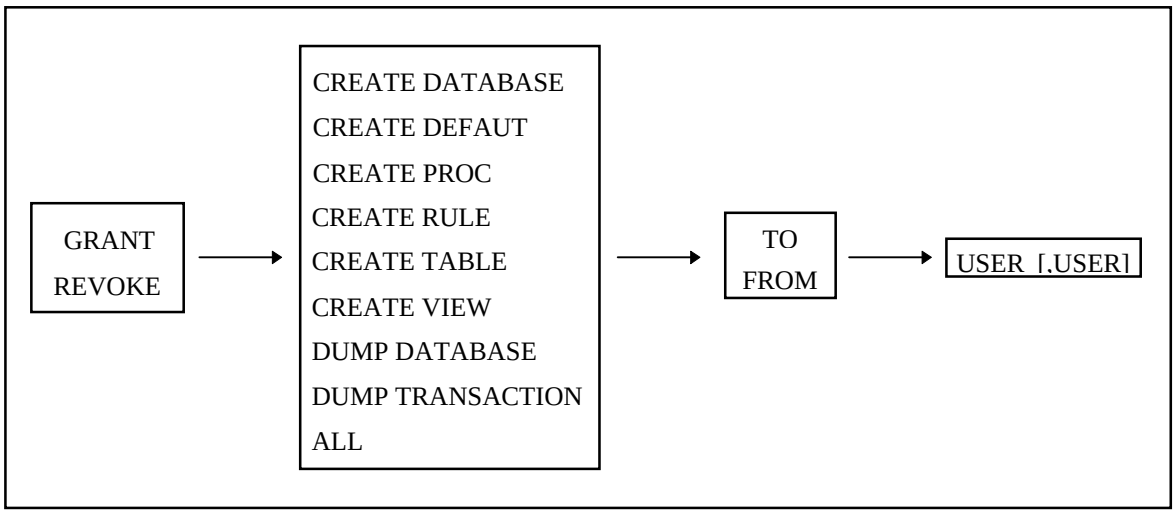
Permissions sur les commandes:

Seul l'administrateur peut lancer des commandes ayant des effets sur l'ensemble du serveur.

Seul le propriétaire de la base peut :

- Ajouter, supprimer ou modifier les comptes utilisateurs de la base.
- checkpoint
- dbcc
- load database
- load transaction
- setuser

Schéma de contrôle d'accès aux commandes:



Syntaxe:

- *permissions sur des objets:*

grant { all [*privilèges* **] |** *liste_de_permissions* **}**
on { ***nom_table* **[(*liste_de_colonnes* **)]**
| *nom_vue* **[(** *liste_de_colonnes* **)]**
| *nom_prcédure_stockée* **}**
to { public | *liste_de_noms* **|** *nom_role* **}**
[with grant option]

- *permissions sur des commandes:*

grant { all [*privilèges* **] |** *liste_de_commandes* **}**
to { public | *liste_de_noms* **|** *nom_role* **}**

Exemples:

Enoncé:

« *Autoriser la lecture de la table AUTEURS pour tout les utilisateurs.* »

Requête:

```
GRANT SELECT ON AUTEURS TO PUBLIC
```

Enoncé:

« *Autoriser toute sorte d'opérations sur la table AUTEURS pour l'utilisateur 'CHEF'.* »

Requête:

```
GRANT ALL ON AUTEURS TO CHEF
```

Enoncé:

« *Autoriser l'exécution de la procédure 'spsel_info' pour tous les membres du groupe 'GROUP_UTIL'.* »

Requête:

```
GRANT EXEC ON spsel_info TO GROUP_UTIL
```

Remarque: Voir la commande REVOKE.

INSERT

Ajoute un nouvel enregistrement (ou ligne) dans une table ou une vue.

Syntaxe:

insert [into] [*base***. [** *propriétaire***.]] {** *nom_table* | *nom_vue* **}**
[(*liste_colonne* **)]**
{ values (*expression* **[,** *expression* **] ...)**
| *commande_select* **}**

Exemple:

Enoncé:
« *Insérer la ligne ci-dessous dans la table TARIFS.* »

reference	***num_edition***	***distrubuteur***	***prix***
L03	1	Z	41

Requête:

```
INSERT   INTO   TARIFS
( reference, num_edition, distributeur, prix )
VALUES
('LO3' , 1, 'Z', 41)
```

REVOKE

Cette commande enlève une permission aux utilisateurs.

La syntaxe de le commande REVOKE est similaire à celle de la commande GRANT.

Remarque: Voir la commande GRANT.

SELECT

Utilisation de SELECT pour des interrogations de tables.

La commande SELECT, permet, entre autre, de sélectionner des champs (colonnes) d'une table.

Syntaxe:

- *Syntaxe complète:*

select [all | **distinct]** *liste _select*
[into [[*base_de_données***.]** *propriétaire***.]** *nom_table* **]**
[from [[*base_de_données***.]** *propriètaire***.] {** *nom_table* | *nom_vue* **}**
[holdlock | **noholdlock] [shared]**
[, [[*base_de_données***.]** *propriètaire***.] {** *nom_table* | *nom_vue* **}**
[holdlock | **noholdlock] [shared]] ...]**
[where *conditions_de _recherche* **]**

[group by [all] *expression_sans_agrégation*
[, *expression_sans_agrégation* **] ...]**
[having *conditions_de _recherche* **]**

[order by
{ [[[*base_de_données***.]** *propriètaire***.] {** *nom_table***.** | *nom_vue***. }]**
nom_colonne| *numéro_liste_select* | *expression* **}**
[asc | **desc]**
[, { [[[*base_de_données***.]** *propriètaire***.] {** *nom_table* | *nom_vue***. }]**
nom_colonne| *numéro_liste_select* | *expression* **}**
[asc | **desc]] ...]**
[compute *agrégation_colonne* **(** *nom_colonne***)**
[, *agrégation_colonne* **(** *nom_colonne***)] ...**
[by *nom_colonne***[,** *nom_colonne***[,** *nom_colonne***] ...]]**
[for browse]

- *Syntaxe simplifiée:*

select *nom_colonne*
[, *nom_colonne* **]...**
from *nom_table*
where *critères_de_selection*

Exemple:

Enoncé:
« *Sélectionner tous les éléments de la tables AUTEURS* »

Requête:

```
SELECT * FROM AUTEURS
```

Remarques: L'ordre des colonnes est donné dans la commande SELECT. En cas d'utilisation du caractère « * », c'est l'ordre de définition des colonnes lors de la création de la table qui est donné.
Voir le chapitre Expression des sélections.

Utilisation de SELECT pour assigner des valeurs aux variables locales.

Syntaxe:

select **@***nom_variable* = **{** *expression* | *commande_de_sélection* **}**
[, @*nom_variable* = **{** *expression* | *commande_de_sélection* **} ...]**
[from *liste_noms_table* **]**
[where *critères_de_selection* **]**
[group by *liste_de_groupements* **]**
[having *conditions_de _recherche* **]**
[order by *liste_de_critères_de_tri* **]**
[compute *liste_de_fonctions* **[by** *liste_de_colonnes* **]]**

Exemple:

Enoncé:

« *Assigner la valeur 10 à la variable locale @quantité* »

Requête:

```
SELECT   @quantité = 10
```

Remarque: Si aucune valeur n'est renvoyée par la commande SELECT, la valeur de la variable reste inchangée. Si plusieurs valeurs sont retournées, c'est la dernière qui est assigné à la variable.
Les commandes SELECT « d'assignation » ne retournent aucune valeur à l'utilisateur.
Les commandes SELECT « d'assignation » ne peuvent pas être combinées avec des sélections retournant des données à l'utilisateur.
Pour de meilleurs performances, essayer de grouper plusieurs assignations dans la même instruction.

SET

La commande SET définit la manière dont sont traitées les requêtes par le serveur.

Elle peut modifier le comportement dans une session interactive ou à l'intérieur d'une procédure ou d'un trigger.

Elle peut être utilisée pour afficher des statistiques sur les stratégies de recherche.

Syntaxe: (simplifiée)

set { { arithabort | arithignore | nocount | noexec | parseonly |
showplan | statistics io | statistics time }
{ on | off } | rowcount number }

Options de la commande SET:

Option	Description de la condition 'on'
arithabort on	Echoue quand il y a overflow ou division par zéro.
arithignore on	Retourne NULL quand il y a overflow ou division par zéro.
nocount on	Evite l'affichage de « (n rows affected) »; **@@rowcount** est toujours mis à jour.
noexec on	Vérifie la syntaxe et crée un plan d'exécution, mais n'exécute pas. A utiliser avec **set showplan** pour voir le plan d'exécution.
parseonly on	Vérifie la syntaxe. N'exécute pas.
rowcount n	Retourne seulement *n* lignes. Si *n* = 0, toutes les lignes sont retournées.
showplan on	Affiche le moyen choisi par le serveur pour traiter la requête puis l'exécute.
statistics io on	Pour les tables : affiche le nombre de balayages, d'accès aux pages (lectures logiques) et d'accès disque (lectures physiques). Pour les commandes : affiche le nombre de pages écrites.
statistics time on	Pour les commandes : affiche le temps de parsing et de compilation. Pour chaque étape de la commande : affiche le temps d'exécution.

Remarque: Les options fixées par la commande SET ont effet pendant la durée de la procédure et sont réinitialisées à sa fin.

UPDATE

Change le contenu de colonnes existantes soit en ajoutant soit en modifiant des données.

Syntaxe

update [[*base_de_données***.] ***propriétaire***.] {** *nom_table* | *nom_vue* **}**
set [[[*base_de_données***.]** *propriétaire***.] {***nom_table***.** | *nom_vue***. }]**
nom_colonne1 **= {** *expression1* | **NULL** | **(***commande_select***) }**
[, *nom_colonne2* **= {** *expression2* | **NULL** | **(** *commande_select* **) }] ...**
[from [[*base_de_données***.]** *propriétaire***.] {** *nom_table* | *nom_vue* **}**
[, [[*base_de_données***.]** *propriétaire***.] {** *nom_table* | *nom_vue* **}] ...]**
[where *conditions_de _recherche* **]**

Exemple:

Enoncé:

« *Augmenter les prix (table TARIFS) du distributeur X de 5 francs.* »

Requête:

```
UPDATE   TARIFS
SET   prix = prix + 5
WHERE   distributeur = 'X'
```

IV. EXPRESSION DES SELECTIONS

Format général d'une sélection

Une commande de sélection contient les clauses suivantes :

```
SELECT ...
        FROM ...
                WHERE ...
                        GROUP BY ...
                                HAVING ...
                                        ORDER BY ...
```

Seules les clauses SELECT et FROM sont obligatoires.

Clauses d'accompagnement

COMPUTE

La clause COMPUTE résume des valeurs dans une commande SELECT avec des fonctions d'agrégation linéaires (**sum**, **avg**, **min**, **max** et **count**).

Les valeurs résumées apparaissent dans des lignes additionnelles dans le retour de la requête (Contrairement aux résultats des fonctions d'agrégation qui apparaissent dans de nouvelles colonnes.)

Il existe deux méthodes pour effectuer des résumés: COMPUTE et COMPUTE BY.

On utilise COMPUTE pour générer des résumés complets.

On utilise COMPUTE BY pour créer des résumés sur les ruptures.

Syntaxe:

compute fonction_d_agrégation_linéaire **(** nom_colonne **)**
[, fonction_d_agrégation_linéaire **(** nom_colonne **)]...**
[by nom_colonne **[,** nom_colonne] **...]**

Exemples:

- Calcul des totaux complets à l'aide d'un clause COMPUTE.

Enoncé:

« *Afficher toutes les éditions (date, référence, numéro d'édition, nombre de pages) et le nombre de page que représente la somme de toutes ces éditions.* »

Requête:

```
SELECT   date_edition, reference, num_edition, nbr_pages
FROM   EDITIONS
COMPUTE   sum (nbr_pages)
```

- Calcul des sous-totaux en utilisant COMPUTE BY en association avec une clause ORDER BY.

Enoncé

« *Afficher toutes les éditions (date, référence, numéro d'édition, nombre de pages) triées par référence et le nombre de page que représente ces éditions par référence.* »

Requête:

```
SELECT   date_edition, reference, num_edition, nbr_pages
FROM   EDITIONS
ORDER BY   reference
COMPUTE   sum (nbr_pages)   BY  reference
```

- Génération à la fois d'un total complet et des sous-totaux en utilisant COMPUTE et COMPUTE BY.

Enoncé:

« *Afficher toutes les éditions (date, référence, numéro d'édition, nombre de pages) triées par référence, le nombre de page que représente ces éditions par référence et le nombre de page que représente la somme de toutes ces éditions.* ». »

Requête:

```
SELECT   date_edition, reference, num_edition, nbr_pages
FROM   EDITIONS
ORDER BY   reference
COMPUTE   sum (nbr_pages)   BY  reference
COMPUTE   sum (nbr_pages)
```

Remarques: On ne peut résumer que les colonnes apparaissant dans la liste des colonnes sélectionnées.
Il est permis d'utiliser toutes les fonctions d'agrégation sauf **count** (*).

GROUP BY et HAVING

Grouper les lignes

La clause GROUP BY organise les lignes en groupes sur la base du contenu de ces lignes.

Chaque agrégat contenu dans la liste SELECT deviendra un agrégat pour le groupe.

Syntaxe:

group by [all] expression_sans_agrégation
[, expression_sans_agrégation **] ...**
[having conditions_de _recherche **]**

Exemples:

Enoncé:
« *Afficher le prix moyen des livres, par livre.* »

Requête:

```
SELECT   reference, AVG (prix)   FROM   TARIFS
GROUP BY   reference
```

Enoncé:
« *Afficher le prix moyen des livres, par livre et par numéro d'édition.* »

Requête:

```
SELECT   reference, num_edition, AVG (prix)   FROM   TARIFS
GROUP BY   reference, num_edition
```

Remarques: Le regroupement ne peut pas se faire sur un titre de colonne.
Les valeurs nulles dans une clause GROUP BY sont traitées comme un groupe.

Restriction sur les lignes utilisées pour construire le groupe

Pour n'utiliser qu'une partie des lignes afin de construire un groupe, il faut une clause **WHERE**.

Exemple:

Enoncé:

« *Afficher le prix moyen de la première édition des livres, par livre.* »

Requête:

```
SELECT   reference, AVG (prix)   FROM   TARIFS
WHERE   num_edition = 1
GROUP BY   reference
```

Pour afficher tous les groupes (même ceux n'ayant aucune ligne correspondante), utilisez le mot clé **ALL.**

Exemple:

Enoncé:

« *Afficher le prix moyen de la deuxième édition des livres, par livre, en affichant tous les livres.* »

Requête:

```
SELECT   reference, AVG (prix)   FROM   TARIFS
WHERE   num_edition = 2
GROUP BY   ALL   reference
```

Extensions Sybase:

- Les expressions nommées dans le GROUP BY n'ont pas besoin d'apparaître dans la clause SELECT.
- Des expressions non mentionnées dans le GROUP BY peuvent apparaître dans la clause SELECT.

Restrictions sur les groupes

Il est possible de restreindre le nombre de groupes retournés en utilisant une clause HAVING.

Une clause HAVING inclus la plupart du temps une fonction d'agrégat.

Exemples:

Enoncé:

« Sélectionner *les livres vendus à une moyenne de plus de 30 francs (regroupés par livres).* »

Requête:

```
SELECT   reference, AVG (prix)   FROM   TARIFS
GROUP BY   reference
HAVING   AVG (prix) > 30
```

Enoncé:

« Sélectionner *les livres vendus à une moyenne de plus de 30 francs lors de leur première édition (regroupés par livres).* »

Requête:

```
SELECT   reference, AVG (prix)   FROM   TARIFS
WHERE   num_edition = 1
GROUP BY   reference
HAVING   AVG (prix) > 30
```

Résumé: WHERE restreint les lignes, HAVING restreint les groupes.

ORDER BY

La clause ORDER BY permet de trier les résultats suivant l'ordre ascendant (mot clé ASC) ou descendant (mot clé DESC) d'un ou plusieurs attributs.

Syntaxe:

order by { [nom_table. | nom_vue. **]** nom_colonne
| numéro_dans_liste_de_sélection | expression **} [asc** | **desc]**
[, { [nom_table. | nom_vue. **]** nom_colonne | numéro_dans_liste_de_sélection |
expression **} [asc** | **desc]] ...**

Remarques: L'attribut sur lequel se fait le tri doit obligatoirement faire partie de la liste des attributs dans la commande SELECT.
Les valeurs indéterminées (NULL) sont affichées ensemble, avant ou après les autres, suivant les S.G.B.D. et le sens croissant ou décroissant du tri.

WHERE

La clause WHERE permet de spécifier un critère de sélection.

La condition dans une clause WHERE est une expression logique composée d'une suite de conditions, prenant chacune la valeur 'vrai' ou 'faux'.

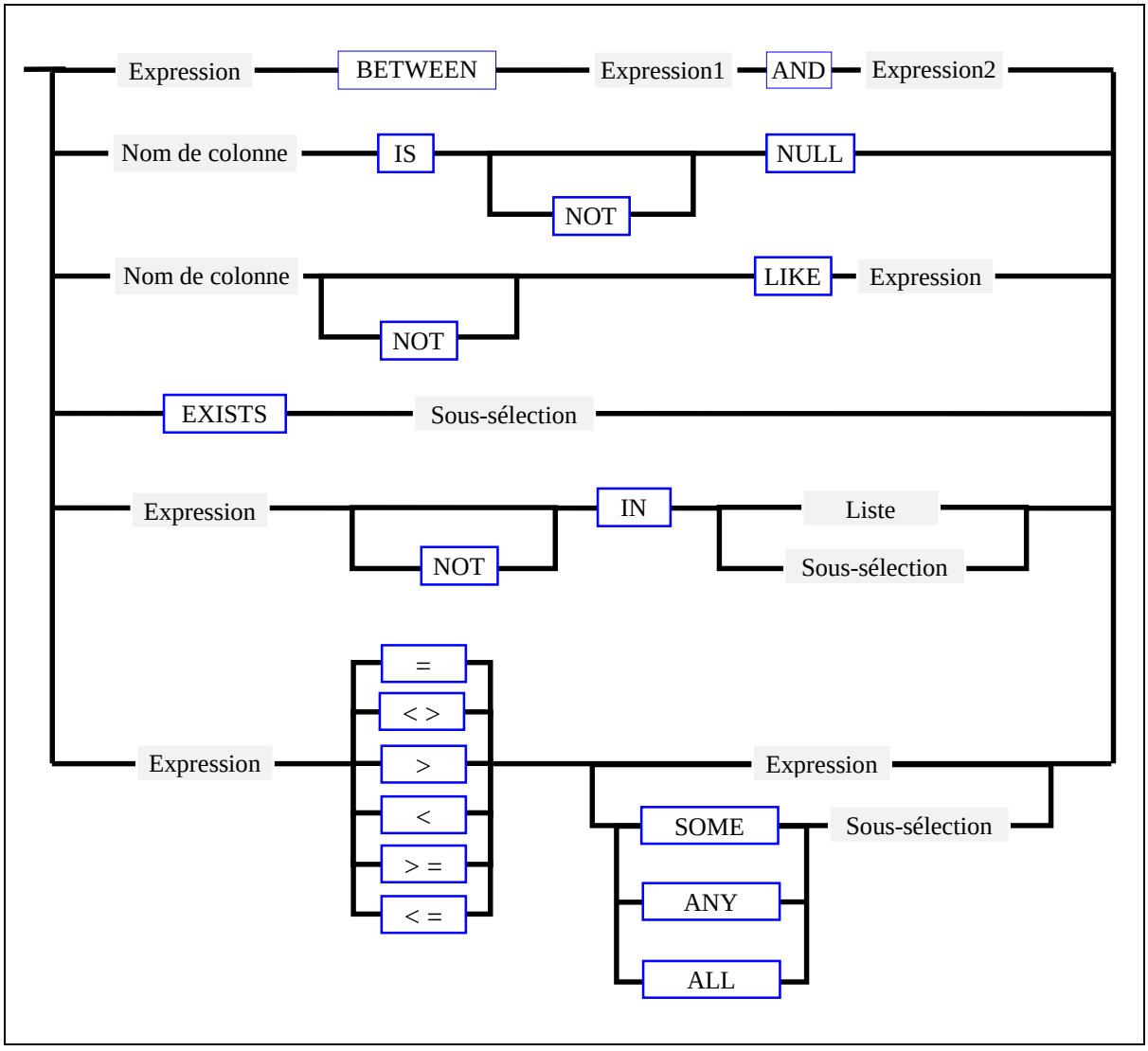
Ces conditions sont combinées entre elles par les opérateurs logiques AND, OR ou NOT. Si la valeur de l'expression logique est 'vrai' pour un tuple considéré, celui-ci fera partie du résultat.

Une condition peut prendre une des formes suivantes:

- comparaison à une valeur **=, <, >, <=, >=, <>**
- comparaison à une fourchette de valeurs BETWEEN
- comparaison à une liste de valeurs IN
- comparaison à un filtre LIKE
- test sur l'indétermination d'une valeur IS NULL
- test 'TOUS' ou 'AU MOINS UN' ALL, ANY
- test existentiel EXISTS

Prédicats de sélection

Il est possible d'utiliser les opérateurs (ou prédicats) suivants pour définir une clause aussi sophistiquée que nécessaire pour extraire les lignes souhaitées :



ALL

Utilisé avec un opérateur de comparaison, ALL sert à tester si une expression est vérifiée dans tous les cas de figure.

L'expression est juste si la comparaison est vérifiée pour toutes les valeurs renvoyées par la clause « sous-sélection ».

Exemple:

Enoncé:

« *Sélectionne la référence de tous les livres vendus si tous les prix sont inférieurs à 100 Francs.* »

Requête:

```
SELECT DISTINCT reference
FROM TARIFS
WHERE 100 > ALL ( SELECT DISTINCT prix FROM TARIFS )
```

ANY

Contrairement à ALL, l'expression est juste si la comparaison est vérifiée dans la clause de sous-sélection pour au moins une valeur.

Exemple:

Enoncé:

« *Sélectionne la référence de tous les livres vendus si au moins un prix est supérieur à 50 Francs.* »

Requête:

```
SELECT DISTINCT reference
FROM TARIFS
WHERE 50 < ANY ( SELECT DISTINCT prix FROM TARIFS )
```

BETWEEN

Ce prédicat est utilisé pour vérifier si la valeur d'une expression est comprise entre deux valeurs.

Syntaxe:

Expression1 [NOT] BETWEEN Expression 2 AND Expression3

Exemple:

Enoncé:

« *Sélectionne les livres (référence) vendus entre 30 et 50 Francs.* »

Requête:

```
SELECT DISTINCT reference
FROM TARIFS
WHERE prix BETWEEN 30 AND 50
```

DISTINCT

La clause DISTINCT permet d'éviter d'obtenir des doublons dans le résultat d'une projection.

Remarque: Une projection en SQL avec omission de la clause DISTINCT n'élimine pas les doubles.

EXISTS

Permet de tester si le résultat d'une sélection contient au moins une ligne (c'est-à-dire si la sélection n'est pas vide).

Syntaxe:

Exists (*sélection* **)**

Exemple:

Enoncé:

« *Si il existe des auteurs du 19ème siècle, sélectionner tous les auteurs (tous les attributs).* »

Requête:

```
SELECT * FROM AUTEURS
WHERE EXISTS ( SELECT * FROM AUTEURS WHERE siecle = 19 )
```

IN

IN permet de sélectionner des lignes contenant une valeur coïncidant avec l'une des valeurs d'une liste.

Syntaxe:

expression **[NOT] IN { ***liste* **|** *sélection* **}**

Exemples

Enoncé:

« *Sélectionner tous les auteurs du 18ème et tous les auteurs du 19ème siècle.* »

Requête:

```
SELECT * FROM AUTEURS
WHERE siecle IN ( 18, 19 )
```

LIKE

La clause LIKE désigne la recherche d'une constante dans une chaîne de caractères.
Elle est utilisable avec les données de type CHAR, VARCHAR, TEXT et DATETIME.

Syntaxe:

LIKE **'**chaine de caractères**%'**

LIKE **'%**chaine de caractères**'**

LIKE **'%**chaine de caractères**%'**

LIKE **'**chaine1_chaine2**'**

LIKE **'[**chaine1**]**chaine2**'**

LIKE **'[~**chaine1**]**chaine2**'**

Le caractère générique % signifie: « N'importe quelle séquence de zéro ou plusieurs caractères peut suivre ou précéder la chaîne indiquée. »

Le caractère générique _ signifie un seul caractère quelconque.

Les caractères génériques [ ] signifient n'importe quel caractère (unique) parmi ceux spécifiés antre les crochets.

Les caractères génériques [~] signifient n'importe quel caractère (unique) excepté un de ceux spécifiées antre les crochets.

Exemples:

- LIKE ‘REL%’ sélectionne les chaînes qui commencent par REL.
- LIKE ‘%ZI’ sélectionne les chaînes qui se terminent par ZI..
- LIKE ‘%PARIS%’ sélectionne sur la présence de la constante PARIS, à n’importe quelle position dans la chaîne.
- LIKE ‘%FAO_’ sélectionne sur la présence d’une constante ‘FAO’ à l’avant dernière place des chaînes consultées
- LIKE ‘[FGT]AO’ sélectionne sur la présence du caractère F, G, ou T au début de la chaîne consultée, et sur la présence des caractères A et O respectivement à la deuxième et troisième place de la chaîne, cette chaîne comportant trois caractères.

Remarque: Pour sélectionner les tuples dont un attribut contient une valeur ne correspondant pas au filtre, il faut employer l’opérateur NOT LIKE.

NULL

Ce prédicat permet de tester la valeur NULL.

Syntaxe:

nom_colonne ***IS [NOT] NULL***

Exemple:

Enoncé:

« S*électionner toutes les éditions dont le nombre de page n'aurait pas été renseigné (c'est à dire que le nombre de pages est NULL).* »

Requête:

```
SELECT   *
FROM   EDITIONS
WHERE   nbr_pages   IS   NULL
```

SOME

Contrairement à ALL, l'expression est juste si la comparaison est vérifiée dans la clause de sous-sélection pour au moins une valeur.

(*Voir:* ANY)

V. PROGRAMMATION STRUCTUREE

Groupes d'instructions

Il est possible de spécifier au serveur de traiter un groupe d'instruction SQL comme un bloc en utilisant une structure « BEGIN ... END »

Un bloc d'instructions est traité comme un ordre SQL.

Syntaxe:

BEGIN
bloc d'instructions
END

Exemple:

Enoncé:
« *Si la moyenne des prix des livres est supérieure à 35 francs diminuer les prix supérieurs à 50 Francs de 10% et les prix inférieurs ou égaux à 50 francs de 5 %.* »

Requête:

```
IF  ( SELECT  AVG( prix )  FROM  TARIFS  ) > 35
      BEGIN
            UPDATE  TARIFS
                  SET  prix = prix * 0.95
                  WHERE  prix <= 50

            UPDATE  TARIFS
                  SET  prix = prix * 0.9
                  WHERE  prix > 50
      END
```

Exécution Conditionnelle

La structure « IF ... ELSE » permet d'exécuter une requête SQL selon une condition basée sur un test.

Syntaxe:

IF *expression_booléenne1*
instruction_si_condition_vraie
[ELSE [IF *expression_booléenne2* **]**
instruction_si_condition1_fausse **]**

Exemple:

Enoncé:
« *Si la moyenne des prix des livres est supérieure à 35 francs diminuer les prix de 10%.* »

Requête:

```
IF  ( SELECT  AVG( prix )  FROM  TARIFS  ) > 35
      UPDATE  TARIFS
            SET  prix = prix * 0.9
```

Remarques: Il n'y a pas de limite dans le nombre de IF imbriqués.
Si les expressions booléens contiennent des commandes SELECT, elles doivent être mises entre parenthèses.
Sans une structure « BEGIN ... END », une seule requête SQL peut suivre un « IF » ou un « ELSE ».

Exécution Répétitive

La structure « WHILE ... BREAK ... CONTINUE » provoque une exécution répétitive.

Le mot clé « WHILE »marque le début de la boucle. La condition de maintient dans la boucle est indiquée immédiatement après.

Le mot clé « BREAK » provoque la sortie de la boucle. Si des boucles sont imbriquées, la sortie ne s'effectue que sur un niveau.

« CONTINUE » renvoi l'exécution au début de la boucle.

Syntaxe:

WHILE *expression_booléenne*
bloc_d'instruction

Exemple:

Enoncé:
« Tant que la moyenne des prix des livres est supérieure à 30 francs diminuer les prix de 10%. »

Requête:

```
WHILE (  SELECT   AVG ( prix ) FROM   TARIFS  ) > 30
        BEGIN
                UPDATE TARIFS
                        SET   prix = prix * 0.9
        END
```

Exécution Evénementielle

La commande WAITFOR Evénementiellesuspend l'exécution en attendant un événement spécifié.

Syntaxe:

WAITFOR { delay '*durée***' | time '***heure***' | errorexit | processexit }**

delay :	Suspend l'exécution pendant une durée donnée (jusqu'à 24 heures).
time :	Suspend l'exécution en attendant un heure donnée de la journée (les dates ne peuvent pas être utilisées).
errorexit:	Suspend l'exécution jusqu'à ce qu'un processus se termine anormalement.
processexit:	Suspend l'exécution e attendant qu'un processus se termine.
mirrorexit:	Si le mirroring échoue sur un ou plusieurs disques.

Exemple:

Enoncé:

« Faire une boucle qui augmente des prix de 10% toutes des 8 heures. »

Requête:

```
WHILE   1 = 1
        BEGIN
                WAITFOR DELAY  '8:00:00'
                UPDATE TARIFS
                        SET   prix = prix * 1.1
        END
```

VI. FONCTIONS

Fonctions d'agrégation

Les fonctions d'agrégation (**sum**, **avg**, **count**, **count**(*), **max** et **min**) sont utilisées pour retourner les valeurs récapitulatives. Elles peuvent être utilisées dans une liste de sélection, dans la clause HAVING d'une commande SELECT, dans une sous-requête, ou encore dans une commande incluant une clause GROUP BY.

Fonction d'agrégation	Retour
sum([all | distinct] expression)	Total des valeurs (distinctes) dans la colonne numérique.
avg([all | distinct] expression)	Moyenne des valeurs (distinctes) dans la colonne numérique.
count([all | distinct] expression)	Nombre de valeurs (distinctes) non-nulles dans la colonne.
count(*)	Nombre de lignes sélectionnées.
max(expression)	Plus grande valeur dans l'expression.
min(expression)	Plus petite valeur dans l'expression.

Remarques: Il existe des fonctions d'agrégation similaires, appelées fonction d'agrégation linéaire, utilisées dans la clause COMPUTE.
Pour les serveurs SQL sécurisés, on ne peut utiliser les fonctions **sum**, **avg**, **max** ou **min** avec des types sécurisés. Néanmoins il est possible d'utiliser **count** ou **count**(*) avec des types sécurisés.

Fonctions de conversion de types

Les fonctions de conversion de types changent des expressions d'un type en un autre et appliquent de nouveaux formats d'affichage pour les dates et les heures.

Les serveurs SQL disposent de trois fonctions de conversion de types (**convert**(), **inttohex**(), et **hextoint**()) qui sont utilisées dans les listes de sélection, dans les clauses WHERE et partout où une expression est utilisée.

Fonction	Paramètres	Retour
convert	**(** type **[(**taille**) | (** précision **[**,échelle**])]**, expression **[**,style**])**	Conversions entre une large variété de types et remise en formes des données calendaires et horaires
hextoint	**(**chaine_hexadécimale**)**	Retourne un entier indépendant de la configuration, équivalent à la chaîne hexadécimale.
inttohex	**(**expression_entière**)**	Retourne un hexadécimal indépendant de la configuration, équivalent à la configuration.

Fonctions de manipulation de dates

Fonction	Paramètres	Retour
getdate	( )	Retourne la date et l'heure courante du système.
datename	(élément_date,date)	Retourne le nom de l'élément spécifié (ex.: le mois « JUIN ») d'une donnée de type **datetime** sous forme de chaîne de caractères.
datepart	(élément_date, date)	Retourne une valeur entière pour l'élément spécifié d'une donnée de type **datetime**.
datediff	(élément_date, date1, date2)	Retourne *date2* - *date1* dans le format de l'élément spécifié.
dateadd	(élément_date, expression_numérique, date)	Retourne la date produite par l'addition du nombre spécifié (concernant l'élément de la date indiqué) et de la date passée en paramètre. L'expression numérique peut être de n'importe quel type numérique; la valeur est alors tronquée en valeur entière.

Fonctions mathématiques

Les fonction mathématiques retournent des valeurs communément utilisées pour les opérations sur des données mathématiques.
Les noms des fonctions mathématiques ne sont pas des mots clés.

Fonction	Paramètres	Retour
abs	(valeur_numérique)	Retourne la valeur absolue d'un expression donnée. Le résultat est du même type, a la même précision et la même échelle que l'expression numérique.
acos	(valeur_numérique_arrondie)	Retourne l'angle (en radians) dont le cosinus est la valeur spécifiée en paramètre.
asin	(valeur_numérique_arrondie)	Retourne l'angle (en radians) dont le sinus est la valeur spécifiée en paramètre.
atan	(valeur_numérique_arrondie)	Retourne l'angle (en radians) dont la tangente est la valeur spécifiée en paramètre.
atn2	(valeur_numérique_arrondie1, valeur_numérique_arrondie2)	Retourne l'angle (en radians) dont la tangente est le rapport des valeurs spécifiées en paramètre (numérique_arrondie1 / numérique_arrondie2).
ceiling	(valeur_numérique)	Retourne le plus petit entier qui soit supérieur ou égal à la valeur spécifiée. Le résultat est du même type que l'expression numérique. Pour les expressions numériques et décimales, les résultats ont des précisions égales à celles des expressions et une échelle de 0.
cos	(valeur_numérique_arrondie)	Retourne le cosinus de l'angle spécifié (en radians).
cot	(valeur_numérique_arrondie)	Retourne la cotangente de l'angle spécifié (en radians).
degrees	(valeur_numérique)	Converti des degrés en radians. Le résultat est du même type que l'expression numérique. Pour les expressions numériques et décimales, les résultats ont une précision interne de 77 et une échelle égale à celle de l'expression. Quand une valeur de type *money* est utilisée, la conversion interne en *float* risque de causer une perte précision.
exp	(valeur_numérique_arrondie)	Retourne l'exponentielle de la valeur spécifiée.

floor	(valeur_numérique)	Retourne le plus grand entier inférieur ou égal à la valeur spécifiée. Le résultat est du même type que l'expression numérique. Pour les expressions numériques et décimales, les résultats ont des précisions égales à celles des expressions et une échelle de 0.
log	(valeur_numérique_arrondie)	Retourne le logarithme naturel de la valeur spécifiée.
log10	(valeur_numérique_arrondie)	Retourne le logarithme de base 10 de la valeur spécifiée.
pi	( )	Retourne la valeur constante 3.1415926535897936
power	(valeur_numérique, puissance)	Retourne la valeur de l'expression numérique élevée à la puissance spécifiée. Le résultat est du même type que l'expression numérique. Pour les expressions numériques et décimales, les résultats ont une précision interne de 77 et une échelle égale à celle de l'expression.
radians	(valeur_numérique)	Converti des radians en degrés. Le résultat est du même type que l'expression numérique. Pour les expressions numériques et décimales, les résultats ont une précision interne de 77 et une échelle égale à celle de l'expression. Quand une valeur de type *money* est utilisée, la conversion interne en *float* risque de causer une perte précision.
rand	([entier])	Retourne une valeur aléatoire *float* comprise entre 0 et 1. La valeur entière spécifiée optionnellement en paramètre est utilisée comme valeur de base.
round	(valeur_numérique, entier)	Arrondi l'expression numérique passée en paramètre avec le nombre de chiffres significatifs passé également en paramètre. Un *integer* positif détermine le nombre de chiffres significatifs à droite du séparateur décimal; un *integer* négatif détermine le nombre de chiffres significatifs à gauche du séparateur décimal. Pour les expressions numériques et décimales, les résultats ont une précision interne de 77 et une échelle égale à celle de l'expression.
sign	(valeur_numérique)	Retourne 1, 0 ou -1. Le résultat est du même type, a la même précision et la même échelle que l'expression numérique.
sin	(valeur_numérique_arrondie)	Retourne le sinus de l'angle spécifié (en radians).
sqrt	(valeur_numérique_arrondie)	Retourne la racine carrée de la valeur spécifiée.
tan	(valeur_numérique_arrondie)	Retourne la tangente de l'angle spécifié (en radians).

Fonctions d'agrégation linéaires

Utilisées dans une commande SELECT avec la clause COMPUTE, les fonctions d'agrégation linéaires(**sum**, **avg**, **min**, **max** et **count**) génèrent des valeurs récapitulatives qui apparaissent comme des lignes complémentaires dans les résultats de requêtes (contrairement aux résultats de fonctions d'agrégation qui apparaissent dans de nouvelles colonnes). Ce qui permet d'obtenir les lignes de détail et de récapitulation dans un même résultat. De plus, il est possible d'effectuer des récapitulations pour des sous-groupes et ainsi calculer plus d'une récapitulation pour le même groupe.

Syntaxe:

compute *fonction_d'agregation_linéaire* **(** *nom_colonne* **)**
[, *fonction_d'agregation_linéaire* **(** *nom_colonne* **)] ...**
[, by *nom_colonne* **[,** *nom _colonne* **] ...]**

Nom	Signification
sum	Total des valeurs dans la colonne (numérique).
avg	Moyenne des valeurs dans la colonne (numérique).
min	Plus petite valeur dans la colonne.
max	Plus grande valeur dans la colonne.
count	Nombre de valeurs non-nulles dans la colonne.

Remarques: Pour les serveurs SQL sécurisés, on ne peut utiliser les fonctions **sum**, **avg**, **max** ou **min** avec des types sécurisés. Néanmoins il est possible d'utiliser **count** ou **count**(*) avec des types sécurisés.

Fonctions de manipulation de chaînes

Ces fonctions opèrent sur des données binaires, des chaînes de caractères et des expressions.

Fonction	Paramètres	Retour
ascii	(expression_alphanumérique)	Retourne le code ASCII pour le premier caractère de l'expression.
char	(expression_entière)	Conversion d'une une valeur entière (1 Octet) en une valeur alphanumérique. (**char** est utilisée comme la fonction réciproque de **ascii**.). La valeur entière doit être comprise entre 0 et 255. Le résultat est du type *char*.
charindex	(expression1, expression2)	Retourne une valeur représentant la position de la première expression dans la deuxième. (Le premier paramètre est l'expression à rechercher)
char_length	(expression_alphanumérique)	Retourne une valeur entière représentant le nombre de caractères de l'expression après élimination des espaces à la fin de l'expression. Pour connaître le nombre d'octets, on utilise la fonction **datalength** (voir « fonctions système »).
difference	(expression_alphanumérique1, expression_alphanumérique2)	Retourne un entier représentant la différence entre deux (voir **soundex** ci-dessous)
lower	(expression_alphanumérique)	Conversion des majuscules en minuscules.
ltrim	(expression_alphanumérique)	Elimine les espaces précédant l'expression.
patindex	("%*recherche* %", Expression_alphanumérique [, **using { bytes | chars | characters}]**)	Retourne un entier représentant la position de départ de la première occurrence de *recherche* dans l'expression spécifié ou zéro si *recherche* n'est pas trouvée. Par défaut, **patindex** retourne l'offset en caractères; pour obtenir l'offset en octets, il faut spécifier **using bytes.** Le caractère générique "%" doit précéder et suivre *recherche* (excepté pour le recherche du premier ou du dernier caractère). (Voir « caractères génériques » pour une description de leur utilisation avec **patindex**). Peut être utilisé avec des données du type *text*.

replicate	(expression_alphanumérique, expression_numérique_entière)	Retourne une chaîne avec le même type de donnée que l'expression alphanumérique, contenant la même expression, répétée le nombre de fois spécifié ou autant de fois que possible tant que le retour ne dépasse pas 255 octets.
reverse	(expression_alphanumérique)	Retourne l'inverse de l'expression alphanumérique. Si l'expression est « abcd », le retour est « dcba ».
right	(expression_alphanumérique, expression_numérique_entière)	Retourne la partie de l'expression commençant à la position spécifiée (nombre de caractères à partir de la droite).
rtrim	(expression_alphanumérique)	Elimine les espaces à la fin de l'expression.
soundex	(expression_alphanumérique)	Retourne un code de 4 caractères pour la chaîne qui est composée d'un séquence continue de couples ou de doubles octets valides en lettres romanes.
space	(expression_numérique_entière)	Retourne une chaîne contenant le nombre spécifié d'espaces.
str	(valeur_numérique_approchée, [, longueur [,décimales]])	Retourne une représentation sous forme de chaîne de caractères d'une valeur numérique décimale. *longueur* indique le nombre de caractères à retourner (incluant le séparateur décimal, tous les chiffres à droite et à gauche, et les espaces); *décimales* indique le nombre de chiffres décimaux retournés.
stuff	(expression_alphanumérique1, début, longueur, expression_alphanumérique2)	Efface le nombre de caractères spécifié par *longueur* dans l'expression1 à partir de la position spécifié par *début*, puis insert l'expression2 dans l'expression1 à la position *début*. Pour effacer certains caractères sans insérer d'autre chaîne, l'expression2 peut être NULL (et non pas "", qui indique un espace).
substring	(expression, début, longueur)	Retourne une partie de l'expression. *début* indique la position à laquelle la nouvelle chaîne commence; *longueur* indique le nombre de caractères dans la nouvelle chaîne.
upper	(expression_alphanumérique)	Conversion des minuscules en majuscules.
+	expression + expression	Concatène deux caractères ou plus ou des expressions binaires.

Fonctions système

Fonction	Paramètres	Retour
col_name	(id_objet,id_colonne [, id_base_de_données])	Retourne le nom de la colonne.
col_length	(nom_objet, nom_colonne)	Retourne la longueur définie pour la colonne. Il faut utiliser **datalenght** pour connaître la taille réelle de la donnée.
curunreservedpgs	(id_base_de_données, numéro_page, pages_libres)	Retourne le nombre de pages libres dans la partie du disque contenant *numéro_page.* Il faut utiliser *pages_libres* pour indiquer une valeur retournée par défaut. Si la base de données est ouverte, **curunreservedpgs** remplace cette valeur par le nombre réel de pages libres stockées en mémoire pour cette partie du disque.
data_pgs	(id_objet, {**doampg** | **ioampg**})	Retourne de pages utilisés par la table (*doampg*) ou l'index (*ioampgh*). Le résultat n'inclus pas les pages utilisées pour les structures internes.
datalength	(expression)	Retourne la longueur de l'expression en octets. L'expression est généralement un nom de colonne. Si l'expression est une constante de caractère, elle doit être entourée de guillemets.
db_id	([nom_base_de_données])	Retourne le numéro (*id*) identifient de la base de données. Si le nom de la base n'est pas spécifié, **db_id** retourne l'identifient de la base courante.
db_name	([id_base_de_donnée])	Retourne le nom de la base de données. Si l'identifient n'est pas spécifié, **db_name** retourne le nom de la base courante.
host_id	( )	Retourne l'identifiant du processus hôte du processus client (pas le processus du serveur).
host_name	( )	Retourne le nom de la machine hôte courante du processus client (pas le processus du serveur).
index_col	(nom_objet, id_index, clé_# [, id_utilisateur])	Retourne le nom de la colonne indexée; retourne NULL si le nom de l'objet n'est ni une table ni une vue.

isnull	(expression1, expression2)	Substitue la valeur spécifié dans *expression2* quand la valeur *d'expression1* est NULL. Si types des expressions ne se convertissent pas implicitement, il faut utiliser la fonction **convert**.
lct_admin	({ { **"lastchance"** | **"logfull"** | **"unsuspended"** }, id_base_de_données } | **"reserve"**, pages_log)	
object_id	(nom_objet)	Retourne l'identificateur (*id*) de l'objet.
object_name	(id_objet [, id_base_de_données])	Retourne le nom de l'objet.
proc_role	(**"sa_role"** | **"sso_role"** | **"oper_role"**)	Recherche si l'utilisateur invoque le rôle correct pour exécuter la procédure. Retourne 1 si l'invocation a requis un rôle. Sinon, retourne 0.
reserved_pgs	(id_objet, { doampg | ioampg })	Retourne le nombre de pages allouées pour la table ou l'index. Cette fonction retourne également les pages utilisées pour les structures internes.
rowcnt	(**doampg**)	Retourne le nombre de lignes dans une table (estimation).
show_role	( )	Retourne les rôles actifs de l'utilisateur. Retourne NULL si l'utilisateur n'a aucun rôle.
suser_id	([nom_utilisateur_serveur]]	Retourne l'identifiant (*id*) de l'utilisateur du serveur (de *syslogins*). Si aucun *id* n'est trouvé, retourne celui de l'utilisateur courant.
suser_name	([id_utilisateur_serveur])	Retourne le nom de l'utilisateur du serveur. Si aucun nom n'est trouvé, retourne le nom de l'utilisateur courant.
user_pgs	(id_objet, doampg, ioampg)	Retourne le nombre total de pages utilisées par une table et son index sectorisé.
tsequal	(timestamp1, timestamp2)	
user		Retourne le nom de l'utilisateur.
user_id	([nom_utilisateur])	Retourne l'identifiant de l'utilisateur, basé sur l'identificateur dans la base de données courante.
user_name	([id_utilisateur])	Retourne le nom de l'utilisateur, basé sur l'identificateur dans la base de données courante.
valid_name	(expression_alphanumérique)	Retourne 0 si l'expression n'est pas un identifiant valide (si elle contient des caractères interdits ou plus de 30 octets); Retourne un nombre différent de 0 si l'expression est un identifiant valide.

Fonctions de manipulation de textes et d'images

Fonction	Paramètres	Retour
patindex	("%recherche%", Expression_alphanumérique [, using { bytes | chars | characters}])	Retourne un entier représentant la position de départ de la première occurrence de recherche dans l'expression spécifiée ou zéro si recherche n'est pas trouvée. Par défaut, **patindex** retourne l'offset en caractères; pour obtenir l'offset en octets, il faut spécifier **using bytes**. Le caractère générique "%" doit précéder et suivre *recherche* (excepté pour le recherche du premier ou du dernier caractère). (Voir « caractères génériques » pour une description de leur utilisation avec **patindex**). Peut être utilisé avec des données du type text.
textptr	(nom_colonne_text)	Retourne un pointeur (16 octets) sur un texte. La valeur retournée pointe sur la première page du type *text*.
textvalid	("nom_table.nom_colonne", pointeur_text)	Test si le pointeur spécifié est valide. Retourne 1 si le pointeur est valide et 0 dans le cas contraire.
set textsize	{ n | 0 }	Spécifie les limites, en octets de la donnée de type *text* ou *image* a retourner dans une commande SELECT. Les limites courantes sont stockées dans la variable globale *@@textsize*. *n* est un entier qui indique la limite en nombre d'octects à retourner; 0 restaure la limite par défaut de 32 K octets.

VII. EXEMPLES DE REQUETES

Description

C'est le thème des livres qui a été retenu pour les exemples de requêtes.
La base présente d'une manière simplifiée la gestion d'une maison d'édition.

Contenu des Tables

Table AUTEURS

nom_auteur	*prenom_auteur*	*siecle*
BALZAC	Honoré	19
CAMUS	Albert	20
DURAS	Marguerite	20
DUMAS	Alexandre	19
DOYLE	Arthur Conan	20
KING	Stephen	20

Table LIVRES

titre	*nom_auteur*	*prenom_auteur*	*reference*
Eugénie Grandet	BALZAC	Honoré	L01
L'amour	DURAS	Marguerite	L02
La peste	CAMUS	Albert	L03
Les justes	CAMUS	Albert	L04
Les chouans	BALZAC	Honoré	L05
Les trois mousquetaires	DUMAS	Alexandre	L06
The case book of Sherlock Holmes	DOYLE	Arthur Conan	L07

Table EDITIONS

reference	*date_edition*	*num_edition*	*langue*	*nbr_pages*
L03	01/01/95	1	FRANCAIS	300
L03	08/07/96	2	FRANCAIS	300
L02	17/04/92	1	FRANCAIS	140
L01	17/04/92	1	FRANCAIS	300
L04	03/06/74	1	FRANCAIS	300
L05	03/06/74	1	FRANCAIS	450
L05	28/08/82	2	FRANCAIS	
L06	05/05/95	1	FRANCAIS	550
L07	02/02/93	1	ANGLAIS	1100

Table TARIFS

reference	*num_edition*	*distrubuteur*	*prix*
L05	1	X	25
L05	2	X	35
L05	2	Y	37
L01	1	Y	30
L02	1	X	82
L04	1	X	20
L06	1	X	40
L07	1	Y	14
L07	1	X	17

Exemples de Requêtes

Requêtes de sélections

Requête 1

Enoncé:

« *Afficher tous les attributs de tous les tuples de la table AUTEURS.* »

Requêtes:

```
SELECT * FROM AUTEURS
```

ou

```
SELECT nom_auteur, prenom_auteur, siecle FROM AUTEURS
```

Retour:

nom_produit	prenom_auteur	siecle
BALZAC	Honoré	19
CAMUS	Albert	20
DURAS	Marguerite	20
DUMAS	Alexandre	19
DOYLE	Arthur Conan	20
KING	Stephen	20

Requête 2

Enoncé:

« *Afficher les noms de tous les auteurs à partir de la table LIVRES.* »

Requêtes:

```
SELECT   nom_auteur   FROM   LIVRES
```

ou

```
SELECT   LIVRES.nom_auteur   FROM   LIVRES
```

Retour:

nom_auteur
BALZAC
DURAS
CAMUS
CAMUS
BALZAC
DOYLE

Remarques: Les « doubles » n'ont pas été éliminés.
Les lignes n'ont pas été triées.

Requête 3

Enoncé:

« *Afficher les noms de tous les auteurs à partir de la table LIVRES avec élimination des doubles.* »

Requêtes:

```
SELECT DISTINCT nom_auteur FROM LIVRES
```

ou

```
SELECT DISTINCT LIVRES.reference FROM LIVRES
```

Retour:

nom_auteur
BALZAC
CAMUS
DOYLE
DUMAS
DURAS

Remarques: Les lignes ont été triées par ordre croissant.

Requête 4

Enoncé:

« *Afficher les titres des livres écrits par BALZAC.* »

Requêtes:

```
SELECT titre FROM LIVRES WHERE nom_auteur = 'BALZAC'
```

ou

```
SELECT LIVRES.titre FROM LIVRES WHERE nom_auteur = 'BALZAC'
```

Retour:

titre
Eugénie Grandet
Le chouans

Requête 5

Enoncé:

« *Afficher la référence des livres dont la deuxième édition est en langue française.* »

Requête:

```
SELECT   reference   FROM   EDITIONS
WHERE   num_edition = 2
AND   langue = 'FRANCAIS'
```

Retour:

reference
L03
L05

Requête 6

Enoncé:

« *Afficher les titres des livres écrits par BALZAC ou par CAMUS.* »

Requêtes:

```
SELECT   titre   FROM   LIVRES
WHERE   nom_auteur = 'BALZAC'   OR   nom_auteur = 'CAMUS'
```

ou

```
SELECT   titre   FROM   LIVRES
WHERE   nom_auteur   IN   ('BALZAC', 'CAMUS')
```

Retour:

titre
Eugénie Grandet
Le chouans
Les justes
Les Chouans

Requête 7

Enoncé:

« *Afficher les titres des livres écrits ni par BALZAC ni par CAMUS (dans la table LIVRES).* »

Requêtes:

```
SELECT titre FROM LIVRES
WHERE nom_auteur < > 'BALZAC' AND nom_auteur < > 'CAMUS'
```

ou

```
SELECT titre FROM LIVRES
WHERE nom_auteur NOT IN ('BALZAC', 'CAMUS')
```

Retour:

titre
L'amour
Les trois mousquetaires
The case book of Sherlock Holmes

Requête 8

Enoncé:

« *Afficher la référence et le numéro d'édition des livres dont le nombre de pages est inconnu.* »

Requête:

```
SELECT reference, num_edition FROM EDITIONS
WHERE nbr_pages IS NULL
```

Retour:

reference	num_edition
L05	2

Requête 9

Enoncé:

« *Sélectionner les auteurs (nom, prénom) dont le nom commence par 'D' (dans la table AUTEURS*). »

Requêtes:

```
SELECT   nom_auteur, prenom_auteur   FROM   AUTEURS
WHERE   nom_auteur   LIKE   'D%' Exemples de Requêtes - Requête 9
```

Retour:

nom_auteur	nom_auteur
DURAS	Marguerite
DUMAS	Alexandre
DOYLE	Arthur Conan

Requête 10

Enoncé:

« *Sélectionner les auteurs (nom, prénom) dont le nom commence par 'D', se termine par un S et est composé de 5 lettres (dans la table AUTEURS*). »

Requêtes:

```
SELECT   nom_auteur, prenom_auteur   FROM   AUTEURS
WHERE   nom_auteur   LIKE   'D___S'
```

Retour:

nom_auteur	nom_auteur
DURAS	Marguerite
DUMAS	Alexandre

Requête 11

Enoncé:

« Combien y a t'il de livres dont le nombre de pages est supérieur à 200 lors de la première édition ? »

Requêtes:

```
SELECT  COUNT (*)  Nombre_de_livres  FROM  EDITIONS
WHERE  num_edition = 1
AND  nbr_pages > 200
```

Retour:

Nombre_de_livres
6

Remarque: Pour nommer ou renommer une colonne Exemples de Requêtes - Requête 11, il faut placer le nouveau nom de celle-ci dans la commande SELECT, après le nom de l'attribut concerné.
Le nom de la colonne de doit pas contenir d'espace.

Requête 12

Enoncé:

« *Sélectionner les éditions (tous les attributs) dont le nombre de pages est le nombre de pages maximum d'une édition.* »

Requêtes:

```
SELECT * FROM EDITIONS
WHERE nbr_pages = ( SELECT MAX (nbr_pages) FROM EDITIONS )
```

ou

```
SELECT * FROM EDITIONS
WHERE nbr_pages IN ( SELECT MAX (nbr_pages) FROM EDITIONS )
```

Retour:

reference	date_edition	num_edition	langue	nbr_pages
L07	02/02/93	1	ANGLAIS	1100

Requête 13

Enoncé:

« *Sélectionner les éditions (tous les attributs) dont le nombre de pages est dans la moyenne (moyenne de toutes les éditions) à 10% près.* »

Requêtes:

```
SELECT * FROM EDITIONS
WHERE nbr_pages BETWEEN
            ( SELECT AVG (nbr_pages) FROM EDITIONS ) * 0.9
      AND
            ( SELECT AVG (nbr_pages) FROM EDITIONS ) * 1.1
```

ou

```
SELECT * FROM EDITIONS
WHERE nbr_pages > =
            ( SELECT AVG (nbr_pages) FROM EDITIONS ) * 0.9
AND nbr_pages < =
            ( SELECT AVG (nbr_pages) FROM EDITIONS ) * 1.1
```

Retour:

reference	date_edition	num_edition	langue	nbr_pages
L05	03/06/74	1	FRANCAIS	450

Requête 14

Enoncé:

« *Afficher les titres des livres (en éliminant les doublons) figurants au tarif.* »

Requêtes:

```
SELECT   DISTINCT   LIVRES.titre   FROM   LIVRES, TARIFS
WHERE   LIVRES.reference = TARIFS.reference
```

ou

```
SELECT   DISTINCT   a.titre   FROM   LIVRES a, TARIFS b
WHERE   a.reference = b.reference
```

Retour:

titre
Eugénie Grandet
L'amour
Les chouans
Les justes
Les trois mousquetaires
The case book of Sherlock Holmes

Remarques: Les tables (LIVRES et TARIFS) sont liés par une jointure (sur l'attribut référence).
On utilise généralement des alias en même temps que les jointures, afin de ne pas répéter inutilement les noms des tables dans une même sélection..

Requête 15

Enoncé:

« *Afficher les titres des livres (en éliminant les doublons) distribués à la fois par le distributeur 'X' et le distributeur 'Y'.* »

Requête:

```
SELECT DISTINCT a.titre FROM LIVRES a, TARIFS b1, TARIFS b2
WHERE a.reference = b1.reference AND b1.reference = b2.reference
AND b1.distributeur = 'X'
AND b2.distributeur = 'Y'
```

Retour:

titre
Les chouans
The case book of Sherlock Holmes

Remarque: Pour effectuer un produit cartésien (ou une jointure) entre une table et elle même, il est indispensable d'utiliser des alias.

Requête 16

Enoncé:

« Sélectionner *les livres (date d'édition, titre, numéro d'édition) édités au moins une fois; trier le retour par date d'édition croissante puis par titre.* »

Requêtes:

```
SELECT   b.date_edition, a.titre, b.num_edition   FROM   LIVRES a, EDITIONS b
WHERE   a.reference = b.reference
ORDER BY   b.date_edition ASC, a.titre ASC
```

ou

```
SELECT   b.date_edition, a.titre, b.num_edition   FROM   LIVRES a, EDITIONS b
WHERE   a.reference = b.reference
ORDER BY b.date_edition, a.titre
```

ou

```
SELECT   date_edition, titre, num_edition   FROM   LIVRES a, EDITIONS b
WHERE   a.reference = b.reference
ORDER BY date_edition, titre
```

Retour:

date_edition	titre	num_edition
03/06/74	Les chouans	1
03/06/74	Les justes	1
28/08/82	Les chouans	2
17/04/92	Eugénie Grandet	1
17/04/92	L'amour	1
02/02/93	The case book of Sherlock Holmes	1
01/01/95	La peste	1
05/05/95	Les trois mousquetaires	1
08/07/96	La peste	2

Remarques: C'est la clause ORDER BY qui permet de trier(sans croissant par défaut) le retour de la requête.
Il n'est pas nécessaire d'indiquer le nom de l'alias avant les attributs lorsque que ceux ci ne figure que dans une des tables concernées par la requête.

Requête 17

Enoncé:

« *Calculer et afficher le nombre de livres écrits par auteur*. »

Requêtes:

```
SELECT   nom_auteur, COUNT (*)livres_écrits
FROM   LIVRES
GROUP BY   nom_auteur
```

Retour:

Nom_auteur	livres_écrits
BALZAC	2
CAMUS	2
DOYLE	1
DUMAS	1
DURAS	1

Requête 18

Enoncé:

« *Afficher le nombre de livres écrits par auteur (nom, prénom) groupés par auteurs ayant un 'A' dans leur nom.* »

Requêtes:

```
SELECT   nom_auteur, prenom_auteur, count(*)
FROM   LIVRES
GROUP BY   nom_auteur, prenom_auteur
HAVING   nom_auteur   LIKE   '%A%'
```

Retour:

nom_auteur	prenom_auteur	nombre_de_livres
BALZAC	Honoré	2
CAMUS	Albert	2
DUMAS	Alexandre	1
DURAS	Marguerite	1

Requête 19

Enoncé:

« *Sélectionner les nom de tous les auteurs de la table AUTEURS, puis sélectionner tous les livres (titre, nom de l'auteur) de la table LIVRES.* »

Requêtes:

```
SELECT   DISTINCT   null titre, nom_auteur, null prenom_auteur
FROM   AUTEURS
UNION
SELECT   titre, nom_auteur, prenom_auteur
FROM   LIVRES
```

Retour:

titre	nom_auteur	prenom_auteur
	BALZAC	
	CAMUS	
	DOYLE	
	DUMAS	
	DURAS	
	KING	
La peste	CAMUS	Albert
Les justes	CAMUS	Albert
Les trois mousquetaires	DUMAS	Alexandre
The case book of Sherlock Holmes	DOYLE	Arthur Conan
Eugénie Grandet	BALZAC	Honoré
Les chouans	BALZAC	Honoré
L'amour	DURAS	Marguerite

Requêtes de mises à jour

Requête 20

Enoncé:

« *Créer la table STOCK avec les champs reference, num_edition, quantite ayant pour styles respectifs varchar(3), tinyint et smallint.* »

Requête:

```
CREATE   TABLE   STOCK
( reference   varchar(3),  num_edition   tinyint,  quantite   smallint )
```

Requête 21

Enoncé:

« *Détruire la table STOCK*. »

Requête:

```
DROP   TABLE   STOCK
```

Requête 22

Enoncé:

« *Créer la table STOCK avec les champs reference, num_edition, quantite ayant pour styles respectifs varchar(3), tinyint et smallint, avec les deux premiers champs obligatoires et le dernier facultatif.* »

Requête:

```
CREATE   TABLE   STOCK
( reference      varchar(3)      NOT  NULL,
  num_edition tinyint          NOT  NULL,
  quantite       smallint         NULL )
```

Requête 23

Enoncé:

« *Insérer la ligne ci-dessous dans la table STOCK.* »

reference	***num_edition***	***quantite***
L01	1	10000

Requête:

```
INSERT   INTO  STOCK
VALUES    ( 'L01', 1, 10000 )
```

Requête 24

Enoncé:

« *Doubler les quantités des premières éditions des livres.* »

Requête:

```
UPDATE   STOCK
SET   quantite = quantite * 2
WHERE   num_edition = 1
```

Requête 25

Enoncé:

« *Enlever du stock les livres de première édition.* »

Requête:

```
DELETE   FROM   STOCK
WHERE   num_edition = 1
```

Requête 26

Enoncé:

« *Ajouter deux colonnes à la table STOCK: une colonne quantite_prec (quantité précédente) de type smallint, et une colonne date_maj (dernière mise à jour) de type smalldatime* »

Requête:

```
ALTER   TABLE   STOCK
ADD     quantite_prec  smallint  NULL,
        date_maj       smalldatetime  NULL
```

Remarques: Les nouvelles colonnes ont la valeurs NULL.
Le mot clé NULL a été placé à la fin des définitions des nouvelles colonnes pour autoriser la valeur NULL dans ces colonnes. Ce mot clé est nécessaire dans le cas présent si la table contient déjà des données.
Le mot clé NOT NULL n'est pas autorisé dans la commande ALTER TABLE.

Requêtes complexes

Requête 27

Enoncé:

« *Créer et renseigner une table temporaire (#TEMP) dont les champs sont décris ci-dessous.* »

Description des champs:

Nom du champ	Champ d'origine	Table d'origine
prenom	prenom_auteur	AUTEURS
nom	nom_auteur	AUTEURS
titre	titre	LIVRES
num_edition	num_edition	EDITIONS
date_edition	date_edition	EDITIONS

Jointures :

Première Table	Seconde Table	Attribut(s)
AUTEURS	LIVRES	nom_auteur, prenom_auteur
LIVRES	EDITIONS	reference

Le résultat devra être trié par prénom, nom, titre puis numéro d'édition.

Requête:

```
SELECT    a.prenom_auteur prenom, a.nom_auteur nom, b.titre,
          c.num_edition, c.date_edition
INTO      #TEMP
FROM      AUTEURS a, LIVRES b, EDITIONS c
WHERE     a.nom_auteur = b.nom_auteur
AND       a.prenom_auteur = b.prenom_auteur
AND       b.reference = c.reference
ORDER BY  a.nom_auteur, a.prenom_auteur, b.titre, c.num_edition
```

Remarques: Le nom d'une table temporaire doit être préfixé d'un caractère « # ».
Les tables temporaires sont automatiquement supprimées par le serveur lors de la déconnexion.

Requête 28

Enoncé:

« *Créer et renseigner une table temporaire (#TEMP2) dont les champs sont décris ci-dessous.* »

Description des champs:

Nom du champ	Champ d'origine	Table d'origine
prenom	prenom_auteur	AUTEURS
nom	nom_auteur	AUTEURS
titre	titre	LIVRES
reference	reference	LIVRES
num_edition	num_edition	EDITIONS
date_edition	date_edition	EDITIONS
prix_moyen	néant	néant
nombre_distributeur	néant	néant

Jointures :

Première Table	Seconde Table	Attribut(s)
AUTEURS	LIVRES	nom_auteur, prenom_auteur
LIVRES	EDITIONS	reference

Le résultat devra être trié par prénom, nom, titre puis numéro d'édition.

Le champ « nombre_distributeur » sera du type **smallint**.

Le champ « prix_moyen » sera du type **smallmoney**.

Requête:

```
CREATE  TABLE  #TEMP2
(       prenom  char(25),  nom  char(25),  titre  char(25),
        reference  varchar(3),  num_edition  smallint,
        date_edition  smalldatetime,  prix_moyen  smallmoney  NULL,
        nombre_distributeur  smallint  NULL  )


INSERT  INTO  #TEMP2

(       prenom,  nom,  titre,  reference,  num_edition,  date_edition  )

SELECT      a.prenom_auteur  prenom,  a.nom_auteur  nom,  b.titre,
            b.reference,  c.num_edition,  c.date_edition
FROM        AUTEURS a, LIVRES b, EDITIONS c
WHERE       a.nom_auteur = b.nom_auteur
AND         a.prenom_auteur = b.prenom_auteur
AND         b.reference = c.reference
ORDER BY    a.nom_auteur,  a.prenom_auteur,  b.titre,  c.num_edition
```

Remarques: Il n'est pas possible ici de créer et de renseigner la table temporaire dans une commande unique (SELECT ... INTO) car les colonnes laissées vides doivent être typées.
Les nouvelles colonnes ont la valeurs NULL.
Le mot clé NULL a été placé à la fin des définitions des nouvelles colonnes pour autoriser la valeur NULL dans ces colonnes. Ce mot clé est nécessaire dans le cas présent si la table contient déjà des données.
Le mot clé NOT NULL n'est pas autorisé dans la commande ALTER TABLE.

Requête 29

Enoncé:

« *Mettre à jour, dans la table temporaire (#TEMP2) les champs décris ci-dessous.* »

Description des champs:

Nom du champ	Description
nombre_distributeur	Nombre de distributeurs pour un livre.
prix_moyen	Moyenne des prix pour ce livre. Fonction AVG du champ *prix*.

Requête:

```
UPDATE   #temp2
SET    nombre_distributeur =
       (       SELECT   COUNT(*)
               FROM   TARIFS a
               WHERE       #temp2.reference=a.reference
               AND         #temp2.num_edition=a.num_edition
       ),
       prix_moyen =
       (       SELECT   AVG(prix)
               FROM   TARIFS a
               WHERE       #temp2.reference=a.reference
               AND         #temp2.num_edition=a.num_edition
       )
```

Requête 30 (Spécifique SYBASE)

Enoncé:

« *Créer une procédure date_edition qui afficher les dates des éditions des livres (dont la référence est passée en paramètre. Le format d'affichage de la date sera : JJ.MM.AAAA.* »

Requête:

```
CREATE PROC date_edition ( @reference varchar(3) )
AS
        SELECT num_edition edition,
                CONVERT ( char(10), date_edition, 104 ) date
        FROM    EDITIONS
        WHERE   reference=@reference
RETURN
```

Remarques: Pour convertir une date (format datetime ou smalldatetime) en une chaîne de caractères selon un style spécifique, il faut utiliser la fonction CONVERT.
Le troisième paramètre de la fonction CONVERT indique le numéro du style de la chaîne retournée; le tableau ci-dessous présente la liste des styles disponibles :

Style	***Format***	***Style***	***Format***
0	mm jj aaa hh:mm		
1	mm/jj/aa	101	mm/jj/aaaa
2	aa.mm.jj	102	aaaa.mm.jj
3	jj/aa/mm	103	jj/mm/aaaa
4	jj.mm.aa	104	jj.mm.aaaa
5	jj-mm-aa	105	jj-mm-aaaa
6	jj moi aa	106	jj moi aaaa
7	moi jj, aa	107	moi jj, aaaa
8	hh:mn:ss		
9	moi jj aa hh:mn:ss.μμμ AM	109	moi jj aaaa hh:mm:ss.μμμ AM
10	mm-jj-aa	110	mm-jj-aaaa
11	aa/mm/jj	111	aaaa/mm/jj
12	aammjj	112	aaaammjj

Requête 31

Enoncé:

« *Créer une table auteurs_bis, jumelle de la table auteurs en une commande. La table auteurs_bis doit avoir le même format que la table auteurs et les mêmes données.* »

Requête:

```
SELECT  *    INTO  auteurs_bis
FROM    auteurs
```

Remarques: Pour que cette commande soit possible, il faut que l'option « Allow select into» soit activée sur la base de données en question.

VIII. ANNEXES

Mots clés réservés

ADD
AFTER
ALL
ALTER
AND
ANY
AS
ASC
AUTHORIZATION
AVERAGE
AVG
BEFORE
BEGIN
BETWEEN
BIND
BREAK
BROWSE
BULK
BY
CASE
CHAR
CHARACTER
CHECKPOINT
CLOSE
CLUSTERED
COBOL
COLUMN
COMMENT
COMMIT
COMPUTE
CONFIRM
CONNECT
COMMIT
CONNECT
CONTINUE
CONTROLROW
CONVERT
COUNT
CREATE
CURRENT
CURSOR
DATA
DATABASE
DATE
DATETIME
DAY

DAYS
DBCC
DEC
DECIMAL
DECLARE
DEFAULT
DELETE
DESC
DESCRIBE
DISK
DISTINCT
DOUBLE
DROP
DUMMY
DUMP
EACH
ELSE
END
ERRLVL
ERROREXIT
EXCEPT
EXCLUSIVE
EXEC
EXECUTE
EXISTS
EXIT
FETCH
FILLFACTOR
FIRST
FLOAT
FOR
FORTRAN
FROM
GO
GOTO
GLOBAL
GRANT
GRAPHIC
GROUP
HAVING
HOLD
HOLDLOCK
IF
IMMEDIATE
IN
INDEX

INSERT
INT
INTEGER
INTERSECT
INTO
IS
KEY
KILL
LABEL
LENGTH
LEVEL
LIKE
LINENO
LOAD
LOCK
LONG
MAX
MICROSECOND
MICROSECONDS
MIN
MINUTE
MIRROREXIT
MIXED
MODE
MONTH
MONTHS
NEXT
NEW
NONCLUSTERED
NONE
NOT
NULL
NUMERIC
OFF
OFFSETS
ON
ONCE
ONLY
OPTION
OR
ORDER
OVER
PAGE
PAGES
PASCAL
PERM

PERMANENT
PLAN
PL1
PRECISION
PREPARE
PRINT
PRIVILEGES
PROC
PROCEDURE
PROCESSEXIT
PROGRAM
PUBLIC
RAISERROR
READTEXT
REAL
RECONFIGURE
REFERENCE
REFECENCES
RELATIVE
RETURN
REVOKE
ROLLBACK
ROWCOUNT
RULE
SAVE
SECOND
SECONDS
SECTION
SELECT
SESSION
SET
SETUSER
SHARE
SHUTDOWN
SMALLINT
SOM
SQL
SQLCODE
SQLDESCRIPTOR
SQLERROR
SQLSTATE
STATISTICS
SUBSTR
SUBSTRINGS
SUM
SYNONYM

TABLE
TABLEGROUP
TABLESPACE
TAPE
TEMP
TEMPORARY
TEXT
TEXTSIZE
THEN
TIME
TIMESTAMP

TO
TRAN
TRANSACTION
TRIGGER
TRUNCATE
TSEQUAL
UNION
UNIQUE
UNITS
UNLOCK
UPDATE

USE
USER
USING
VALUE
VALUES
VARCHAR
VARIABLES
VIEW
VOLUME
WAIT
WAITFOR

WHEN
WHENEVER
WHERE
WHILE
WITH
WORK
WRITE
WRITETEXT
YEAR
YEARS

Bibliographie

Titre	Auteur	Editeur
Bases de données	G.GARDARIN	EYROLLES
Guide du langage SQL	John Viescas	Microsoft Press
Le livre de SQL	Suzy Pasleau	PSI
SQL AIDE-MEMOIRE		CEDIC / NATHAN
SQL Data System Concepts and Facilities		IBM Corporation
SYBASE SQL Server Quick Reference Guide		SYBASE
TRANSACT SQL		SYBASE

Lexique

Clé	Une clé primaire (appelée tout simplement clé) est un champ ou un groupe de champs qui contient des données n'identifiant qu'un seul qu'un seul enregistrement de la table. Une clé nécessite une seule valeur par enregistrement (ligne) de la table, ce qui évite les doublons. Les tables qui possèdent des clés sont dites indexées. Une clé définit l'ordre de tri par défaut de la table.
Index	Un index détermine l'ordre dans lequel on accède aux enregistrements de la table. Un index est une table (ou plusieurs tables) permettant d'associer à chaque clé d'enregistrement, l'adresse relative de cet enregistrement.
Requête	Une requête est une question formulée sur les données contenues dans des tables.

S.G.B.D.	Système de Gestion de Bases de Données.
S.Q.L.	(Structured Query Language.) Langage de requêtes structuré. Il permet de créer, modifier et récupérer des données à partir d'une base de données relationnelle.
Table	Les tables recueillent les données de la base. Une table est composée de lignes et de colonnes. Chaque ligne contient toute l'information disponible sur un sujet particulier. On l'appelle enregistrement. Chaque colonne contient une catégorie d'information composant l'enregistrement. On l'appelle champ.

Index

S

T

U

V

W