



Bases de données : SQL

Support de Formation n°1



Mise à jour 12 janvier 2015 (MF)
Création 3 mai 2001 (LR, PG, MF)

Objectifs

- Présentation du langage SQL et de l'algèbre relationnel
- Interrogation et mise à jour d'une base de données en SQL interactif
- Construction complète d'une base de données
- Notions d'administration d'une base de données en Client/Serveur
- Présentation du modèle client/serveur
- La connexion ODBC et l'utilisation d'ACCESS comme client

Mode d'emploi

Pour chacun des thèmes abordés, ce support de formation :

- présente les points essentiels
- renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur, par le symbole : 
- propose des exercices pratiques par le symbole : 

Autres symboles utilisés :



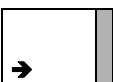
Point important qui mérite d'être souligné



A éviter à tout prix !



Manip à réaliser avec un utilitaire graphique (qui devrait normalement simplifier le travail !)



Approfondissement : travail de documentation ou exercice supplémentaire, à faire individuellement (non intégré dans le temps de formation moyen)

Documents connexes

Pour l'utilisation de l'outil graphique lire le poly

***Administration sql serveur 2005 avec l'outil Microsoft sql serveur management studio
(AdministrationSqlServeur2005.doc)***

Sommaire

1. PRESENTATION DU LANGAGE SQL	6
1.1 Ressources	6
1.2 Historique du langage	6
1.3 Notion de relation	6
1.4 Rappels d'algèbre relationnel	11
1.5 Place de SQL dans les SGBDR	15
2. DECOUVERTE PRATIQUE DU LANGAGE SQL	16
2.1 Connexion à la base de données PUBLI	16
2.2 Description de la base de données PUBLI	16
2.3 Consultation d'une base de données	26
2.4 Mise à jour d'une base de données.....	39
3. CREATION DE LA BASE «COMPAGNIE AERIENNE»	42
3.1 Introduction	42
3.2 Création des domaines (types de données utilisateur)	42
3.3 Création et suppression de tables. Contraintes d'intégrité	46
3.4 Création de vues	50
3.5 Définition des permissions sur les objets d'une base de donnée	53
3.6 Les fonctions.....	56
3.7 Les procédures stockées	58
3.8 Les déclencheurs (triggers)	67
3.9 Les curseurs.....	71
3.10 Les index	73
3.11 Récapitulation	75
4. LES TRANSACTIONS	85
5. LE MODELE CLIENT/SERVEUR	95

5.1	Introduction	95
5.2	Les principes de base	95
5.3	Le dialogue entre client et serveur : l'IPC	96
5.4	Les mécanismes utilisés par les interfaces de communication.....	97
5.5	Les différentes mises en oeuvre du modèle	98
	Le Client-Serveur de présentation.....	99
	Le client-Serveur de données	99
	Le Client-Serveur de procédure.....	99
5.6	Conclusion	99
6.	LE CLIENT/SERVEUR AVEC ODBC ET ACCESS.....	101
6.1	Création d'une source de donnée ODBC	101
6.2	Connexion à une source de données sous Access	104
	ANNEXES	106
1.	GLOSSAIRE.....	107
2.	Corrigés des exercices complémentaires.....	111
3.	Les fonctions.....	113
4.	Les éléments du langage TRANSACT-SQL.....	119

1. PRESENTATION DU LANGAGE SQL

1.1 Ressources

- Pour la syntaxe du langage SQL : consulter le livre « SQL2 Initiation/Programmation » de Christian Marée et Guy Ledant
- Pour l'utilisation de l'environnement SQL Server : utiliser « Documentation en ligne de SQL Server » et « Aide de Transact SQL » sous « Analyseur de requête SQL »

1.2 Historique du langage

- S.Q.L. (Structured Query Language) est un langage structuré permettant d'interroger et de modifier les données contenues dans une **base de données relationnelle**.
- Il est issu de SEQUEL : Structured English Query Language. C'est le premier langage pour les S.G.B.D Relationnels. Il a été développé par IBM en 1970 pour système R, son 1^{er} SGBDR.
- S.Q.L. a été reconnu par l'ANSI puis imposé comme norme. Il n'existe pas de S.G.B.D.R sans S.Q.L. ! Malheureusement, malgré la norme SQL, il existe un ensemble de « dialectes » qui respectent un minimum commun.
- Ce cours s'appuiera sur le langage « Transact SQL » de Microsoft, en essayant de rester le plus standard possible.

1.3 Notion de relation



SQL 2 : Partie I Approche p. 1 à 25

- SQL est basé sur la théorie des ensembles : il manipule des « tables » qui représentent le graphe d'une relation entre plusieurs ensembles (colonnes).
- Chaque ligne ou « tuple » est un élément du graphe de la relation

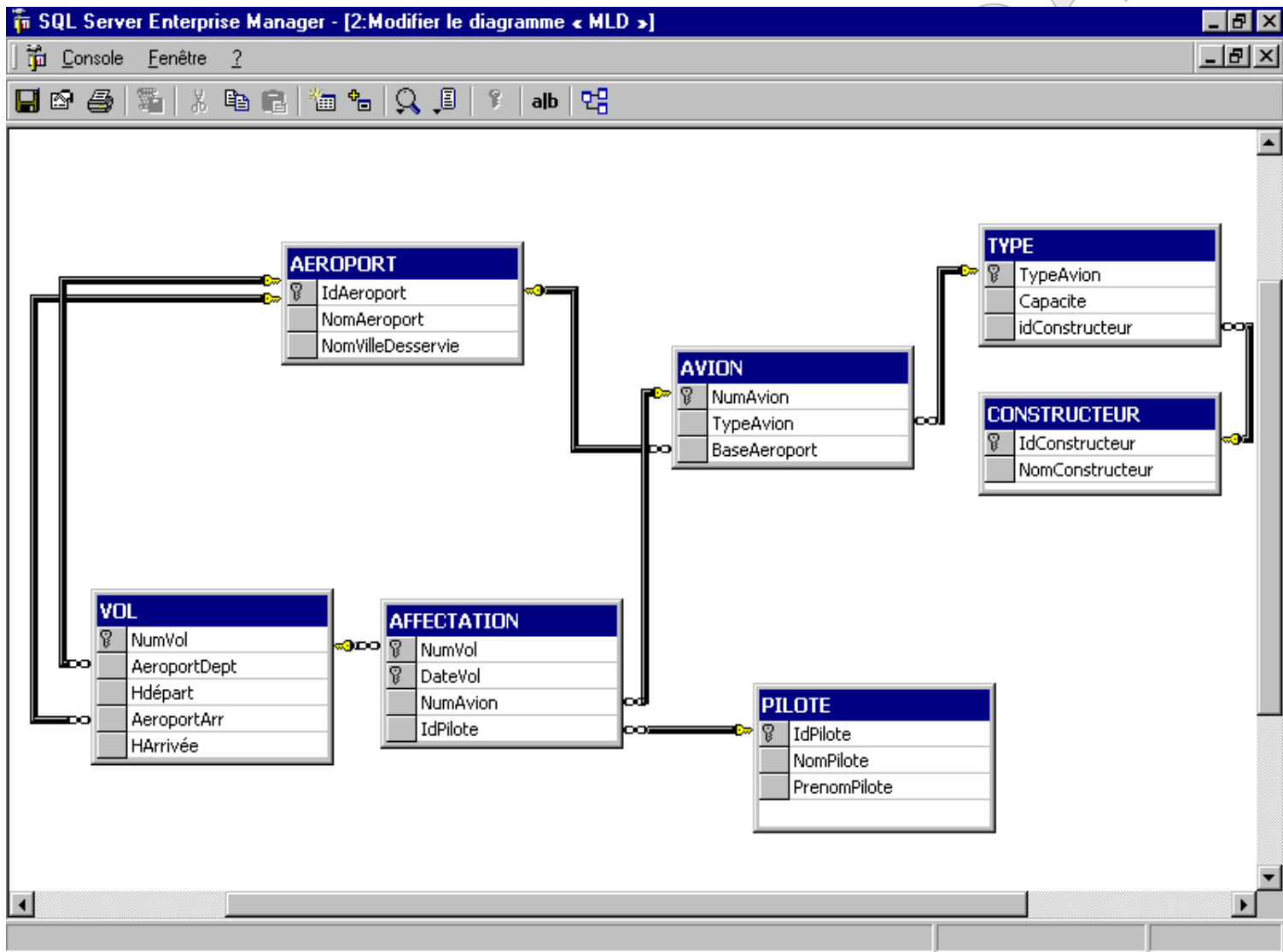
IdPilote	NomPilote	PrenomPilote
1	GAINSBURG	Serge
2	FERRAT	Jean
3	NOUGARO	Claude
4	SCHUMMAN	Robert
5	STROGOFF	Michel
6	SORREL	Lucien
7	TAVERNIER	Bertrand
8	FAYOLLE	Marc
9	LECU	Régis

Ligne ou tuple

Colonne ou attribut

- Exemple : la base de données « Compagnie aérienne » que l'on créera sous SQL Server dans la suite de ce cours :

Schéma général de la base



Structure et contenu des tables

AVION (NumAvion, TypeAvion, BaseAeroport)

NumAvion : numéro d'avion (clé primaire, numérique)

TypeAvion : type d'avion : A320, B707... (Clé étrangère vers la colonne TypeAvion de la table TYPE, alphanumérique)

BaseAeroport : identificateur de l'aéroport où est basé l'avion (clé étrangère vers la colonne IdAeroport de la table AEROPORT, 3 lettres)

NumAvion	TypeAvion	BaseAeroport
100	A320	NIC
101	B707	CDG
102	A320	BLA
103	DC10	BLA
104	B747	ORL
105	A320	GRE
106	ATR42	CDG
107	B727	LYS
108	B727	NAN
109	A340	BAS

Tous les avions de même type ont des caractéristiques communes : tous les A320 possèdent le même nombre de places et sont construits par « AirBus ». Pour ne pas introduire de redondance dans notre base, il faut donc créer une table TYPE pour stocker le nombre de places et le nom du constructeur.

TYPE (TypeAvion, Capacite, IdConstructeur)

TypeAvion : type d'avion (clé primaire, alphanumérique)

Capacité : nombre de places (numérique)

IdConstructeur : identificateur du constructeur (clé étrangère vers la colonne IdConstructeur de la table CONSTRUCTEUR, numérique)

TypeAvion	Capacite	idConstructeur
A320	300	1
A340	350	1
ATR42	50	1
B707	250	2
B727	300	2
B747	400	2
DC10	200	4

Les noms des constructeurs doivent être connus avant de renseigner les types d'avion : il faut les stocker dans une table indépendante CONSTRUCTEUR, pour pouvoir les présenter à l'utilisateur dans une liste déroulante. Par ailleurs, cette solution économise de la place dans la base de données (N entiers au lieu de N fois 50 caractères)

CONSTRUCTEUR (IdConstructeur, NomConstructeur)

IdConstructeur : identificateur du constructeur (clé primaire, numérique)

NomConstructeur : nom du constructeur (alphanumérique)

IdConstructeur	NomConstructeur
1	Aérospatiale
2	Boeing
3	Cessna
4	Douglas

Tous les avions sont basés dans un aéroport. Les vols effectués par la compagnie aérienne vont d'un aéroport de départ à un aéroport d'arrivée. La table AEROPORT doit regrouper toutes les caractéristiques qui concernent directement l'aéroport : identificateur, nom, ville desservie

AEROPORT (IdAeroport, NomAeroport, NomVilleDesservie)

IdAeroport : identificateur de l'aéroport (clé primaire, 3 lettres)

NomAeroport : nom de l'aéroport (alphanumérique)

NomVilleDesservie : ville desservie par l'aéroport (alphanumérique)

IdAeroport	NomAeroport	NomVilleDesservie
BAS	Poretta	Bastia
BLA	Blagnac	Toulouse
BRI	Brive	Brive
CDG	Roissy	Paris
GRE	Saint Geoir	Grenoble
LYS	Saint exupéry	Lyon
NAN	Saint Herblain	Nantes
NIC	Nice cote d'azur	Nice
ORL	Orly	Paris

Un vol, décrit par un numéro de vol unique, relie un aéroport de départ à un aéroport d'arrivée, en partant à une heure donnée et en arrivant à une heure donnée.

Le même vol est proposé par la compagnie aérienne à des dates différentes, avec des moyens (pilote et avion) éventuellement différents : ne pas confondre la table VOL qui décrit les caractéristiques générales du vol, avec la table AFFECTATION qui décrit les moyens mis en œuvre pour un VOL proposé à une date donnée.

VOL (NumVol, AeroportDept, Hdépart, AeroportArr, HArrivée)

NumVol : numéro de vol (clé primaire, "IT" + 3 chiffres)

AeroportDept : identificateur de l'aéroport de départ (clé étrangère vers la colonne IdAeroport de la table AEROPORT, 3 lettres)

Hdépart : heure de départ (type heure)

AeroportArr : identificateur de l'aéroport d'arrivée (clé étrangère vers la colonne IdAeroport de la table AEROPORT, 3 lettres)

Harrivée : heure d'arrivée (type heure)

NumVol	AeroportDept	Hdépart	AeroportArr	HArrivée
IT100	NIC	7:00	CDG	9:00
IT101	ORL	11:00	BLA	12:00
IT102	CDG	12:00	NIC	14:00
IT103	GRE	9:00	BLA	11:00
IT104	BLA	17:00	GRE	19:00
IT105	LYS	6:00	ORL	7:00
IT106	BAS	10:00	ORL	13:00
IT107	NIC	7:00	BRI	8:00
IT108	BRI	19:00	ORL	20:00
IT109	NIC	18:00	ORL	19:00

IT110	ORL	15:00	NIC	16:00
IT111	NIC	17:00	NAN	19:00

Pour connaître une « desserte » de façon unique, il faut connaître le numéro de vol et la date : la table AFFECTATION aura donc une « clé primaire composée », constituée par la concaténation des colonnes NumVol et DateVol.

AFFECTATION (NumVol, DateVol, NumAvion, IdPilote)

NumVol : numéro de vol (clé étrangère vers la colonne NumVol de la table VOL)

DateVol : date du vol (type date). Clé primaire : NumVol + DateVol

NumAvion : numéro de l'avion qui assure le vol (clé étrangère vers la colonne NumAvion de la table Avion, numérique)

IdPilote : identificateur du pilote en charge du vol (clé étrangère vers la colonne IdPilote de la table Pilote, numérique)

NumVol	DateVol	NumAvion	IdPilote
IT100	6 avril 2001	100	1
IT100	7 avril 2001	101	2
IT101	6 avril 2001	100	2
IT101	7 avril 2001	103	4
IT102	6 avril 2001	101	1
IT102	7 avril 2001	102	3
IT103	6 avril 2001	105	3
IT103	7 avril 2001	104	2
IT104	6 avril 2001	105	3
IT104	7 avril 2001	107	8
IT105	6 avril 2001	107	7
IT105	7 avril 2001	106	7
IT106	6 avril 2001	109	8
IT106	7 avril 2001	104	5
IT107	6 avril 2001	106	9
IT107	7 avril 2001	103	8
IT108	6 avril 2001	106	9
IT108	7 avril 2001	106	5
IT109	6 avril 2001	107	7
IT109	7 avril 2001	105	1
IT110	6 avril 2001	102	2
IT110	7 avril 2001	104	3
IT111	6 avril 2001	101	4
IT111	7 avril 2001	100	8

- Les éléments d'une colonne appartiennent tous au même ensemble appelé « domaine »
- Une clé qui fait référence à la clé primaire d'une autre table est appelée « clé étrangère » : NumAvion dans Affectation...

1.4 Rappels d'algèbre relationnel

Toutes les requêtes SQL correspondent à une combinaison des 7 opérateurs d'algèbre relationnel.

Les Opérateurs Ensemblistes

Ne s'appliquent qu'à des relations « unicompatibles » : possédant le même nombre d'attributs sur les mêmes domaines :

- l'**union** permet le regroupement de tuples :

R1 Vols à destination de Paris charles de Gaulle

NumVol	AéroportDept	Hdépart	AéroportArr	HArrivée
IT100	NIC	7:00	CDG	9:00

R2 Vols à destination de Paris Orly

NumVol	AéroportDept	Hdépart	AéroportArr	HArrivée
IT105	LYS	6:00	ORL	7:00
IT106	BAS	10:00	ORL	13:00
IT108	BRI	19:00	ORL	20:00
IT109	NIC	18:00	ORL	19:00

R3 = R1 U R2 Vols à destination de Paris

NumVol	AéroportDept	Hdépart	AéroportArr	HArrivée
IT100	NIC	7:00	CDG	9:00
IT105	LYS	6:00	ORL	7:00
IT106	BAS	10:00	ORL	13:00
IT108	BRI	19:00	ORL	20:00
IT109	NIC	18:00	ORL	19:00

- l'**intersection** permet la création d'une table à partir de tuples communs à 2 tables :

R4 Vols Départ Nice

NumVol	AéroportDept	Hdépart	AéroportArr	HArrivée
IT100	NIC	7:00	CDG	9:00
IT107	NIC	7:00	BRI	8:00
IT109	NIC	18:00	ORL	19:00
IT111	NIC	17:00	NAN	19:00

R5 Vols Arrivée Paris Orly

NumVol	AéroportDept	Hdépart	AéroportArr	HArrivée
IT105	LYS	6:00	ORL	7:00
IT106	BAS	10:00	ORL	13:00

IT108	BRI	19:00	ORL	20:00
IT109	NIC	18:00	ORL	19:00

R6 = R4 ^ R5 Vols départ Nice, arrivée Orly

NumVol	AéroportDept	Hdépart	AéroportArr	HArrivée
IT109	NIC	18:00	ORL	19:00

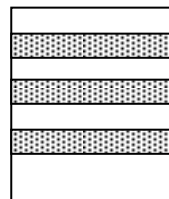
- la **différence** sélectionne les tuples d'une table en éliminant les tuples présents dans une autre table.

R7 = R4 - R3 Vols départ Nice, arrivée Province

NumVol	AéroportDept	Hdépart	AéroportArr	HArrivée
IT107	NIC	7:00	BRI	8:00
IT111	NIC	17:00	NAN	19:00

Opérateurs Relationnels Unaires

- la **sélection** réalise un découpage horizontal de la table en ne conservant que les tuples satisfaisant une condition définie sur les valeurs d'un attribut.
⇒ **Certains enregistrements et tous les attributs.**

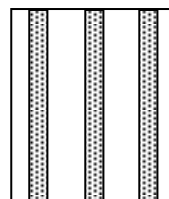


Exemples :

- Descriptif complet des vols pour Nice : toutes les lignes de la table VOL telles que AeroportArr = 'NIC'.
- Descriptif complet des avions de type A320 : toutes les lignes de la table AVION telles que TypeAvion = 'A320'

- la **projection** permet de ne conserver que les attributs (colonnes) intéressants ; c'est un découpage vertical de la table.

⇒ **Certains attributs et tous les enregistrements.**



Exemples :

- liste de tous les numéros d'avions

- liste des noms et des prénoms des pilotes

Opérateurs Relationnels Binaires

- le **produit cartésien** réalise la juxtaposition ou concaténation de tous les tuples d'une table avec tous les tuples d'une autre table. Si les 2 tables ont M et N tuples le résultat aura M * N tuples.

A1	B1	A1	B1
A2	B2	A2	B1
A3		A3	B1
		A1	B2
		A2	B2
		A3	B2

- Le **jointure (join)**, est possible seulement sur 2 tables possédant un domaine commun. La jointure consiste à juxtaposer les tuples dont la valeur d'un attribut est identique dans les deux tables. On constate que souvent, la jointure porte sur des clés étrangère et primaire liées.

Exemple : Faire un Planning des Vols indiquant le nom et le prénom des Pilotes

Il faut concaténer les lignes de la table **Affectation** avec celles de la table **Pilote**, pour lesquelles **Affectation.IdPilote = Pilote.IdPilote**

Table Affectation

NumVol	DateVol	NumAvion	IdPilote
IT100	6 avril 2001	100	1
IT100	7 avril 2001	101	2
IT101	6 avril 2001	100	2
IT101	7 avril 2001	103	4
IT102	6 avril 2001	101	1
IT102	7 avril 2001	102	3
IT103	6 avril 2001	105	3
IT103	7 avril 2001	104	2
IT104	6 avril 2001	105	3
IT104	7 avril 2001	107	8
IT105	6 avril 2001	107	7
IT105	7 avril 2001	106	7
IT106	6 avril 2001	109	8
IT106	7 avril 2001	104	5
IT107	6 avril 2001	106	9
IT107	7 avril 2001	103	8
IT108	6 avril 2001	106	9
IT108	7 avril 2001	106	5
IT109	6 avril 2001	107	7
IT109	7 avril 2001	105	1

IT110	6 avril 2001	102	2
IT110	7 avril 2001	104	3
IT111	6 avril 2001	101	4
IT111	7 avril 2001	100	8

Table Pilote

IdPilote	NomPilote	PrenomPilote
1	GAINSBURG	Serge
2	FERRAT	Jean
3	NOUGARO	Claude
4	SCHUMMAN	Robert
5	STROGOFF	Michel
6	SORREL	Lucien
7	TAVERNIER	Bertrand
8	FAYOLLE	Marc
9	LECU	Régis

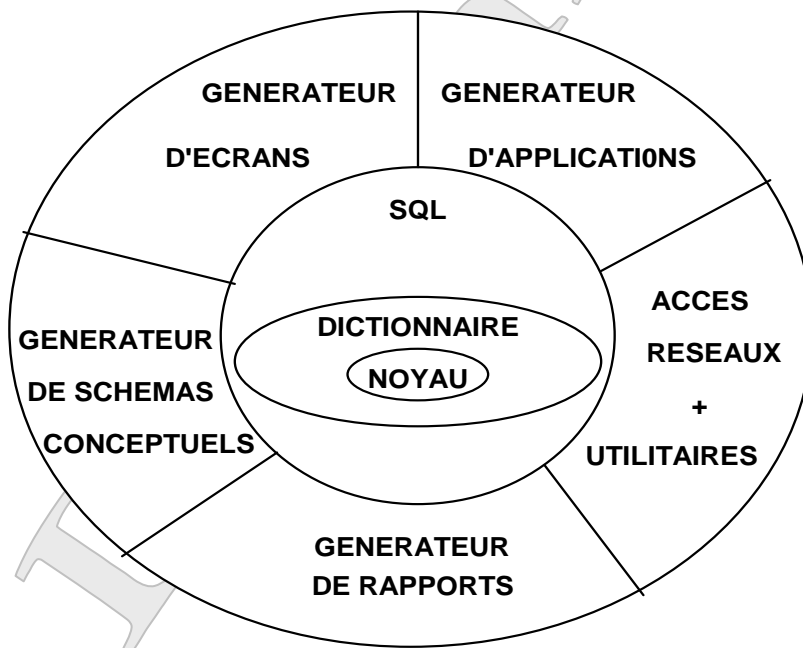
Planning : jointure Affectation - Pilotes

NumVol	DateVol	NumAvion	NomPilote	PrenomPilote
IT100	6 avril 2001	100	GAINSBURG	Serge
IT100	7 avril 2001	101	FERRAT	Jean
IT101	6 avril 2001	100	FERRAT	Jean
IT101	7 avril 2001	103	SCHUMMAN	Robert
IT102	6 avril 2001	101	GAINSBURG	Serge
IT102	7 avril 2001	102	NOUGARO	Claude
IT103	6 avril 2001	105	NOUGARO	Claude
IT103	7 avril 2001	104	FERRAT	Jean
IT104	6 avril 2001	105	NOUGARO	Claude
IT104	7 avril 2001	107	FAYOLLE	Marc
IT105	6 avril 2001	107	TAVERNIER	Bertrand
IT105	7 avril 2001	106	TAVERNIER	Bertrand
IT106	6 avril 2001	109	FAYOLLE	Marc
IT106	7 avril 2001	104	STROGOFF	Michel
IT107	6 avril 2001	106	LECU	Régis
IT107	7 avril 2001	103	FAYOLLE	Marc
IT108	6 avril 2001	106	LECU	Régis
IT108	7 avril 2001	106	STROGOFF	Michel
IT109	6 avril 2001	107	TAVERNIER	Bertrand
IT109	7 avril 2001	105	GAINSBURG	Serge
IT110	6 avril 2001	102	FERRAT	Jean
IT110	7 avril 2001	104	NOUGARO	Claude
IT111	6 avril 2001	101	SCHUMMAN	Robert
IT111	7 avril 2001	100	FAYOLLE	Marc

1.5 Place de SQL dans les SGBDR

La plupart des SGBDR du marché (DB2-IBM, ORACLE, INFORMIX, SYBASE...) offrent plus qu'un langage de requête puissant (en général SQL). On y trouve toute une panoplie d'outils dits de 4ème génération :

- générateur d'écrans : facilitant la saisie et l'affichage de données par imbrication de requêtes,
- générateur d'états : pour la sortie d'état de la BD sur papier ou écran,
- générateur d'application : qui fait appel aux écrans et états précédents, et qui permet la création de menus ainsi que des traitements sur la BD par requêtes simples (en SQL) ou en imbriquant des appels à des programmes développés en L3G (ce qui permet de combiner puissance des outils de 4ème génération et souplesse des L3G).
- générateur de schéma conceptuel : pour la description de la structure des données (entités, attributs, liens...). A tous les niveaux, le langage SQL normalisé par l'ANSI est reconnu comme le langage de requête par excellence qui permet la création, la manipulation et le contrôle des données ; il constitue le lien entre les divers composants du SGBDR comme le montre le schéma suivant :



2. DECOUVERTE PRATIQUE DU LANGAGE SQL

2.1 Connexion à la base de données PUBLI

- **SQL Server Management Studio** est un outil client-serveur qui permet d'envoyer des requêtes SQL et d'administrer les serveurs à distance.
- Choisissez « **Utilisez l'authentification SQL Server** » :
 - Nom de connexion : *Stage*
 - Mot de passe : *stage*
- A la connexion, vous êtes dans la base de données par défaut *master*
- Pour accéder à la base de données de démonstration *publi*, sélectionnez la dans la liste déroulante, ou utilisez l'instruction **USE**:

USE PUBLI

- Pour vérifier la connexion, faites une interrogation simple :
- Dans l'onglet Requête tapez :

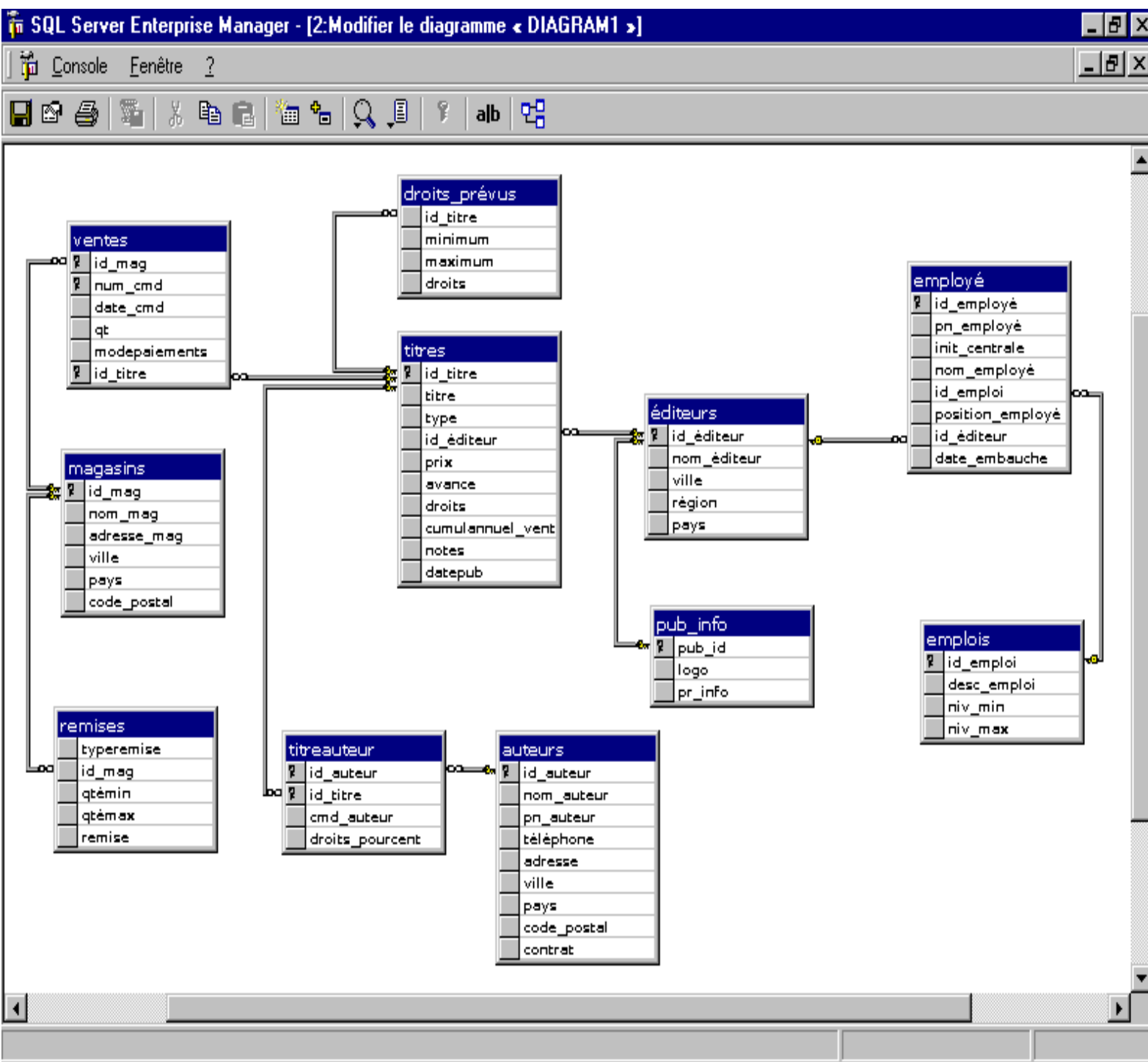
```
select *
from magasins
```

Cliquez sur la flèche verte pour lancer la requête, qui affiche la description des magasins :

<i>id_mag</i>	<i>nom_mag</i>	<i>adresse_mag</i>	<i>ville</i>	<i>pays</i>	<i>code_postal</i>
6380	<i>Eric the Read Books</i>	<i>788, Catamagus Ave.</i>	<i>Seattle</i>	<i>WA</i>	<i>98056</i>
7066	<i>Librairie spécialisée</i>	<i>567, Av. de la Victoire</i>	<i>Paris</i>	<i>FR</i>	<i>75016</i>
7067	<i>Moissons livresques</i>	<i>577, Boulevard Anspach.</i>	<i>Bruxelles</i>	<i>BE</i>	<i>1000</i>
<i>...etc...</i>					

2.2 Description de la base de données PUBLI

- Il y a parfois des petites différences entre la documentation et le contenu réel des tables => pensez à vérifier vos programmes en visualisant les contenus des tables par des requêtes simples
- A partir de SQL Server 7.0, la base originale sur laquelle s'appuient tous les exemples de la documentation Microsoft, est en anglais : **PUBS**. Vous pouvez l'utiliser pour tester des syntaxes documentées dans l'aide, mais les exercices porteront toujours sur la « version française » de la base : **PUBLI**



Publi décrit la base de données d'un groupe d'édition.

- Tous les éditeurs appartenant au groupe sont décrits dans la table **éditeurs**.
- Les éditeurs ont des logo (table **pub_info**), emploient du personnel (table **employé**), et éditent des livres (table **titre**).
- Chaque employé occupe un emploi (table **emplois**)
- Chaque livre est écrit par un ou plusieurs auteurs (table **auteurs** et table intermédiaire **titreauteur**)
- Pour chaque livre vendu, son/ses auteurs touchent des droits, qui sont définis en pourcentage du prix de vente, par tranche, en fonction de la quantité de livres vendus (table **droits_prévus**).
- Les livres sont vendus dans des magasins (table **ventes** et **magasins**)

- Différents types de remise sont consentis sur les livres vendus (table **remises**)

Les commentaires de détail seront donnés table par table.

Table éditeurs

Nom_colonne	Type de données	NULL	Par défaut	Check	Clé/index
<i>id_éditeur</i>	<i>char(4)</i>	non		oui (1)	CP, ordonné.
<i>nom_éditeur</i>	<i>varchar(40)</i>	oui			
<i>ville</i>	<i>varchar(20)</i>	oui			
<i>région</i>	<i>char(2)</i>	oui			
<i>pays</i>	<i>varchar(30)</i>	oui	'USA'		

(1) La contrainte CHECK *id_éditeur* est définie comme (id_éditeur in ('1389', '0736', '0877', '1622', '1756'))
OR id_éditeur LIKE '99[0-9][0-9]'

<i>id_éditeur</i>	<i>nom_éditeur</i>	<i>ville</i>	<i>région</i>	<i>pays</i>
0736	New Moon Books	Boston	MA	USA
0877	Binnet & Hardley	Washington	DC	USA
1389	Algodata Infosystems	Bruxelles	NULL	BE
1622	Five Lakes Publishing	Chicago	IL	USA
1756	Ramona éditeur	Lausanne	NULL	CH
9901	GGG&G	Munich	NULL	GER
9952	Scootney Books	New York	NY	USA
9999	Editions Lucerne	Paris	NULL	FR

Table pub_info

Nom_colonne	Type de données	NULL	Par défaut	Check	Clé/index
<i>pub_id</i>	<i>char(4)</i>	non			CP, ordonné., CE éditeurs(id_éditeur)
<i>logo</i>	<i>image</i>	oui			
<i>pr_info</i>	<i>text</i>	oui			
<i>id_éditeur</i>	<i>logo</i> (1)	<i>info_rp</i> (2)			

0736 NEWMOON.BMP Exemple de données de type text pour New Moon Books, éditeur 0736 dans la base de données pubs. New Moon Books est situé à Boston, Massachusetts.

0877 BINNET.BMP Exemple de données de type text pour Binnet & Hardley, éditeur 0877 dans la base de données pubs. Binnet & Hardley est situé à Washington, D.C.

1389 ALGODATA.BMP Exemple de données de type text pour Algodata Infosystems, éditeur 1389 dans la base de données pubs. Algodata Infosystems est situé à Bruxelles, Belgique.

1622 5LAKES.BMP Exemple de données de type text pour Five Lakes Publishing, éditeur 1622 dans la base de données pubs. Five Lakes Publishing est situé à Chicago, Illinois.

1756 RAMONA.BMP Exemple de données de type text pour Ramona éditeur, éditeur 1756 dans la base de données pubs. Ramona éditeur est situé à Lausanne, Suisse.

9901 GGGG.BMP Exemple de données de type text pour GGG&G, éditeur 9901 dans la base de données pubs. GGG&G est situé à Munich, Allemagne.

9952 SCOOTNEY.BMP Exemple de données de type text pour Scootney Books, éditeur 9952 dans la base de données pubs. Scootney Books est situé à New York City, New York.

9999 LUCERNE.BMP Exemple de données de type text pour Éditions Lucerne, éditeur 9999 dans la base de données pubs. Les Éditions Lucerne sont situées à Paris, France.

(1) Les informations présentées ici NE sont PAS les données réelles. Il s'agit du nom du fichier d'où provient le bitmap (données graphiques).

(2) Le texte présenté ici NE constitue PAS la totalité des données. Lors de l'affichage de données *text*, l'affichage est limité à un nombre fini de caractères. Ces informations présentent les 120 premiers caractères de la colonne de texte.

Table employé

Tous les employés ont un coefficient actuel (colonne **position_employé**), compris entre le coefficient minimum et le coefficient maximum correspondant à leur type d'emploi (**niv_min** et **niv_max** dans la table emplois)

Nom_colonne	Type de données	NULL	Par défaut	Check	Clé/index
id_employé	empid	non		oui (1)	CP, non ordonné.
pn_employé	varchar(20)	non			Composé, ordonné. (2)
init_centrale	char(1)	oui			Composé, ordonné. (2)
nom_employé	varchar(30)	non			Composé, ordonné. (2)
id_emploi	smallint	non	1		CE emplois(id_emploi)
position_employé	tinyint	non	10		
id_éditeur	char(4)	non	'9952'		CE éditeurs(id_éditeur)
date_embauche	datetime	non	GETDATE()		

(1) La contrainte CHECK est définie comme (id_employé LIKE '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][FM]') OR (id_employé LIKE '[A-Z][A-Z][1-9][0-9][0-9][0-9][FM]')

(2) L'index composé, ordonné est défini sur **nom_employé**, **pn_employé**, **init_centrale**.

Les tableaux suivants présentent le contenu de la table **employé**. La première colonne (**id_employé**) est répétée dans la liste du dessous, devant les colonnes 6 à 8. Elle n'est répétée qu'à des fins de lisibilité.

id_employé (#1)	pn_employé (#2)	init_centrale (#3)	nom_employé (#4)	id_emploi (#5)
PMA42628M	Paolo	M	Accorti	13
PSA89086M	Pedro	S	Alfonso	14
VPA30890F	Victoria	P	Ashworth	6
PHB50241M	Patrick	H	Brognon	9
L-B31947F	Lesley	NULL	Brown	7
F-C16315M	Francisco	NULL	Chang	4
PTC11962M	Philip	T	Cramer	2
A-C71970F	Aria	NULL	Cruz	10
PJD25939M	Philippe	J	De Bueger	5
R-D39728F	Renelde	NULL	Depré	8
AMD15433F	Ann	M	Devon	3
ARD36773F	Anabela	R	Domingues	8
PHF38899M	Peter	H	Franken	10
PXH22250M	Paul	X	Henriot	5
PDI47470M	Palle	D	Ibsen	7
KFJ64308F	Karin	F	Josephs	14
MGK44605M	Matti	G	Karttunen	6
POK93028M	Pirkko	O	Koskitalo	10
JYL26161F	Janine	Y	Labruno	5
M-L67958F	Maria	NULL	Larsson	7
Y-L77953M	Yoshi	NULL	Latimer	12
LAL21447M	Laurence	A	Lebihan	5
ENL44273F	Elizabeth	N	Lincoln	14
PCM98509F	Patricia	C	McKenna	11
R-M53550M	Roland	NULL	Mendel	11
HAN90777M	Helvetius	A	Nagy	7
TPO55093M	Timothy	P	O'Rourke	13
SKO22412M	Sven		Ottlieb	5
MAP77183M	Miguel	A	Paolino	11
PSP68661F	Paula	S	Parente	8
M-P91209M	Manuel	NULL	Pereira	8
LCQ23061M	Luc	C	Querton	5
LJR92907F	Laurence	J	Radoux	9
M-R38834F	Martine		Rancé	9
DWR65030M	Diego	W	Roel	6
A-R89858F	Annette	NULL	Roulet	6
CAS28514M	Carlos	A	Santana	5
MMS49649F	Mary	M	Saveley	8
CGS88322F	Carine	G	Schmitt	13
MAS70474F	Margaret	A	Smith	9
HAS54740M	Howard	A	Snyder	12
MFS52347M	Martin	F	Sommer	10
DBT39435M	Daniel	B	Tonini	11

id_employé (#1)	position_employé (#6)	id_éditeur (#7)	date_embauche (#8)
PMA42628M	35	0877	08/12/92
PSA89086M	89	1389	12/12/90
VPA30890F	140	0877	09/10/90
PHB50241M	170	0736	08/09/88
L-B31947F	120	0877	02/11/91
F-C16315M	227	9952	11/02/90

PTC11962M	215	9952	11/11/89
A-C71970F	87	1389	10/12/91
PJD25939M	246	1756	03/01/89
R-D39728F	35	0877	09/11/89
AMD15433F	200	9952	07/12/91
ARD36773F	100	0877	01/12/93
PHF38899M	75	0877	05/12/92
PXH22250M	159	0877	08/11/93
PDI47470M	195	0736	05/09/93
KFJ64308F	100	0736	10/12/92
MGK44605M	220	0736	05/01/94
POK93028M	80	9999	11/11/93
JYL26161F	172	9901	05/12/91
M-L67958F135		1389	03/12/92
Y-L77953M	32	1389	06/11/89
LAL21447M	175	0736	06/03/90
ENL44273F	35	0877	07/12/90
PCM98509F	150	9999	08/01/89
R-M53550M	150	0736	09/05/91
HAN90777M	120	9999	03/11/93
TPO55093M	100	0736	06/11/88
SKO22412M	150	1389	04/05/91
MAP77183M	112	1389	12/07/92
PSP68661F	125	1389	01/12/94
M-P91209M	101	9999	01/09/89
LCQ23061M	198	1622	10/09/93
LJR92907F	170	9999	03/11/94
M-R38834F	75	0877	02/05/92
DWR65030M	192	1389	12/12/91
A-R89858F	152	9999	02/11/90
CAS28514M	211	9999	04/11/89
MMS49649F	175	0736	06/12/93
CGS88322F	64	1389	07/07/92
MAS70474F	78	1389	09/11/88
HAS54740M	100	0736	11/11/88
MFS52347M	165	0736	04/11/90
DBT39435M	75	0877	01/01/90

Table emplois

A chaque type d'emploi, correspond un coefficient minimal (**niv_min**) et un coefficient maximal (**niv_max**)

Nom_colonne	Type de données	NULL	Par défaut Check	Clé/index
<i>id_emploi</i>	<i>smallint</i>	non	IDENTITY(1,1)	CP, ordonné.
<i>desc_emploi</i>	<i>varchar(50)</i>	non	oui (1)	
<i>niv_min</i>	<i>tinyint</i>	non	oui (2)	
<i>niv_max</i>	<i>tinyint</i>	non	oui (3)	

<i>id_emploi</i>	<i>desc_emploi</i>	<i>niv_min</i>	<i>niv_max</i>
1	Nouveau collaborateur — Poste non spécifié	10	10
2	Directeur général	200	250
3	Directeur des opérations commerciales	175	225
4	Responsable financier	175	250
5	Editeur	150	250
6	Editeur en chef	140	225
7	Directeur du marketing	120	200
8	Directeur des relations publiques	100	175
9	Directeur des achats	75	175
10	Directeur de la production	75	165
11	Directeur des opérations	75	150
12	Rédacteur	25	100
13	Représentant commercial	25	100
14	Graphiste	25	100

(1) La contrainte DEFAULT est définie comme ("Nouveau poste - pas de dénomination officielle").

(2) La contrainte CHECK *niv_min* est définie comme (*niv_min* >= 10). (3) La contrainte CHECK *niv_max* est définie comme (*niv_max* <= 250).

Table auteurs

Certains auteurs travaillent sous contrat avec leur éditeur (colonne **contrat** de type **bit**)

Nom_colonne	Type de données	NULL	Par défaut	Check	Clé/Index
<i>id_auteur</i>	id	non		oui (1)	CP, .
<i>nom_auteur</i>	varchar(40)	non			Composé, non ordonné (3)
<i>pn_auteur</i>	varchar(20)	non			Composé, non ordonné (3)
<i>téléphone</i>	char(12)	non	'INCONNU'		
<i>adresse</i>	varchar(40)	oui			
<i>ville</i>	varchar(20)	oui			
<i>pays</i>	char(2)	oui			
<i>code_postal</i>	char(5)	oui		oui (2)	
<i>contrat</i>	bit	non			

(1) La contrainte CHECK *id_auteur* est définie comme (id_auteur LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]').

(2) La contrainte CHECK *code_postal* est définie comme (code_postal LIKE '[0-9][0-9][0-9][0-9][0-9]').

(3) L'index composé non ordonné est défini sur *nom_auteur*, *pn_auteur*.

Les tables ci-dessous présentent le contenu de la table *auteurs*. La première colonne (*id_auteur*) est répétée dans la liste du dessous, devant les colonnes 5 à 9. Elle n'est répétée qu'à des fins de lisibilité.

<i>id_auteur</i> (#1)	<i>nom_auteur</i> (#2)	<i>pn_auteur</i> (#3)	<i>téléphone</i> (#4)			
172-32-1176	Bourne	Stéphanie	45.33.08.49			
213-46-8915	Mathieu	Charles	93.11.73.35			
238-95-7766	Chartier	Laurent	38.66.24.53			
267-41-2394	Médina	Marguerite	48.96.77.44			
274-80-9391	Merrell	Patricia	42.27.44.35			
341-22-1782	Lorense	Danielle	02.32.89.34			
409-56-7008	Bucchia	Patrice	42.01.84.27			
427-17-2319	Logerot	Philippe	42.24.89.91			
472-27-2349	Schildwachter	Xavier	47.04.44.77			
486-29-1786	Vue	Jessica	42.71.09.71			
527-72-3246	Posey	William	69.15.11.00			
648-92-1872	Sorensen	Christophe	56.79.96.29			
672-71-3249	De Verne	Vincent	45.48.12.23			
712-45-1867	Vilc	Benjamin	93.30.24.68			
722-51-5454	Hall	Catherine	54.43.36.78			
724-08-9931	D'Autricourt	Alain	46.36.37.99			
724-80-9391	Lacouture	Gilles	45.04.48.78			
756-30-7391	Jalabert	Marc	45.22.88.91			
807-91-6654	Bec	Arthur	22.47.87.11			
846-92-7186	Letournec	Benoît	48.54.31.99			
893-72-1158	Facq	Jean-Rémy	56.48.05.44			
899-46-2035	Chevalier	Anne	20.24.54.21			
998-72-3567	Chevalier	Bernard	20.24.54.20			

<i>id_auteur</i> (#1)	<i>adresse</i> (#5)	<i>ville</i> (#6)	<i>pays</i> (#7)	<i>code_postal</i> (#8)	<i>contrat</i> (#9)
172-32-1176	4, square Gauguin	Nice	FR	06014	1
213-46-8915	18, avenue Arbabi	Paris	FR	75017	1
238-95-7766	4, impasse Lamoix	Bordeaux	FR	33000	1
267-41-2394	48, passage Sainte Anne	Toulouse	FR	31002	1
274-80-9391	2, place du Général Catroux	Paris	FR	75015	1
341-22-1782	14, impasse Lacarte	Liège	BE	01548	0
409-56-7008	201, boulevard de Clichy	Bordeaux	FR	33000	1
427-17-2319	32, rue de l'Amiral Cloué	Luxembourg	LU	01016	1
472-27-2349	11, rue Buffon	Bruxelles	BE	02530	1
486-29-1786	62-64, rue Vieille du Temple	Chevette	FR	91450	1
527-72-3246	57, avenue des tapis	Zurich	CH	91450	0
648-92-1872	55, rue Pierre-Louis Coll	Zurich	CH	91450	1
672-71-3249	113, rue du Cherche-Midi	Vers-La-Petite	FR	91712	1
712-45-1867	18, faubourg Saint Jean	Monte Carlo	MC	98000	1
722-51-5454	12, rue Astarte	Luxembourg	LU	02016	1
724-08-9931	75, rue des Couronnes	Paris	FR	75017	0
724-80-9391	20, rue de la Pompe	Marseille	FR	13016	1
756-30-7391	94, rue de la Condamine	Paris	FR	75020	1
807-91-6654	4, chemin de la Tour de Campel	Bruges	BE	05006	1
846-92-7186	32, rue du Mont Thabor	Lille	FR	59000	1
893-72-1158	59, rue Lisleferme	Paris	FR	75018	0
899-46-2035	48, rue de Valmy	Genève	CH	91712	1
998-72-3567	48, rue de Valmy	Genève	CH	91712	1

Table *titreauteur*

Attention : certains auteurs peuvent être décrits dans la table *Auteurs* sans être présents dans la table *titreauteur*, s'ils n'ont encore rien écrit (cas des auteurs sous contrat qui écrivent leur premier livre)

Nom_colonne	Type de données	NULL	Par défaut	Check	Clé/index
<i>id_auteur</i>	<i>id</i>	non			Composé CP, ordonné., (1) CE auteurs(<i>id_auteur</i>) (2)
<i>id_titre</i>	<i>tid</i>	non			Composé CP, ordonné., (1) CE titres(<i>id_titre</i>) (3)
<i>cmd_auteur</i>	<i>tinyint</i>	oui			
<i>droits_pourcent</i>	<i>int</i>	oui			

- (1) L'index composé, clé primaire, ordonné est défini sur *id_auteur*, *id_titre*.
 (2) Cette clé étrangère a également un index non ordonné sur *id_auteur*.
 (3) Cette clé étrangère a également un index non ordonné sur *id_titre*.

<i>id_auteur</i>	<i>id_titre</i>	<i>cmd_auteur</i>	<i>droits_pourcent</i>
172-32-1176	PS3333	1	100
213-46-8915	BU1032	2	40
213-46-8915	BU2075	1	100
238-95-7766	PC1035	1	100
267-41-2394	BU1111	2	40
274-80-9391	BU7832	1	100
409-56-7008	BU1032	1	60
427-17-2319	PC8888	1	50
486-29-1786	PC9999	1	100
486-29-1786	PS7777	1	100
712-45-1867	MC2222	1	100
722-51-5454	MC3021	1	75
724-80-9391	BU1111	1	60
724-80-9391	PS1372	2	25
756-30-7391	PS1372	1	75
846-92-7186	PC8888	2	50
899-46-2035	MC3021	2	25
899-46-2035	PS2091	2	50
998-72-3567	PS2091	1	50
998-72-3567	PS2106	1	100

Table *titres*

Certains auteurs perçoivent une avance sur recette, pendant l'écriture de leur livre (colonne *avance*). Pour chaque titre, on tient à jour le nombre d'ouvrages vendus, tous magasins confondus (colonne *cumulannuel_ventes*), et la tranche de droit d'auteur atteinte par ce livre (colonne *droits*)

Nom_colonne	Type de données	NULL	Par défaut	Check	Clé/index
<i>id_titre</i>	<i>tid</i>	non			CP, ordonné.
<i>titre</i>	<i>varchar(80)</i>	non			Non ordonné.
<i>type</i>	<i>char(12)</i>	non		'UNDECIDED'	
<i>id_éditeur</i>	<i>char(4)</i>	oui			CE éditeurs(<i>id_éditeur</i>)
<i>prix</i>	<i>money</i>	oui			
<i>avance</i>	<i>money</i>	oui			
<i>droits</i>	<i>int</i>	oui			
<i>cumulannuel_ventes</i>	<i>int</i>	oui			
<i>notes</i>	<i>varchar(200)</i>	oui			
<i>datepub</i>	<i>datetime</i>	non		GETDATE()	

Les tableaux ci-après présentent le contenu de la table *titres*. La première colonne (*id_titre*) est répétée dans les listes du dessous, d'abord devant les colonnes 5 à 8, ensuite devant les colonnes 9 et 10. Elle n'est répétée qu'à des fins de lisibilité.

<i>id_titre</i> (#1)	<i>titre</i> (#2)	<i>t</i>	<i>type</i> (#3)	<i>id_éditeur</i> (#4)	<i>prix</i> (#5)
BU1032	Guide des bases de données du gestionnaire pressé		gestion	1389	140,00
BU1111	La cuisine - l'ordinateur : bilans clandestins		gestion	1389	82,00
BU2075	Le stress en informatique n'est pas une fatalité!		gestion	0736	24,00
BU7832	Toute la vérité sur les ordinateurs		gestion	1389	136,00
MC2222	Les festins de Party 2		cui_moderne	0877	136,00
MC3021	Les micro-ondes par gourmandise		cui_moderne	0877	21,00
MC3026	La psychologie des ordinateurs de cuisine		SANS TITRE	0877	NULL
PC1035	Est-ce vraiment convivial?		informatique	1389	156,00
PC8888	Les secrets de la Silicon Valley		informatique	1389	136,00
PC9999	Guide des bonnes manières sur un réseau		informatique	1389	NULL
PS1372	Phobie et passion informatique : éventail de comportements		psychologie	0877	147,00
PS2091	La colère : notre ennemie?		psychologie	0736	76,00
PS2106	Vivre sans crainte		psychologie	0736	49,00

PS3333	Privation durable d'informations : étude de quatre cas représentatifs	psychologie	0736	136,00
PS7777	Equilibre émotionnel : un nouvel algorithme	psychologie	0736	54,00

id_titre (#1)	avance (#6)	droits (#7)	cumulannuel_ventes (#8)
BU1032	35.000	10	4095
BU1111	34.000	10	3876
BU2075	69.000	24	18722
BU7832	34.000	10	4095
MC2222	0.00	12	2032
MC3021	102.000	24	22246
MC3026	NULL	NULL	NULL
PC1035	48.000	16	8780
PC8888	54.000	10	4095
PC9999	NULL	NULL	NULL
PS1372	48.000	10	375
PS2091	15.000	12	2045
PS2106	41.000	10	111
PS3333	14.000	10	4072
PS7777	27.000	10	3336

id_titre (#1)	notes (#9)	datepub (#10)
BU1032	Vue d'ensemble illustrée des systèmes de gestion de base de données disponibles sur le marché. L'accent est mis sur les applications de gestion courantes.	06/12/85
BU1111	Conseils utiles vous permettant de tirer le meilleur parti possible de vos ressources informatiques.	06/09/85
BU2075	Exposé des techniques médicales et psychologiques les plus récentes permettant de survivre dans le bureau électronique. Explications claires et détaillées.	06/12/85
BU7832	Analyse commentée des possibilités offertes par les ordinateurs : un guide impartial pour l'utilisateur critique.	06/12/85
MC2222	Recueil de recettes rapides, faciles et élégantes, testées et goûtées par des gens qui n'ont jamais le temps de manger. Aide précieuse pour le cuisinier occasionnel.	06/09/85
MC3021	Adaptation de recettes traditionnelles des provinces françaises - la cuisine au micro-ondes.	06/09/85
MC3026	NULL	17/05/96
PC1035	Etude comparative des progiciels les plus répandus. S'adressant aux utilisateurs débutants, cet ouvrage établit un palmarès des logiciels en fonction de leur convivialité.	06/12/85
PC8888	Deux femmes courageuses dévoilent tous les scandales qui jonchent l'irrésistible ascension des pionniers de l'informatique. Matériel et logiciel : personne n'est épargné.	06/12/85
PC9999	La bible des débutants dans un environnement réseau.	17/05/96
PS1372	Lecture indispensable pour le spécialiste : cet ouvrage étudie les différences entre ceux qui détestent et craignent les ordinateurs et ceux qui les trouvent épatants.	10/11/85
PS2091	Etude approfondie des conséquences somatiques des émotions fortes. De nombreux schémas du métabolisme illustrent l'exposé et en facilitent la compréhension.	06/11/85
PS2106	Comment amortir le choc des interactions quotidiennes par la gymnastique, la méditation et la diététique (nombreux exemples de menus). Bandes vidéo sur commande pour les exercices physiques.	10/05/85
PS3333	Que se passe-t-il quand les données viennent à manquer? Analyse scientifique des effets du manque d'information sur les grands consommateurs.	06/12/85
PS7777	Comment se protéger contre le stress exagéré du monde moderne. Parmi les remèdes proposés : utilisation de l'ordinateur et alimentation judicieusement choisie.	06/12/85

Table droits_prévus

Les auteurs perçoivent des droits croissants sur leurs livres, en fonction de la quantité vendue : les droits sont exprimés en pourcentage (colonne **droits**), pour chaque tranche de livres vendus (nombre de livres entre **minimum** et **maximum**). Exemple : Si l'on vend 3500 livres "BU1035", son auteur percevra 10% sur les 2000 premiers livres, 12% du livre 2001 au 3000, 14% du 3001 au 3500.

Nom_colonne	Type de données	NULL	Par défaut	Check	Clé/index
id_titre	idt	non			CE titres(id_titre)
minimum	int	oui			
maximum	int	oui			
droits	int	oui			

id_titre minimum maximum droits

BU1032	0	5000	10
BU1032	5001	50000	12
PC1035	0	2000	10
PC1035	2001	3000	12
PC1035	3001	4000	14
PC1035	4001	10000	16
PC1035	10001	50000	18
BU2075	0	1000	10
BU2075	1001	3000	12
BU2075	3001	5000	14
BU2075	5001	7000	16
BU2075	7001	10000	18
BU2075	10001	12000	20
BU2075	12001	14000	22
BU2075	14001	50000	24
PS2091	0	1000	10
PS2091	1001	5000	12
PS2091	5001	10000	14
PS2091	10001	50000	16
PS2106	0	2000	10
PS2106	2001	5000	12
PS2106	5001	10000	14
PS2106	10001	50000	16
MC3021	0	1000	10
MC3021	1001	2000	12
MC3021	2001	4000	14
MC3021	4001	6000	16
MC3021	6001	8000	18
MC3021	8001	10000	20
MC3021	10001	12000	22
MC3021	12001	50000	24
PC8888	0	5000	10
PC8888	5001	10000	12
PC8888	10001	15000	14
PC8888	15001	50000	16
PS7777	0	5000	10
PS7777	5001	50000	12
PS3333	0	5000	10
PS3333	5001	10000	12
PS3333	10001	15000	14
PS3333	15001	50000	16
BU1111	0	4000	10
BU1111	4001	8000	12
BU1111	8001	10000	14
BU1111	12001	16000	16
BU1111	16001	20000	18
BU1111	20001	24000	20
BU1111	24001	28000	22
BU1111	28001	50000	24
MC2222	0	2000	10
MC2222	2001	4000	12
MC2222	4001	8000	14
MC2222	8001	12000	16
MC2222	12001	20000	18
MC2222	20001	50000	20
BU7832	0	5000	10
BU7832	5001	10000	12
BU7832	10001	15000	14
BU7832	15001	20000	16
BU7832	20001	25000	18
BU7832	25001	30000	20
BU7832	30001	35000	22
BU7832	35001	50000	24
PS1372	0	10000	10
PS1372	10001	20000	12
PS1372	20001	30000	14
PS1372	30001	40000	16
PS1372	40001	50000	18

Table ventes

Les magasins envoient des commandes aux éditeurs. Chaque commande porte sur un ou plusieurs livres. Chaque magasin possède ses propres conventions de numérotation des commandes. Pour décrire une ligne de commande de façon unique, il faut donc connaître : le magasin qui l'a émise (colonne **id_mag**), le numéro de commande (**num_cmd**), le titre du livre commandé (**id_titre**). La table Ventes possède donc une clé primaire composée, constituée de ces trois colonnes.

Nom_colonne	Type de données	NULL	Clé/index
<i>id_mag</i>	char(4)	non	Composé CP, ordonné. (1) , CE magasins(<i>id_mag</i>)
<i>num_cmd</i>	varchar(20)	non	Composé CP, ordonné. (1)
<i>date_cmd</i>	datetime	non	
<i>qt</i>	smallint	non	
<i>modepaiements</i>	varchar(12)	non	
<i>id_titre</i>	idt	non	Composé CP, ordonné., (1) CE titres(<i>id_titre</i>)

(1) L'index composé, clé primaire, ordonné est défini sur *id_mag*, *num_cmd*, *id_titre*.

<i>id_mag</i>	<i>num_cmd</i>	<i>date_cmd</i>	<i>qt</i>	<i>modepaiements</i>	<i>id_titre</i>
6380	722a	09/11/94	3	Net 60	PS2091
6380	6871	09/12/94	5	Net60	BU1032
7066	GAR001	24/03/04	25	Net30	PS2091
7066	A2976	05/12/93	50	Net 30	PC8888
7066	QA7442.3	09/12/94	75	Comptant	PS2091
7067	D4482	09/12/94	10	Net 60	PS2091
7131	N914008	09/12/94	20	Net 30	PS2091
7131	N914014	09/12/94	25	Net 30	MC3021
7131	P3087a	05/12/93	20	Net 60	PS1372
7131	P3087a	05/12/93	25	Net 60	PS2106
7131	P3087a	05/12/93	15	Net 60	PS3333
7131	P3087a	05/12/93	25	Net 60	PS7777
7896	QQ2299	10/12/93	15	Net 60	BU7832
7896	TQ456	12/12/93	10	Net 60	MC2222
7896	X999	02/11/93	35	Comptant	BU2075
8042	423LL922	09/12/94	15	Comptant	MC3021
8042	423LL930	09/12/94	10	Comptant	BU1032
8042	P723	03/11/93	25	Net 30	BU1111
8042	QA879.1	05/11/93	30	Net 30	PC1035

Table magasins

Nom_colonne	Type de données	NULL	Par défaut	Check	Clé/index
<i>id_mag</i>	char(4)	non			CP, ordonné.
<i>nom_mag</i>	varchar(40)	oui			
<i>adresse_mag</i>	varchar(40)	oui			
<i>ville</i>	varchar(20)	oui			
<i>pays</i>	char(2)	oui			
<i>code_postal</i>	char(5)	oui			

<i>id_mag</i>	<i>nom_mag</i>	<i>adresse_mag</i>	<i>ville</i>	<i>pays</i>	<i>code_postal</i>
6380	Eric the Read Books	788 Catamaugus Ave.	Seattle	WA	98056
7066	Librairie spécialisée	567, Av. de la Victoire	Paris	FR	75016
7067	Moissons livresques	577, Boulevard Anspach.	Bruxelles	BE	1000
7131	Doc-U-Mat: Quality Laundry and Books	24-A Avrogado Way	Remulade	WA	98014
7896	Fricative Bookshop	89 Madison St.	Fremont	CA	90019
8042	Bookbeat	679 Carson St.	Portland	OR	89076

Table remises

Les éditeurs consentent trois types de remises : des « remises client » à certains magasins privilégiés (référéncés par la colonne *id_mag*) ; des « remises en volume », en fonction de la quantité commandée (entre *qtémin* et *qtémax*) ; une remise initiale (type FNAC) à tous les magasins du groupe.

Nom_colonne	Type de données	NULL	Par défaut	Check	Clé/index
<i>typeremise</i>	varchar(40)	non			
<i>id_mag</i>	char(4)	oui			CE magasins(<i>id_mag</i>)
<i>qtémin</i>	smallint	oui			
<i>qtémax</i>	smallint	oui			
<i>remise</i>	decimal	non			

<i>typeremise</i>	<i>id_mag</i>	<i>qtémin</i>	<i>qtémax</i>	<i>remise</i>
Remise initiale	NULL	NULL	NULL	10.5

Remise Volume	NULL	100	1000	6.7
Remise client	8042	NULL	NULL	5.0

2.3 Consultation d'une base de données



Avec sql server management studio express

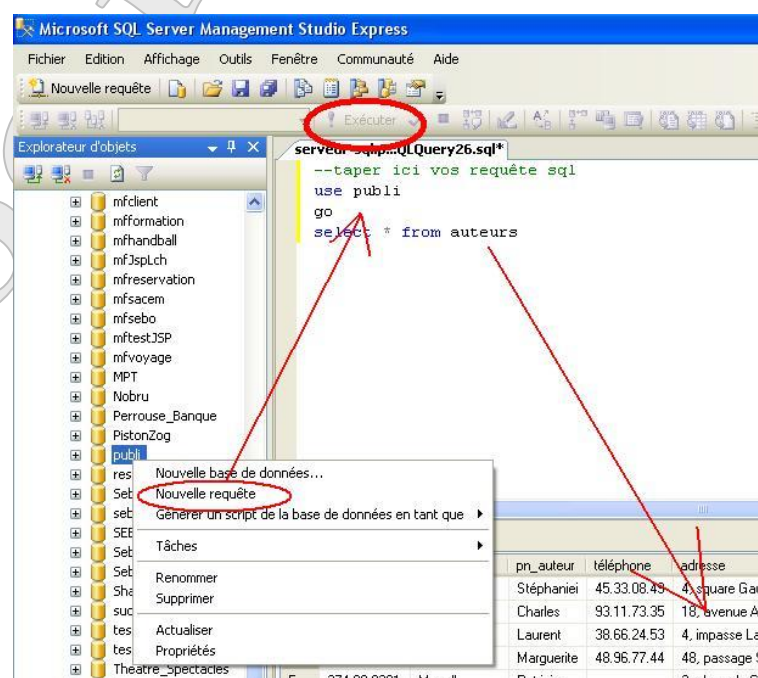


Microsoft Sql management studio express pages 5 et 12 (pour prendre en main l'outil et exécuter les exercices ci après)

- lancez l'outil par le menu « SQL Server 2005 → Management studio express »
- A l'invite loguez vous avec votre identifiant sql serveur (user :votre nom / password :votre nom).



- Développez « Bases de données » : seules les bases de données auxquelles vous avez accès apparaissent. Développer la base publi.
- Ouvrez une fenêtre requête sql pour tester vos requêtes



Requête Select simple sur une seule table



Doc. en Ligne : « Principe de bases des requêtes »
SQL 2 : p. 43-60

- L'instruction SELECT spécifie les colonnes que vous voulez récupérer. La clause FROM spécifie les tables dans lesquelles se trouvent les colonnes. La clause WHERE spécifie les lignes que vous voulez visualiser dans les tables. Syntaxe simplifiée de l'instruction SELECT :

```
SELECT liste_de_sélection
FROM liste_de_tables
WHERE critères_de_sélection
```

Par exemple, l'instruction SELECT suivante extrait les nom et prénom des écrivains de la table *auteurs* vivant à *Paris*.

```
SELECT pn_auteur, nom_auteur
FROM auteurs
WHERE ville = 'Paris'
```

<i>pn_auteur</i>	<i>nom_auteur</i>
Charles	Mathieu
Patricia	Merrell
Alain	D'Autricourt
Marc	Jalabert
Jean-Rémy	Facq



Exercice 1 : Afficher le nom, la ville et la région de tous les éditeurs

- La syntaxe complète de l'instruction SELECT :

```
SELECT [ALL | DISTINCT] liste_de_sélection
      [INTO [nouveau_nom_de_table]]
      [FROM {nom_de_table | nom_de_vue}[(conseils-optimiseur)]
      [[, {nom_de_table2 | nom_de_vue2}[(conseils-optimiseur)]
      [..., {nom_de_table16 | nom_de_vue16}[(conseils-optimiseur)]]]
      [clause WHERE]
      [clause GROUP BY]
      [clause HAVING]
      [clause ORDER BY]
      [clause COMPUTE]
      [FOR BROWSE ]
```

- Les clauses d'une instruction SELECT doivent être utilisées dans l'ordre décrit ci-dessus. (Par exemple, si l'instruction comprend une clause GROUP BY et une clause ORDER BY, la clause GROUP BY précède la clause ORDER BY).

- Le nom des objets de la base de données doit être déterminé si le choix d'un objet est ambigu : vous pouvez alors préciser le nom de la base de données, du propriétaire, de la table, en les séparant par des points :

base_de_données.propriétaire.nom_de_table.nom_de_colonne

Par exemple, si l'utilisateur *suzanne* possède la table *auteurs* dans la base de données *pubs*, l'identificateur unique de la colonne *ville* dans cette table est:

pubs.suzanne.auteurs.ville

- Vous pouvez demander un tri en fonction du nom de colonne ou de la position de colonne dans la liste de sélection. Les deux requêtes suivantes sont équivalentes:

```
SELECT pn_auteur, nom_auteur
FROM auteurs
ORDER BY nom_auteur
```

```
ORDER BY 2
```

Pour plus d'informations sur SELECT, consultez l'instruction SELECT dans le *Manuel de référence Transact-SQL* (choisissez *Select Examples (T-SQL)*)

Sélection de lignes : clause Where

Les critères de sélection, ou conditions, de la clause WHERE peuvent inclure:

- Des opérateurs de comparaison (tels que =, <>, <, et >)

```
WHERE avance * 2 > cumulannuel_ventes * prix
```

- Des intervalles (BETWEEN et NOT BETWEEN)

```
WHERE cumulannuel_ventes between 4095 and 12000
```

- Des listes (IN, NOT IN)

```
WHERE state in ('BE', 'CH', 'LU')
```

- Des concordances avec des modèles (LIKE et NOT LIKE)

```
WHERE téléphone not like '45%'
```

% Toute chaîne de zéro caractère ou plus

_ Tout caractère unique

[a-f] Tout caractère de l'intervalle ([a-f]) ou de l'ensemble spécifié ([abcdef])

[^a-f] Tout caractère en dehors de l'intervalle ([^a-f]) ou de l'ensemble spécifié ([^abcdef])

- Des valeurs inconnues (IS NULL et IS NOT NULL)

```
where avance is null
```

SQL sait gérer la notion de valeur non définie

- Des combinaisons de ces critères (AND, OR)

```
where avance < 30000 or (cumulannuel_ventes > 2000 and
cumulannuel_ventes < 2500)
```



Exercice 2 : WHERE

Afficher le titre des livres dont le prix est supérieur à 100.

Exercice 3 : LIKE, BETWEEN, AND

Afficher le nom, le prénom, et la date d'embauche des employés embauchés en 90, dont le nom commence par 'L', et la position est comprise entre 10 et 100

Exercice 4 : ORDER BY

Afficher le nom et la date d'embauche des employés, classés par leur identificateur d'éditeur, puis par leur nom de famille (sous-critère)

Exercice 5 : IN, ORDER BY

Afficher le nom, le pays et l'adresse des auteurs Français, Suisse ou Belge, classés par pays.

Expressions et fonctions

- Les 4 opérateurs arithmétiques de base peuvent être utilisés dans les clauses SELECT, WHERE et ORDER pour affiner les recherches partout où il est possible d'utiliser une valeur d'attribut :

Livres qui reçoivent une avance sur vente supérieure à 500 fois leur prix :

```
SELECT titre, prix, avance  
FROM titres  
WHERE avance >= 500 * prix
```

- Suivant les systèmes la gamme des opérateurs et des fonctions utilisables peut être très évoluée : elle comporte toujours des fonctions de traitement de chaînes de caractères, des opérateurs sur les dates, etc...

Cet exemple détermine la différence en jours entre la date courante et la date de publication :

```
SELECT newdate = DATEDIFF(day, datepub, getdate())  
FROM titres
```



Etudiez les différentes fonctions existantes, en annexe 3 de ce document

**Manipulation de fonctions :**

Exercice 6 : Affichez l'année d'embauche de Janine Labrune (fonction DATEPART).

Exercice 7 : Affichez le mois d'embauche de Janine Labrune (fonction DATENAME).

Exercice 8 : Affichez le jour d'embauche de Janine Labrune (fonction DAY).

Exercice 9 : Affichez la date d'embauche de Janine Labrune au format 'JJ/MM/AAAA' (fonction CONVERT)

Exercice 10 : Affichez la date d'embauche de Janine Labrune au format 'JJ-MM-AA' (fonction CONVERT)

Exercice 11 : Affichez la date d'embauche de Janine Labrune au format 'JJ.MM.AA' (fonctions DAY, MONTH, YEAR et CAST)

Exercice 12 : Affichez tous les noms, prénoms et date d'embauche des employés embauchés après le 24/03/1991 (fonction DATEDIFF)

Exercice 13 : Affichez le numéro du département (2 premiers caracteres du code postal) dans la table auteur (fonction SUBSTRING)

Exercice 14 : Affichez tous les noms des auteurs en majuscule

Exercice 15 : Affichez tous les noms et prénoms des auteurs en enlevant les accents (remplacer é par e).

Requête sur les groupes : GROUP BY, fonctions de groupe, HAVING



Doc. en ligne « Concepts de requêtes avancées » : « Regroupement de lignes à l'aide de Group by »

- La clause **GROUP BY** est employée dans les instructions **SELECT** pour diviser une table en groupes. Vous pouvez regrouper vos données par nom de colonne ou en fonction des résultats des colonnes calculées lorsque vous utilisez des données numériques. L'instruction suivante calcule l'avance moyenne et la somme des ventes annuelles cumulées pour chaque type de livre:

```
SELECT type, AVG(avance), SUM(cumulannuel_ventes)
FROM titres
GROUP BY type
```

type

cui_moderne	51,000.00	24278
cui_traditio	43,000.00	19566
gestion	43,000.00	30788
informatique	51,000.00	12875
psychologie	29,000.00	9939
SANS TITRE	(null)	(null)
(6 ligne(s) affectée(s))		

- Les **fonctions de groupe** (ou « d'agrégation ») effectuent un calcul sur l'ensemble des valeurs d'un attribut d'un groupe de tuples. Un groupe est un sous ensemble des tuples d'une table tel que la valeur d'un attribut y reste constante ; un groupe est spécifié au moyen de la clause **GROUP BY** suivi du nom de l'attribut à l'origine du groupement. En l'absence de cette clause tous les tuples sélectionnés forment le groupe.
 - COUNT**, compte les occurrences pour un attribut,
 - SUM**, somme les valeurs de l'attribut (de type numérique),
 - AVG**, fait la moyenne (Average) des valeurs de l'attribut,
 - MAX**, **MIN**, donne la valeur MAX et la valeur MIN de l'attribut.
- Ces fonctions imposent la spécification de l'attribut en tant qu'argument. Si cet argument est précédé du mot clé **DISTINCT** les répétitions sont éliminées. D'autre part, les valeurs indéterminées (= NULL) ne sont pas prises en compte.

- La clause HAVING est équivalente à WHERE mais elle se rapporte aux groupes. Elle porte en général sur la valeur d'une fonction de groupe : seuls les groupes répondant au critère spécifié par HAVING feront parti du résultat.

Regrouper les titres en fonction du type, en éliminant les groupes qui contiennent un seul livre

```
SELECT type
FROM titres
GROUP BY type
HAVING COUNT(*) > 1
```

```
type
-----
cui_moderne
cui_tradition
gestion
informatique
psychologie
```

Regrouper les titres en fonction du type, en se limitant aux types qui débutent par la lettre «c»

```
SELECT type
FROM titres
GROUP BY type
HAVING type LIKE 'c%'
```

```
type
-----
cui_moderne
cui_tradition
```

Regrouper les titres en fonction du type par éditeur, en incluant seulement les éditeurs dont le numéro d'identification est supérieur à 0800 et qui ont consenti des avances pour un total supérieur à 90 000 FF, et qui vendent des livres pour un prix moyen inférieur à 150 FF:

```
SELECT id_éditeur, SUM(avance), AVG(prix)
FROM titres
GROUP BY id_éditeur
HAVING SUM(avance) > 90000
      AND AVG(prix) < 150
      AND id_éditeur > '0800'
```

Même exemple, en éliminant les titres dont le prix est inférieur à 60 FF, et en triant les résultats en fonction des numéros d'identification des éditeurs:

Remarque que la clause where sélectionne des lignes, la clause having sélectionne des groupes

```
SELECT id_éditeur, SUM(avance), AVG(prix)
FROM titres
WHERE prix >= 60
GROUP BY id_éditeur
HAVING SUM(avance) > 90000
      AND AVG(prix) < 150
      AND id_éditeur > '0800'
ORDER BY id_éditeur
```



Exercice 16 : GROUP BY, COUNT

Afficher les pays et les nombres d'auteur par pays (table auteurs).

Exercice 17 : GROUP BY, SUM

Afficher les identifiants magasin et le cumul des quantités vendues par magasin (table ventes).

Exercice 18 : GROUP BY, COUNT, MIN, MAX

Pour chaque niveau d'emploi (table employés, colonne position_employé) affichez le nombre d'employés de ce niveau, la date d'embauche du salarié le plus ancien et du plus récent dans le niveau

Exercice 19 : GROUP BY, clause sur un sous-ensemble HAVING

Afficher le nombre des éditeurs regroupés par pays, en se limitant aux pays dont le nom contient un 'S' ou un 'R'

Exercice 20 : GROUP BY, HAVING

Afficher les pays ayant plus de 3 auteurs, et le nombre d'auteurs de chacun de ces pays.

Travail sur plusieurs tables : les jointures



Doc. en ligne : « Principe de base des jointures »
SQL2 : p. 61 à 78

- Les opérations de jointure permettent d'extraire des données à partir de plusieurs tables ou vues dans la même base de données ou dans des bases différentes en n'effectuant qu'une seule opération. Joindre deux ou plusieurs tables revient à comparer les données de colonnes spécifiques, et ensuite à utiliser les lignes sélectionnées dans les résultats de cette comparaison pour créer une nouvelle table.
- Une instruction de jointure:
 - spécifie une colonne dans chaque table ;
 - compare les valeurs de ces colonnes ligne par ligne ;
 - forme de nouvelles lignes en combinant les lignes qui contiennent les valeurs retenues dans la comparaison.

Exemple de jointure : nom des auteurs et des éditeurs vivant dans la même ville :

```
SELECT pn_auteur, nom_auteur, nom_éditeur
FROM auteurs, éditeurs
WHERE auteurs.ville = éditeurs.ville
```

pn_auteur	nom_auteur	nom_éditeur
Xavier	Schildwachter	Algodata Infosystems
Alain	D'Autricourt	Editions Lucerne
Charles	Mathieu	Editions Lucerne
Jean-Rémy	Facq	Editions Lucerne
Marc	Jalabert	Editions Lucerne
Patricia	Merrell	Editions Lucerne

Ou avec l'opérateur **JOIN** dans la syntaxe SQL 2 :


```
SELECT pn_auteur, nom_auteur, nom_éditeur
FROM auteurs
JOIN éditeurs on auteurs.ville = éditeurs.ville
```



Exercice 21 : Jointure entre deux tables

Afficher les noms des éditeurs parisiens, et les titres et les prix des livres qu'ils éditent.

Exercice 22 : Jointure entre trois tables

Afficher les noms des auteurs parisiens, et les titres et les prix des livres qu'ils écrivent.

Exercice 23 : Jointure , COUNT

Afficher les nombre de livres écrits par des auteurs sous contrat (colonne contrat = 1)

Exercice 24 : Jointure sur quatre tables, ORDER BY

Afficher le nom de l'éditeur, les titres des livres qu'il publie, les noms des magasins où ils sont vendus, le nombre d'exemplaires vendus dans chaque magasin

nom_éditeur	titre	nom_mag	qt
Algodata Infosystems	Guide des bases de données du gestionnaire pressé	Eric the Read Books	5
Algodata Infosystems	Guide des bases de données du gestionnaire pressé	Bookbeat	10
Algodata Infosystems	La cuisine - l'ordinateur : bilans clandestins	Bookbeat	25
Algodata Infosystems	Toute la vérité sur les ordinateurs	Fricative Bookshop	15
Binnet & Hardley	Les festins de Parly 2	Fricative Bookshop	10
Binnet & Hardley	Les micro-ondes par gourmandise	Doc-U-Mat: Quality Laundry and Books	25
Etc...			

Exercice 25 : jointure sur 4 tables, GROUP BY, HAVING, SUM

Afficher les noms des auteurs qui ont vendu au moins 20 livres, et le nombre de livres qu'ils ont vendus (tables auteurs, titreauteur, titres, ventes)

Exercice 26 : jointure

Afficher la liste des livres vendus dans la même ville que leur auteur

Exercice 27 : jointure

Afficher les noms des éditeurs vivants dans le même pays qu'un auteur

Exercice 28 : jointure

Afficher les noms des éditeurs vivants dans le même pays que les auteurs des livres qu'ils éditent

Les sous-requêtes



Doc. en ligne : « Concepts de requêtes avancées » : « Principe de base des sous-requêtes »

- Une sous-requête est une instruction SELECT imbriquée à l'intérieur d'une instruction SELECT, INSERT, UPDATE ou DELETE, ou d'une autre sous-requête : une sous-requête peut porter sur la même table que la requête externe ou sur une autre table.
- Dans Transact-SQL, une sous-requête qui renvoie une seule valeur peut s'employer dans toutes les circonstances où une expression est autorisée.

- Les instructions SELECT qui incluent au moins une sous-requête sont parfois appelées *requêtes imbriquées* ou *instructions SELECT imbriquées*. Le fait d'imbriquer une instruction SELECT à l'intérieur d'une autre explique la présence du mot «structuré» dans l'expression «langage d'interrogation structuré» (SQL, *Structured Query Language*).
- Une sous-requête imbriquée dans une instruction SELECT externe présente la syntaxe suivante:
 (SELECT [ALL | DISTINCT] liste_de_sélection_de_la_sous-requête
 [FROM {nom_de_table | nom_de_vue}[conseils-optimiseur]
 [[, {nom_de_table2 | nom_de_vue2}[conseils-optimiseur]
 [..., {nom_de_table16 | nom_de_vue16}[conseils-optimiseur]]]
 [clause WHERE]
 [clause GROUP BY]
 [clause HAVING])
- L'instruction SELECT d'une sous-requête se place toujours entre parenthèses. Elle ne peut pas contenir de clause ORDER BY ou COMPUTE.
- Une sous-requête peut s'imbriquer dans une clause WHERE, ou HAVING d'une instruction externe SELECT, INSERT, UPDATE ou DELETE, ou dans une autre sous-requête. Le niveau d'imbriication n'est pas limité.
- A noter le cas particulier des requêtes imbriquées placées après le SELECT, qui permettent d'afficher une colonne calculée (Cf. Exercice 14).
- Il existe deux catégories de sous-requêtes : les sous-requêtes qui renvoient une et une seule valeur directement manipulable par des opérateurs de comparaison, et les requêtes ensemblistes qui renvoient des listes manipulables par les opérateurs ensemblistes IN, ANY , ALL, EXISTS

Méthode de construction des sous-requêtes renvoyant une et une seule valeur

Pour trouver tous les livres de même prix que le livre « *Toute la vérité sur les ordinateurs* », on peut procéder en deux étapes, en cherchant d'abord le prix du livre « *Toute la vérité sur les ordinateurs* » :

```
SELECT prix
FROM titres
WHERE titre = 'Toute la vérité sur les ordinateurs'
```

```
prix
-----
136,00
```

Puis, en utilisant ce résultat dans une seconde requête pour trouver les livres qui ont le même prix :

```
SELECT titre, prix
FROM titres
WHERE prix = 136
```

titre	prix
Toute la vérité sur les ordinateurs	136,00
Les festins de Parly 2	136,00
Les secrets de la Silicon Valley	136,00
Privation durable d'informations : étude...	136,00

En substituant à la constante 136, la requête qui calcule ce prix, on obtient la solution avec sous-requête :

```
SELECT titre, prix
FROM titres
WHERE prix = (SELECT prix FROM titres
              WHERE titre = 'Toute la vérité sur les ordinateurs')
```

Avec la même démarche de construction progressive, on peut répondre à des questions plus complexes : dans tous les cas, il faut repérer dans la question, la proposition principale qui fournit la requête principale et les propositions subordonnées qui fournissent les critères de choix ; chaque subordonnée devient une requête imbriquée qu'il faudra d'abord tester indépendamment, sur un cas particulier.

Exemple : *Afficher les titres des livres dont le prix est supérieur ou égal au tiers du prix maximum, et inférieur ou égal à la moyenne des prix.*

-- 1) trouver le prix maximum

```
SELECT max(prix)
FROM titres
--      => 156.00
```

-- 2) trouver la moyenne des prix

```
SELECT avg(prix)
FROM titres
--      => 99.4
```

-- 3) Ecrire la requête principale avec ces constantes

```
SELECT titre
FROM titres
WHERE (prix >= 156.0/3)
and (prix <= 99.4)
```

-- 4) Réécrire la requête principale en substituant les requêtes imbriquées aux 2 constantes

```
SELECT titre
FROM titres
WHERE prix >= (SELECT max(prix)
                FROM titres ) /3
and   prix <= (SELECT avg(prix)
                FROM titres )
```

Sous-requêtes ensemblistes

- Utilisation de IN et EXISTS pour tester l'appartenance d'un nuple à une liste construite par une requête imbriquée

Exemple : *Afficher les noms et les prénoms des auteurs qui ont écrit au moins un livre (donc qui figurent dans la table titreauteur)*

Avec l'opérateur IN :

```
SELECT nom_auteur, pn_auteur
FROM auteurs au
```

```
WHERE id_auteur IN (SELECT id_auteur FROM titreauteur)
Order by nom_auteur
```

Avec l'opérateur **EXISTS** :

```
SELECT nom_auteur, pn_auteur
FROM auteurs au
WHERE EXISTS (SELECT id_auteur FROM titreauteur
               WHERE id_auteur = au.id_auteur)
Order by nom_auteur
```



Remarquer la jointure supplémentaire entre la table auteurs de la requête principale et la table titreauteur de la requête imbriquée, par rapport à la solution avec l'opérateur IN

Dans ce cas particulier, le problème est réductible à une requête simple, sans sous-requête :

```
SELECT DISTINCT nom_auteur, pn_auteur
FROM auteurs au, titreauteur ti
WHERE au.id_auteur = ti.id_auteur
ORDER BY nom_auteur, pn_auteur
```

- Sous-requêtes opérant sur des listes, introduites par un opérateur de comparaison modifié par ANY ou ALL :

Exemple 1 : *Afficher les noms et les prénoms des auteurs dont tous les livres ont un prix de 136 F*

```
SELECT nom_auteur, pn_auteur
FROM auteurs au
WHERE 136 = ALL (SELECT prix
                  FROM titres t, titreauteur ta
                  WHERE t.id_titre = ta.id_titre
                  AND au.id_auteur = ta.id_auteur)
ORDER BY nom_auteur, pn_auteur
```



Lorsqu'une requête imbriquée ne renvoie rien, toutes les expressions avec l'opérateur ALL sont vraies : on peut tout dire de l'ensemble vide. La requête ci-dessus trouve donc les auteurs recherchés, dont tous les livres valent 136 F, mais aussi les auteurs qui n'ont pas encore publié de livres (et ne figurent donc pas dans la table titreauteur)

Pour s'assurer du bon fonctionnement de l'opérateur ALL, il faut toujours doubler la sous-requête avec ALL d'une sous-requête d'existence :

```
SELECT nom_auteur, pn_auteur
FROM auteurs au
WHERE id_auteur IN (SELECT id_auteur FROM titreauteur)
```

```

and      136 = ALL ( SELECT prix
                      FROM    titres t, titreauteur ta
                      WHERE    t.id_titre = ta.id_titre
                      AND      au.id_auteur = ta.id_auteur)
ORDER BY nom_auteur, pn_auteur

```

Exemple 2 : *Afficher les noms et les prénoms des auteurs dont au moins un livre a un prix de 136F*

```

SELECT nom_auteur, pn_auteur
FROM auteurs au
WHERE 136 = ANY (SELECT prix
                  FROM    titres t, titreauteur ta
                  WHERE    au.id_auteur = ta.id_auteur
                  AND      t.id_titre = ta.id_titre)
ORDER BY nom_auteur, pn_auteur

```

Si l'ensemble construit par la sous-requête est vide, toutes les comparaisons contruites sur l'opérateur ANY sont fausses : il est inutile de doubler les sous-requêtes avec ANY d'un test d'existence.



Exercice 29 : sous-requêtes

Afficher la liste des livres vendus dans la même ville que leur auteur

Exercice 30 : sous-requêtes, EXIST, ALL

Afficher les noms et prénoms par ordre alphabétique des auteurs qui possèdent 100% de droits sur tous leurs livres ! (titreauteur.droits_pourcent = 100 pour tous les livres)

Exercice 31 : 1 sous-requête, MAX

Afficher le titre du livre le plus cher (maximum de titre.prix)

Exercice 32 : 1 sous-requête utilisée dans la clause SELECT, SUM

Afficher la liste des titres dans l'ordre alphabétique et le cumul de leurs ventes, tous magasins confondus (tables titres et ventes)

Exercice 33 : 1 sous-requête, MAX (ou Group By, Max)

Afficher le maximum des ventes par magasin et titre.

Exercice 34 :

Afficher le nom et l'identificateur des éditeurs qui éditent de la gestion et pas d'informatique

L'opérateur ensembliste UNION



Doc. en ligne : « Concepts de requêtes avancées » : « Combinaison des résultats avec UNION »

Exemple : Noms, prénoms des auteurs habitant Genève ou Paris

```
SELECT nom_auteur, pn_auteur
FROM auteurs
WHERE ville = 'Genève'
UNION
SELECT nom_auteur, pn_auteur
FROM auteurs
WHERE ville = 'Paris'
```

Synthèse sur les principes d'écriture d'une requête SELECT

- ☞ Déterminer les tables utilisées et les indiquer dans la clause FROM,
- ☞ Lister les attributs à visualiser dans la clause SELECT,
- ☞ Etablir les clauses WHERE de jointure,
- ☞ Etablir les clauses WHERE d'énoncé,
- ☞ Si la clause SELECT comporte des fonctions de groupes, utiliser une clause GROUP BY reprenant tous les attributs cités dans le SELECT, sauf les fonctions de groupe,
- ☞ Fixer les critères de recherche par HAVING sur les groupes et par WHERE sur des lignes individuelles,
- ☞ Pour fusionner les résultats de deux clauses SELECT utiliser l'opérateur UNION,
- ☞ Préciser l'ordre de rangement des résultats par une clause ORDER BY
- ☞ Jointures ou SELECT imbriquées ?

- Une **jointure** est indispensable pour traduire une requête où il s'agit de visualiser des informations émanant de plusieurs tables. Une requête imbriquée rend le même service, mais est moins élégante.
- Dans certains cas une **requête imbriquée** s'impose : clause de non existence par exemple.

2.4 Mise à jour d'une base de données



Doc. en ligne : « Accéder aux données et les modifier » : Ajout, Modification, Suppression de données

SQL2 : p. 79 à 82

- Dans SQL Server, vous pouvez ajouter ou modifier des données au moyen des instructions de modification de données INSERT, DELETE, et UPDATE
- INSERT ajoute une nouvelle ligne à une table ;
- DELETE supprime une ou plusieurs lignes ;
- UPDATE modifie les lignes ;
- Vous pouvez modifier des données dans une seule table par instruction. Transact-SQL vous permet de baser vos modifications sur des données contenues dans d'autres tables, même celles d'autres bases de données.

Ajout de lignes : INSERT

- Le mot-clé VALUES est utilisé pour spécifier les valeurs de certaines ou de toutes les colonnes dans une nouvelle ligne :

```
INSERT nom_de_table
VALUES (constante1, constante2, ...)
```

- Ajout de toutes les colonnes d'une ligne :

```
INSERT titres
VALUES ('BU2222', 'Plus vite!', 'gestion', '1389',
      NULL, NULL, NULL, NULL, 'ok', '14/06/95')
```

- Ajout de certaines colonnes : les colonnes ignorées doivent être définies pour permettre des valeurs NULL. Si vous sautez une colonne avec la contrainte DEFAULT, sa valeur par défaut est utilisée.

```
INSERT INTO magasins (id_mag, nom_mag)
VALUES ('1229', 'Les livres de Marie')
```

- Vous pouvez également utiliser une instruction SELECT dans une instruction INSERT pour insérer des valeurs sélectionnées dans une ou plusieurs autres tables ou vues. Voici la syntaxe simplifiée :

```
INSERT nom_de_table
SELECT liste_de_colonne
FROM liste_de_table
WHERE critères_de_sélection
```

- Insertion de données provenant de la même table par une instruction SELECT :

```
INSERT éditeurs
SELECT '9980', 'test', ville, région, pays
FROM éditeurs
WHERE nom_éditeur = 'New Moon Books'
```

Nouveau contenu de la table :

<i>id_éditeur</i>	<i>nom_éditeur</i>	<i>ville</i>	<i>région</i>	<i>pays</i>
0736	New Moon Books	Boston	MA	USA
0877	Binnet & Hardley	Washington	DC	USA
1389	Algodata Infosystems	Bruxelles		BE
1622	Five Lakes Publishing	Chicago	IL	USA
1756	Ramona, éditeur	Lausanne	CH	
9901	GGG&G	Munich	(null)	GER
9952	Scootney Books	New York	NY	USA
9980	test	Boston	MA	USA
9999	Editions Lucerne	Paris	(null)	FR

- Insertion de données provenant d'une autre table, colonnes dans le même ordre :

```
INSERT auteurs
SELECT *
FROM nouveaux_auteurs
```

- Insertion de données provenant d'une autre table, colonnes dans un autre ordre : la table *auteurs* contient les colonnes *id_auteur*, *pn_auteur*, *nom_auteur* et *adresse*, la table *nouveaux_auteurs* contient les colonnes *id_auteur*, *adresse*, *pn_auteur* et *nom_auteur*.

```
INSERT auteurs (id_auteur, adresse, nom_auteur, pn_auteur)
SELECT * FROM nouveaux_auteurs
ou
INSERT auteurs
SELECT id_auteur, pn_auteur, nom_auteur, adresse
FROM nouveaux_auteurs
```



Exercice 35 :

Rentrer vos noms, prénoms, dans la table auteurs, avec un identificateur qui n'existe pas déjà. Tous les champs obligatoires (non NULL) doivent être renseignés, sauf ceux qui possèdent une valeur par défaut (téléphone). Réafficher la table. Relancer la même requête et interpréter le message d'erreur.

Exercice 36 :

*Créer un nouveau livre de type 'informatique' dans la table titres, en lui attribuant un *id_titre*, un titre et une date de publication (datepub).*

Exercice 37 :

Recopier toutes les caractéristiques d'un auteur en lui donnant un nouvel identificateur, et un nouveau nom.

Modification de lignes : UPDATE

- Arthur Bec décide par exemple de modifier son nom en Roland Perceval. Voici comment modifier sa ligne dans la table *auteurs* :

```
UPDATE auteurs
SET nom_auteur = 'Perceval', pn_auteur = 'Roland'
WHERE nom_auteur = 'Bec'
```



Exercice 38 :

Augmenter de 10% tous les prix des livres de l'éditeur « Algodata Infosystems ». Vérifier l'opération par une commande Select adéquate, avant et après l'augmentation.

Exercice 39 :

Modifier l'adresse de l'auteur qui a écrit 'Vivre sans crainte'.

Suppression de lignes : DELETE

- Supposons qu'une opération complexe débouche sur l'acquisition de tous les auteurs de Bruxelles et de leurs ouvrages par un autre éditeur. Vous devez immédiatement supprimer tous ces ouvrages de la table *titres*, mais vous ne connaissez pas leur titre ni leur numéro d'identification. La seule information dont vous disposez concerne le nom de la ville où résident ces auteurs.
- Vous pouvez effacer les lignes dans la table *titres* en retrouvant les numéros d'identification des auteurs pour les lignes qui ont *Bruxelles* comme ville dans la table *auteurs*, et en utilisant ensuite ces numéros pour trouver les numéros d'identification des ouvrages dans la table *titreauteur*. En d'autres termes, une triple jointure est requise pour trouver les lignes que vous souhaitez effacer dans la table *titres*.
- Les trois tables sont donc comprises dans la clause FROM de l'instruction DELETE. Toutefois, seules les lignes dans la table *titres* qui répondent aux conditions de la clause WHERE sont effacées. Vous devez effectuer des effacements séparés pour supprimer les lignes pertinentes dans les autres tables que la table *titres*.

```
DELETE titres
FROM auteurs, titres, titreauteur
WHERE titres.id_titre = titreauteur.id_titre
      AND auteurs.id_auteur = titreauteur.id_auteur
      AND ville = 'Bruxelles'
```

- **Remarque** La relation clé primaire/clé étrangère entre *titres.id_titre* et *titreauteur.id_titre* vous empêche d'exécuter cet effacement parce que la clé étrangère ne permettra pas que vous effaciez des titres encore référencés dans *titreauteur* : effacez en premier ces références



Exercice 40 :

Détruire les lignes créées dans la table *auteur*, dans les exercices 35 et 37, afin de remettre la table dans son état initial.

3. CREATION DE LA BASE «COMPAGNIE AERIENNE»



Doc. en ligne : « Créer et maintenir des bases de données »

3.1 Introduction

- Dans ce chapitre, nous allons construire complètement la base de données « Compagnie Aérienne » qui a été présentée au début.
- L'administrateur (compte **sa**) vous a accordé le droit de créer une nouvelle base de donnée, dont vous serez propriétaire (compte **dbo**, « data base owner ») : le **dbo** a tous les droits sur sa base, de même que l'administrateur a tous les droits sur la totalité du SGBDR.
- Tous les exercices ci après peuvent être exécutés :
 - En mode texte en tapant des ordres sql *Create*
 - Avec l'outil graphique.

Bien que l'outil graphique soit particulièrement performant, il est intéressant de savoir construire une base en mode texte, car tous les SGBD ne proposent pas d'outils graphiques, et aussi ils sont tous différents. Nous proposons de faire d'abord les exercices en mode texte, puis ensuite avec l'outil graphique.



Loguez vous avec Management studio et ouvrez une fenêtre sql (nouvelle requête)

```
CREATE DATABASE AirXXX
```

3.2 Création des domaines (types de données utilisateur)



Doc. en ligne : « Accéder aux données et les modifier » : « Eléments de syntaxe Transact SQL » -> « Utilisation des types de données »

- La première étape consiste à créer des « domaines », c'est-à-dire des types de données, qui s'appliqueront aux colonnes ou attributs des tables et des colonnes. Comme en C ou en Pascal, les domaines sont définis par le propriétaire de la base de données, à partir des types prédéfinis du langage SQL :

bit	entier dont la valeur est 1 ou 0.
tinyint	entier dont la valeur est comprise entre 0 et 255.
smallint	entier dont la valeur est comprise entre -2^{15} (-32 768) et $2^{15} - 1$ (32 767).
int	entier dont la valeur est comprise entre -2^{31} (-2 147 483 648) et $2^{31} - 1$ (2 147 483 647).
decimal	Données numériques fixes de précision et d'échelle comprises entre $-10^{38} - 1$ et $10^{38} - 1$.
numeric	Synonyme de decimal .

smallmoney	Valeurs de données monétaires comprises entre - 214 748,3648 et +214 748,3647, avec une précision d'un dix-millième d'unité monétaire.
money	Valeurs de données monétaires comprises entre -2^{63} (-922 337 203 685 477,580 8) et $2^{63} - 1$ (+922 337 203 685 477,580 7), avec une précision d'un dix-millième d'unité monétaire.
real	Données numériques de précision en virgule flottante comprises entre $-3.40E + 38$ et $3.40E + 38$.
float	Données numériques de précision en virgule flottante comprises entre $-1.79E + 308$ et $1.79E + 308$.
smalldatetime	Données de date et d'heure comprise entre le 1 ^{er} janvier 1900 et le 6 juin 2079, avec une précision d'une minute.
datetime	Données de date et d'heure comprises entre le 1 ^{er} janvier 1753 et le 31 décembre 9999, avec une précision de trois centièmes de seconde ou de 3,33 millisecondes.
char	chaîne de caractères de longueur fixe d'un maximum de 8 000 caractères.
varchar	chaîne de caractères de longueur variable d'un maximum de 8 000 caractères.
text	texte de longueur variable ne pouvant pas dépasser $2^{31} - 1$ (2 147 483 647) caractères.
binary	données binaires de longueur fixe ne pouvant pas dépasser 8 000 octets.
varbinary	Données binaires de longueur variable ne pouvant pas dépasser 8 000 octets.
image	Données binaires de longueur variable ne pouvant pas dépasser $2^{31} - 1$ (2 147 483 647) octets.

Fonction CREATE TYPE

L'instruction Transact-SQL CREATE TYPE (remplace la procédure stockée système **sp_addtype** de SQL Server 2000) permet de créer ces types de données :

Syntaxe sous Transact-SQL :

```
CREATE TYPE [nom_schema.] nom_de_type
{
    FROM type_système [(precision[, échelle])]
    [NULL | NOT NULL ]
}
```

nom_de_type : Nom du type de données défini par l'utilisateur.

type_système : Nom du type de données physique prédéfini de SQL Server sur lequel repose le type de données défini par l'utilisateur.

contrainte_de_valeur_NULL : Paramètre indiquant comment gérer des valeurs **NULL** dans un type de données défini par l'utilisateur (vaut **NULL** ou **NOT NULL**).

- SQL Server ne fait pas de différence entre les majuscules et les minuscules pour les noms de type de données système.



Exercice 41

*Les exercices qui suivent vont vous guider pour écrire un script de création de base de données : **Creer.SQL**.*

*On commentera chacune des étapes du script (commentaire Transact SQL : **/*** plusieurs lignes ***/** ou **--** fin de ligne)*

*On développera en parallèle un script **Supprimer.SQL** qui supprime tous les objets que l'on vient de traiter. Après chaque exercice, on relancera les scripts de suppression et de création, pour s'assurer que tous les objets se créent correctement.*

A la fin de ce chapitre, on doit donc aboutir à deux scripts complets qui permettent de maintenir la base (en la recréant complètement en cas de crash système...)

Etape 1 : création des types utilisateurs :

TypeDate : type prédéfini **datetime**. Obligatoire. (=> date du vol dans la table *Affectation*).

TypeHoraire : type prédéfini **smalldatetime**. Obligatoire (=> heures de départ et d'arrivée, dans la table *Vol*)

TypeNom : chaîne variable limitée à 50 caractères. Obligatoire. (=> nom du pilote dans la table *Pilote* et du constructeur dans la table *Constructeur*)

TypePrenom : chaîne variable limitée à 30 caractères. Optionnel. (=> prénom du pilote dans la table *Pilote*)

TypeAvion : chaîne variable limitée à 20 caractères. Obligatoire. (=> type d'avion dans les tables *Type* et *Avion*)

TypeIdConstructeur : entier (**smallint**). Obligatoire (=> identificateur du constructeur dans la table *Constructeur*)

TypeIdAéroport : chaîne fixe de 3 caractères. Obligatoire. (=> identificateur de l'aéroport dans la table *Aéroport*)

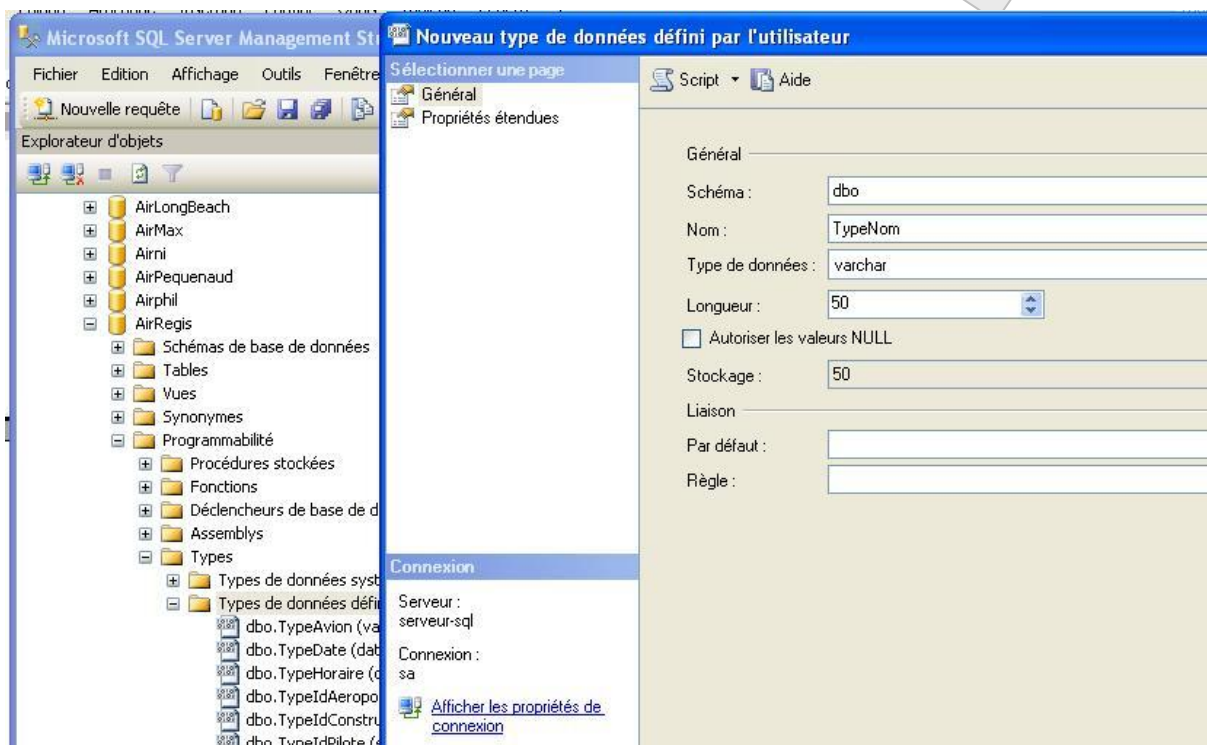
TypeIdPilote : entier (**smallint**). Obligatoire (=> identificateur du pilote dans la table *Pilote*)

TypeNumVol : chaîne fixe de 5 caractères. Obligatoire (=> identificateur du vol dans la table *Vol*)

TypeVille : chaîne variable limitée à 50 caractères. Optionnel. (=> nom de l'aéroport et de la ville desservie dans la table *Aéroport*)

☺ Avec sql server management studio express (voir page 10)

- lancer l'outil par le menu « SQL Server 2005 → Management studio express »
- A l'invite loguer vous avec votre identifiant sql serveur (user :votre nom / password :votre nom).
- Développer « Bases de données » : seules les bases de données auxquelles vous avez accès apparaissent. Développer la base en cours de construction, et le répertoire « Types de données utilisateurs ».



3.3 Création et suppression de tables. Contraintes d'intégrité



Doc. en ligne : « Créer et maintenir des bases de données » : « Tables »

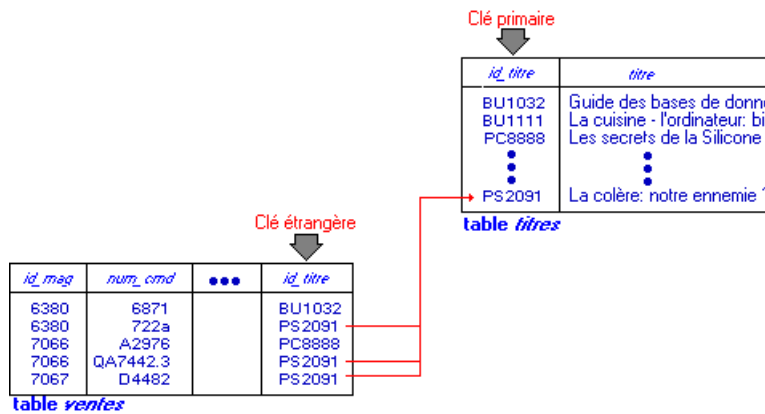
Clés primaires et étrangères

- La *clé primaire* est une colonne ou une combinaison de colonnes dont les valeurs identifient de façon unique chaque ligne dans la table : elle ne peut pas être nulle.
- On crée une clé primaire grâce à la contrainte PRIMARY KEY lors de la création ou de la modification d'une table. SQL crée automatiquement un objet « index » pour la clé primaire : l'index assure l'unicité de la clé primaire dans la table, et permet un accès rapide aux données par la clé primaire.
- Une *clé étrangère* est une colonne ou une combinaison de colonnes dont les valeurs correspondent à la clé primaire d'une autre table. Elle n'est pas obligatoirement unique ni définie (peut être NULL), mais ses valeurs doivent correspondre à des valeurs existantes de la clé primaire. On crée une clé étrangère par la contrainte FOREIGN KEY lors de la création ou de la modification d'une table.

Intégrité des données dans une base

- Assurer l'intégrité des données, c'est préserver la cohérence et l'exactitude des données stockées dans une base de données en validant le contenu des différents champs, en vérifiant la valeur des champs l'un par rapport à l'autre, en validant les données dans une table par rapport à une autre, et en vérifiant que la mise à jour d'une base de données est efficacement et correctement effectuée pour chaque transaction.
- 4 contraintes d'intégrité :
 - ⇒ Intégrité d'entité : unicité de la clé primaire (PRIMARY KEY) ou d'autres clés (UNIQUE)
 - ⇒ Intégrité de domaine : plage des valeurs possibles pour une colonne. On peut restreindre cette plage en attribuant à la colonne un type défini par l'utilisateur, ou par une contrainte CHECK avec une règle.
 - ⇒ L'intégrité référentielle garantit la relation entre une clé primaire unique dans une table et les clés étrangères qui y font référence dans les autres tables. Exemple : avant de supprimer une ligne dans la table PILOTE, il faut supprimer toutes les lignes de la table VOL qui font référence à ce pilote.
- Pour assurer l'intégrité référentielle, SQL Server interdit de :
 - ajouter des enregistrements à une table liée lorsqu'il n'y a aucun enregistrement associé dans la table primaire ;
 - changer des valeurs ou effacer des enregistrements dans une table primaire qui engendrerait des enregistrements « orphelins » dans une table liée ;

Par exemple, avec les tables *ventes* et *titres* dans la base de données *pubs*, l'intégrité référentielle est basée sur la relation entre la clé étrangère (*id_titre*) de la table *ventes* et la clé primaire (*id_titre*) de la table *titres*, comme le montre l'illustration suivante :



Exemples de création de table : CreateTable

- Lorsque vous créez une table, nommez ses colonnes, définissez un type de donnée pour chaque colonne, précisez si elles peuvent contenir des valeurs NULL, définissez des contraintes par l'option CHECK, des valeurs par défaut par DEFAULT

CREATE TABLE emplois

(

id_emploi smallint
desc_emploi varchar(50)
niv_min tinyint
niv_max tinyint

)

PRIMARY KEY,
NOT NULL DEFAULT 'Nouveau poste',
NOT NULL CHECK (niv_min >= 10),
NOT NULL CHECK (niv_max <= 250)

Clé primaire

Valeur par défaut

Contrainte

- FOREIGN KEY** : la clé fait référence à la clé primaire d'une autre table.

id_emploi smallint NOT NULL DEFAULT 1 REFERENCES emplois(id_emploi)

ou id_emploi smallint et plus loin FOREIGN KEY (id_emploi) REFERENCES emplois (id_emploi)

- UNIQUE** : assure l'unicité d'une clé secondaire

pseudonyme varchar(30) NULL UNIQUE

- DEFAULT** : valeur par défaut lorsqu'aucune valeur n'est donnée dans INSERT ou UPDATE

- CHECK** :

CHECK (id_éditeur IN ('1389', '0736', '0877', '1622', '1756'))
OR id_éditeur LIKE '99[0-9][0-9]')

CHECK (niv_min >= 10)

Nom contrainte

ou avec mot clé CONSTRAINT :

CONSTRAINT CK_id_employé CHECK (id_employé LIKE '[0-9][0-9][0-9][0-9][FM]')

- **IDENTITY** : valeur numérique autoincrémentale (valeur initiale et valeur de l'incrément)

IDENTITY(10,3) (=> valeurs: 10, 13, 16...)



Exercice 42 : Création des tables et des contraintes

Compléter le script de création pour créer les tables de la base « Compagnie aérienne », décrite dans le schéma général. Pour chaque table, on définira les liens clé étrangère-clé primaire en suivant le schéma.

Définitions de contraintes supplémentaires sur les tables :

AVION : la clé primaire NumAvion est autoincrémentée à partir de 100, par pas de 1.

TYPE : la clé primaire TypeAvion commence obligatoirement par une lettre
la capacité est comprise entre 50 et 400 : cette colonne est obligatoire, avec un défaut de 100

CONSTRUCTEUR : la clé primaire IdConstructeur est autoincrémentée à partir de 1, par pas de 1

PILOTE : la clé primaire IdPilote est autoincrémentée à partir de 1, par pas de 1.
La combinaison (nom, prénom) est unique.

AEROPORT : la clé primaire IdAeroport ne comporte que des lettres
la colonne NomVilleDesservie est optionnelle
la colonne NomAeroport est obligatoire

VOL : la clé primaire NumVol est constituée du préfixe " IT" suivi de trois chiffres

AFFECTATION : la clé primaire est composée des colonnes NumVol et DateVol
la clé étrangère IdPilote peut être NULL (en attente d'affectation à un pilote)

Commencer par établir la liste de dépendance :

- numéroté 0 les tables indépendantes, 1 les tables qui ne dépendent que de tables indépendantes, 2 celles qui dépendent des précédentes...
- créer les tables en commençant par les indépendantes, et en suivant l'ordre de la liste de dépendance.

Vérifier la création des tables sous SQL Server Management Studio

Exercice 43 : Script de test

Tester les contraintes d'intégrité en faisant des essais de remplissage par l'instruction INSERT et des suppressions par DELETE : vérifier les valeurs par défaut, les domaines de validité des colonnes...

Partir des valeurs décrites dans le chapitre I, en remplissant les tables dans l'ordre de la liste de dépendance.

Exercice 44 : Script de destruction des types utilisateurs et des tables

Pour détruire une table :

```
DROP TABLE AVION
GO
```


Le GO oblige le moteur à faire la suppression immédiatement et permet de recréer la table (sinon erreur « Il y a déjà un objet Avion dans la base de données »)

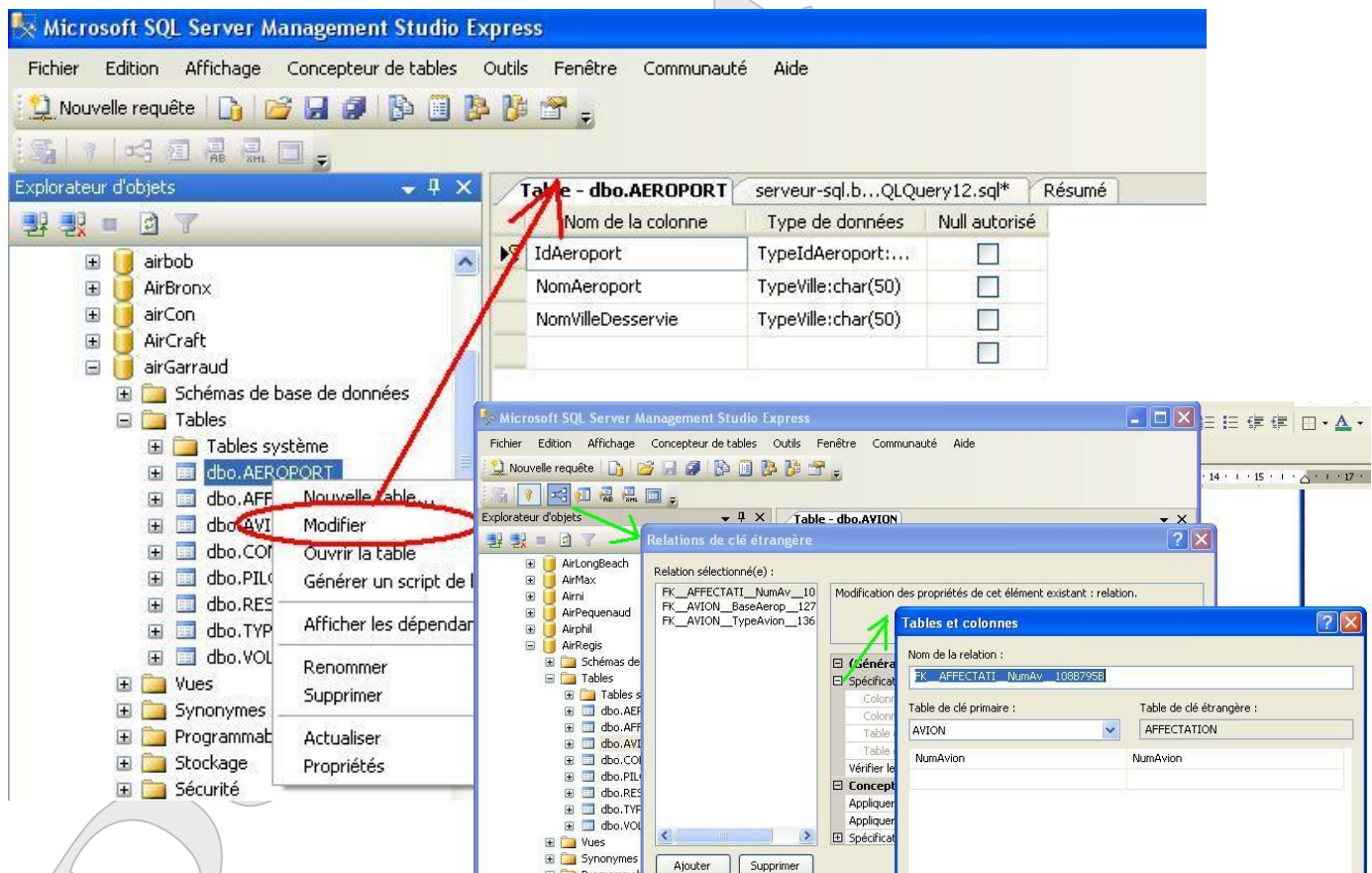
Attention : il faut détruire les tables dans l'ordre inverse de la création, en supprimant d'abord les tables dépendantes

**Détruire les types à la fin, quand ils ne sont plus utilisés par aucune table :
DROP TYPE TypeNom**

☺ Avec sql server management studio express (voir page 7 à 11)

Dans le répertoire « Bases de données », « Tables », visualisez et vérifiez les tables et leurs contraintes.

- ⇒ Pour visualiser les propriétés d'une table, avec le bouton droit de la souris, choisir « Modifier une table »
- ⇒ Pour obtenir les détails sur les contraintes, et les clés étrangères : cliquer l'outil relation.



3.4 Création de vues



Doc. en ligne : « Créer et maintenir des bases de données » : « Vues »
SQL2 : p. 133 à 149

Quand utiliser des vues ?

- Une *vue* est une table virtuelle dont le contenu est défini par une requête **SELECT**. Comme une table réelle, la vue possède des colonnes nommées et des lignes de données, mais elle n'est pas stockée dans la base de données, et n'a pas d'existence propre, indépendamment des tables.
- Les vues améliorent la sécurité en permettant de contrôler les données que les utilisateurs peuvent visualiser, et l'ergonomie, en présentant les données sous une forme familière, indépendamment de la structure interne de la base.
- Une vue peut être manipulée exactement comme une table : lorsqu'on modifie une vue, on modifie en fait les tables sur lesquelles elle s'appuie. Inversement, si on modifie les tables, les modifications sont reportées automatiquement dans toutes les vues qui leur font référence. La suppression d'une vue n'influe pas sur le contenu de la table associée. Par contre la suppression d'une table supprime toutes les vues qui lui font référence.

<i>id_titre</i>	<i>titre</i>	<i>type</i>	<i>id_éditeur</i>	<i>prix</i>	<i>avance</i>	<i>droits</i>	<i>cumulannuel_ventes</i>
BU1023	Guide des bases de données du ges	gestion	1389	140,00	35.000,00	10	4095
BU1111	La cuisine - l'orinateur: bilans clande	gestion	1389	82,00	34.000,00	10	3876
BU2075	Le stress en informatique n'est pas	gestion	0736	24,00	69.000,00	24	18722
BU7832	Toute la vérité sur les ordinateurs	gestion	1389	136,00	34.000,00	10	4095
MC2222	Les festins de Parly 2	cui_moderne	0877	136,00	0,00	12	2032
MC3021	Les micro-ondes par gourmandise	cui_moderne	0877	21,00	102.000,00	24	22246

table titres

<i>titre</i>	<i>prix</i>	<i>nom_éditeur</i>
Guide des bases de données du gestion	140,00	Algodata Infosystems
La cuisine - l'ordinateur: bilans clandest	82,00	Algodata Infosystems
Le stress en informatique n'est pas un	24,00	New Moon Books

Vue

<i>id_éditeur</i>	<i>nom_éditeur</i>	<i>ville</i>	<i>région</i>
0736	New Moon Books	Boston	DC
0877	Binnet & Hardley	Washington	
1389	Algodata Infosystems	Bruxelles	
1622	Five Lakes Publishing	Chicago	IL
1756	Ramona, éditeur	Lausanne	

table éditeurs

Fonction CREATE VIEW

```
CREATE VIEW vuetitres
AS
SELECT titre, type, prix, datepub
FROM titres
Ou avec les colonnes explicites
CREATE VIEW vueEtiquette (TitreOuvrage, PrixOuvrage)
AS
SELECT titre, prix
FROM titres
```

Limitations :

- Vous pouvez créer des vues uniquement dans la base de données courante.
 - Vous pouvez construire des vues à partir d'autres vues ou de procédures qui font référence à des vues.
 - Si vous remplissez une vue par une instruction SELECT, vous n'avez en général pas besoin de donner les noms des colonnes, qui seront ceux du SELECT, sauf dans les cas suivants :
 - colonnes dérivées d'une expression arithmétique, d'une fonction ou d'une constante.
 - plusieurs colonnes issues de tables différentes auraient le même nom.
 - on veut donner un nom à la colonne de la vue, différent du nom de la colonne dans la table (présentation pour l'utilisateur de la base de données)
 - Vous ne pouvez pas associer des règles, des valeurs par défaut ou des Déclencheurs à des vues, ni construire des Index à partir de vues.
 - Vous ne pouvez pas créer de vues temporaires et vous ne pouvez pas créer de vues à partir de tables temporaires.
 - Vous ne pouvez pas modifier par UPDATE, INSERT ou DELETE les vues qui dérivent de plusieurs tables par jointure, ou qui contiennent les clauses GROUP BY, ou DISTINCT, ou qui omettent un attribut obligatoire de la table (déclaré NOT NULL) : ces vues ne seront accessibles qu'en lecture.
 - Pour une table modifiable, la clause WITH CHECK OPTION contrôle les modifications sur la vue.
-



Exercice 45 création d'une vue « Depart »

On veut disposer d'une vue qui visualise les départs du jour, à partir de Paris, avec la présentation ci-dessous :

NumVol	De	A	Heure de Départ	Heure d'arrivée
IT101	ORL	BLA	11H15	12H40
IT102	CDG	NIC	12H31	14H00
IT110	ORL	NIC	15H24	16H00

Remarques :

- La table AEROPORT donne la liste des aéroports desservant chaque ville : ORLY (ORL) et Charles de Gaulle (CDG) pour la ville de Paris. La table AFFECTATION donne les dates de tous les vols.
- Utiliser la fonction **DATENAME()** de Transact SQL pour extraire l'heure et les minutes des colonnes VOL.Hdépart et VOL.HArrivée, qui sont stockées en smalldatetime : **DATENAME (hh, Hdépart)** renvoie l'heure du départ, **DATENAME (mi, Hdépart)** renvoie les minutes
- On obtient la date du jour par la fonction **GETDATE()**. Cette date ne peut pas être comparée directement à AFFECTATION.DateVol, car elle comprend les heures, minutes, secondes. Le plus simple est de comparer le quantième du jour de la date courante, au quantième de la date du vol : la fonction **DATENAME (dayofyear, uneDate)** extrait le quantième du jour de la date « uneDate »

Tester la vue par un select simple. Rajouter la création de la vue dans le script de création de la base, et sa suppression dans le script de suppression.

3.5 Définition des permissions sur les objets d'une base de donnée

Contrôle d'accès aux données dans un SGBD

- L'accès aux données est contrôlé par le SGBD au moyen de l'identification de l'utilisateur qui lors de la connexion doit fournir le nom d'utilisateur et le mot de passe associé. Chaque objet d'une base de données mémorise la liste des utilisateurs autorisés à le manipuler : son créateur et les autres utilisateurs qui auront été habilités par lui.
- L'administrateur système de SQL SERVER (compte sa) est le seul à posséder tous les droits sur toutes les bases de données : création d'une nouvelle base, suppression... Ces droits ne peuvent pas lui être retirés (indépendants des droits explicites sur la base)
- Le propriétaire d'une base de données (DBO : Data Base Owner) a tous les droits sur les objets de sa base, et peut les donner à d'autres utilisateurs par la commande GRANT ou leur retirer des droits par la commande REVOKE

Attribution de droits : GRANT



SQL2 : p. 171-173

- La requête GRANT permet de donner à un utilisateur ou un groupe d'utilisateurs la permission d'utiliser certaines instructions de création ou de suppression d'objet (CREATE TABLE, CREATE VIEW, DROP TABLE, DROP VIEW...) ou des droits d'accès sur des objets donnés par SELECT, INSERT, UPDATE, DELETE, ALTER, ou ALL .
- Pour autoriser l'accès à un objet pour tous les utilisateurs on utilisera le groupe PUBLIC
- Permissions d'instruction :

```
GRANT { ALL | liste_d'instructions } TO { PUBLIC | liste_de_noms }
```

```
GRANT CREATE DATABASE, CREATE TABLE
TO Marie, Jean
```

- Permissions d'objet :

```
GRANT { ALL | liste_de_permissions } ON { nom_de_table [(liste_de_colonnes)] | nom_de_vue
[(liste_de_colonnes)] | nom_de_procedure_stockée | nom_de_procedure_stockée_étendue } TO { PUBLIC |
liste_de_noms }
```

- Il faut d'abord accorder des permissions générales au groupe public, puis préciser en accordant ou en retirant des permissions spécifiques à certains utilisateurs :

```
GRANT SELECT
ON auteurs
TO public
go
```

```
-- Marie , Jean et Pierre auront tous les privilèges sur la table auteurs
GRANT INSERT, UPDATE, DELETE
ON auteurs
TO Marie, Jean, Pierre
```

Suppression de droits : REVOKE



Doc. en ligne : chercher « REVOKE (T-SQL) »

- Permissions d'instruction :

REVOKE {ALL | liste_instructions} FROM {PUBLIC | liste_de_noms}

*REVOKE CREATE TABLE, CREATE DEFAULT
FROM Marie, Jean*

- Permissions d'objet :

REVOKE {ALL | liste_de_permissions} ON {nom_de_table [(liste_de_colonnes)] | nom_de_vue [(liste_de_colonnes)] | nom_de_procedure_stockée | nom_de_procedure_stockée_étendue} FROM {PUBLIC | liste_de_noms}



Exercice 46 : définition des permissions sur les tables et les vues

Dans chaque groupe de quatre, un des comptes représentera l'utilisateur de base qui consulte les horaires, les deux autres seront des employés de l'aéroport dotés de permissions limitées, qui assistent le propriétaire de la base : un responsable des horaires et un responsable de la planification

Sous le compte du propriétaire de la base (DBO) qui a tous les droits sur la base :

- **Créer les trois nouveaux utilisateurs dans votre base (usager, Responsable_horaires, Responsable_planification), correspondant aux trois comptes d'accès de votre groupe, avec la syntaxe :**

exec sp_adduser nom_de_connexion, nom_utilisateur_base

- **Pour l'utilisateur, permettre la lecture de la vue Depart, pour consultation dans l'aéroport. Aucun droit sur les tables !**
- **Le responsable des horaires a tous les droits sur Vol, et peut consulter Avion, Aeroport et Type**
- **Le responsable planification a le droit de lire, modifier, ajouter et supprimer sur la table Affectation, et de lire les tables Avion, Vol, Pilote, et Type**
- **Les deux assistants peuvent créer des vues .**

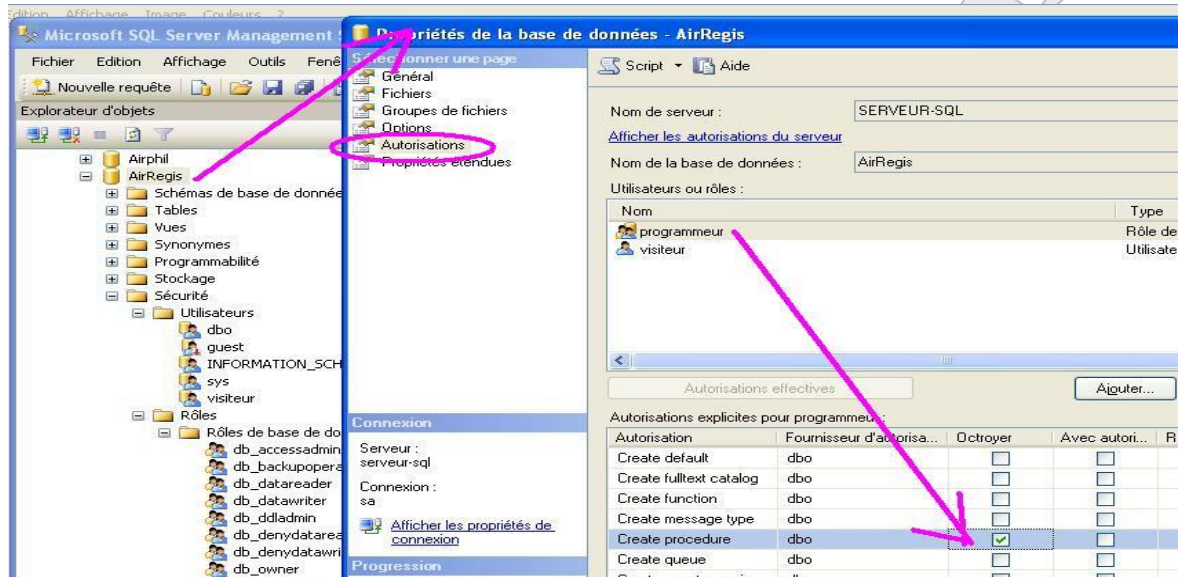
En se connectant sous les trois comptes, faire des requêtes manuelles, pour vérifier que les permissions fonctionnent comme prévu.

Rajouter la création des nouveaux utilisateurs et la définition des permissions à la fin du script de création de la base, et la suppression des utilisateurs dans le script de suppression.

☺ Avec sql server management studio express (voir page 15 à 23)

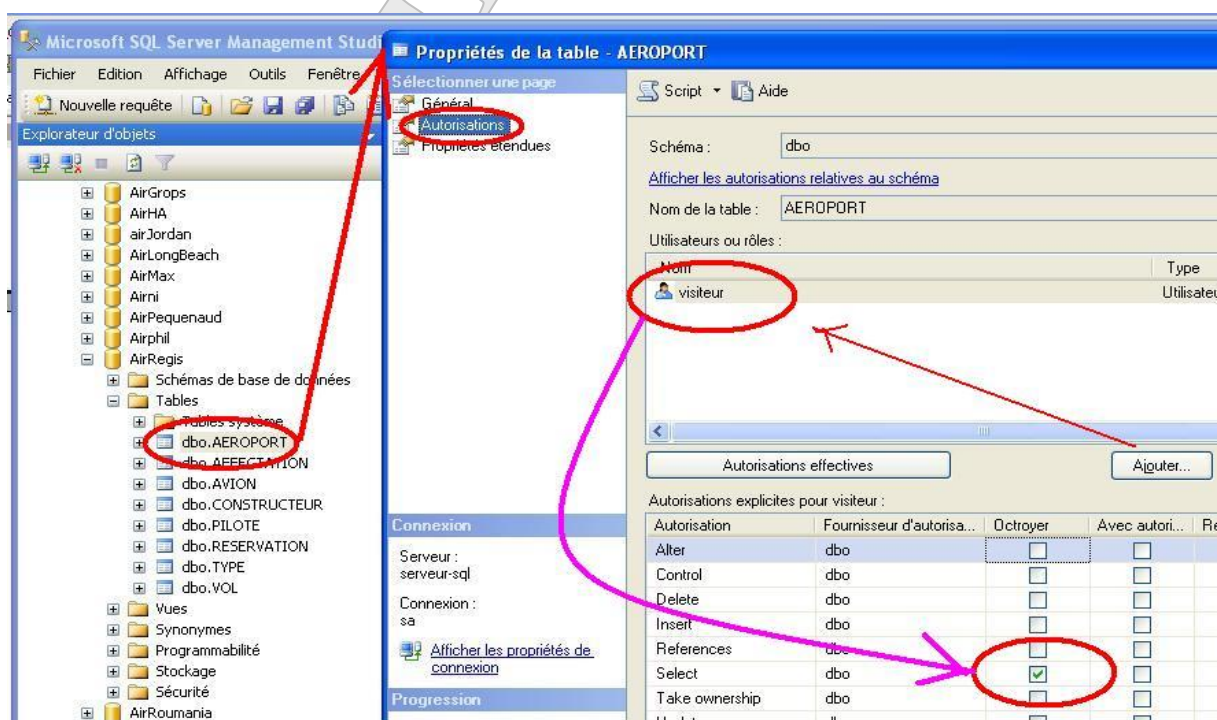
Permissions sur la base :

- En vous connectant avec le compte propriétaire de la base (dbo), vous pouvez visualiser et modifier les permissions d'instruction (CREATE TABLE, CREATE VIEW...) des différents utilisateurs de votre base : sélectionnez le nom de votre base, et choisissez « Propriétés » avec le bouton droit de la souris.



Permission sur les objets

- Vous pouvez aussi visualiser et modifier les **permissions d'objet** sur les tables (SELECT, INSERT, UPDATE, DELETE) :
 - choisissez « Propriétés » avec le bouton droit de la souris.
 - puis cliquez sur le bouton « Autorisations »



3.6 Les fonctions

Création des fonctions

Il existe deux types principaux de fonctions :

- Les fonctions scalaires renvoyant une seule valeur
- Les fonctions table, renvoyant un ensemble de lignes et de colonnes que l'on manipule comme une table

Par le biais d'une fonction, aucune modification de table de la base de données n'est possible. Seuls les objets locaux à la fonction peuvent être modifiés.

Les fonctions sont créées de la même manière qu'une requête SQL, et se retrouvent dans l'arborescence de **Studio Management** dans le dossier de la base de données, sous-dossier **Programmabilité / Fonctions**.

1. Les fonctions scalaires

Une fonction scalaire est identique à une fonction mathématique : elle peut avoir plusieurs paramètres d'entrée (jusqu'à 1024), et renvoie une seule valeur. Elle sera utilisée dans la clause SELECT ou la clause WHERE d'une requête.

Exemple: la fonction **fnDateFormat** formate à l'aide d'un séparateur entré en paramètre, une date et la retourne sous forme de chaîne de caractères.

```
CREATE FUNCTION fnDateFormat (@pdate datetime, @psep char (1))
RETURNS char (10)
AS
BEGIN
    RETURN
        CONVERT (varchar(10), datepart (dd, @pdate))
        + @psep + CONVERT (varchar (10), datepart (mm, @pdate))
        + @psep + CONVERT (varchar (10), datepart (yy, @pdate))
END
```

- Chaque paramètre est défini par son nom obligatoirement préfixé du caractère @ et son type, et séparé du suivant par une virgule
- La clause **RETURNS** spécifie le type de données à retourner : les données de type timestamp, table, text, ntext et image, et définies par l'utilisateur ne peuvent être renvoyées.
- La valeur renvoyée par la fonction est toujours définie par le mot-clé **RETURN**
- La fonction est définie dans un bloc **BEGIN ... END**

La fonction sera forcément appelée par son nom, préfixé éventuellement du nom de son propriétaire:

```
SELECT dbo.fnDateFormat(GETDATE(), '/')
```

Une fonction peut contenir un appel à une autre fonction, y compris elle-même, à une procédure stockée, et peut contenir toutes les instructions TRANSACT SQL.

2. Les fonctions table

- contient une seule instruction **SELECT** déterminant le format de la table renvoyée.
- Accepte des paramètres en entrée, et s'utilise comme une table dans une clause **FROM** : elles remplacent totalement les vues en y ajoutant la possibilité de passer des paramètres.

Exemple : la fonction table en ligne, **fnNomArticles** renvoie tous les articles d'une certaine couleur à partir d'une table **ARTICLES** de structure

ARTICLES (art_num, art_nom, art_coul, art_pa, art_pv, art_qte, art_frs)

```
CREATE FUNCTION fn_NomArticles (@pcoul char(10)) RETURNS table
AS
RETURN (SELECT art_nom,art_frs FROM dbo.articles WHERE art_coul =@pcoul)
```

- La clause **RETURNS** spécifie table comme type de données à retourner
- La valeur renvoyée par la fonction toujours définie par le mot-clé **RETURN** est une instruction **SELECT**
- On n'utilise pas de bloc **BEGIN ... END**

La fonction sera appelée par son nom :

```
SELECT * from fn_NomArticles('ROUGE')
```

Modification-suppression des fonctions

La commande de modification **ALTER FUNCTION** accepte les mêmes paramètres que **CREATE FUNCTION** :

```
ALTER FUNCTION fn_DateFormat (@pdate datetime, @psep char(1))
RETURNS char(10)
AS
BEGIN
    RETURN CONVERT (varchar(10), datepart(dd,@pdate))
    + @psep + CONVERT (varchar(10), datepart(mm, @pdate))
    + @psep + CONVERT (varchar(10), datepart(yy, @pdate))
END
```

La commande de suppression **DROP FUNCTION** permet de supprimer une fonction :

```
DROP FUNCTION dbo.fn_DateFormat
```



Doc. en ligne : « Créer et maintenir des bases de données » : « Fonctions »
Aide *Transact-SQL* : CREATE FUNCTION



Exercice 47 création d'une fonction scalaire

Faire un select qui affiche la somme des heures de vol d'un pilote donné, pour un numéro de semaine donné, avec la présentation suivante :

Semaine 49, Heures de Vols de LECU Régis = 3

On créera une fonction *fn_HeuresDeVol*, qui aura en paramètres le nom et le prénom du pilote et le numéro de semaine. Elle utilisera la fonction *DATEDIFF* pour calculer la différence entre l'heure de départ et d'arrivée, et *DATENAME* avec l'option *week* pour extraire le numéro de semaine de la date du vol.



Exercice 48 création d'une fonction « Depart »

Reprendre l'exercice n°24, qui permettait la création d'une vue Départ . On va maintenant écrire une fonction qui va nous ramener les départs d'un jour passé en paramètre, à partir d'une ville elle aussi passée en paramètre.

<i>NumVol</i>	<i>De</i>	<i>A</i>	<i>Heure de Départ</i>	<i>Heure d'arrivée</i>
<i>IT101</i>	<i>ORL</i>	<i>BLA</i>	<i>11H15</i>	<i>12H40</i>
<i>IT102</i>	<i>CDG</i>	<i>NIC</i>	<i>12H31</i>	<i>14H00</i>
<i>IT110</i>	<i>ORL</i>	<i>NIC</i>	<i>15H24</i>	<i>16H00</i>

3.7 Les procédures stockées

Utilité des procédures stockées

- Une procédure stockée est un ensemble d'instructions compilées, qui peut s'exécuter plus rapidement. Les procédures stockées augmentent la puissance et la performance du langage SQL. Elles peuvent :
 - recevoir et renvoyer des paramètres à une procédure appelante,
 - appeler d'autres procédures,
 - renvoyer une valeur de sortie à la procédure appelante

Elles supportent les boucles et les if traditionnels de l'algo. Elles allient donc la puissance de SQL et de la programmation. Les noms de variables commencent par '@' afin de les distinguer des noms de table et de champs. Les variables sont typées.

- Les procédures stockées sur d'autres serveurs SQL auxquels le processus client n'est pas directement connecté peuvent être exécutées si le serveur distant autorise les accès distants.
- Les procédures stockées diffèrent des autres instructions et lots d'instructions SQL parce qu'elles sont préanalysées et prénormalisées. Lorsque vous exécutez une procédure pour la première fois, le programme de traitement des requêtes de SQL Server l'analyse et prépare une structure interne

normalisée pour la procédure qui est stockée dans une table système. Lors de la première exécution de la procédure au démarrage du serveur SQL, elle est chargée en mémoire et compilée complètement (elle n'est plus analysée ou mise en ordre puisque cela a été fait lors de sa création). Le plan compilé reste alors en mémoire (sauf si le besoin de mémoire l'oblige à en sortir) pour que sa prochaine exécution (effectuée par le même client ou par un autre) puisse être traitée sans que le plan de compilation consomme des ressources système.

- Les procédures stockées peuvent servir de mécanismes de sécurité, car un utilisateur peut avoir la permission d'exécuter une procédure stockée même s'il n'a aucune permission sur les tables ou les vues auxquelles elle fait référence. Il est de bonne pratique de faire toutes les mises à jour via des procédures stockées.
- Un certain nombre de procédures existent, qui permettent de manipuler la base de données : ce sont les **procédures stockées système** qui sont stockées dans la base de données master et identifiées par le préfixe sp_
- Pour obtenir des informations supplémentaires sur tous les types de procédures stockées, vous pouvez utiliser les procédures stockées système suivantes :
 - **Sp_help** nom_proc : Affiche la liste de paramètres et leur type
 - **Sp_helptext** nom_proc : Affiche le texte de la procédure stockée si non cryptée
 - **Sp_depends** nom_proc : Enumère les objets qui dépendent de la procédure et les objets dont dépend cette procédure
 - **Sp_stored_procedures** : Renvoie la liste des procédures stockées de la base de données

Création de procédures stockées : **CREATE PROCEDURE**.



Doc. en ligne : « Créer et maintenir des bases de données » : « Procédures Stockées »
Aide Transact-SQL : CREATE PROCEDURE et EXECUTE ou SQL2 : p.114

- Les procédures stockées sont des objets de base de données : la permission d'exécuter CREATE PROCEDURE revient par défaut au propriétaire de base de données qui peut la transmettre à d'autres utilisateurs.
- Vous ne pouvez créer une procédure stockée que dans la base de données courante.
- Une procédure stockée peut contenir toutes les instructions SQL, sauf CREATE. Elle peut appeler une autre procédure stockée jusqu'à 16 niveaux d'imbrications.
- **Les principaux éléments de syntaxe sont annexés en annexe 4 de ce document.**
- **Pour créer une procédure stockée**
 - Dans SQL Server management Studio, dans votre base de données, choisissez l'option « Programmabilité », puis « Procédures stockées », et ensuite « Nouvelle Procédure Stockée » avec le bouton droit de la souris.

Ou

- Utilisez l'instruction SQL **CREATE PROC** :
- la procédure stockée info_auteur reçoit en paramètres le nom et le prénom d'un auteur, et affiche le titre et l'éditeur de chacun des livres de cet auteur :

```
CREATE PROC info_auteur @nom varchar(40), @prénom varchar(20)
AS
```

@ avant le nom
du paramètre

```

SELECT nom_auteur, pn_auteur, titre, nom_éditeur
FROM auteurs, titres, éditeurs, titreauteur
WHERE pn_auteur = @prénom
      AND nom_auteur = @nom
      AND auteurs.id_auteur = titreauteur.id_auteur
      AND titres.id_titre = titreauteur.id_titre
      AND titres.id_éditeur = éditeurs.id_éditeur
go

```

- Pour exécuter la procédure :

Execute info_auteur 'Chevalier', 'Bernard'

nom_auteur	pn_auteur	titre	nom_éditeur
Chevalier	Bernard	La colère : notre ennemie ?	New Moon Books
Chevalier	Bernard	Vivre sans crainte	New Moon Books

- la procédure info_éditeur2 affiche le nom des auteurs qui ont écrit un livre publié par l'éditeur passé en paramètre : si aucun nom d'éditeur n'est précisé, la procédure fournit les auteurs publiés par Algodata Infosystems.

```

CREATE PROC info_éditeur2 @nom_éditeur varchar(40) = 'Algodata Infosystems'
AS
SELECT nom_auteur, pn_auteur, nom_éditeur
FROM auteurs a, éditeurs e, titres t, titreauteur ta
WHERE @nom_éditeur = e.nom_éditeur
      AND a.id_auteur = ta.id_auteur
      AND t.id_titre = ta.id_titre
      AND t.id_éditeur = e.id_éditeur

```

Paramètre
par défaut

- Si la valeur par défaut est une chaîne de caractères contenant des espaces ou des marques de ponctuation, ou si elle débute par un nombre (par exemple, 6xxx), elle doit figurer entre guillemets anglais simples.
- La procédure peut spécifier les actions à accomplir si l'utilisateur ne fournit pas de paramètre :

```

CREATE PROC montre_index3 @table varchar(30) = NULL
AS
IF @table is NULL
PRINT 'Entrez un nom de table'
ELSE
-- traitement du paramètre...

```

L'instruction 'select' permet de créer un recordset en sortie.

L'instruction 'print' permet d'afficher un message sous isql.

Dans la pratique les procédures stockées seront appelées par un client VB et l'instruction print sera inutilisable.

- L'option OUTPUT derrière un paramètre, indique un paramètre en sortie, qui est renvoyé à la procédure appelante : *somme_titres* est créée avec un paramètre d'entrée facultatif *titre_sélectionné* et un paramètre de retour *somme*

```

CREATE PROC somme_titres @titre_sélectionné varchar(40) = '%', @somme money OUTPUT
AS
SELECT 'Liste des Titres ' = titre
FROM titres

```

WHERE titre LIKE @titre_sélectionné

SELECT @somme = SUM(prix)
FROM titres
WHERE titre LIKE @titre_sélectionné

go

- Appel d'une procédure stockée avec un paramètre en sortie :

```
DECLARE @coût_total money
EXECUTE somme_titres 'Les%', @coût_total OUTPUT
IF @coût_total < 1000
BEGIN
    PRINT ''
    PRINT 'L'ensemble de ces titres peut s'acheter pour moins de 1 000 FF.'
END
ELSE
    PRINT 'Le coût total de ces livres s'élève à ' + convert (varchar(20), @coût_total)
```

go

Liste des Titres

Les festins de Parly 2
Les micro-ondes par gourmandise
Les secrets de la Silicon Valley

L'ensemble de ces livres peut s'acheter pour moins de 1 000 FF.

- Dans une procédure stockée, vous pouvez aussi retourner une valeur par l'instruction **RETURN**, et la récupérer dans l'appelant par la syntaxe suivante :

```
DECLARE @coût_total money
EXECUTE @coût_total = somme_titre 'Les%'
```

Ce code retour exprime souvent le bon ou mauvais déroulement de la procédure.

Création de procédures stockées : La gestion des erreurs.

Plusieurs techniques et bonnes pratiques peuvent être mises en œuvre lors de la gestion des erreurs.

- **Message d'erreur en OUTPUT**

Il est de bonne pratique de faire en sorte que la notion de gravité de l'erreur (type d'erreur) soit renvoyée par RETURN. Un code de retour égal à 0 indique un succès.

En parallèle, on renvoie en paramètre OUTPUT le libellé de l'erreur.

Exemple:

```
CREATE PROCEDURE Test @IdPilote TypeIdPilote, @message varchar(50) OUTPUT
AS
    IF NOT EXISTS (SELECT IdPilote From PILOTE Where IdPilote = @IdPilote)
```

```

BEGIN  /* Pilote inexistant */
    SET @message = 'Pilote inexistant'
    RETURN -2
END
ELSE
    RETURN 0

```

- **La variable @@ERROR** contient le numéro de l'erreur de la dernière instruction Transact-SQL exécutée. En effet, dans l'exemple précédent, on peut avoir une erreur d'exécution sur la requête SELECT, erreur liée à des contraintes externes à notre procédure stockée (connection avec la base impossible, absence de droits sur cette table, etc...). Cette variable @@ERROR va nous permettre de tester la bonne exécution d'une instruction. Si l'instruction s'est exécutée avec succès, @@ERROR renvoie la valeur 0.

Exemple:

```

CREATE PROCEDURE Test @nom Typenom,
                    @prenom Typenom,
                    @message varchar(50) OUTPUT
AS
....
    INSERT Pilote (NomPilote, PrenomPilote) VALUES (@nom, @prenom)
    IF @@ERROR <> 0
        BEGIN
            SET @message ='Erreur système : insertion du nouveau pilote !'
            RETURN -1
        END
    ...

```

- **Le bloc TRY...CATCH** permet d'éviter les tests fastidieux à chaque exécution. Ce mécanisme d'exception a été introduit par Microsoft depuis SQL SERVER 2005.

Exemple:

```

CREATE PROCEDURE Test @nom Typenom,
                    @prenom Typenom ,
                    @message varchar(50) OUTPUT
AS
    BEGIN TRY
        ....
        INSERT Pilote (NomPilote, PrenomPilote) VALUES (@nom, @prenom)
        ...
    END TRY
    BEGIN CATCH
        SET @message ='Erreur système '
        RETURN -1
    END CATCH

```

- **L'instruction RAISERROR** renvoie un message d'erreur défini par l'utilisateur et place un indicateur système pour enregistrer le fait qu'une erreur s'est produite.

Elle permet à l'application de lire une entrée dans la table système **sysmessages** ou de construire un message dynamiquement en spécifiant sa gravité et son état. Cette instruction peut écrire des messages d'erreur dans le journal des erreurs de SQL Server et dans le journal des applications de Windows.

Les valeurs de gravité comprises entre 1 et 14 sont utilisées par SQL SERVER pour indiquer différentes causes d'échec.

Exemple:

```
CREATE PROCEDURE Test @IdPilote TypeIdPilote
AS
IF NOT EXISTS (SELECT IdPilote From PILOTE Where IdPilote = @IdPilote)
BEGIN /* Pilote inexistant */
    RAISERROR ('N° pilote incorrect', 16, 1)
    RETURN -3
END
```

Après l'exécution de ce code, la valeur de retour contiendra la valeur -3 si le pilote existe. Un message d'erreur sera écrit dans le journal Applications de Windows et dans le journal des erreurs de SQL SERVER.

Pour utiliser un message d'erreur préenregistré, on codera : RAISERROR (n°message, 16, 1)

Pour inclure le code de l'employé dans le texte du message, on codera :

RAISERROR (n°message, 16, 1, @codemp)

- **Structure type d'une procédure stockée.**

- Il est de bonne pratique que la procédure commence par vérifier que tous les paramètres sont renseignés. Noter qu'il n'est pas vraiment possible de se protéger contre un paramètre non transmis à l'appel, mais il faut vérifier que les valeurs ne sont pas nulles.
- Encapsuler tous les accès base par un TRY-CATCH
- Commencer ensuite si besoin les vérifications fonctionnelles (existence du pilote, etc...). pour toute erreur fonctionnelle, renvoyer un code retour indiquant le type d'erreur, et un message en OUTPUT.
- (Nous améliorerons ce modèle type de procédure stockée en intégrant ultérieurement la gestion des transactions).

Voici un premier exemple complet de procédure :

```
-- =====
-- Author:      Lécu
-- MAJ:         12 janvier 2015
-- Description:  faire un retrait d'espèces
--
-- Tous les paramètres doivent être fournis dans l'appel,
-- mais les valeurs nulles sont vérifiées.
-- Code Retour
-- 1 : un paramètre possède une valeur nulle
-- 2 : le compte n'existe pas ou le solde n'est pas suffisant
-- 3 : erreur base de données (exception)
-- 0 : OK
-- Sortie : @Message : contient le message d'erreur
-- =====
Create PROCEDURE retrait @NumCompte int,
                        @Montant int,
                        @Message varchar(50) OUTPUT
AS
```

```

declare @soldeAvantOperation int;
declare @codeRet int;

--hors transaction
--Vérifier que les paramètres ne sont pas nuls
if @NumCompte is Null
begin
    set @codeRet=1;
    set @Message = 'Paramètre numcompte absent';
end
else if @Montant is Null
begin
    set @codeRet=1;
    set @Message = 'Paramètre montant absent';
end
else
begin
begin try

    --vérifier l'existence du compte
    if not EXISTS (select * from COMPTE
                    where NumCompte = @NumCompte)
    begin
        -- compte inexistant
        Set @message='compte inexistant';
        set @codeRet=2;
    end
    else
    begin
        SELECT @soldeAvantOperation = solde
        FROM COMPTE
        WHERE NumCompte = @NumCompte;

        if (@soldeAvantOperation < @Montant)
        begin
            -- solde insuffisant
            Set @message='solde insuffisant';
            set @codeRet=2;
        end
        else
        begin
            UPDATE COMPTE
            SET Solde = (Solde - @Montant)
            where NumCompte = @NumCompte;

            set @codeRet=0;          --OK
        end
    end
end try
begin catch
    --KO erreur base de données
    Set @message='erreur base de données' + ERROR_MESSAGE() ;
    set @codeRet=3;
end catch
end
return (@codeRet);

go

```

Dans tous les exercices suivants, nous créerons des procédures respectant les bonnes pratiques décrites ci-dessus.



Exercice 49 : création et test de procédures stockées

Créer une procédure stockée « Planning_perso » qui affiche, pour un Id pilote passé en paramètre, les numéros de vol, avec les aéroports de départ et d'arrivée, classés par date et heure de départ.

**** Planning personnel de LECU Régis

NumVOL	De	Vers	partant le
IT105	LYS	ORL	6 avril 6:0
IT102	CDG	NIC	6 avril 12:0
IT108	BRI	ORL	6 avril 19:0
IT103	GRE	BLA	7 avril 9:0
IT107	NIC	BRI	6 mai 7:0
IT107	NIC	BRI	7 juin 7:0

Si le paramètre est omis, la procédure affichera le message d'erreur : « Erreur : Idpilote non défini ». Si le pilote n'existe pas, la procédure affichera « Erreur : pilote inexistant ». Tester la procédure, et rajouter la au script de création de la base.

NB dans la pratique, pour plus d'un paramètre, il n'est pas possible de toujours se protéger contre un paramètre manquant, un paramètre manquant dans la séquence d'appel provoque souvent une exception dans la procédure stockée. Par contre il est de bonne pratique de vérifier que les paramètres d'entrée n'ont pas de valeur nulle.



Exercice 50 : Procédures imbriquées avec passage de paramètres, et valeur de retour

1) En utilisant la requête développée lors de l'exercice 25bis, faire une procédure Horaire qui renvoie la somme des heures de vol d'un pilote quelconque (connu par son nom et son prénom) pour une semaine donnée, et tester-la en affichant son résultat.

Une procédure stockée doit être une boîte noire : ne pas afficher de messages d'erreurs mais retourner un code d'erreur à l'appelant : 0 si succès, -1 si pilote inexistant. Le nom, le prénom du pilote et le numéro de semaine seront définis comme paramètres obligatoires.

2) Réaliser une procédure ObjectifHebdomadaire qui

- affiche le numéro de semaine demandée, le nom et le prénom du pilote, et son nombre d'heures hebdomadaire (renvoyé par Horaire)
- compare cette somme à un objectif passé en paramètre : nombre d'heures demandées
- affiche « Objectif atteint » ou « non atteint »

Ajouter Horaire et ObjectifHebdomadaire, aux scripts de création et de suppression

Remarques :

Les procédures stockées ci avant donneront des résultats faux s'il existe des pilotes de même nom et prénom. Il sera donc prudent de vérifier qu'il y a un et un seul pilote.

Une alternative intéressante consiste à fournir en paramètre d'entrée l' ID du pilote, plutôt que nom et prénom.

**Exercice 51 : Procédures imbriquées avec passage de paramètres, et valeur de retour**

- 1) *Elaborer une procédure VérifierPilote() qui en paramètre reçoit un identifiant pilote, un numéro de vol, et une date. Cette procédure va vérifier si le pilote peut être affecté à ce vol, c'est-à-dire si à cette date et pour ces heures là, il n'est pas déjà affecté à un autre vol. (On ne se préoccupera pas de la vraisemblance de son affectation liée à sa position géographique : par exemple s'il vient d'atterrir à Hong-Kong, on pourra l'affecter à un vol qui décolle de Los Angeles...). Cette procédure renvoie un booléen indiquant si l'affectation est possible ou non.*
- 2) *Elaborer une procédure VérifierAvion() qui en paramètre reçoit un numéro d'avion, un numéro de vol et une date. Cette procédure va vérifier si l'avion peut être affecté à ce vol, c'est-à-dire si à cette date et pour ces heures là, il n'est pas déjà affecté à un autre vol. Cette procédure renvoie un booléen indiquant si l'affectation est possible ou non.*
- 3) *Réaliser une procédure Affecter(), qui en paramètre reçoit un numéro d'avion, un numéro de pilote, un numéro de vol et une date. Cette procédure va vérifier que le pilote et l'avion peuvent être affectés (en utilisant les procédures précédemment développées), et va ensuite créer l'affectation.*

3.8 Les déclencheurs (triggers)



Doc. en ligne : « Créer et maintenir des bases de données » : « Respect des règles d'entreprise à l'aide de déclencheurs »
Aide Transact SQL : CREATE TRIGGER
SQL2 : p. 107 à 111

Définition

- Un *déclencheur* (trigger) est un type particulier de procédure stockée qui s'exécute automatiquement lorsque vous modifiez ou supprimez des données dans une table.
- Les triggers garantissent la cohérence des données liées logiquement dans différentes tables : par exemple, un trigger permettra de mettre à jour toutes les lignes de la table AFFECTATION, liées à l'avion que l'on va supprimer dans la table AVION.
- Chaque trigger est spécifique à certaines opérations sur les données : UPDATE, INSERT ou DELETE. Il s'exécute immédiatement après l'opération qui l'a lancé : le trigger et l'opération forment une seule transaction qui peut être annulée par le trigger. Si une erreur grave est détectée, toute la transaction est automatiquement annulée (en langage de bases de données, une transaction désigne une suite indivisible d'instructions SQL, qui s'effectuent toutes ou pas du tout ; cette notion sera détaillée dans les chapitres suivants)

Exemples d'utilisation :

- Pour effectuer des changements en cascade dans des tables liées de la base de données : un déclencheur UPDATE sur **titres.id_titre** provoque une mise à jour des lignes correspondant à ce titre dans les tables **titreauteur**, **ventes** et **droits_prévus**.
- Pour assurer des restrictions plus complexes que par la contrainte CHECK : contrairement à CHECK, les déclencheurs peuvent faire référence à des colonnes dans d'autres tables.
- Pour trouver la différence entre l'état d'une table avant et après une modification des données, et accomplir une ou plusieurs actions en fonction de cette différence.

Syntaxe du CREATE TRIGGER

- Un déclencheur est un objet de base de données. Pour créer un déclencheur, vous devez spécifier la table courante et les instructions de modification des données qui l'activent. Ensuite, vous devez spécifier la ou les actions que le déclencheur devra entreprendre.
- Une table peut avoir au maximum trois déclencheurs: un déclencheur UPDATE, un déclencheur INSERT et un déclencheur DELETE.
- Chaque déclencheur peut avoir de nombreuses fonctions et appeler jusqu'à 16 procédures. Un déclencheur ne peut être associé qu'à une seule table, mais il peut s'appliquer aux trois opérations de modifications de données (UPDATE, INSERT et DELETE).



De manière standard, les triggers peuvent se déclencher avant ou après le traitement déclencheur (on parle de TRIGGER AFTER ou de TRIGGER BEFORE). SQL-Server n'admet que des TRIGGER AFTER.

- Exemple 1 :

```
CREATE TRIGGER pense_bête
ON titres
FOR INSERT, UPDATE
AS
  RAISERROR ('Insertion Livres',15,9)
go
```

Nom de la table

Conditions de déclenchement

Actions associées

Ce trigger envoie un message au client lorsqu'on ajoute ou on modifie des données dans *titres*.
L'instruction SQL :

```
insert titres (id_titre, titre)
values ('BX100', 'test')
```

provoque le déclenchement du trigger, avec le message :

```
insert titres (id_titre, titre)
values ('BX100', 'test')
```

Msg. 50000, niveau 15, état 9

Insertion Livres

- Exemple 2 : ce déclencheur INSERT met à jour la colonne *cumulannuel_ventes* de la table *titres* chaque fois qu'une nouvelle ligne est ajoutée à la table *ventes*.

```
CREATE TRIGGER déclench_ins
ON ventes
FOR INSERT
AS
  UPDATE titres
  SET cumulannuel_ventes = cumulannuel_ventes + qt
  FROM inserted
WHERE titres.id_titre = inserted.id_titre
Go
```

Table temporaire = lignes modifiées de VENTES

```
-- Affichage du cumul de ventes par livres, avant la nouvelle vente
SELECT id_titre, cumulannuel_ventes
```

FROM titres
WHERE cumulannuel_ventes is NOT NULL

<i>id_titre</i>	<i>cumulannuel_ventes</i>
-----	-----
<i>BU1032</i>	<i>15</i>
<i>MC3021</i>	<i>40</i>
<i>PS2091</i>	<i>30</i>

-- Insertion d'une nouvelle vente : qt = 100 sur le livre 'BU1032'

INSERT ventes
VALUES ('6380', '6700', '1994', 100, 'Net 60', 'BU1032')

-- Provoque la mise à jour automatique de la table TITRES par le trigger, et donc aussi le déclenchement du trigger d'avertissement précédent :

Msg. 50000, niveau 15, état 9

Insertion Livres

--Réaffichage du cumul de ventes par livres, après la vente

<i>id_titre</i>	<i>cumulannuel_ventes</i>
-----	-----
<i>BU1032</i>	<i>115</i>
<i>MC3021</i>	<i>40</i>
<i>PS2091</i>	<i>30</i>

☺ Avec Management studio

Pour visualiser et mettre à jour un Trigger :

Dans l'explorateur du serveur, ouvrir la base, déployer les tables, puis déclencheur, dans la table qui déclenche.

**Exercice 52 : sécurisation des suppressions, avec modification en cascade**

Dans la table PILOTE, créer un trigger en suppression, qui met à NULL tous les champs IdPilote de la table AFFECTATION, liés au pilote à supprimer : avec cette méthode, on connaît les vols sans affectation, qui devront être affectés à un autre pilote. Tester le trigger en supprimant un pilote, et en réaffichant la table AFFECTATION.

Les lignes à supprimer sont stockées dans la table temporaire deleted. Pour faire cet exercice, il faut supprimer la contrainte de clé étrangère sur la colonne IdPilote de la table AFFECTATION : comme les contraintes de clé étrangère sont évaluées avant le lancement des triggers, le lien entre AFFECTATION et PILOTE interdit la suppression des pilotes référencés dans la table AFFECTATION.

A la fin, il faudra remettre la contrainte de clé étrangère sur la colonne IdPilote de la table AFFECTATION.



Ceci est un exercice ! Dans la réalité, lorsque les contraintes de clés étrangères ont été définies, il est bien entendu hors de question de les enlever ponctuellement pour effectuer un traitement !!!

**Exercice 53 : test de cohérence en insertion sur la table VOL**

Dans la table VOL, créer un trigger en insertion qui vérifie que la ville d'arrivée est différente de la ville de départ, et que l'heure d'arrivée est postérieure à l'heure de départ. Dans le cas contraire, on annule l'insertion en appelant l'instruction ROLLBACK.

Exercice 54 : test de cohérence en insertion sur la table AFFECTATION

Interdire à un pilote de voler plus de 35 heures par semaine, par un Trigger en insertion sur la table AFFECTATION.

Lors de l'affectation d'un pilote à un vol, on vérifie que la somme des heures de vol hebdomadaire de ce pilote, pour la semaine considérée, ne dépasse pas 35 heures

La somme des heures de vol du pilote pour une semaine représentée par son numéro, est donnée par la procédure Horaire.

Remarques**Le rollback dans un trigger**

Le rollback annule les actions du trigger et celles de la requête déclenchante

Si la requête appelante est elle-même dans une transaction, la transaction est annulée

De plus l'ensemble du lot appelant est abandonné, il est donc impossible de gérer l'incident.

Le raise error dans un trigger

Si la gravité est >10, cela provoque une exception dans le lot appelant.

Conclusion.

Compte tenu des complications évoquées ci avant, il paraît souhaitable de ne pas utiliser de triggers dans des contextes où un risque d'erreur existe.

3.9 Les curseurs



Doc. en ligne : « Créer et maintenir des bases de données » : « Curseurs »
Aide Transact SQL : DECLARE CURSOR

Les instructions SQL Server produisent un jeu de résultats complet mais il arrive que dans des procédures stockées ou lors de certains traitements, on ait besoin de balayer ces jeux d'enregistrements et de les traiter ligne à ligne.

L'ouverture d'un curseur sur un jeu de résultats permet de traiter celui-ci ligne par ligne.

Utilisation des curseurs

- L'instruction **DECLARE CURSOR** définit les attributs d'un curseur Transact-SQL, notamment s'il permet ou non le défilement, et la requête utilisée pour créer le jeu de résultats sur lequel le curseur opère :

```
DECLARE nom_cur CURSOR FOR  
SELECT ...
```

On peut aussi l'utiliser avec possibilité de modification en ajoutant **FOR UPDATE** à la fin de la requête :

```
DECLARE nom_cur CURSOR FOR  
SELECT ... FOR UPDATE
```

- L'instruction **OPEN** remplit le jeu de résultats :

```
OPEN nom_cur;
```

- L'instruction **FETCH** renvoie une ligne à partir de ce jeu de résultats, et est donc utilisée pour balayer le curseur.

```
FETCH NEXT FROM nom_cur : lit la ligne suivante.  
FETCH FIRST FROM nom_cur : lit la première ligne.  
FETCH LAST FROM nom_cur : lit la dernière ligne.  
FETCH ABSOLUTE i FROM nom_cur : lit la ligne i.  
FETCH RELATIVE n FROM nom_cur : lit la nième ligne suivante.  
FETCH NEXT FROM nom_cur into @var1, @var2 : lit la ligne  
suivante et place les données issues de l'extraction dans des  
variables.
```

Pour parcourir un curseur, on peut employer une boucle **WHILE** qui teste la valeur de la fonction **@@FETCH_STATUS** qui renvoie 0 tant que l'on n'est pas à la fin.

- L'instruction **CLOSE** libère le jeu de résultats actuel associé au curseur :

```
CLOSE nom_cur;
```

- L'instruction **DEALLOCATE** libère les ressources utilisées par le curseur :

```
DEALLOCATE nom_cur;
```

Exemple d'utilisation d'un curseur :

```

DECLARE Employee_Cursor CURSOR FOR
SELECT LastName, FirstName
FROM AdventureWorks.HumanResources.vEmployee
WHERE LastName like 'B%';

OPEN Employee_Cursor;

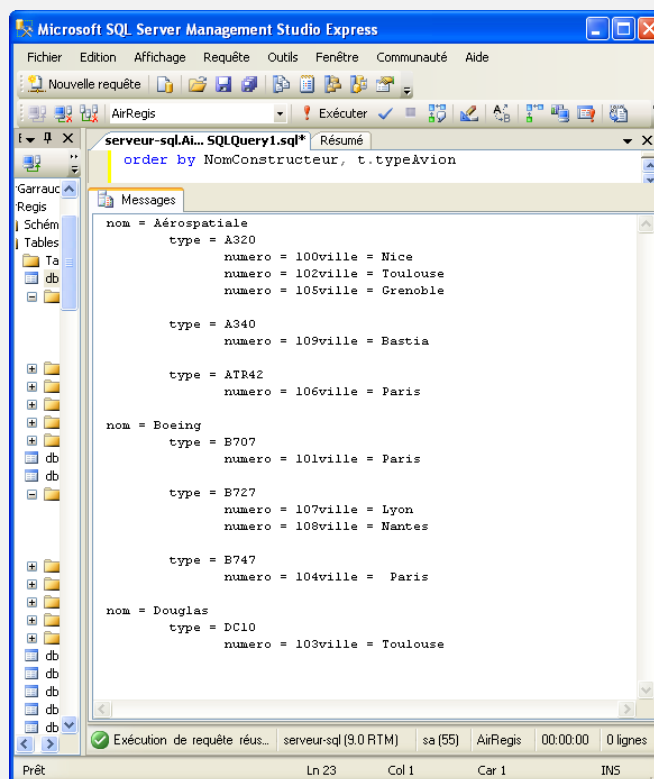
FETCH NEXT FROM Employee_Cursor;
WHILE @@FETCH_STATUS = 0
BEGIN
    FETCH NEXT FROM Employee_Cursor
END;

CLOSE Employee_Cursor;
DEALLOCATE Employee_Cursor;

```

**Exercice 55 : utilisation d'un curseur**

Utilisez un curseur pour afficher (avec des print), l'état suivant (liste des numéros d'avion avec la ville ou ils sont basés, classés par type d'appareil et par nom de constructeur. Il s'agit de respecter le formalisme d'affichage, et donc de faire des ruptures par constructeur et par type d'avion.



3.10 Les index



Doc. en ligne : « Créer et maintenir des bases de données » : « Index »
 Doc. en ligne : « Optimiser la performance des bases de données »
 Aide Transact SQL : CREATE INDEX

Présentation simplifiée et conseils

- Les index permettent d'optimiser le fonctionnement d'une base de données : ils accélèrent les opérations SELECT, INSERT, UPDATE et DELETE en fournissant un accès direct sur certaines colonnes ou ensembles de colonnes.
- Un index ordonné (option CLUSTERED) fournit un accès direct aux lignes physiques de la table, sur le disque. La création d'un index ordonné modifie l'ordre physique des lignes : il faut donc le créer avant les index non ordonnés, et il ne peut y avoir qu'un index ordonné par table (souvent sur la clé primaire).
- Un index non ordonné (sans l'option CLUSTERED) est un index qui s'appuie sur l'ordre logique de la table, en mémoire. Une table peut contenir plusieurs index non ordonnés. Employez un index non ordonné pour :
 - des colonnes qui contiennent un nombre élevé de valeurs uniques;
 - des requêtes qui renvoient de petits ensembles de résultats
- La recherche de données au moyen d'un index ordonné est presque toujours plus rapide qu'avec un index non ordonné. L'index ordonné est particulièrement utile lorsque l'on veut extraire en une seule instruction plusieurs lignes qui possèdent des clés contiguës : l'index garantit que ces lignes seront physiquement adjacentes.
- Pour écrire un index efficace, il faut visualiser la « sélectivité » des colonnes = le nombre de valeurs uniques par rapport au nombre de lignes de la table, données par l'instruction SQL :

SELECT COUNT (DISTINCT *nom_de_colonne*) FROM *nom_de_table*

Conseils pour une table de 10 000 lignes:

Nombre de valeurs uniques	Choix de l'index
5000	Index non ordonné
20	Index ordonné
3	Pas d'index

- **Index simple** : sur la colonne id_d'auteur de la table auteurs.

```
CREATE INDEX index_id_auteur
ON auteurs (id_auteur)
```

- **Index ordonné unique** : sur la colonne id_d'auteur de la table auteurs pour assurer l'unicité des données. L'option CLUSTERED étant spécifiée, cet index classera physiquement les données sur le disque.

```
CREATE UNIQUE CLUSTERED INDEX index_id_auteur  
ON auteurs (id_auteur)
```

- Index composé simple : crée un index sur les colonnes id_d'auteur et id_titre de la table titreauteur.

```
CREATE INDEX ind1  
ON titreauteur (id_auteur, id_titre)
```



Exercice 56 : optimisation de la base « Compagnie aérienne »

Créer un Index non ordonné composé sur les champs NomPilote et PrenomPilote de la table Pilote.

Vérifier l'existence de l'index sous Microsoft SQL Management Studio : sélectionner la table, puis index.

3.11 Récapitulation

- **Reprendre tous les exercices et compléter le script de création de la base, dans l'ordre :**

- définition des types utilisateurs
- création des tables, en commençant par les tables indépendantes
- création des vues
- création des fonctions
- création des procédures stockées
- définition des permissions sur les tables, les vues, les procédures stockées
- création des triggers
- création des index

- **Ecrire un script d'initialisation complète des tables (INSERT...)**

- **Ecrire un script de destruction complète de la base, dans l'ordre inverse de la création :**

- suppression des index, triggers, procédures stockées
- suppression des fonctions
- suppression des vues
- suppression des tables, en commençant par les tables avec des clés étrangères
- suppression des types utilisateurs



Avec SQL Management studio

Conservez votre base et créez une autre base vide, que vous allez gérer entièrement sous SQL Server Management Studio. Pour cela utilisez le document complémentaire :

Administration sql serveur 2005 avec l'outil Microsoft sql serveur management studio

- **Types de données utilisateur** : bouton droit => « Nouveau type de données défini par l'utilisateur »

- **Tables** : sélectionnez l'objet « Tables » : bouton droit => « Nouvelle Table »

L'utilisation est assez intuitive : pour définir les contraintes de clé primaire => cliquez sur la clé jaune dans la barre d'outil (comme sous ACCESS)

- **Vues** : sélectionnez l'objet « Vues » : bouton droit => « Nouvelle Vue ».

Vous pouvez définir votre Vue en langage SQL, ou avec un outil graphique comparable à celui d'ACCESS et de VISUAL BASIC.

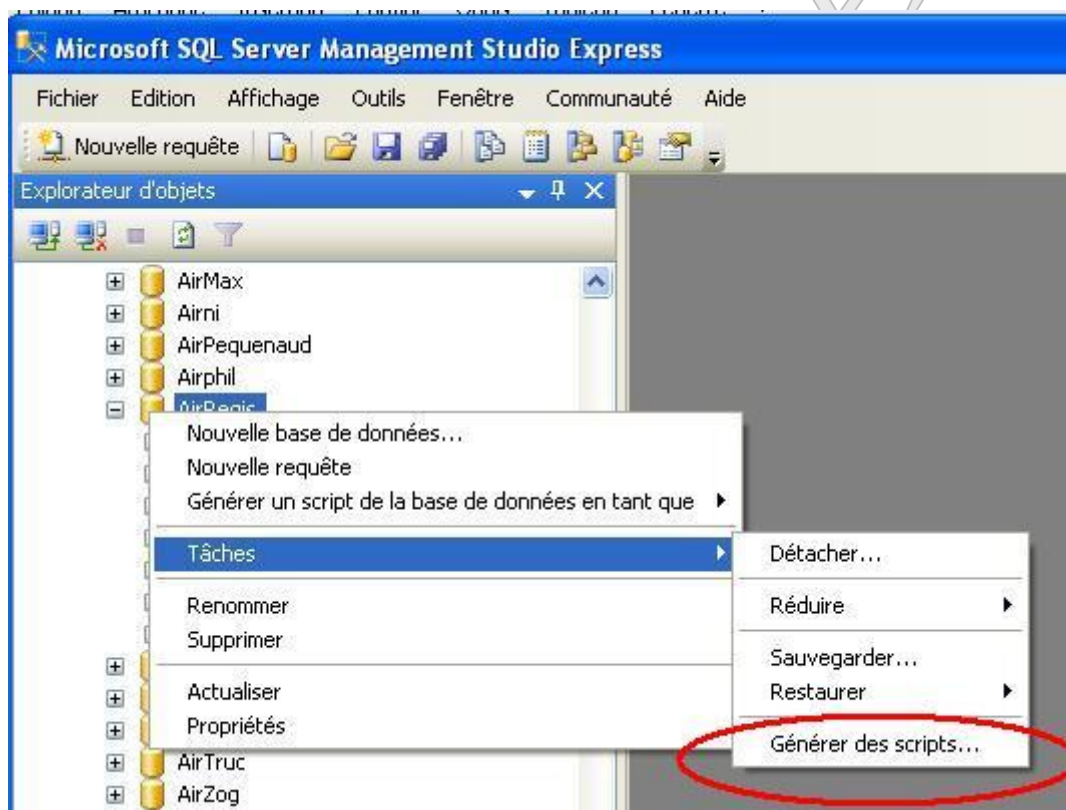
- **Procédures stockées** : sélectionnez l'objet « Procédures Stockées » : bouton droit => « Nouvelle Procédure Stockée ».

Pas d'aide graphique : il faut l'écrire en SQL dans la fenêtre !

- **Triggers** : sélectionnez une à une les tables concernées : bouton droit « Modifier une table », puis bouton « Déclencheurs » dans la barre d'outil de la fenêtre « Créer la table... »

Demander une création automatique de scripts de création et de destruction, et comparez les scripts générés aux scripts que vous venez d'écrire

Sélectionner votre base : avec le bouton droit, choisir « Toutes Tâches », « Générer des scripts SQL »



A titre d'exemple : lire le script de création de la base PUBLI utilisé au début de ce cours (Extrait)

Faire des recherches documentaires sur les syntaxes qui n'ont pas été vues dans la construction guidée. Ce script peut aussi vous servir d'exemples pour la plupart des syntaxes.

```
-- - InstPubli.SQL    2001

GO
set nocount      on
set dateformat   mdy

USE master
GO
declare @dtm varchar(55)
select  @dtm=convert(varchar,getdate(),113)
raiserror('Début de InstPubli.SQL à %s ....',1,1,@dtm) with nowait
if exists (select * from sysdatabases where name='publi')
    DROP database publi
GO

CHECKPOINT
GO
CREATE DATABASE publi
    on master = 3
GO
CHECKPOINT
GO
USE publi
GO
if db_name() = 'publi'
    raiserror('Base de données ''publi'' créée et contexte actuellement utilisé.',1,1)
else
    raiserror('Erreur dans InstPubli.SQL, ''USE publi'' a échoué!  Arrêt immédiat du SPID.'
        ,22,127) with log
go

execute sp_dboption 'publi' , 'trunc. log on chkpt.' , 'true'

EXECUTE sp_addtype id, 'varchar(11)', 'NOT NULL'
EXECUTE sp_addtype idt, 'varchar(6)', 'NOT NULL'
EXECUTE sp_addtype empid, 'char(9)', 'NOT NULL'
go

raiserror('Etape de création de la section de table ....',1,1)
GO

CREATE TABLE auteurs
(
    id_auteur    id
        CHECK (id_auteur like '[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9]')
        CONSTRAINT UPKCL_auidind PRIMARY KEY CLUSTERED,
    nom_auteur   varchar(40)    NOT NULL,
    pn_auteur    varchar(20)    NOT NULL,
    téléphone    char(12)      NOT NULL
        DEFAULT ('INCONNU'),
    adresse      varchar(40)    NULL,
    ville        varchar(20)    NULL,
    pays         char(2)        NULL,
    code_postal  char(5)        NULL
        CHECK (code_postal like '[0-9][0-9][0-9][0-9][0-9]'),
    contrat      bit            NOT NULL
)
go
```

```

CREATE TABLE éditeurs
(
    id_éditeur char(4) NOT NULL
        CONSTRAINT UPKCL_pubind PRIMARY KEY CLUSTERED
        CHECK (id_éditeur in ('1389', '0736', '0877', '1622', '1756')
            OR id_éditeur like '99[0-9][0-9]'),
    nom_éditeur varchar(40) NULL,
    ville varchar(20) NULL,
    région char(2) NULL,
    pays varchar(30) NULL
        DEFAULT('FR')
)
go

CREATE TABLE titres
(
    id_titre idt
        CONSTRAINT UPKCL_titleidind PRIMARY KEY CLUSTERED,
    titre varchar(80) NOT NULL,
    type char(12) NOT NULL
        DEFAULT ('SANS TITRE'),
    id_éditeur char(4) NULL
        REFERENCES éditeurs(id_éditeur),
    prix money NULL,
    avance money NULL,
    droits int NULL,
    cumulannuel_ventes int NULL,
    notes varchar(200) NULL,
    datepub datetime NOT NULL
        DEFAULT (getdate())
)
go

CREATE TABLE titreauteur
(
    id_auteur id
        REFERENCES auteurs(id_auteur),
    id_titre idt
        REFERENCES titres(id_titre),
    cmd_auteur tinyint NULL,
    droits_pourcent int NULL,
    CONSTRAINT UPKCL_taind PRIMARY KEY CLUSTERED(id_auteur, id_titre)
)
go

CREATE TABLE magasins
(
    id_mag char(4) NOT NULL
        CONSTRAINT UPKCL_storeid PRIMARY KEY CLUSTERED,
    nom_mag varchar(40) NULL,
    adresse_mag varchar(40) NULL,
    ville varchar(20) NULL,
    pays char(2) NULL,
    code_postal char(5) NULL
)
go

CREATE TABLE ventes
(
    id_mag char(4) NOT NULL
        REFERENCES magasins(id_mag),
    num_cmd varchar(20) NOT NULL,
    date_cmd datetime NOT NULL,
    qt smallint NOT NULL,
    modepaiements varchar(12) NOT NULL,
    id_titre idt
        REFERENCES titres(id_titre),
    CONSTRAINT UPKCL_sales PRIMARY KEY CLUSTERED (id_mag, num_cmd, id_titre)
)

```

go

```
CREATE TABLE droits_prévus
(
    id_titre      idt
    REFERENCES titres(id_titre),
    minimum int    NULL,
    maximum int    NULL,
    droits int     NULL
)
go
```

```
CREATE TABLE remises
(
    typeremise    varchar(40)    NOT NULL,
    id_mag char(4) NULL
    REFERENCES magasins(id_mag),
    qtémin smallint    NULL,
    qtémax smallint    NULL,
    remise dec(4,2) NOT NULL
)
go
```

```
CREATE TABLE emplois
(
    id_emploi smallint
    IDENTITY(1,1)
    PRIMARY KEY CLUSTERED,
    desc_emploi varchar(50) NOT NULL
    DEFAULT 'Nouveau poste - pas de dénomination officielle',
    niv_min tinyint NOT NULL
    CHECK (niv_min >= 10),
    niv_max tinyint NOT NULL
    CHECK (niv_max <= 250)
)
go
```

```
CREATE TABLE pub_info
(
    pub_id char(4) NOT NULL
    REFERENCES éditeurs(id éditeur)
    CONSTRAINT UPKCL_pubinfo PRIMARY KEY CLUSTERED,
    logo image NULL,
    pr_info text NULL
)
go
```

```
CREATE TABLE employé
(
    id_employé empid
    CONSTRAINT PK_id_employé PRIMARY KEY NONCLUSTERED
    CONSTRAINT CK_id_employé CHECK (id_employé LIKE
    '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-9][FM]' or
    id_employé LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-9][0-9][FM]'),
    pn_employé varchar(20) NOT NULL,
    init_centrale char(1) NULL,
    nom_employé varchar(30) NOT NULL,
    id_emploi smallint NOT NULL
    DEFAULT 1
    REFERENCES emplois(id_emploi),
    position_employé tinyint
    DEFAULT 10,
    id_éditeur char(4) NOT NULL
    DEFAULT ('9952')
    REFERENCES éditeurs(id_éditeur),
    date_embauche datetime NOT NULL
)
```

```

        DEFAULT (getdate())
    )
go

raiserror('Etape de création de la section de trigger ....',1,1)
GO

CREATE TRIGGER employé_insupd
ON employé
FOR INSERT, UPDATE
AS
--Get the range of level for this job type from the jobs table.
DECLARE @niv_min tinyint,
        @niv_max tinyint,
        @niv_emploi tinyint,
        @id_emploi smallint
SELECT @niv_min = niv_min,
       @niv_max = niv_max,
       @niv_emploi = i.position_employé,
       @id_emploi = i.id_emploi
FROM employé e, emplois j, inserted i
WHERE e.id_employé = i.id_employé AND i.id_emploi = j.id_emploi
IF (@id_emploi = 1) and (@niv_emploi <> 10)
BEGIN
    RAISERROR ('L'identificateur d'emploi 1 attend le niveau par défaut 10.',16,-1)
    ROLLBACK TRANSACTION
END
ELSE
IF NOT (@niv_emploi BETWEEN @niv_min AND @niv_max)
BEGIN
    RAISERROR ('Le niveau de id_emploi:%d doit se situer entre %d et %d.',
               16, -1, @id_emploi, @niv_min, @niv_max)
    ROLLBACK TRANSACTION
END
go

raiserror('Etape d'insertion des auteurs ....',1,1)
GO

INSERT auteurs
VALUES('807-91-6654','Bec','Arthur','22.47.87.11','4, chemin de la Tour de Campel',
      'Bruges','BE','05006',1)
go

raiserror('Etape d'insertion des éditeurs ....',1,1)
GO
INSERT éditeurs VALUES('9901', 'GGG&G', 'Munich', NULL, 'GER')
INSERT éditeurs VALUES('9999', 'Editions Lucerne', 'Paris', NULL, 'FR')
go

raiserror('Etape d'insertion des pub_info ....',1,1)
GO
INSERT pub_info VALUES('9901', 0xFFFFFFFF, 'Aucune information actuellement')
INSERT pub_info VALUES('9999', 0xFFFFFFFF, 'Aucune information actuellement')
go

raiserror('Etape d'insertion des titres ....',1,1)
GO
INSERT titres VALUES('PC8888','Les secrets de la Silicon
Valley','informatique','1389',$136.00,$54000.00,10,4095,'Deux femmes courageuses dévoilent tous
les scandales qui jonchent l'irrésistible ascension des pionniers de l'informatique. Matériel
et logiciel : personne n'est épargné.','12/06/85' )
go
raiserror('Etape d'insertion de titreauteur ....',1,1)
GO
INSERT titreauteur VALUES('172-32-1176', 'PS3333', 1, 100)
INSERT titreauteur VALUES('213-46-8915', 'BU1032', 2, 40)

```



```

go

raiserror('Etape d''insertion des magasins ....',1,1)

GO
INSERT magasins VALUES('7066','Librairie spécialisée','567, Av. de la
Victoire','Paris','FR','75016')
go

raiserror('Etape d''insertion des ventes ....',1,1)

GO
INSERT ventes VALUES('7066', 'QA7442.3', '11/09/94', 75, 'Comptant','PS2091')
go

raiserror('Etape d''insertion de droits_prévus ....',1,1)

GO
INSERT droits_prévus VALUES('BU2075', 1001, 3000, 12)
INSERT droits_prévus VALUES('BU2075', 3001, 5000, 14)
go

raiserror('Etape d''insertion des remises ....',1,1)

GO
INSERT remises VALUES('Remise volume', NULL, 100, 1000, 6.7)
INSERT remises VALUES('Remise client', '8042', NULL, NULL, 5.0)
go

raiserror('Etape d''insertion des emplois ....',1,1)

GO
INSERT emplois VALUES ('Directeur des opérations', 75, 150)
INSERT emplois VALUES ('Rédacteur', 25, 100)
go

raiserror('Etape d''insertion des employé ....',1,1)

GO
INSERT employé VALUES ('M-P91209M', 'Manuel', '', 'Pereira', 8, 101, '9999', '09/01/89')
go

raiserror('Etape de création de la section d''index ....',1,1) with nowait

GO
CREATE CLUSTERED INDEX employé_ind ON employé(nom_employé, pn_employé, init_centrale)
go

CREATE NONCLUSTERED INDEX aunmind ON auteurs (nom_auteur, pn_auteur)
go
CREATE NONCLUSTERED INDEX titleidind ON ventes (id_titre)
go
CREATE NONCLUSTERED INDEX titleind ON titres (titre)
go
CREATE NONCLUSTERED INDEX auidind ON titreauteur (id_auteur)
go
CREATE NONCLUSTERED INDEX titleidind ON titreauteur (id_titre)
go
CREATE NONCLUSTERED INDEX titleidind ON droits_prévus (id_titre)
go

raiserror('Etape de création de la section d''view ....',1,1) with nowait
GO
CREATE VIEW vuetitre
AS
SELECT titre, cmd_auteur, nom_auteur, prix, cumulannuel_ventes, id_éditeur
FROM auteurs, titres, titreauteur
WHERE auteurs.id_auteur = titreauteur.id_auteur
      AND titres.id_titre = titreauteur.id_titre
go

```

```
raiserror('Etape de création de la section de procedure ....',1,1) with nowait

GO

CREATE PROCEDURE pardroits @pourcentage int
AS
SELECT id_auteur FROM titreauteur
WHERE titreauteur.droits_pourcent = @pourcentage
go

raiserror('(Ensuite, premier octroi incorporé dans la section de proc.)',1,1)

GRANT EXECUTE ON pardroits TO public
go

CREATE PROCEDURE reptq1 AS
SELECT id_éditeur, id_titre, prix, datepub
FROM titres
WHERE prix is NOT NULL
ORDER BY id_éditeur
COMPUTE avg(prix) BY id_éditeur
COMPUTE avg(prix)
go

GRANT EXECUTE ON reptq1 TO public
go

CREATE PROCEDURE reptq2 AS
SELECT type, id_éditeur, titres.id_titre, cmd_auteur,
       Name = substring (nom_auteur, 1,15), cumulannuel_ventes
FROM titres, auteurs, titreauteur
WHERE titres.id_titre = titreauteur.id_titre AND auteurs.id_auteur = titreauteur.id_auteur
       AND id_éditeur is NOT NULL
ORDER BY id_éditeur, type
COMPUTE avg(cumulannuel_ventes) BY id_éditeur, type
COMPUTE avg(cumulannuel_ventes) BY id_éditeur
go

GRANT EXECUTE ON reptq2 TO public
go

CREATE PROCEDURE reptq3 @limite_inférieure money, @limite_supérieure money,
@type char(12)
AS
SELECT id_éditeur, type, id_titre, prix
FROM titres
WHERE prix >@limite_inférieure AND prix <@limite_supérieure AND type = @type OR type LIKE
'%cui%'
ORDER BY id_éditeur, type
COMPUTE count(id_titre) BY id_éditeur, type
go

GRANT EXECUTE ON reptq3 TO public
go
GRANT CREATE PROCEDURE TO public
go

raiserror('Etape de la section de GUEST ....',1,1)

GO

EXECUTE sp_adduser guest
go
```

```
GRANT ALL ON éditeurs TO guest
... idem sur toutes les tables...
```

```
GRANT EXECUTE ON pardroits TO guest
GRANT CREATE TABLE TO guest
GRANT CREATE VIEW TO guest
GRANT CREATE RULE TO guest
GRANT CREATE DEFAULT TO guest
GRANT CREATE PROCEDURE TO guest
go
```

```
UPDATE STATISTICS éditeurs
... idem sur toutes les tables...
GO
```

```
CHECKPOINT
```

```
go
```

```
declare @dtm varchar(55)
```

```
select @dtm=convert(varchar,getdate(),113)
```

```
raiserror('Fin de InstPubli.SQL à %s ....',1,1,@dtm) with nowait
```

**Exercices complémentaires sur la base "Compagnie aérienne"**

Pour faire cette suite d'exercices, commencer par ajouter une colonne Ville à la table Pilote : ville où réside le pilote. Remplir cette colonne avec un jeu de test, qui comprend certaines des villes où sont localisés des avions (corrigés en annexe)

- 1 Noms des pilotes qui habitent dans la ville de localisation d'un A320 (dans une ville desservie par un aéroport où sont basés des A320)
- 2 Noms des pilotes planifiés pour des vols sur A320
- 3 Noms des pilotes planifiés sur A320, qui habitent dans la ville de localisation d'un A320
- 4 Noms des pilotes planifiés sur A320, qui n'habitent pas la ville de localisation d'un A320
- 5 Noms des pilotes planifiés sur A320 ou qui habitent la ville de localisation d'un A320
- 6 Pour chaque ville desservie, donner la moyenne, le minimum et le maximum des capacités des avions qui sont localisés dans un aéroport desservant cette ville
- 7 Même question, limitée aux villes où sont localisés plus de deux avions
- 8 Afficher les noms des pilotes qui ne pilotent que des A320
- 9 Afficher les noms des pilotes qui pilotent tous les A320 de la compagnie

4. LES TRANSACTIONS



Doc. en ligne : « Accéder aux données et les modifier » : « Transactions »
SQL2 : chap. 13, « La gestion des transactions »

Principes

- Une transaction est une unité logique de travail, un ensemble d'instructions d'interrogation ou de mise à jour des données que SQL traite comme une seule action indivisible : soit toutes les instructions comprises dans la transaction sont traitées, soit aucune.
- Ce mécanisme garantit la fiabilité des accès au SGBD et la cohérence des données : il pare à toute interruption non désirée d'une séquence d'instructions (arrêt programme, panne machine ou volonté d'annuler les dernières opérations).
- Une unité logique de travail doit posséder quatre propriétés appelées propriétés **ACID** (Atomicité, Cohérence, Isolation et Durabilité), pour être considérée comme une transaction :
 - Atomicité : une transaction doit être une unité de travail indivisible ; soit toutes les modifications de données sont effectuées, soit aucune ne l'est.

Soit une transaction bancaire qui débite un compte A et crédite un compte B : sans le mécanisme de transaction, si une erreur survenait entre le débit et le crédit, la somme versée serait perdue pour tout le monde. Le mécanisme de transaction va lier les deux opérations : en cas de crash du système central ou de coupure du réseau entre l'agence bancaire et le central, les deux opérations sont annulées.

- Cohérence : lorsqu'elle est terminée, une transaction doit laisser les données dans un état cohérent.

C'est une conséquence de l'atomicité : si l'on arrive sans erreur à la fin de la transaction, les comptes débiteur et créditeur sont actualisés par une opération indivisible ; si une erreur survient pendant la transaction, les deux opérations sont annulées. Dans les deux cas, les deux soldes demeurent cohérents.

Remarquons que le mécanisme de transaction ne fait pas tout : si le problème survient immédiatement après la fin de la transaction, le virement a eu lieu, et il n'est pas question de le faire une seconde fois ; au contraire, si l'erreur survient juste avant, il faut refaire le virement.

Ce sera donc à l'application de tenir à jour un « journal » des virements, pour savoir dans tous les cas si le virement a eu lieu ou non : pour restituer fidèlement les opérations, le journal doit évidemment être actualisé dans la transaction.

- Isolation : les modifications effectuées par des transactions concurrentes doivent être isolées transaction par transaction. Une transaction accède aux données soit dans l'état où elles étaient avant d'être modifiées par une transaction concurrente, soit telles qu'elles se présentent après exécution de cette dernière, mais jamais dans un état intermédiaire.

Tant qu'une transaction n'est pas finie, on ne peut pas être sûr qu'elle aboutisse sans erreur : un utilisateur qui consulte les deux soldes depuis un autre terminal de gestion, doit donc voir les anciens soldes = les dernières données dont la cohérence est garantie.

- **Durabilité** : Lorsqu'une transaction est terminée, ses effets sur le système sont permanents. Les modifications sont conservées même en cas de défaillance du système.

Remarques :

- 1) Ce type de fonctionnement est possible parce que le S.G.B.D. dispose d'une copie des tables avant modification (souvent appelée SNAPSHOT = l'état instantané. C'est le journal des transactions). Lors du COMMIT, le SGBDR recopie le fichier SNAPSHOT sur le fichier de bases de donnée, en une seule opération indivisible.
- 2) Un ROLLBACK automatique est exécuté en cas de panne de la machine.
- 3) ROLLBACK et COMMIT ne concernent pas les commandes de définition de données : CREATE, DROP, ALTER. Toute modification portant sur la structure de la base de données, est donc irréversible.
- 4) L'exécution fréquente d'un COMMIT peut éviter la perte d'informations.

Mise en œuvre en Transact SQL

- Selon les SGBD, le début d'une transaction est déclenché différemment : sous SQL SERVER, il est marqué par l'instruction :

BEGIN TRANSACTION

- La fin d'une transaction est marquée par les instructions **COMMIT** ou **ROLLBACK**.
- Les modifications ne sont rendues permanentes que lors de l'instruction **COMMIT** : tant que les changements ne sont pas validés par un **COMMIT**, seul l'utilisateur travaille sur les données qu'il vient de modifier alors que les autres utilisateurs travaillent sur les valeurs avant modification.
- L'instruction **ROLLBACK** annule toutes les modifications effectuées depuis le début de la transaction en cours.
- En l'absence de transaction explicite marquée par BEGIN TRANSACTION, toute instruction SQL constitue une transaction élémentaire.



Exercice 57 : vérification du mécanisme ACID, observation des verrouillages

Travaillez sur votre base « Compagnie Aérienne » avec l'analyseur de requête : on ouvrira deux sessions par binôme à deux noms différents.

- **pour que SQL SERVER gère correctement les transactions, commencer par spécifier dans les deux sessions :**

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

- 1) Créer un nouveau pilote, en terminant la transaction par **ROLLBACK**. Vérifier que le nouveau pilote n'est pas conservé dans la base (faire un **SELECT** avant et après le **ROLLBACK**)
- 2) Ouvrir une transaction par session. Faire une lecture sur la table Pilote (**SELECT**) dans chaque session. Que constatez-vous ?
- 3) Ouvrir une transaction par session. Dans une des sessions, on lit les données (**SELECT** sur Pilote) ; dans l'autre on essaie de les changer (**UPDATE** sur Pilote). Que constatez-vous ?
- 5) Ouvrir une transaction par session. On commence par modifier dans une session et on lit dans l'autre. Que constatez-vous ?

Pour expliquer ce qui a été observé, nous allons détailler les deux utilisations des transactions :

- dans la gestion des erreurs (utilisation du **ROLLBACK**) ;
- dans la gestion des accès « concurrents » (plusieurs clients accédant aux mêmes tables en lecture ou en écriture, entraînant un verrouillage)

L'utilisation des transactions dans la gestion d'erreur

- Pour écrire une transaction correcte, il faut faire un **ROLLBACK** dès qu'une erreur est détectée.
- Nous allons reprendre notre modèle de procédure stockée et y intégrer la gestion des transactions :

Voici un exemple complet de procédure avec gestion des transactions :

```
-- =====
-- Author:          Lécu
-- MAJ:             12 janvier 2015
-- Description:      faire un retrait d'espèces
--
-- Tous les paramètres doivent être fournis dans l'appel,
-- mais les valeurs nulles sont vérifiées.
-- Code Retour
--      1 : un paramètre possède une valeur nulle
--      2 : le compte n'existe pas ou le solde n'est pas suffisant
--      3 : erreur base de données (exception)
--      0 : OK
-- Sortie : @Message : contient le message d'erreur
-- =====
Create PROCEDURE retrait @NumCompte int,
                        @Montant int,
                        @Message varchar(50) OUTPUT
AS
    declare @soldeAvantOperation int;
    declare @codeRet int;

    --hors transaction
    --Vérifier que les paramètres ne sont pas nuls
    if @NumCompte is Null
    begin
        set @codeRet=1;
        set @Message = 'Paramètre numcompte absent';
    end
    else if @Montant is Null
    begin
        set @codeRet=1;
```

```

        set @Message = 'Paramètre montant absent';
    end
    else
    begin
        begin try
            begin transaction
            --vérifier l'existence du compte, dans la transaction
            if not EXISTS (select *
                           from COMPTE with (holdlock, tablockx)
                           where NumCompte = @NumCompte)
            begin
                -- compte inexistant
                Set @message = 'compte inexistant';
                set @codeRet = 2;
                rollback transaction;
            end
            else
            begin
                SELECT @soldeAvantOperation = solde
                FROM COMPTE
                WHERE NumCompte = @NumCompte;

                if (@soldeAvantOperation < @Montant)
                begin
                    -- solde insuffisant
                    Set @message = 'solde insuffisant';
                    set @codeRet = 2;
                    rollback transaction;
                end
                else
                begin
                    UPDATE COMPTE
                    SET Solde = (Solde - @Montant)
                    where NumCompte = @NumCompte;

                    set @codeRet=0;          --OK
                    commit transaction;
                end
            end
        end try
        begin catch
            --KO erreur base de données
            Set @message='erreur base de données' + ERROR_MESSAGE() ;
            set @codeRet=3;
            rollback transaction;
        end catch
    end
    return (@codeRet);
go

```



Exercice 58 : réalisation d'une procédure stockée fiable, avec transaction

On veut gérer par une procédure stockée les départs de la société, et les réaffectations des vols au pilote remplaçant. La procédure doit être fiable : un pilote ne doit être supprimé des listes de la compagnie aérienne, qu'une fois le remplaçant entré dans les listes et tous ses vols réaffectés. Ces trois opérations doivent être indivisibles.

On propose l'interface suivante pour la procédure stockée :

ENTREE

@idAncien : identificateur de l'ancien pilote (OBLIGATOIRE)

@nom : nom du nouveau pilote (OBLIGATOIRE)

@prenom : prenom du nouveau pilote (OBLIGATOIRE)

SORTIE

@idNouveau : identificateur du nouveau pilote

RETOUR

(0) *Remplacement OK,* (1) *Erreur : paramètres incorrects,*

(2) *Erreur : l'ancien pilote, n'existe pas,* (3) *Erreur système*

La procédure sera livrée avec son programme de test...

Conseils pour une procédure stockée bien conçue

- Affecter une valeur de code d'erreur, pour chaque anomalie possible.
- Vérifier hors transaction la vraisemblance des paramètres d'entrée.
- Dans une transaction.
- Vérifier que la mise à jour envisagée est compatible avec l'intégrité de la base
- Dans la négative faire un rollback.
- Utiliser les try catch, en cas d'échec faire un rollback.
- L'algo de la procédure comprendra des nombreuses branches, seule la branche succès comporte un commit

Malgré toutes nos précautions, une exception durant l'exécution d'une procédure stockée reste possible, en ce cas la transaction peut rester ouverte. Il faudra en tenir compte au niveau appelant en testant le nombre de transactions ouvertes

(La fonction @@**TRANCOUNT** vaut zéro lorsque aucune transaction n'est ouverte ; une instruction **BEGIN TRAN** incrémente @@**TRANCOUNT** de 1, et une instruction **ROLLBACK TRAN** décrémente @@**TRANCOUNT** de 1).

L'accès « concurrent » : verrouillages dans les transactions

- Un SGBD étant souvent utilisé sur une machine multi-utilisateurs, ou en Client-Serveur à partir de plusieurs clients, on peut imaginer ce qui arriverait si deux transactions pouvaient modifier en même temps la même ligne d'une table !

Reprenons l'exemple du débit/crédit, en supposant que la banque ne tolère aucun découvert. L'application qui effectue le virement doit tester si le compte A est approvisionné, avant de faire le débit.

Si l'on n'utilise pas de transaction, on tombe sur un problème classique en programmation « concurrente » : deux applications veulent débiter le même compte ; elles commencent par tester le solde qui est suffisant pour chacune d'entre elles, et débitent ensuite la somme demandée, en aboutissant à un solde négatif.

Le compte à débiter a un solde initial de 1200 F

Application A : Demande un débit de 1000 F	Application B : Demande un débit de 500 F
(1) Test du solde OK : $1200 > 1000$	(2) Test du solde OK : $1200 > 500$
(3) Débit de 1000 F \Rightarrow nouveau solde = 200 F	(4) Débit de 500 F \Rightarrow solde final = -300 F

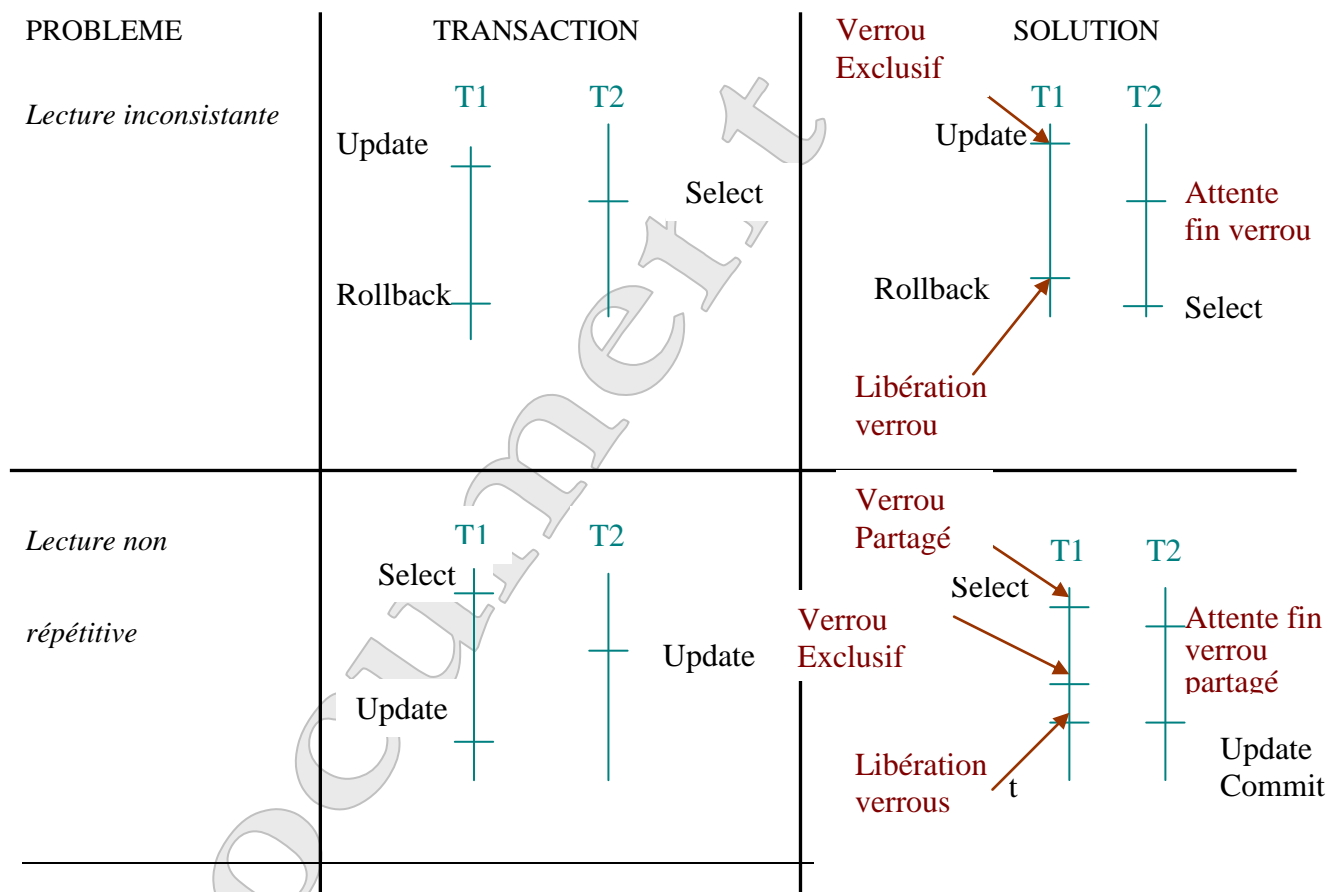
- Ce problème fondamental de l'accès concurrentiel aux données est géré au moyen d'un mécanisme de verrouillage, des lignes ou des tables entières (lock).
- Ce mécanisme permet de bloquer l'accès aux lignes ou aux tables concernées soit automatiquement (ORACLE, SQL Server), soit sur commande explicite (DB2, RDB). L'accès à une ligne ou une table verrouillées peut être traité soit par attente du déblocage (mode WAIT) soit par ROLLBACK de la transaction (mode NOWAIT) : le déblocage est effectif dès la fin de la transaction.

Reprise de l'exemple précédent avec transaction et verrouillage de la table Comptes

Application A : Demande un débit de 1000 F	Application B : Demande un débit de 500 F
(1) Début transaction avec verrouillage exclusif de la table Comptes (passant)	(2) Début transaction avec verrouillage exclusif de la table Comptes (bloqué)
(3) Test du solde OK : $1200 > 1000$	En attente
(4) Débit de 1000 F \Rightarrow solde = 200 F	En attente
(5) Fin transaction \Rightarrow déverrouillage de la table Comptes	(6) Test du solde : $200 < 500 \Rightarrow$ débit refusé
	(7) Fin transaction \Rightarrow déverrouillage de la table Comptes

Les niveaux d'isolation dans les transactions

- La norme SQL2 ne prévoit pas de demande explicite des verrous en début de transaction : les programmeurs risquant d'oublier de réserver les verrous, la norme considère que la prise automatique de verrous par le SGBD est préférable. Le SGBD choisit automatiquement les verrous à poser sur les tables et les lignes en fonction des requêtes reçues :
- **SELECT** : le SGBD pose un « verrou partagé », c'est-à-dire un verrou qui autorise les autres lecteurs à consulter la table, mais qui bloque les demandes d'écriture (INSERT, DELETE, UPDATE)
- **INSERT, DELETE, UPDATE** : le SGBD pose un « verrou exclusif » qui interdit tout autre opération sur l'objet, y compris les lectures



- Le SGBD pose des verrous différents selon le « niveau d'isolation des transactions » :

1. Read Uncommitted : Aucun verrou. On peut lire les mises à jour dans d'autres applications, même si elles n'ont pas encore été validées par un COMMIT. Fortement déconseillé !

2. Read Committed (défaut) : verrou exclusif sur les lignes modifiées. Le SGBD cache les modifications en cours aux autres utilisateurs, tant qu'elles n'ont pas été validées par COMMIT. Mais il ne pose pas de verrous sur le SELECT : dans une même transaction, les lectures ne sont pas

nécessairement « répétitives », les données pouvant être modifiées par une autre transaction entre des lectures qui se suivent.

3. Repeatable Read (conseillé) : corrige le problème précédent des lectures non répétitives, en posant un verrou partagé sur les lignes lues, en plus du verrou exclusif sur les lignes modifiées. Les lignes ne pourront donc pas être modifiées entre le premier SELECT et la fin de la transaction.

4. Serializable : le SGBD prend un verrou de table sur toute ressource utilisée

- En résumé, le SGBD manipule quatre sortes de verrous, selon les opérations et le niveau d'isolation demandés :

	Verrou partagé	Verrou exclusif
Verrou de pages (lignes)	Lecture dans le cas 3	modification dans les cas 2 et 3
Verrou de tables	Lecture dans le cas 4	modification dans le cas 4

- Verrou de pages : en pratique, un SGBD ne pose jamais un verrou sur une seule ligne, mais sur la page qui contient cette ligne.



Norme SQL 2 sur les niveaux d'isolation : p. 162

Mise en œuvre pratique en Transact SQL

- Par défaut SQL Server travaille dans un niveau intermédiaire entre « Read UnCommitted » et « Read Committed » : il pose des verrous exclusifs sur les pages modifiées; il pose des verrous partagés sur les pages lues (comme dans le cas « Repeatable Read ») mais il les libère dès la fin du SELECT. Les verrous partagés sont donc bloquants si une modification est en cours dans une autre transaction, mais ils n'empêchent pas les autres de modifier les données qui viennent d'être lues.
- Trois options pour modifier le comportement des verrous : **HOLDLOCK**, **TABLOCK** et **TABLOCKX**

HOLDLOCK maintient le verrou jusqu'au COMMIT (au lieu de le relâcher après le SELECT). Il interdit donc toute modification externe, et garantit la cohérence de toutes les lectures

TABLOCK pose un verrou partagé de table, au lieu d'un verrou de page

TABLOCKX pose un verrou exclusif de table

Exemple de transaction bancaire qui garantit la cohérence du solde (abandon si solde insuffisant) et la « répétitivité » des différentes lectures (solde non modifié par une autre transaction entre deux lectures)

-- fixer le mode d'isolation

set transaction isolation level SERIALIZABLE

-- début de la transaction

begin transaction

-- lecture de la table Compte avec verrouillage de la table en mode exclusif : écriture et lecture interdites jusqu'au COMMIT

Select solde From Comptes With (HOLDLOCK TABLOCKX) Where num = 1

-- modification de la table Comptes

Update Comptes Set solde = solde – 500 Where num = 1

-- lecture de vérification, pour trouver le nouveau solde

Select solde From Comptes Where num = 1

-- fin transaction

commit transaction

Attention aux « deadlocks »

L'utilisation des verrous peut conduire à des "deadlock": situation où deux transactions ou plus peuvent se bloquer mutuellement.

<i>Application A</i>	<i>Application B</i>
begin transaction	begin transaction
Select solde From Comptes with (HOLDLOCK) Where num = 1 verrou partagé, PASSANT	
	Select solde From Comptes with (HOLDLOCK) Where num = 1 verrou partagé, PASSANT
Update Comptes Set solde = solde – 500 Where num = 1 Demande de verrou exclusif, BLOQUANT (sur verrou partagé de B)	
	Update Comptes Set solde = solde – 500 Where num = 1 Demande de verrou exclusif, BLOQUANT (sur verrou partagé de A) => DEADLOCK



Exercice 59 : provoquer un DEADLOCK

Lancer deux fois le script SQL fourni Deadlock.sql sous deux connexions différentes.

Exécuter uniquement le **SELECT** sous chacune des connexions. Puis exécuter le **UPDATE** sous chacune des connexions. Que se passe-t-il ? Comment le SGBD sort-il du deadlock ?

Même exercice en observant les verrous qui sont posés par le SGBD. Sous Management studio, connectez vous sous le compte « Développeur » qui vous donne des droits suffisants pour observer l'état du système. Dans « Gestion / Moniteur d'activité », vous trouverez des informations sur les processus en cours, et les verrous posés sur les différents objets de la base.

Modifier le script avec l'option **TABLOCKX** qui évite le Deadlock en bloquant le deuxième processus dès le **SELECT** (=> c'est la solution à retenir)

Microsoft SQL Server Management Studio Express

Explorateur d'objets

Moniteur d'activité - serveur-sql

11/07/2007 15:41:14

Prochaine actualisation : Manuel

Afficher les paramètres d'actualisation

Filtre : Appliqué

Afficher les paramètres du filtre

Connexion

Serveur : serveur-sql

Connexion : sa

Afficher les propriétés de connexion

Progression

Terminé

9 éléments affichés sur un total de 29 éléments.

ID de proces...	Process...	Utilisateur	Base de données	État	Bloqué par	Blocage	Ouvrir
51	non	SPECIFI...	msdb	sleeping	0	0	0
52	non	SPECIFI...	msdb	sleeping	0	0	0
53	non	SPECIFI...	msdb	sleeping	0	0	0
54	non	sa	master	sleeping	0	0	0
55	non	garraud	GarraudSchemas	sleeping	0	0	0
57	non	sa	tempdb	runnable	0	0	2
58	non	garraud	banqueGarraud	sleeping	0	1	1
60	non	sa	master	sleeping	0	0	0
61	non	garraud	banqueGarraud	suspended	58	0	2

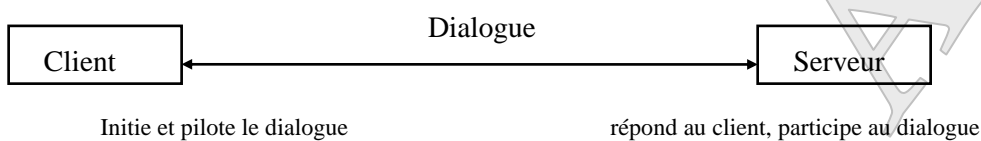
Le processus 61 est bloqués par le processus 58

Fermer

5. LE MODELE CLIENT/SERVEUR

5.1 Introduction

- A la base du dialogue, il y a un besoin du client : le client est l'application qui débute la "conversation", afin d'obtenir de la part du serveur (le programme "répondant") des données ou un résultat.
- Le serveur est un programme générique dont la seule fonction est de répondre aux demandes des applications clientes, en fournissant en retour les données ou le résultat. Les demandes des applications sont appelées généralement des requêtes. Le serveur est, par nature, indépendant des applications qui l'interrogent.



- Est conforme au modèle client-serveur, une application qui fait appel à des services distants par des requêtes, plutôt que par un échange de fichiers. Ce modèle exige que les ordinateurs soient reliés en réseau, et repose sur un protocole qui assure le bon fonctionnement des échanges :
 - au niveau du client : émission de requêtes (les appels de service)
 - au niveau du serveur : émission des réponses ou résultats

5.2 Les principes de base

Comment peut-on définir un service?

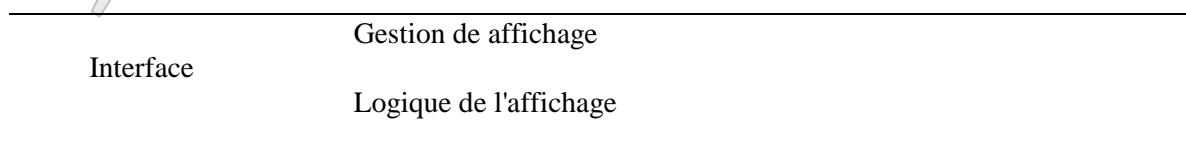
- La notion de service doit être comprise à travers la notion de traitement. Le modèle client-serveur permet de répartir les services utilisés par une application et non l'application elle-même.
- Les traitements auxquels font appel les applications, requièrent souvent beaucoup de ressources machine : il est donc préférable, dans un souci de performance et d'efficacité, de les faire résider chacun sur un ordinateur dédié, appelé serveur.

Comment peut-on distinguer le cœur d'une application et les services annexes de celle-ci que l'on doit déporter?

Une application informatique se compose fondamentalement de 3 niveaux :

- * l'interface avec l'utilisateur,
- * les traitements,
- * les données.

Ces niveaux sont eux-mêmes composés de couches :



Traitements	Logique fonctionnelle
	Exécution des procédures
Données	Intégrité des données
	Gestion des données

Quel est le rôle des différentes couches?

- * La gestion de l'affichage : est assurée par l'environnement d'exploitation (ex: Windows),
- * La logique de l'affichage : transmet à la gestion de l'affichage les directives de présentation (ex : les feuilles et les contrôles graphiques dans une application Visual Basic sous Windows),
- * La logique fonctionnelle : représente l'arborescence algorithmique de l'application et aiguille le traitement vers les procédures qui sont déroulées et exécutées dans la couche suivante (ex : les fonctions événementielles en Visual Basic associées aux messages souris, clavier..)
- * L'exécution des procédures : effectue les traitements de l'application (ex : des ordres SQL lancés par les fonctions événementielles, ou mieux des appels de procédures stockées),
- * L'intégrité des données : vérifie l'utilisation cohérente des données par les différents processus,
- * La gestion des données : permet la manipulation des données (sélection ou mise à jour des données).

5.3 Le dialogue entre client et serveur : l'IPC

Le dialogue s'effectue à travers le réseau qui relie le client et le serveur :

- Une conversation de type client-serveur est un dialogue inter-processus qui s'appuie de part et d'autre sur 2 interfaces de niveau système. Ce dialogue s'appelle un IPC (Inter Processus Communication) ou encore Middleware.
- L'IPC permet l'établissement et le maintien du dialogue. C'est un ensemble de couches logicielles qui :
 - prend en compte les requêtes de l'application cliente,
 - les transmet de façon transparente à travers le réseau,
 - retourne les données ou les résultats à l'application cliente.
- L'IPC est constitué par :
 - des API (Application Programming Interface) ou interface de programmation au niveau applicatif : l'API est une interface entre un programme et un système. L'API encapsule des fonctions qui vont permettre à l'application de faire appel aux services proposés par le serveur (ex : les WinSockets pour faire de la communication IP sous Windows) ;
 - des FAP (Format And Protocols) ou protocoles de communication et format des données : ce sont des procédures chargées d'assurer la présentation, les conversions de formats et de codification.

Application
Interface de programmation (API)
Protocole de communication et format (FAP)
Protocole de transport lié au réseau

- L'application ne voit que l'API, l'API n'est en contact qu'avec la FAP et la FAP fait la liaison avec les couches réseau.
- L'IPC est l'élément fondamental du fonctionnement client-serveur. Deux approches d'IPC existent :
 - Les RPC (Remote Procédure Call ou appels de procédure à distance) pour les dialogues sans connexion ;
 - Par Messages : il faut alors établir une session de communication entre Client et Serveur, et gérer cette session.

5.4 Les mécanismes utilisés par les interfaces de communication

Les RPC

- Il s'agit de communication par échange synchrone : chacun reste en attente de l'autre et ne peut rien faire pendant ce temps. La fiabilité est médiocre : en cas d'incident, il n'y a pas de reprise. Mais la mise en œuvre est simple.

Client	Serveur
Appel de la procédure serveur et Attente de la réponse	→
	prise en compte de la requête
	réveil du serveur
	exécution de la procédure
	émission d'un message réponse unique
	←
Réception du résultat et reprise de l'exécution	

Les messages

- Le Client et le Serveur effectuent un échange de messages. Le dialogue est asynchrone. C'est la demande de connexion du programme client qui est le point de départ du dialogue. Ce mode de communication représente un "investissement" important, et nécessite un bon niveau d'expertise. Il n'est justifié que pour des échanges intensifs.

Client	Serveur
Demande de connexion: envoi d'un message	

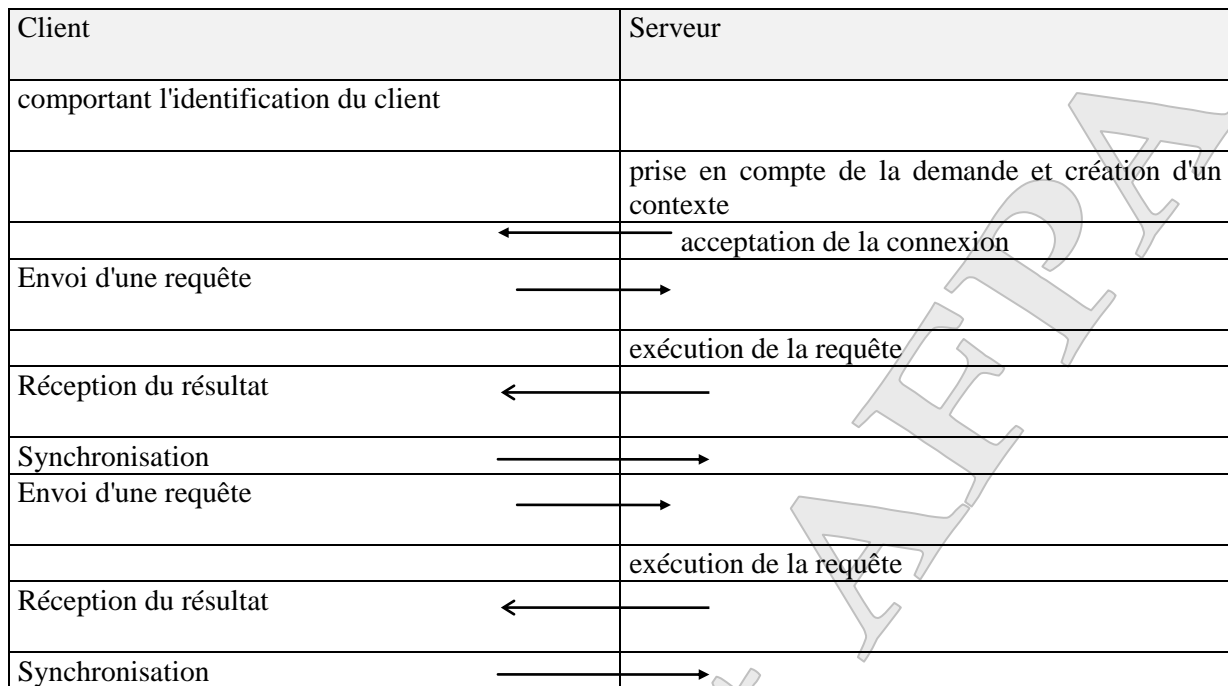


Schéma en couches d'un échange client-serveur :

Application	API
Présentation	FAP
Session	FAP
Transport	TCP
Réseau	IP
Liaison	Ethernet, Token Ring
Physique	Coaxial...

- Le **protocole de communication** est contenu dans la couche FAP de l'IPC. Il gère l'ordonnement de l'échange :
 - ouvrir une session,
 - envoyer une requête,
 - faire remonter les résultats.
- Le **protocole de transport** prend les messages émis par les applications pour les insérer dans une trame, qui va circuler sur le réseau. Ex : TCP/IP

5.5 Les différentes mises en oeuvre du modèle

- Le type de relation Client-Serveur est déterminé par l'activité du site serveur, c'est à dire le service "déporté" du cœur de l'application. Selon la répartition évoquée entre Données, Traitements et Présentation, on peut distinguer plusieurs types de liaison Client-Serveur :
 - le client-serveur de présentation,
 - le client-serveur de données,
 - le client-serveur de procédures.
- Ces types ne sont pas exclusifs les uns des autres.

Le Client-Serveur de présentation

C'est le module Gestion de l'affichage, déporté sur un système spécialisé, qui est le service. Ceci suppose de pouvoir séparer la gestion de l'affichage et la logique de l'affichage : par exemple sous X-Windows. Windows est un système monolithique qui ne permet pas ce découpage.

Client	Serveur
logique de l'affichage	gestion de l'affichage
logique fonctionnelle	
Exécution des procédures	
intégrité des données	
gestion des données	

Le client-Serveur de données

C'est le module de gestion des données qui est déporté sur un serveur dédié ; on peut également déporter le module d'intégrité des données. Ex : les SGBDR.

Client	serveur
Gestion de l'affichage	intégrité des données
Logique de l'affichage	gestion des données
Logique fonctionnelle	
Exécution des procédures	

Le Client-Serveur de procédure

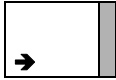
- Ce type requiert plus de savoir faire pour sa mise en œuvre : il s'agit là de déporter vers le serveur une partie des traitements, sans toucher à la localisation de la Gestion des données et de la présentation.
- La logique fonctionnelle de l'application réside toujours sur la station cliente. Cette partie logique effectue des appels ponctuels à des traitements ou services de la station serveur. Ces services sont soit des applicatifs écrits dans des langages classiques, soit des procédures cataloguées toutes prêtes, écrites en SQL (ex : « procédures stockées » appelées en Visual Basic).

Client	Serveur
Gestion de l'affichage	exécution des procédures
Logique de l'affichage	
Logique fonctionnelle	
Intégrité des données	
Gestion des données	

5.6 Conclusion

- Le Client-Serveur représente la solution pour aller vers une utilisation uniforme des moyens informatiques.

- Si le Client-Serveur ne permet pas de faire baisser les couts de projets, les avantages techniques apportés par le modèle client-serveur se traduisent par des atouts "stratégiques":
 - une approche "évolutive" de la rénovation des systèmes d'information,
 - un meilleur parti des ressources matérielles éparpillées dans l'entreprise,
 - la prise en compte des nouvelles technologies.



Pour une présentation détaillée des techniques Client/Serveur, lire la documentation « Le Client/Serveur » de Marc Fayolle.

6. LE CLIENT/SERVEUR AVEC ODBC ET ACCESS

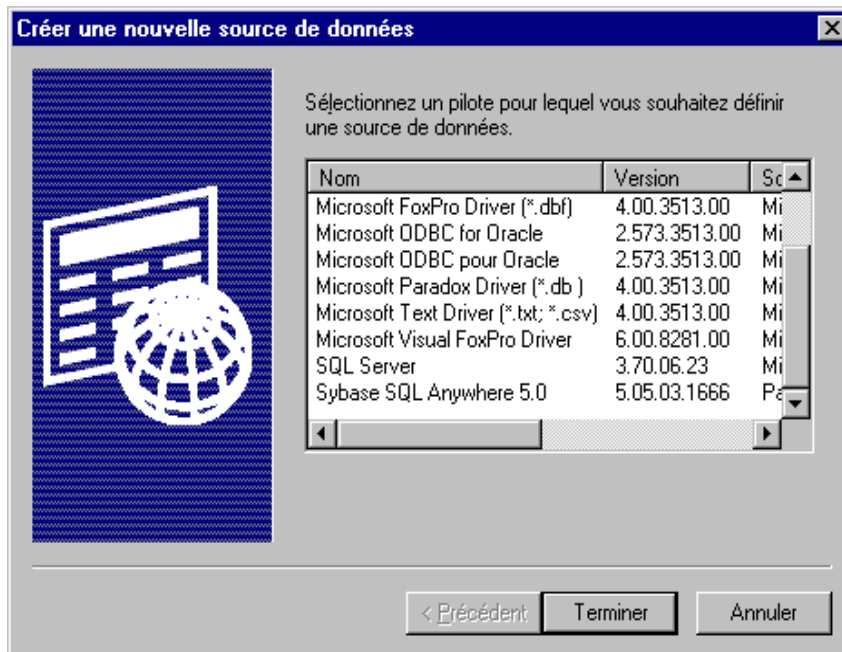
- Dans les chapitres qui précèdent, nous avons déjà utilisé deux clients, sans nous préoccuper de la manière dont ils communiquent avec le serveur de bases de données SQL SERVER : « SQL Server Enterprise Manager », et « Analyseur de Requêtes ». Ces clients propriétaires Microsoft gèrent leur propre connexion au serveur, et ne peuvent pas se connecter à des serveurs non Microsoft.
- Au contraire, l'interface ODBC (Open Database Connectivity) permet à de nombreux clients de se connecter vers des serveurs Microsoft, Oracle, Informix... Cette norme de fait, qui a été créée par Microsoft est aujourd'hui reconnue par la plupart des serveurs de base de données.
- Ce chapitre montre comment configurer une source ODBC entre un client et un serveur, et utiliser ACCESS comme client, en liant les tables ACCESS à une base distante sous SQL SERVER : il ne demande pas de connaissances particulières sur ACCESS. Dans la suite du module, nous verrons comment créer nos propres clients, en VISUAL BASIC avec la nouvelle interface ADO (ActiveX Data Object) qui encapsule ODBC
- En annexe, vous trouverez des exemples de programmes client écrits en langage C, utilisant l'API ODBC

6.1 Création d'une source de donnée ODBC

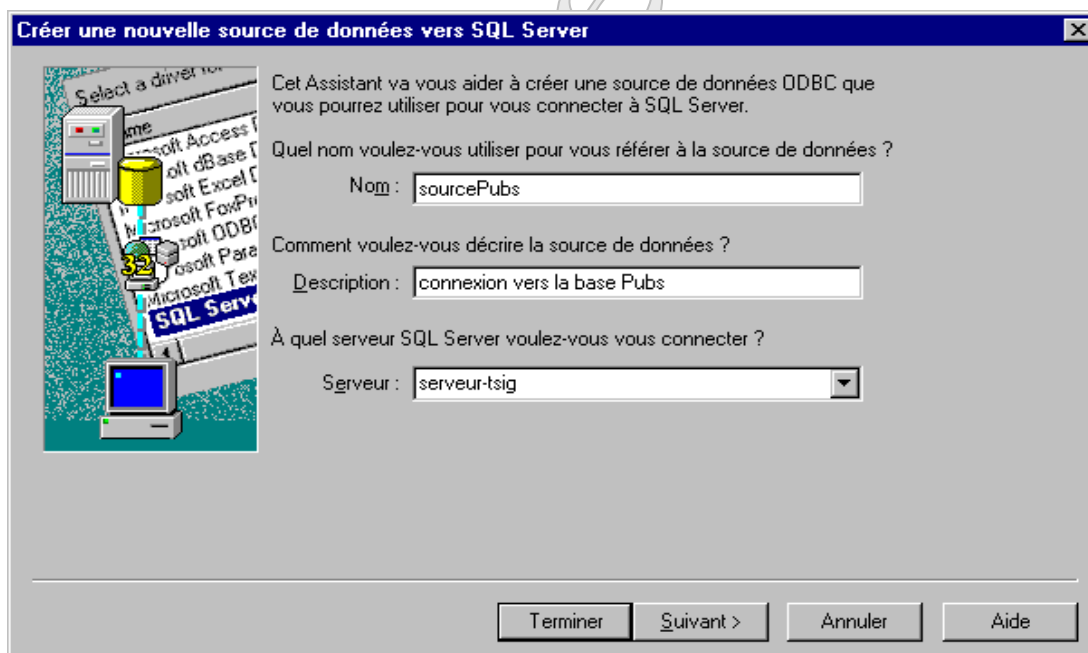
- Choisissez ODBC/32 dans le panneau de configuration



- Cliquez sur Ajouter (une source de données utilisateur)

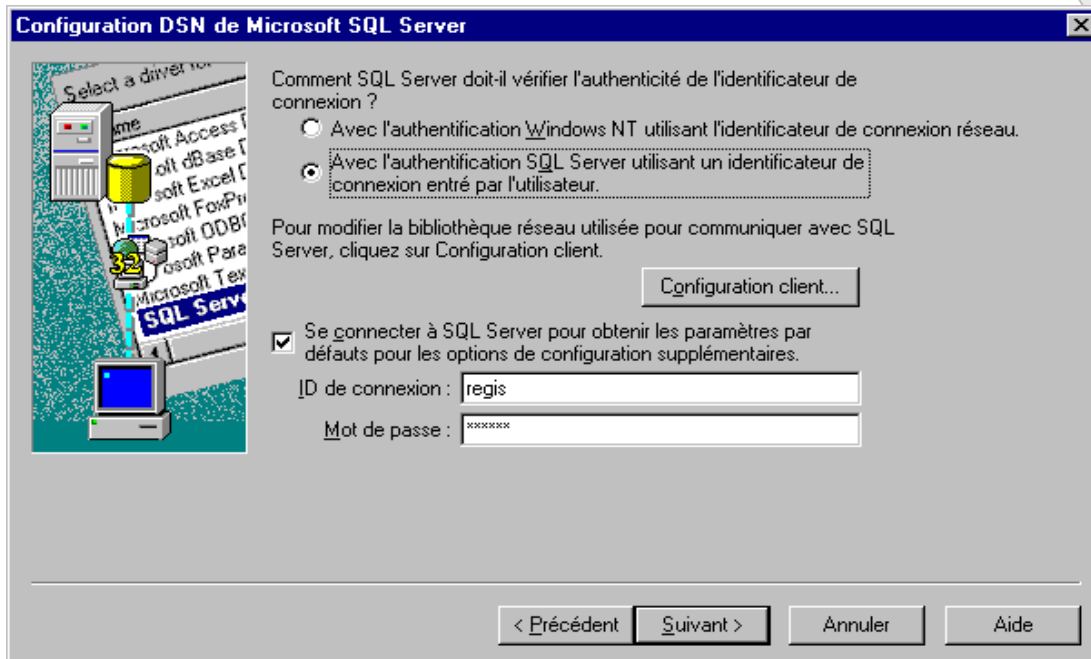


- Dans la fenêtre « Créer une nouvelle source de données », choisir « SQL SERVER », puis « Terminer »

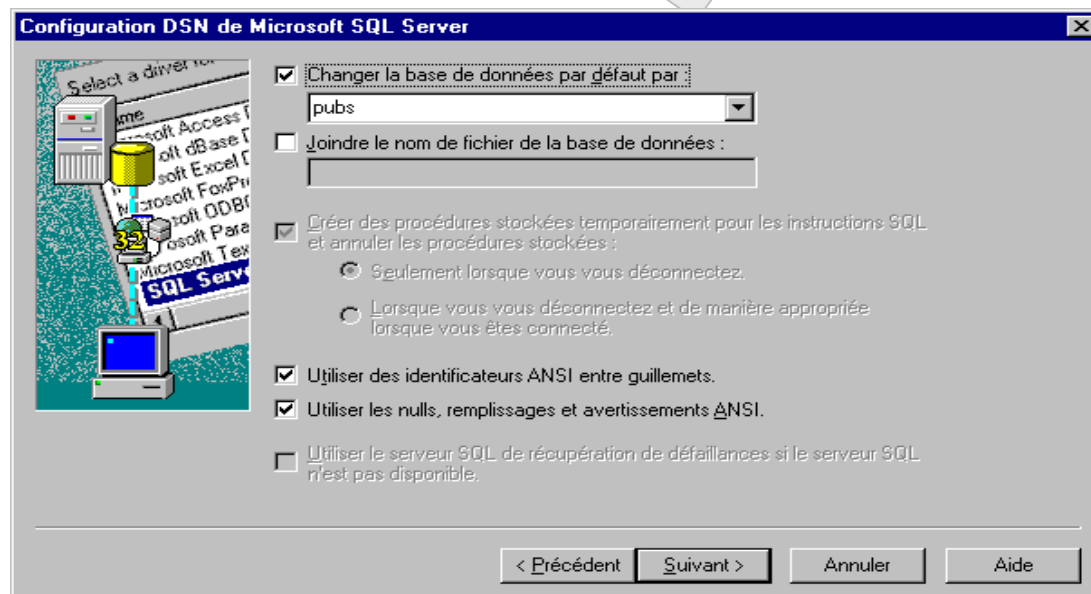


- Nom de la source de données : nom de votre choix, qui référencera la base de données sur votre PC client (ex : *sourcePubs*)
- Description : commentaire libre
- Serveur : nom de la machine, où est installée la base de données (ex : *serveur-tsig*)

- Dans la fenêtre suivante : choisissez l'authentification par SQL Server, et indiquer votre compte d'accès et mot de passe.

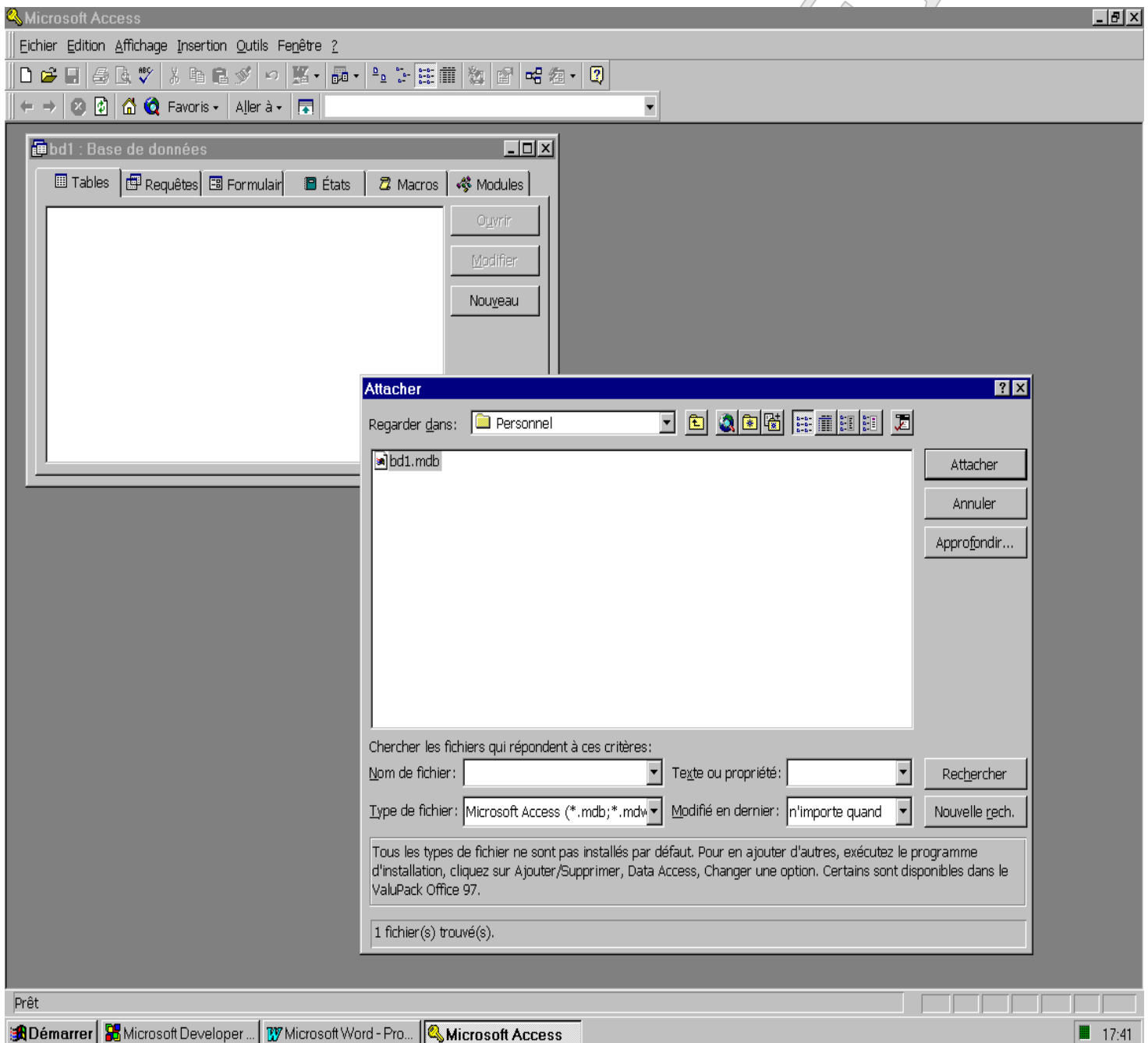


- Choisir le nom de la base de données sur le serveur (*pubs*)

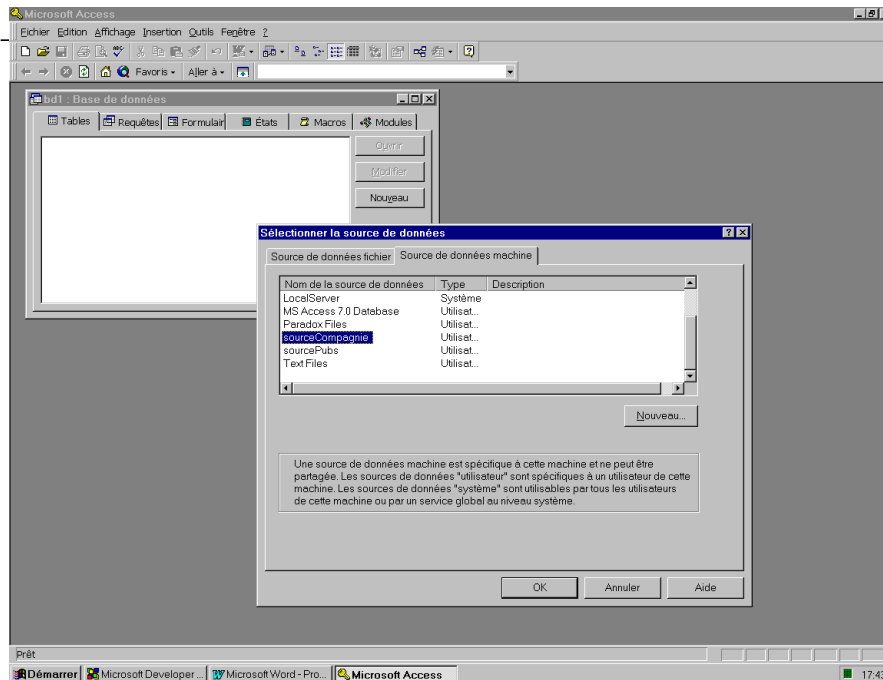


6.2 Connexion à une source de données sous Access

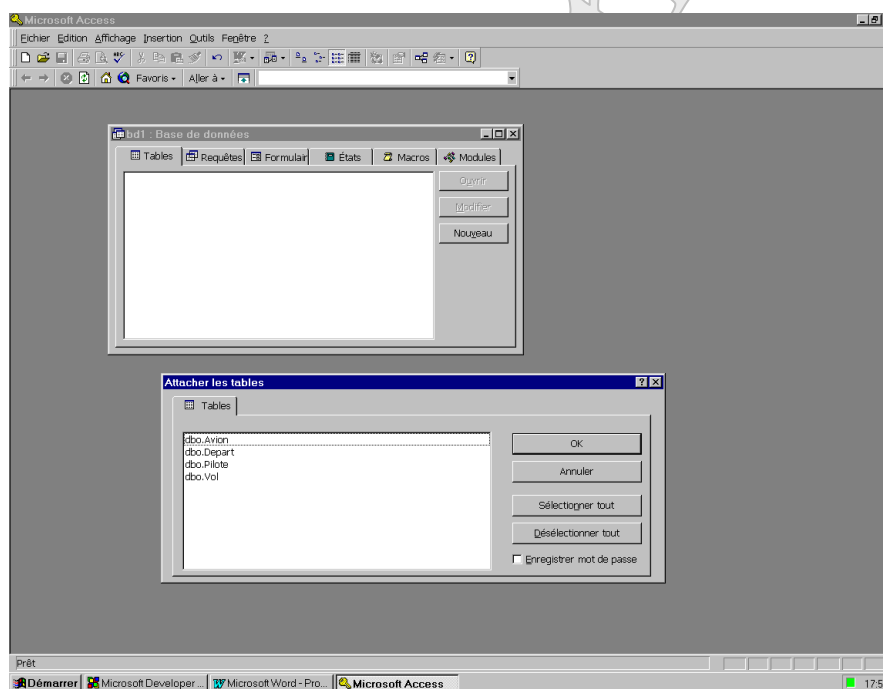
- Créer une nouvelle base vide : <CTRL N>, OK pour « Base de données vide », puis « Créer »
- Menu Insertion : Table
- Dans la fenêtre « Nouvelle Table », choisir « Attacher la table »



- Dans le menu « Type de fichier », choisir « Bases de données ODBC () »



- Sélectionner l'onglet « Source de données machine » et la source de données que vous recherchez dans la fenêtre : choisir la source « sourceCompagnie », après l'avoir créée par la méthode indiquée dans ce qui précède.
- Dans le menu « Connexion à SQL SERVER », loguez-vous sous le compte du propriétaire de la base de données.



- Sélectionner les tables seulement : Avion, Pilote, Vol, Affectation...
- Vérifier que vous pouvez effectuer des opérations licites de consultation, de création, de suppression de Nuples et que les erreurs dues aux contraintes sur les tables SQL SERVER sont bien rapatriées vers ACCESS : doublons, pages permises, contraintes référentielles...
- Consulter la définition des tables (*avec l'icône Equerre Bleu*), en lecture seulement, sous ACCESS et comparez le contenu des colonnes par rapport à SQL Enterprise Manager : le type des colonnes, les contraintes de clé primaire, d'index, les colonnes obligatoires (NULL interdit) sont connues d'ACCESS, mais pas les autres contraintes de plage, de clé étrangères, qui sont évaluées uniquement par le serveur.

ANNEXES

Document AFPA

1. GLOSSAIRE

Alias : Nom temporaire donné à une table ou à un attribut du résultat d'une requête. Un alias dure le temps de l'exécution de la requête, et n'a de sens que dans la requête où il est défini.

Attribut (*attribute*) : Chaque colonne d'une relation. Les attributs d'une relation doivent être uniques et sont normalement non ordonnés. Un attribut est toujours au moins caractérisé par son type et sa longueur. L'ensemble des valeurs que peut prendre un attribut est le domaine de l'attribut. Le degré d'une relation est le nombre d'attributs de la relation.

Base de données (*Data Base*) : Ensemble logique de données constitué de tables ou de fichiers avec leur index, et d'un dictionnaire centralisé permettant une gestion efficace des données.

Base de données relationnelle (*Relational Data Base*) : Base de données perçue par ses utilisateurs comme une collection de tables. Ces tables sont manipulées par un langage d'accès de quatrième génération, ou en SQL. L'utilisateur précise ce qu'il cherche dans la base et non comment il faut l'obtenir. Dans une base de données relationnelle, le SGBDR choisit le meilleur chemin d'accès pour optimiser l'utilisation des ressources (mémoire, disque, temps).

Champ (*Field*) : Synonyme d'attribut mais s'applique plutôt pour un enregistrement dans un système de gestion de fichiers classique (aussi appelé zone).

Clause (*Clause*) : Mot réservé du langage SQL utilisé pour spécifier ce qu'une commande doit faire, et souvent suivi d'une valeur. Dans la commande :

`'SELECT DISTINCT clt_nom FROM clients'`

les mots SELECT, DISTINCT et FROM sont des clauses.

Clé (*Key*) : Attribut ou combinaison d'attributs utilisé pour accéder à une table sur un critère particulier. Un index est construit sur une clé.

Clé étrangère (*Foreign Key*) : Attribut ou groupe d'attributs d'une relation, clé primaire dans une autre relation, et constituant un lien privilégié entre relations.

Clé primaire (*Primary Key*) : Clé donnant accès de façon univoque, pour chacune de ses valeurs, à un tuple de la table, (on dit parfois aussi « identifiant »).

Clé secondaire (*Secondary Key*) : Clé donnant accès pour chacune de ses valeurs, à un ensemble de tuples de la table, ceux-ci sont alors accédés en séquence.

Cluster (*Cluster*) : Voir Groupement (de tables).

Colonne (*Column*) : Voir attribut.

Corrélée (sous-requête) (*Correlated query*) : Sous-requête exécutée de façon répétitive, pour chaque tuple sélectionné par la requête extérieure. A chaque exécution de la sous-requête, le(s) tuple(s) traité(s) dépend(ent) de la valeur de un ou plusieurs attributs du tuple sélectionné dans la requête extérieure.

Curseur (*Cursor*) : En SQL intégré, sorte de pointeur vers un tuple particulier de la table temporaire contenant le résultat d'une requête. Le curseur fait le lien entre SQL, langage ensembliste, et le langage hôte (C, COBOL,...) qui ne traite qu'un tuple à la fois.

Dictionnaire de données (*Data Dictionary*) : Ensemble de tables et de vues contenant toutes les informations sur la structure d'une base de données. Le dictionnaire est utilisé à chaque requête pour vérifier la légitimité et la cohérence de la commande.

Domaine (*Domain*) : Ensemble des valeurs que peut prendre un attribut.

Fonction de groupe (*Group Function*) : Fonction mathématique s'appliquant sur toutes les valeurs d'un attribut ou d'une expression combinant plusieurs attributs, dans les tuples des groupes générés par une requête. La fonction calcule une seule valeur pour chaque groupe.

Groupement (de tables) (*cluster*) : groupe de deux ou plusieurs tables ayant des valeurs d'attributs identiques, et partageant le même espace disque. Utilisé pour accélérer les requêtes multi-tables et pour diminuer l'espace occupé.

Index (*Index*) : Objet d'une base de données permettant de trouver un tuple ou un ensemble de tuples dans une table sans examiner la table entièrement. L'index possède une entrée pour chaque valeur de la clé associée, couplée à un (des) pointeur(s) vers le(s) tuple(s) qui ont cette valeur.

Intégrité d'entité (ou de relation) (*Integrity*) : Règle obligatoire dans une base de données relationnelle, spécifiant que les valeurs d'une clé primaire doivent être uniques et non NULL.

Intégrité de référence (ou référentielle) (*Referential Integrity*) : Règle de cohérence hautement souhaitée dans une base de données relationnelle, obligeant à vérifier la

correspondance entre la valeur d'une clé étrangère et celle de la clé primaire associée. Cette règle doit être vérifiée à chaque mise à jour de la base.

Jointure (*join*) : Opération relationnelle couramment utilisée en SQL pour retrouver simultanément des parties de tuples dans plusieurs tables. On associe à cette jointure une condition (dite de jointure) spécifiant quels tuples des différentes tables doivent être associés.

Jointure 'externe' : Voir 'outer join'.

Ligne (d'une table) (*row*) : Voir tuple.

'Outer join' : Jointure incluant tous les tuples d'une table, même si l'autre table ne comprend aucun tuple satisfaisant la condition de jointure. Lorsque la condition de jointure ne peut être respectée, des valeurs NULL sont insérées pour les attributs manquants.

NULL : Valeur particulière d'un attribut, qui signifie "n'a pas de valeur" ou "valeur non applicable". Dans le premier cas, on ne connaît pas la valeur au moment de l'encodage, et dans le second, l'attribut n'a pas de sens pour le tuple concerné. Une requête peut sélectionner ou ignorer les tuples avec valeurs NULL. La valeur NULL se distingue de "blanc" et zéro.

Propriété (*Property*) : Voir attribut. S'emploie en conception de bases de données, notamment dans les méthodes MERISE, AXIAL, pour désigner les divers éléments d'une entité (ou d'une relation).

Privilège (*Privilege*) : Chacune des opérations qu'un utilisateur peut être autorisé à effectuer sur un objet de la base ; le privilège est accordé par le créateur de l'objet.

Relation (*Relation*) : Ensemble non ordonné de tuples. Une relation contient toujours une clé primaire. Chaque tuple est unique et est accessible de façon univoque par une valeur de la clé primaire. Une relation est toujours représentée au niveau conceptuel, donc en SQL, comme une table à deux dimensions, composées de lignes constituées d'attributs. Pour être manipulable par SQL, une relation doit être en "première forme normale" (ou "normalisée"), c'est à dire que chaque attribut doit avoir une valeur atomique.

SGBDR (*RDBMS*) : Voir "Système de gestion de bases de données relationnelles".

Sous-requête ou **requête imbriquée** (*Subquery*) : Requête entourée de parenthèses apparaissant dans une autre requête (appelée requête extérieure), et permettant de conditionner la sélection des tuples de cette dernière.

Synonyme (*Synonym*) : Nom donné à une table ou une vue dans le but de simplifier les références à cet objet. Un synonyme est personnel et persiste jusqu'à ce que son créateur le détruise. Le synonyme est d'application dans toutes les requêtes où l'objet intervient. A ne pas confondre avec 'alias'.

Système de gestion de bases de données relationnelles : ensemble de programmes permettant la mémorisation, la manipulation et le contrôle des données dans une ou plusieurs bases de données relationnelles. Outre le noyau relationnel qui gère les tables et optimise les requêtes, un SGBDR possède un langage d'exploitation de quatrième génération ainsi que divers outils (générateurs d'écrans, de documents, d'applications,...) favorisant le développement rapide d'applications. CODD et DATE ont défini un ensemble de règles strictes permettant de déterminer le niveau de 'relationnalité' d'un SGBDR.

Table (*Table*) : Voir relation.

Transaction (*Transaction*) : Ensemble logique d'événements et d'actions qui modifient le contenu d'une base de données, faisant passer la base d'un état cohérent à un autre état cohérent.

Tuple (*Tuple*) : Une ligne dans une relation (ou table). Un tuple est constitué de valeurs correspondant aux attributs qui constituent la relation. La cardinalité d'une relation est le nombre de tuples de la relation.

Vue (*View*) : Objet d'une base de données relationnelle et du langage SQL qui peut généralement être utilisé comme une table. Une vue réalise un résumé, une simplification, ou une personnalisation de la base de données en regroupant des données sélectionnées de plusieurs tables.

2. Corrigés des exercices complémentaires

Use AirRegis
go

-- ajout d'une colonne Ville à la table Pilote

```
ALTER TABLE Pilote
ADD Ville TypeVille NULL
go
```

-- initialisation des Villes

```
UPDATE Pilote
SET ville = 'Bastia'
WHERE IdPilote = 1
```

...etc..

-- 1° Noms des pilotes qui habitent dans la ville de localisation d'un A320 (dans une ville desservie par un aéroport où est basé un A320)

```
SELECT NomPilote
FROM AVION av, AEROPORT ae, Pilote p
WHERE TypeAvion = 'A320'
AND BaseAeroport = IdAeroport
AND NomVilleDesservie = Ville
```

-- 2° Noms des pilotes planifiés pour des vols sur A320

```
SELECT Distinct NomPilote
FROM AVION av, AFFECTATION af, Pilote p
WHERE TypeAvion = 'A320'
AND af.NumAvion = av.NumAvion
AND af.IdPilote = p.IdPilote
```

-- 3° Noms des pilotes planifiés sur un A320, qui habitent dans la ville de localisation d'un A320

```
SELECT Distinct NomPilote
FROM AVION av, AEROPORT ae, Pilote p, AFFECTATION af
WHERE TypeAvion = 'A320'
AND BaseAeroport = IdAeroport
AND NomVilleDesservie = Ville
AND af.NumAvion = av.NumAvion
AND af.IdPilote = p.IdPilote
```

-- 4° Noms des pilotes planifiés sur A320, qui n'habitent pas dans la ville de localisation d'un A320

```
SELECT Distinct NomPilote
FROM AVION av, Pilote p, AFFECTATION af
WHERE TypeAvion = 'A320'
AND af.NumAvion = av.NumAvion
AND af.IdPilote = p.IdPilote
AND NomPilote not IN ( SELECT NomPilote
                        FROM AVION av, AEROPORT ae, Pilote p
                        WHERE TypeAvion = 'A320'
                        AND BaseAeroport = IdAeroport
                        AND NomVilleDesservie = Ville)
```

-- 5° Noms des pilotes planifiés sur A320 ou qui habitent la ville de localisation d'un A320

```

SELECT NomPilote
FROM AVION av, AEROPORT ae, Pilote p
WHERE TypeAvion = 'A320'
AND BaseAeroport = IdAeroport
AND NomVilleDesservie = Ville
UNION
SELECT Distinct NomPilote
FROM AVION av, Pilote p, AFFECTATION af
WHERE TypeAvion = 'A320'
AND af.NumAvion = av.NumAvion
AND af.IdPilote = p.IdPilote
ORDER BY NomPilote

```

-- 6° Pour chaque ville desservie, donner la moyenne des capacités des avions qui sont localisés dans un aéroport desservant cette ville

```

SELECT 'Ville ' = NomVilleDesservie, 'Moyenne des capacités' = AVG (Capacite),
'Minimum' = MIN (Capacite), 'Maximum' = MAX (Capacite)
FROM AEROPORT, AVION av, TYPE ty
WHERE IdAeroport = BaseAeroport
AND av.TypeAvion = ty.TypeAvion
Group By NomVilleDesservie

```

-- 6° Même question, limitée aux villes où sont localisés plus de deux avions

```

SELECT 'Ville ' = NomVilleDesservie, 'Moyenne des capacités' = AVG (Capacite),
'Minimum' = MIN (Capacite), 'Maximum' = MAX (Capacite)
FROM AEROPORT, AVION av, TYPE ty
WHERE IdAeroport = BaseAeroport
AND av.TypeAvion = ty.TypeAvion
Group By NomVilleDesservie
Having count(NumAvion) > 2

```

-- 7° Afficher les noms des pilotes qui ne pilotent que des A320

```

SELECT NomPilote
FROM Pilote p
WHERE IdPilote IN (SELECT IdPilote FROM AFFECTATION)
AND 'A320' = ALL (SELECT Distinct TypeAvion
From AFFECTATION af, AVION av
WHERE af.NumAvion = av.NumAvion
AND p.IdPilote = af.IdPilote)

```

-- 8° Afficher les noms des pilotes qui pilotent tous les A320 de la compagnie

```

SELECT NomPilote
FROM Pilote p
WHERE (SELECT count (NumAvion) FROM Avion WHERE TypeAvion = 'A320')
=
(SELECT count (DISTINCT af.NumAvion)
From AFFECTATION af, AVION av
WHERE af.NumAvion = av.NumAvion
AND p.IdPilote = af.IdPilote
AND TypeAvion = 'A320')

```


3. Les fonctions

SQL offre la possibilité d'intégrer dans les expressions des fonctions retournant :

- Une valeur dépendant du contenu de colonnes (fonction sur les colonnes)
- Une valeur dépendant d'opérandes (fonctions scalaires)

Exemple 1: Lister le salaire maximum du département E01

```
SELECT MAX (SALAIRE) FROM EMPLOYES  
WHERE WDEPT = 'E01'
```

Lorsqu'une fonction d'agrégation est exécutée, SQL effectue la synthèse des valeurs d'une colonne spécifique pour la table complète ou pour des groupes de lignes de la table (clause GROUP BY) : une valeur d'agrégation unique est alors générée pour la table complète ou pour chaque groupe de lignes.

Exemple 2: Lister le nombre de caractères du nom des employés.

```
SELECT LEN(NOM) FROM EMPLOYES
```

Les fonctions scalaires effectuent une opération sur une valeur unique et renvoie ensuite une valeur unique. Elles peuvent être utilisées pour autant que l'expression est valide.

a) Les fonctions d'agrégation

Les fonctions d'agrégation effectuent un calcul sur un ensemble de valeurs et renvoient une valeur unique. A l'exception de la fonction COUNT(*), les fonctions d'agrégation ignorent les valeurs NULL.

Pour les fonctions pour lesquelles ces valeurs sont précisées :

ALL applique la fonction d'agrégation à toutes les valeurs. (ALL est l'argument par défaut)

DISTINCT spécifie que la fonction doit seulement être appliquée à chaque instance unique d'une valeur, quel que soit le nombre d'occurrences de la valeur.

AVG ([ALL | DISTINCT] expr) : Calcul de la moyenne des valeurs de la collection de nombres précisés par l'expression entre parenthèses.

```
SELECT AVG(SALAIRE) FROM EMPLOYES
```

```
SELECT AVG(DISTINCT SALAIRE) FROM EMPLOYES
```

SUM ([ALL | DISTINCT] expr) : Somme des valeurs des nombres d'une colonne de type numérique

```
SELECT SUM(SALAIRE) FROM EMPLOYES WHERE WDEPT = 'E01'
```

MAX(expr) et MIN (expr) : Obtention de la valeur maximum (minimum) d'une collection de valeurs.

COUNT (*) : Dénombrement d'une collection de lignes

```
SELECT COUNT (*) FROM EMPLOYES
```

La fonction COUNT renvoie le nombre d'employés.

La fonction COUNT(*) ne requiert aucun paramètre et calcule le nombre de lignes de la table spécifiée sans supprimer les doublons. Elle compte chaque ligne séparément, même les lignes contenant des valeurs NULL.

COUNT ([ALL | DISTINCT] expr) : Dénombrement de toutes les expressions non nulles ou non nulles uniques

SELECT COUNT (DISTINCT WDEPT) FROM EMPLOYES

La fonction COUNT renvoie le nombre de département non nuls uniques.

COUNT_BIG ([ALL | DISTINCT] expr) : Idem count, mais la valeur de retour est au format bigint au lieu de int.

VAR (expr) : Renvoie la variance de toutes les valeurs de l'expression numérique donnée

STDEV(expr) : Renvoie l'écart type de toutes les valeurs de l'expression numérique donnée

b) Les fonctions scalaires mathématiques

Les fonctions scalaires effectuent une opération sur une valeur ou sur un nombre fixe de valeurs, et non sur une collection de valeurs.

*Les fonctions scalaires **mathématiques** effectuent un calcul, généralement basé sur les valeurs d'entrée transmises comme arguments et elles renvoient une valeur numérique*

Mathématiques:

ABS (expr) valeur absolue

CEILING (expr), **FLOOR**(expr) : Plus petit (grand) entier supérieur(inférieur) ou égal à *expr*

SIGN (expr) : Renvoie 1 si *expr* positive, -1 si *expr* est négative, 0 si nulle

SQRT (expr) racine carrée

POWER (expr,n) élève *expr* à la puissance n

SQUARE (expr) calcul le carré de *expr*

Trigonométriques :

SIN(expr), **COS**(expr), **TAN**(expr), **COTAN**(expr)

ASIN(expr), **ACOS**(expr), **ATAN**(expr)

PI() renvoie la valeur PI : 3.14159265358979

DEGREES(expr), **RADIANS**(expr) : Conversion de degrés (radians) en radians (degrés)

Logarithmiques :

LOG(expr), **EXP**(expr), **LOG10**(expr)

Diverses:

RAND (expr) rend un nombre aléatoire entre 0 et 1 , *expr* représente la valeur de départ

ROUND (expr, n) arrondit *expr* à la précision n

c) Les fonctions scalaires de chaîne

Les fonctions scalaires **de chaîne** effectuent une opération sur une valeur d'entrée de type chaîne et renvoient une valeur numérique ou de type chaîne.

CHAR (expr), NCHAR (expr) : Convertit *expr* en un caractère ASCII ou un caractère UNICODE

CHAR(65) donne A

ASCII (expr), UNICODE (expr) : Renvoie le code ASCII ou unicode de *expr*

ASCII (A) donne 65

LEN (expr) : Renvoie le nombre de caractères d'une expression de chaîne

LEN ('MARTIN') donne 6

LOWER (expr), UPPER (expr) : Renvoie *expr* après avoir transformé les caractères majuscules en caractères minuscules, ou minuscules en majuscules

LOWER ('MARTIN') donne martin

UPPER ('martin') donne MARTIN

LEFT (expr, n) RIGHT (expr, n) : Renvoie les *n* caractères les plus à gauche (droite) de *expr*.

LEFT ('MARTIN', 2) donne MA

RIGHT ('MARTIN', 2) donne IN

LTRIM (expr), RTRIM (expr) : Supprime les espaces à gauche (droite) de *expr*.

LTRIM (' MARTIN') donne MARTIN

RTRIM ('MARTIN ') donne MARTIN

SUBSTRING (expr, p, n) : Renvoie *n* caractères de la chaîne. *expr* à partir de la position spécifiée par *p*

SUBSTRING ('HUGO', 2, 3) donne 'UGO'

STUFF (expr1, p, n, expr2) : Supprime *n* caractères de *expr1* à partir d'une position donnée et insère *expr2*

STUFF ('ABCDEF', 2, 3, 'IJKLMN') renvoie AIJKLMNEF

REPLACE (expr1, expr2, expr3) : Remplace dans *expr1* toutes les occurrences de *expr2* par *expr3*

REPLACE ('ABCDAB', 'AB', 'IJK') renvoie IJKCDIJK

d) Les fonctions scalaires de date et heure

Les fonctions scalaires **de date et heure** effectuent une opération sur une valeur d'entrée de type date et heure et renvoient soit une valeur numérique, de type date ou heure, ou de type chaîne.

CURRENT_TIMESTAMP, GETDATE () : Renvoie la date et l'heure courante

DAY (expr), MONTH (expr), YEAR (expr) : Obtention d'un jour, mois ou année à partir de *expr*

DAY (CURRENT_TIMESTAMP) donne le jour courant

YEAR (CURRENT_TIMESTAMP) extrait l'année courante

Les fonctions **DAY, MONTH** et **YEAR** sont synonymes de

DATEPART (dd, date), DATEPART (mm, date) et DATEPART (yy, date), respectivement.

Dans les fonctions suivantes, le paramètre *partie_de_date* indique la partie de date à renvoyer suivant les abréviations :

Partie de date	Nom Complet	Abréviation
Année	Year	yy, yyyy
Quart	Quart	qq, q
Mois	Month	mm, m
Quantième	DayofYear	dy
Jour	Day	d, dd
Semaine	Week	wk, ww
Jour de la semaine	WeekDay	dw
Heure	Hour	hh
Minute	Minute	mi
Secondes	Second	ss, s
Millisecondes	Millisecond	ms

DATEPART (partie_de_date, date)

Renvoie un entier représentant l'élément de date précisé dans la date spécifiée.

DATEPART (mm, CURRENT_TIMESTAMP) donne le mois courant
équivalent à

DATEPART (month, CURRENT_TIMESTAMP)

DATEPART (mm, '21/06/2001') renvoie 6

DATEPART (dw, '21/06/2001') renvoie 4

DATENAME (partie_de_date, date)

Renvoie le nom de l'élément de date précisé dans la date spécifiée.

DATENAME (mm, '21/06/2001') renvoie 'juin'

DATENAME (dw, '21/06/2001') renvoie 'jeudi'

DATEADD (partie_de_date, nombre, date)

Renvoie une nouvelle valeur **datetime** calculée en ajoutant un intervalle à la date spécifiée.

DATEADD (dd, 15, '21/06/2001') renvoie '06/07/2001'

DATEADD (mm, 7, '21/06/2001') renvoie '21/01/2002'

DATEDIFF(partie_de_date, date_début, date_fin)

Renvoie un entier spécifiant le nombre d'intervalles compris entre date_début et date_fin.

DATEDIFF (dd, '21/06/2001', '01/07/2001') renvoie 10

DATEDIFF (mm, '21/06/2001', '01/01/2002') renvoie 7

Il est possible de configurer le premier jour de la semaine par l'intermédiaire de la fonction

SET DATEFIRST (numero_jour)

Les jours sont numérotés de 1 pour le lundi à 7 pour le dimanche ; il est possible de connaître la configuration en interrogeant la fonction @@datefirst.

e) Les fonctions de conversion

Conversion explicite d'une expression d'un type de données en une expression d'un type de données différent.

CAST et CONVERT offrent la même fonctionnalité

CAST (expression as type_de_donnée) : Permet de convertir une valeur dans le type spécifié.

CAST (DATEPART (hh, DATE) as char(2))

renvoie le nombre d'heures sous forme d'une chaîne de caractères de longueur 2, extraite d'un champ DATE de type datetime.

CONVERT (type_de_donnée, expr, style)

Permet de convertir l'expression dans le type spécifié, avec la possibilité d'utilisation d'un style (facultatif)

CONVERT (char, DATE, 8)

renvoie sous la forme d'une chaîne de caractères (hh :mm :ss) l'heure d'un champ DATE de type datetime avec application du style 8

Sans siècle (aa)	Avec siècle (aaaa)	Standard	Entrée/Sortie
1	101	États-Unis	mm/jj/aaaa
2	102	ANSI	aa.mm.jj
3	103	Anglais/Français	jj/mm/aa
4	104	Allemand	jj.mm.aa
5	105	Italien	jj-mm-aa
6	106 ⁽¹⁾	-	jj mois aa
7	107 ⁽¹⁾	-	mois jj, aa
8	108	-	hh:mm:ss

Extrait de styles (Voir aide en ligne pour plus d'informations)

f) Les fonctions de classement

La fonction ROW_NUMBER permet de numérotter séquentiellement chaque ligne rendue par une instruction Select, sur la base d'un critère donné, codé sur une clause Over.

```
SELECT ROW_NUMBER () OVER (ORDER BY SALAIRE DESC) AS NUMCOL,
PRENOM, NOM, SALAIRE FROM EMPLOYES
```

NUMCOL	PRENOM	NOM	SALAIRE
1	ANDRE	BOULIN	2100
2	JULIEN	DELMAS	1834
3	MARIE	VIALA	1650
4	ARTHUR	ZOLA	1600
5	PIERRE	DURAND	1530
6	ANNE	MARTIN	1500
7	PAUL	BALZAC	1200
8	JEAN	CAZENEUVE	

On ne peut malheureusement pas faire de la sélection sur la colonne obtenue (clause WHERE) pour limiter le nombre de lignes renvoyées, en une seule instruction. Une solution est de passer par l'intermédiaire d'une table temporaire, au moyen d'une procédure stockée.

RANK()

Avec la fonction RANK, le résultat obtenu est le même qu'avec ROW_NUMBER à la différence qu'une ligne peut se voir attribuer le même numéro (n) que la ligne qui la précède, la ligne suivante prenant le numéro (n+2).

dans l'exemple si deux salaires sont égaux

DENSE_RANK()

Le résultat obtenu est le même qu'avec RANK à la différence que la ligne suivant les 2 lignes dont le numéro de colonne est égal prendra le numéro (n+1).

4. Les éléments du langage TRANSACT-SQL

Les principaux éléments complémentaires de Transact-SQL sont les commentaires, les variables, les fonctions, et les instructions de contrôle du déroulement.

- **Les commentaires**

Les **commentaires en ligne** sont situés après deux tirets -- ;

Il est également possible de créer un **bloc de commentaires** en plaçant /* en début et */ fin de bloc

- **Les Variables locales**

On définit une variable locale à l'aide d'une instruction **DECLARE**, on lui affecte une valeur initiale à l'aide de l'instruction **SET** et on l'utilise dans le cadre de l'instruction, du lot ou de la procédure dans laquelle elle a été déclarée.

Le nom d'une variable locale commence toujours par @.

- **Les variables système**

TRANSACT-SQL propose de nombreuses variables qui renvoient des informations ; certaines d'entre elles demandant des paramètres en entrée.

Elles se distinguent des variables utilisateur par le double @ :

Quelques exemples ...

@@VERSION renvoie des informations sur les versions de SQL SERVER et de système d'exploitation utilisées. (SELECT @@VERSION)

@@TRANCOUNT renvoie le nombre de transactions ouvertes.

@@ROWCOUNT renvoie une valeur représentant le nombre de lignes affectées par la dernière requête effectuée.

@@ERROR contient le numéro de l'erreur de la dernière instruction TRANSACT-SQL exécutée.

- **Les éléments de contrôle du déroulement**

CASE évalue une liste de conditions et renvoie la valeur de l'expression de résultat correspondant à la condition sélectionnée.

Syntaxe

CASE expression_en_entrée

WHEN expression_cas_particulier **THEN** expression_resultat

 [...n]

[ELSE expression_resultat_dans_les_autres_cas]

END

La structure BEGIN ... END

C'est une structure de bloc permettant de délimiter une série d'instructions (utilisé avec IF et WHILE).

L'instruction RETURN

Elle permet d'afficher un message.

L'instruction PRINT

Elle permet de sortir d'une procédure en renvoyant éventuellement une variable entière.

L'instruction IF

C'est la structure alternative qui permet de tester une condition et d'exécuter une instruction si le test est vrai.

Syntaxe

IF condition

Instruction | bloc

[ELSE]

Instruction | bloc

L'instruction WHILE

C'est la structure répétitive qui permet d'exécuter une série d'instructions tant qu'une condition est vraie. L'exécution des instructions de la boucle peut être contrôlée par les instructions **BREAK** et **CONTINUE**.

L'instruction **BREAK** permet de sortir de la boucle.

CONTINUE permet de repartir à la première instruction de la boucle.

Syntaxe

WHILE condition

- **Utilisation des lots**

Un lot (batch) est un ensemble d'instructions Transact SQL envoyées ensemble à SQL Server et exécutées collectivement. Un lot peut être soumis de façon interactive ou dans le cadre d'un script.

Le séparateur de lot **GO** n'est pas une instruction Transact-SQL ; il n'est reconnu que par l'Analyseur de requête et signale qu'il doit soumettre la ou les requêtes précédentes.

SQL Server optimise, compile et exécute ensemble les instructions d'un même lot ; la portée des variables est limitée à un seul lot.

Les instructions exécutées au sein d'un même lot doivent respecter un certain nombre de règles :

- Il n'est pas possible de combiner les instructions **CREATE DEFAULT**, **CREATE VIEW**, **CREATE TRIGGER**, **CREATE PROCEDURE** avec d'autres instructions dans un même lot
- Si une table est modifiée, les nouvelles colonnes ne pourront être utilisées que dans le lot suivant