



SQL SERVER 2012 SUPPORT DE FORMATION PARTIE 1

26/05/2014

Concepteur Developpeur Informatique

TABLE DES MATIERES

Table des matières

PRESENTATION DU LANGUAGE	1
Ressources.....	1
Historique du langage	1
Notion de relation	1
Rappels d'algèbre relationnel	9
Les Opérateurs Ensemblistes	9
Opérateurs Relationnels Unaires.....	11
Opérateurs Relationnels Binaires	12
Place de SQL dans les SGBDR.....	15
DECOUVERTE DU SQL PAR LA PRATIQUE	16
Une base « <i>jouet</i> » pour decouvrir sql	16
Les tables.....	16
Quelques requêtes	16
Decouverte de T Sql avec Sql Server Management Studio	16
Base de données « PUBLI »	17
Consultation d'une base de donnees	23
Requête SELECT simple sur une seule table	27
Sélection de lignes : clause WHERE	28
Expressions et fonctions.....	29
Requête sur les groupes avec GROUP BY, fonctions de groupe, HAVING	30
Infos de détails ET infos globales : COMPUTE	32
Travail sur plusieurs tables : les jointures	33
Les jointures externes.....	35
Les sous-requêtes	36
L'opérateur ensembliste UNION	40
Synthèse sur les principes d'écriture d'une requête SELECT	40

TABLE DES MATIERES

Mise à jour d'une base de données	41
Ajout de lignes : INSERT.....	41
Modification de lignes : UPDATE	43
Suppression de lignes : DELETE	43

PRESENTATION DU LANGAGE

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 2012 » et « Aide de Transact SQL » sous « Analyseur de requête SQL »

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 1er 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.

NOTION DE RELATION

Le terme relationnel provient de la définition mathématique d'**algèbre relationnelle** (Codd 70).

Une **relation** est un ensemble de tuples (*tuple est la généralisation de couple à un nombre quelconque d'éléments : Les couples, les triplets, les quadruplets, etc sont des tuples*) de données, on peut alors définir des opérations algébriques sur les.

Nous parlerons dans la suite de **table**, et non pas de relation.

Une **table** est un ensemble de tuples de données distincts deux à deux. Une table est constituée d'une **clef primaire** et de plusieurs attributs (ou colonnes) qui dépendent de cette clef.

La **clef primaire** d'une table est un attribut ou un groupe d'attributs de la table qui détermine tous les autres de façon unique.

PRESENTATION DU LANGAGE

Une table possède toujours une et une seule clef primaire. Par contre, une table peut présenter plusieurs clefs candidates qui pourraient jouer ce rôle.

Le domaine d'un attribut est l'ensemble des valeurs que peut prendre cet attribut. Le domaine est constitué d'un type, d'une longueur et de contraintes qui réduisent l'ensemble des valeurs permises.

Une clef étrangère dans une table est une clef primaire ou candidate dans une autre table.

Les contraintes d'intégrité font partie du schéma logique. Parmi celles-ci, on distingue :

- les **contraintes de domaine** qui restreignent l'ensemble des valeurs que peut prendre un attribut dans une table,
 - les **contraintes d'intégrité d'entité** qui précisent qu'une table doit toujours avoir une clef primaire et
 - les **contraintes d'intégrité référentielle** qui précisent les conditions dans lesquelles peuvent être ajoutés ou supprimés des enregistrements lorsqu'il existe des associations entre tables par l'intermédiaire de clefs étrangères
- *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.*

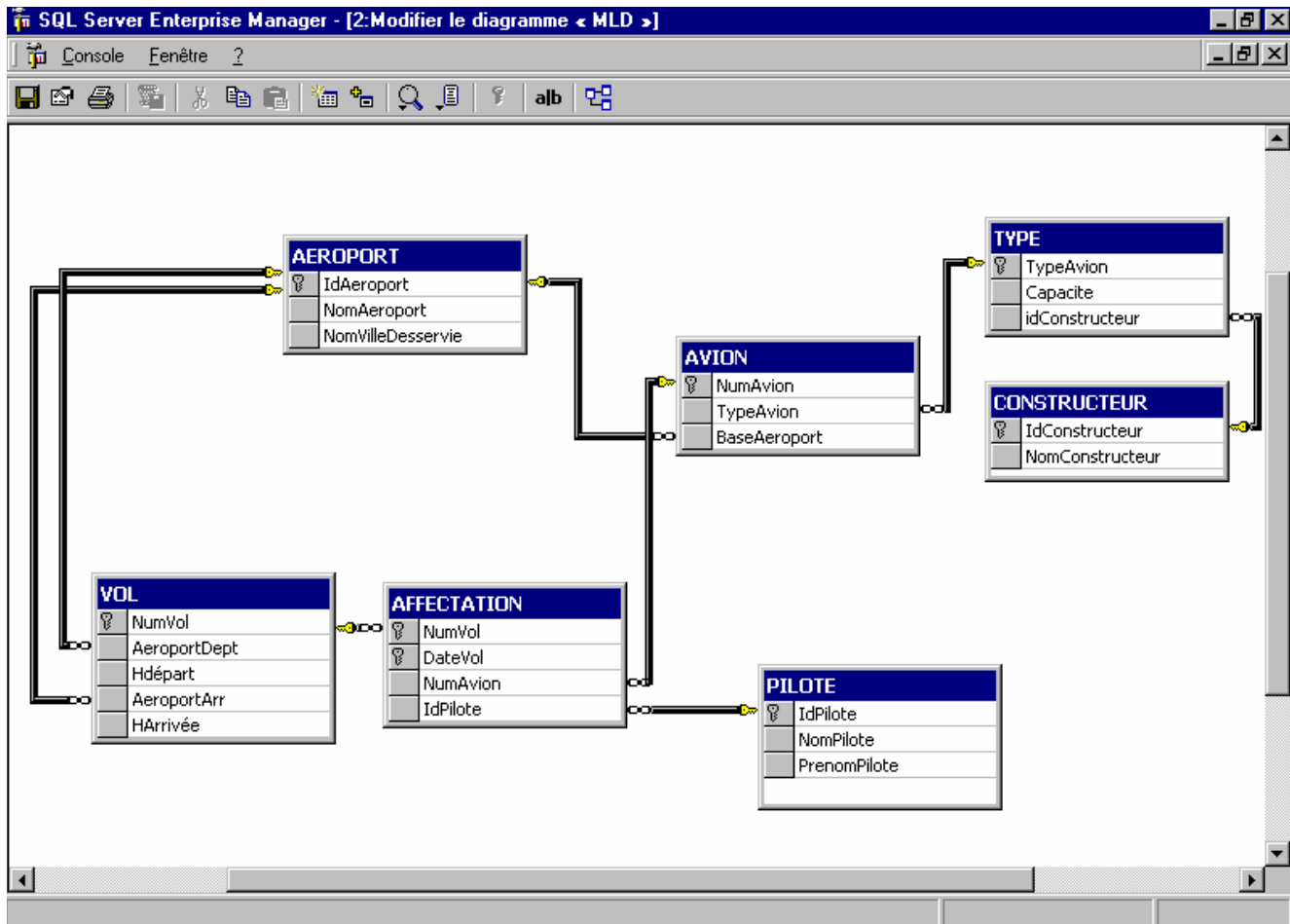
IdPilote	NomPilote	PrenomPilote
1	GAINSBURG	Serge
2	FERRAT	Jean
3	NOUGARO	Claude
4	SCHUMMAN	Robert
5	STROGOFF	Michel
6	SORREL	Lucien
7	TAVERNIER	Bertrand
8	FAYOLLE	Marc
9	LECU	Régis

Ligne ou Tuple

Colonne ou Attribut

PRESENTATION DU LANGAGE

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



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)

PRESENTATION DU LANGAGE

NumAvion	TypeAvion	BaseAeroport
100	A320	NIC
101	B707	CDG
102	A302	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

PRESENTATION DU LANGAGE

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 (IdAéroport, NomAéroport, NomVilleDesservie)

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

NomAéroport : nom de l'aéroport (alphanumérique)

NomVilleDesservie : ville desservie par l'aéroport (alphanumérique)

IdAéroport	NomAéroport	NomVilleDesservie
BAS	Poretta	Bastia
BLA	Blagnac	Toulouse
BRI	Brive	Brive
CDG	Roissy	Paris
GRE	Saint Geoir	Grenoble

PRESENTATION DU LANGAGE

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, AéroportDept, Hdépart, AéroportArr, HArrivée)

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

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

Hdépart : heure de départ (type heure)

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

Harrivée : heure d'arrivée (type heure)

NumVol	AéroportDept	Hdépart	AéroportArr	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

PRESENTATION DU LANGAGE

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

PRESENTATION DU LANGAGE

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...

PRESENTATION DU LANGAGE

RAPPELS D'ALGEBRE RELATIONNEL

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

Les Opérateurs Ensemblistes

Ils ne s'appliquent qu'à des relations « *unicompatibles* » c'est à dire possédant le même nombre d'attributs sur les mêmes domaines. On a :

1. 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

PRESENTATION DU LANGAGE

2. 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

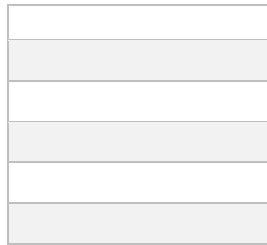
3. LA **DIFFERENCE** : sélectionne les tuples d'une table en éliminant les tuples présents dans une autre table.

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

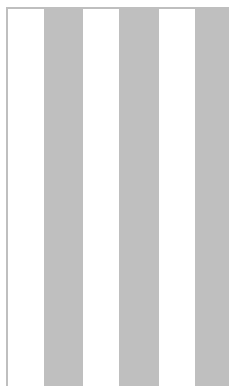
Opérateurs Relationnels Unaires

4. LA **SELECTION** 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'
5. 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

PRESENTATION DU LANGAGE

Opérateurs Relationnels Binaires

6. LE **PRODUIT CARTESIEN** : 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 x N tuples.

A1
A2
A3

B1
B2

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

7. LA **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

PRESENTATION DU LANGAGE

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

PRESENTATION DU LANGAGE

Table Planning, Jointure de Affectation - Pilote

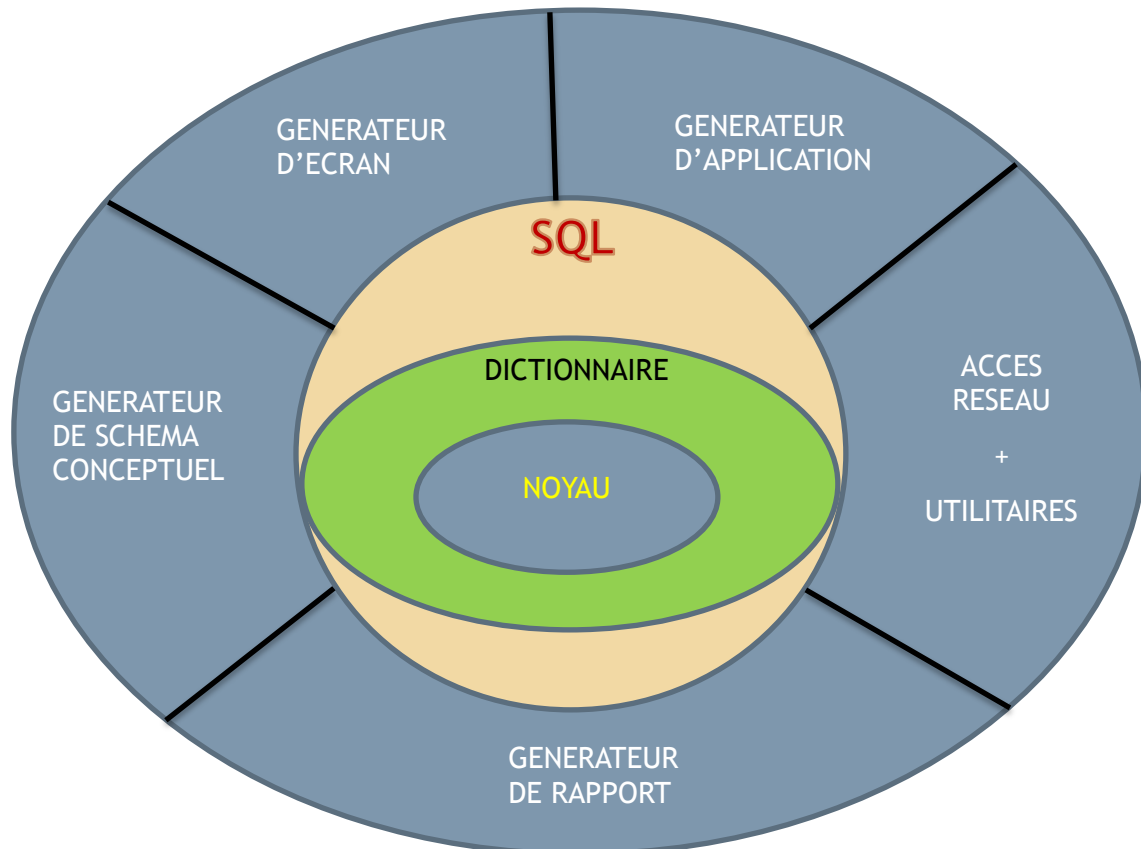
NumVol	DateVol	NumA	NomPilote	PrenomPil
IT100	6 avril 2001	100	GAINSBURG	Serge
IT100	7 avril 2001	101	FERRAT	Jean
IT101	6 avril 2001	100	FERRAT	Jean
IT101	7 avril 2001	103	SCHUMMAN	Robert
IT102	6 avril 2001	101	GAINSBURG	Serge
IT102	7 avril 2001	102	NOUGARO	Claude
IT103	6 avril 2001	105	NOUGARO	Claude
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

PRESENTATION DU LANGAGE

PLACE DE SQL DANS LES SGBDR

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

- ❖ Un générateur d'écrans pour faciliter la saisie et l'affichage de données par imbrication de requêtes.
- ❖ Un générateur d'états pour generer la sortie d'état de la BD sur papier ou écran.
- ❖ Un 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).
- ❖ Un 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 :



DECOUVERTE DU SQL PAR LA PRATIQUE

Une base « *jouet* » pour decouvrir sql

Les tables

a	b	c
x	m	2
x	n	1
y	m	4
z	p	1

d	e
x	8
y	4
x	1

Quelques requêtes

Exercice 1

Calculer à la main le résultat des requêtes suivantes :

1. SELECT * FROM un ;
2. SELECT a FROM un ;
3. SELECT a FROM un WHERE c=1 ;
4. SELECT a FROM un WHERE c=1 OR c=2 ;
5. SELECT DISTINCT a FROM un WHERE c=1 OR c=2 ;
6. SELECT a FROM un ORDER BY b ;
7. SELECT a, e FROM un, deux ;
8. SELECT a, e FROM un, deux WHERE c=e ;

Decouverte de T Sql avec Sql Server Management Studio

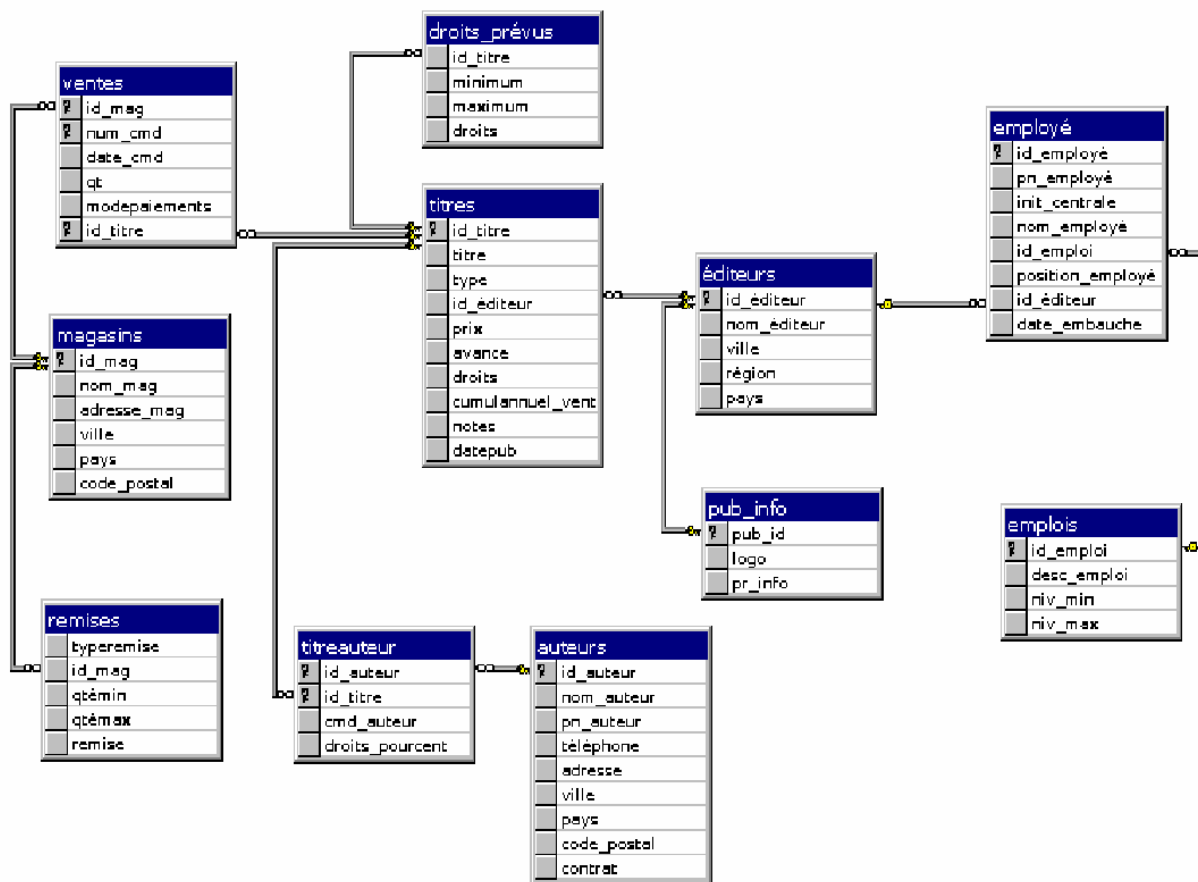
Exercice 2

A l'aide de SSMS :

1. créez la base de données « *jouet* »
2. créez les tables « UN » et « DEUX »
3. inserez les données des différentes tables
4. vérifiez les résultats de l'exercice précédent

DECOUVERTE DU SQL PAR LA PRATIQUE

Base de données « 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]')

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(<i>id_éditeur</i>)
<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)			

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][0-9][FM]') OR (id_employé LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-9][0-9][FM]')
2. L'index composé, ordonné est défini sur nom_employé, pn_employé, init_centrale.

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
id_emploi	smallint	non	IDENTITY(1,1)	CP, ordonné.
desc_emploi	varchar(50)	non	oui (1)	
niv_min	tinyint	non	oui (2)	
niv_max	tinyint	non	oui (3)	

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][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*.

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_titre</i> <i>cmd_auteur</i> <i>droits_pourcent</i>	<i>id</i> <i>tid</i> <i>tinyint</i> <i>int</i>	non non oui oui		Composé CP, ordonné., (1) CE auteurs(<i>id_auteur</i>) (2) Composé CP, ordonné., (1) CE titres(<i>id_titre</i>) (3)

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(id_éditeur)
<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()	

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
<i>id_titre</i>	<i>idt</i>	non		CE titres(id_titre)
<i>minimum</i>	<i>int</i>	oui		
<i>maximum</i>	<i>int</i>	oui		
<i>droits</i>	<i>int</i>	oui		

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>	<i>char</i> (4)	non	Composé CP, ordonné. (1) , CE magasins(<i>id_mag</i>)
<i>num_cmd</i>	<i>varchar</i> (20)	non	Composé CP, ordonné. (1)
<i>date_cmd</i>	<i>datetime</i>	non	
<i>qt</i>	<i>smallint</i>	non	
<i>modepaiements</i>	<i>varchar</i> (12)	non	
<i>id_titre</i>	<i>idt</i>	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*.

Table magasins

NOM_COLONNE	TYPE DE DONNEES	NULL	PAR DEFAULT CHECK	CLE/INDEX
<i>id_mag</i>	<i>char</i> (4)	non		CP, ordonné
<i>nom_mag</i>	<i>varchar</i> (40)	Oui		
<i>adresse_mag</i>	<i>varchar</i> (40)	Oui		
<i>ville</i>	<i>varchar</i> (20)	Oui		
<i>pays</i>	<i>char</i> (2)	Oui		
<i>code_postal</i>	<i>char</i> (5)	oui		

DECOUVERTE DU SQL PAR LA PRATIQUE

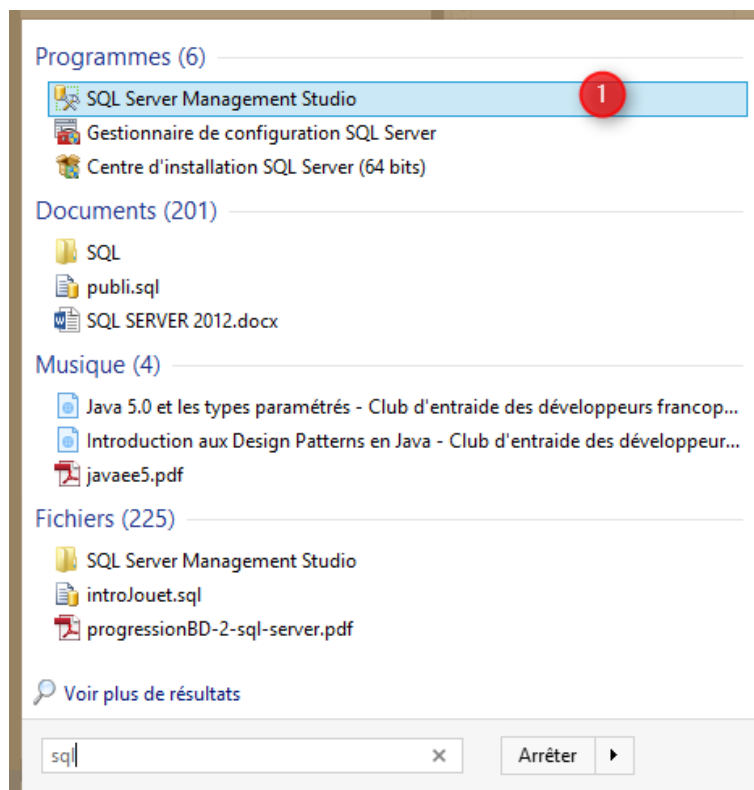
Table remises

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.

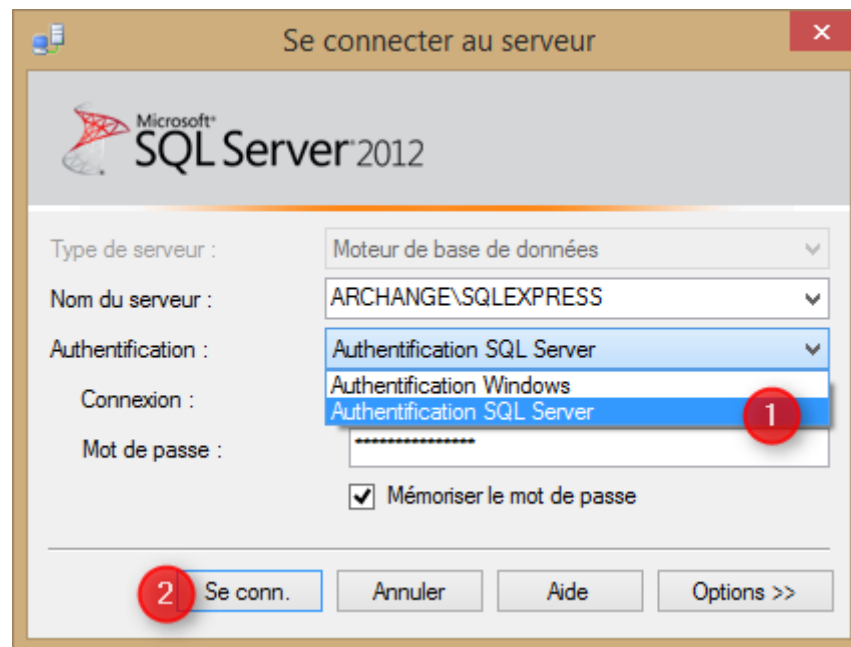
NOM_COLONNE	TYPE DE DONNEES	NULL	PAR DEFAULT CHECK	CLE/INDEX
typeremise	varchar(40)	non		
id_mag	char(40)	Oui		CE magasins(id_mag)
qtémin	smallint	Oui		
qtémax	smallint	Oui		
remise	decimal	non		

Consultation d'une base de donnees

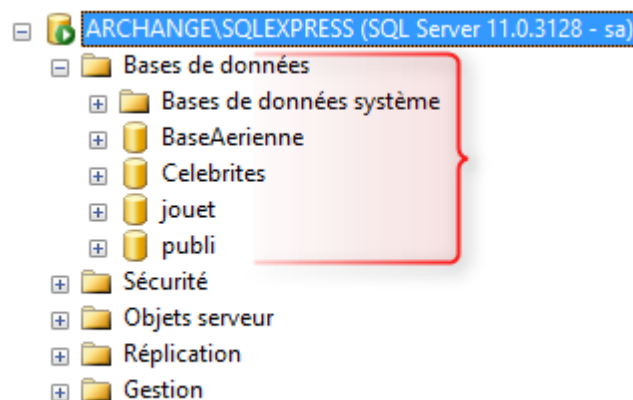
Microsoft Sql management studio est l'outil utilisé pour exécuter les exercices ci après



DECOUVERTE DU SQL PAR LA PRATIQUE

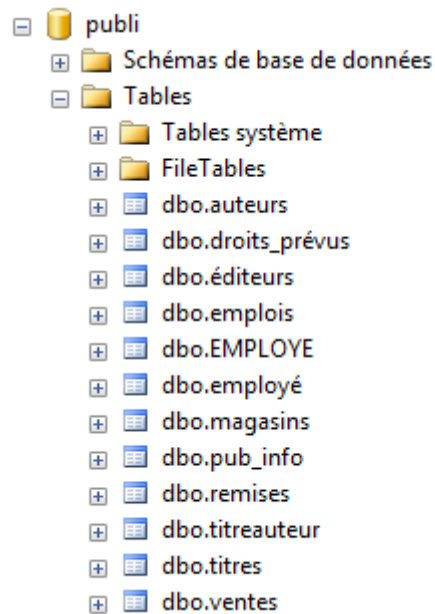


- Développer « Bases de données » : seules les bases de données auxquelles vous avez accès apparaissent.

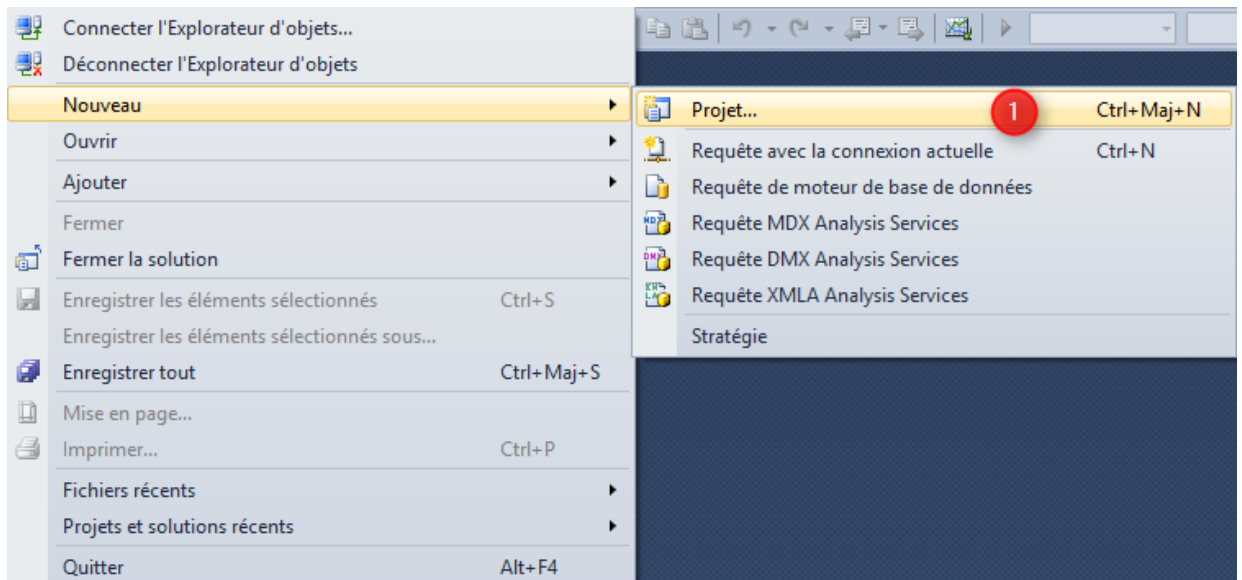


DECOUVERTE DU SQL PAR LA PRATIQUE

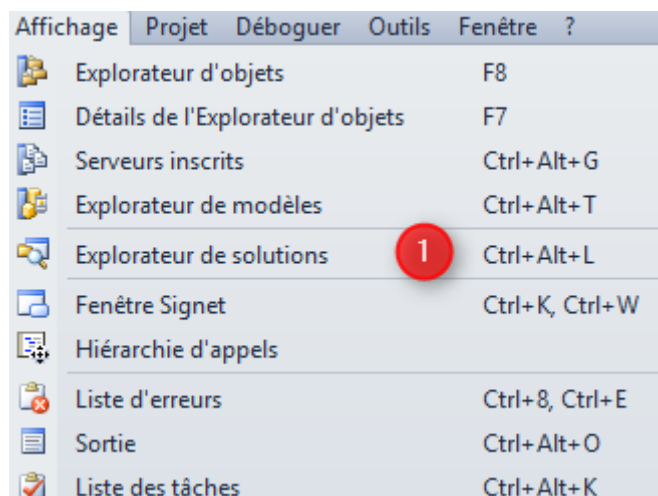
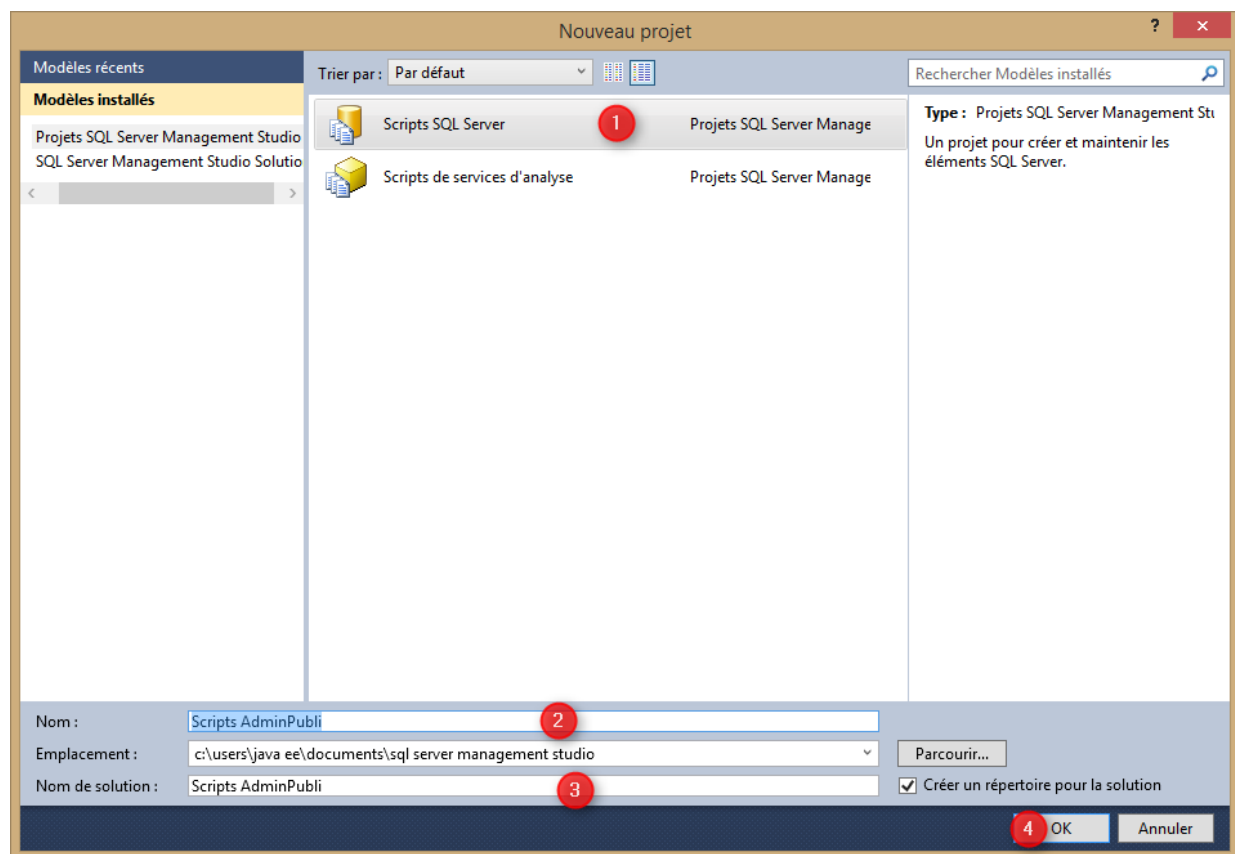
➤ Développer la base publi

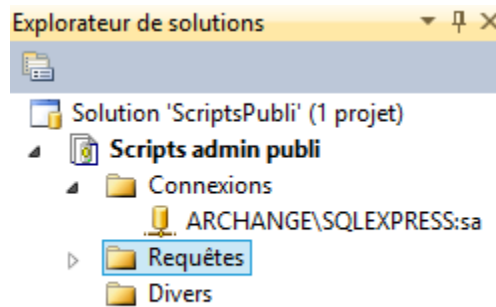


➤ Creation d'un projet



DECOUVERTE DU SQL PAR LA PRATIQUE





Requête SELECT simple sur une seule table

- 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

Exemple

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'
```

pn_auteur	nom_auteur
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.

Pour plus d'informations sur SELECT, consultez l'instruction SELECT dans le Manuel de référence Transact-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 : 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 3 : 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 4 : 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  
go
```

- 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  
go
```


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

- 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
GO
```

- 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.

DECOUVERTE DU SQL PAR LA PRATIQUE

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
GO
```

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%'
GO
```

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'
GO
```

DECOUVERTE DU SQL PAR LA PRATIQUE

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 :

Remarquer que la clause where selectionne des lignes, la clause having selectionne 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
GO
```

Exercice 5 : GROUP BY, COUNT, MIN, MAX

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

Exercice 6 : GROUP BY, MAX

Pour chaque Identificateur de titre, calculer les droits prévus maximum (table droits_prévus, colonne droits)

Exercice 7 : 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'

Infos de détails ET infos globales : COMPUTE

La clause **COMPUTE** est employée dans des instructions SELECT avec des fonctions d'agrégation ligne « SUM, AVG, MIN, MAX et COUNT » pour générer des valeurs statistiques à partir de valeurs contenues dans des groupes de lignes.

- Ces valeurs statistiques apparaissent sous forme de lignes supplémentaires dans les résultats de la requête, ce qui vous permet de visualiser les lignes détail et les lignes de statistiques dans un seul ensemble de résultats..

Exemple : somme des prix des différents types de livres de cuisine

Travail sur plusieurs tables : les jointures

- Une jointure est un produit cartésien avec une restriction. Une jointure permet d'associer logiquement des lignes de tables différentes. Les jointures sont généralement (pour des raisons de performances) utilisées pour mettre en relation les données de lignes comportant une clé étrangère avec les données de lignes comportant une clé primaire. Voyons-le en détail avec un exemple concret :

Il existe deux manières de faire une jointure :

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

- a. La syntaxe classique :

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

- b. La syntaxe SQL ANSI :

INNER JOIN dans la syntaxe SQL 2

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

Le résultat est le même pour les deux instructions précédentes SQL

- 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

DECOUVERTE DU SQL PAR LA PRATIQUE

- 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.

Exercice 9 : Jointure entre trois tables

Afficher les noms des auteurs parisiens, les titres et les prix de leurs livres

Exercice 10 : Jointure sur quatre tables, ORDER BY, COMPUTE

Pour chaque éditeur, 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, et le nombre d'exemplaires vendus au total par chaque éditeur.

<i>nom éditeur</i>	<i>titre</i>	<i>nom mag</i>	<i>qt</i>
<i>Algodata Infosystems</i>	<i>Guide des bases de données du gestionnaire pressé</i>	<i>Eric the Read Books</i>	<i>5</i>
<i>Algodata Infosystems</i>	<i>Guide des bases de données du gestionnaire pressé</i>	<i>Bookbeat</i>	<i>10</i>
<i>Algodata Infosystems</i>	<i>La cuisine - l'ordinateur : bilans clandestins</i>	<i>Bookbeat</i>	<i>25</i>
<i>Algodata Infosystems</i>	<i>Toute la vérité sur les ordinateurs</i>	<i>Fricative Bookshop</i>	<i>15</i>
			<i>sum</i>
			<i>=====</i>
			<i>55</i>
<i>nom éditeur</i>	<i>titre</i>	<i>nom mag</i>	<i>qt</i>
<i>Binnet & Hardley</i>	<i>Les festins de Parly 2</i>	<i>Fricative Bookshop</i>	<i>10</i>
<i>Binnet & Hardley</i>	<i>Les micro-ondes par gourmandise</i>	<i>Doc-U-Mat: Quality Laundry and Books</i>	<i>25</i>
			<i>sum</i>
			<i>=====</i>
			<i>35</i>

Exercice 11 : 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)

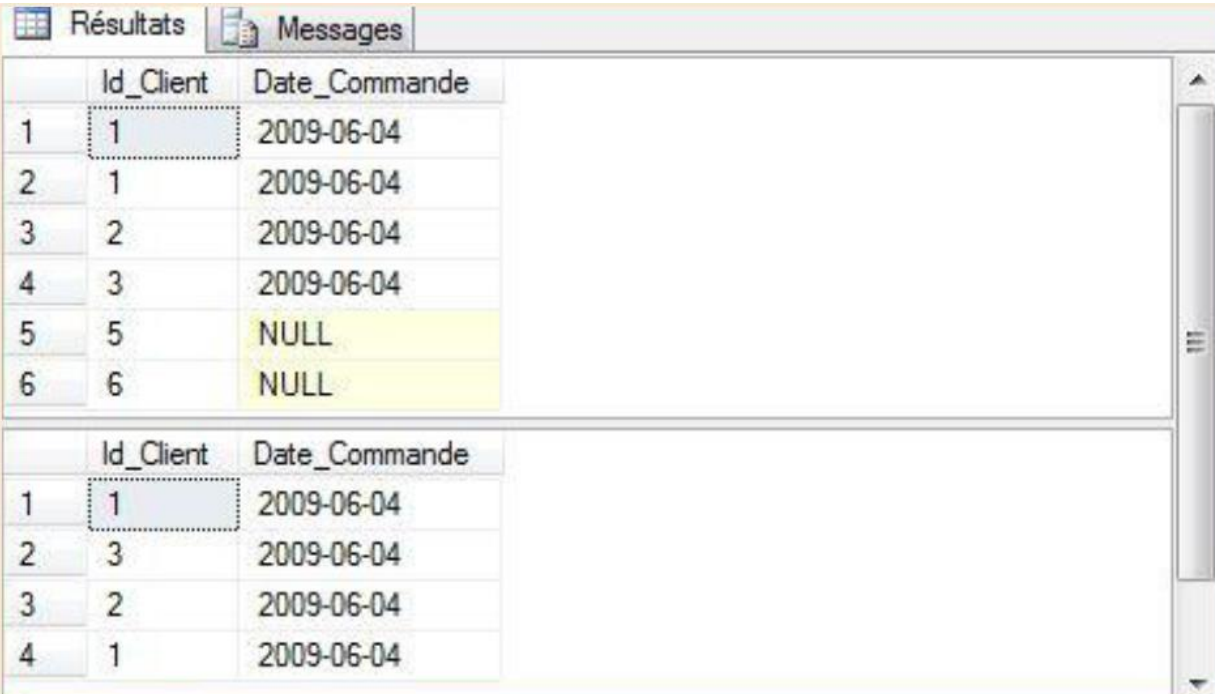
Les jointures externes

Les jointures externes sont des jointures dans lesquelles la condition est fausse. Dans ce cas, le résultat retourné sera celui d'une des deux tables. Le résultat sera celui de la première table citée si on utilise l'option `LEFT`, et celui de la seconde table citée si l'on utilise l'option `RIGHT`. La syntaxe est la suivante :

```
SELECT Client.Id_Client, Date_Commande
FROM Client LEFT OUTER JOIN Commande
ON Client.Id_Client = Commande.Id_Client
```

```
SELECT Client.Id_Client, Date_Commande
FROM Client RIGHT OUTER JOIN Commande
ON Client.Id_Client = Commande.Id_Client
```

Et le résultat est le suivant :



	Id_Client	Date_Commande
1	1	2009-06-04
2	1	2009-06-04
3	2	2009-06-04
4	3	2009-06-04
5	5	NULL
6	6	NULL

	Id_Client	Date_Commande
1	1	2009-06-04
2	3	2009-06-04
3	2	2009-06-04
4	1	2009-06-04

On remarque alors clairement que suivant qu'on utilise l'option `RIGHT` ou `LEFT`, le résultat est différent, et qu'il respecte le comportement décrit auparavant. Les valeurs `NULL` présentes dans le premier résultat sont dues au fait que les clients dont l'id est 5 et 6 n'ont pas de commandes. Ces valeurs `NULL` disparaissent dans le second résultat, tout simplement parce qu'il n'existe pas de commande qui n'a pas de client, alors que l'inverse existe. En revanche, il est obligatoire d'utiliser les jointures externes avec la syntaxe ANSI, c'est pourquoi je vous recommande d'apprendre les jointures selon le modèle ANSI et non le modèle classique, bien que le modèle classique soit plus logique. Dans les versions antérieures, le modèle classique était supporté grâce aux signes `*=` et `=*`, mais ceci ne sont plus supportés depuis SQL Server 2008.

Les sous-requêtes

Il est possible d'imbriquer une requête `SELECT` dans une requête `SELECT` (`UPDATE` ou `DELETE`). Les sous requêtes peuvent être utilisées avec les clauses `HAVING` ou `WHERE`.

Il existe trois types de sous requêtes différentes :

1. Les sous requêtes qui ne renvoient qu'une seule valeur unique (sous-requête scalaire) :

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'
```

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
```

En substituant à la constante 136, la requête qui calcule ce prix, on obtient la solution avec sousrequête.

```
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.

DECOUVERTE DU SQL PAR LA PRATIQUE

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.

- trouver le prix maximum

```
SELECT max(prix)
FROM titres
```

- Trouver la moyenne des prix

```
SELECT avg(prix)
FROM titres
```

- Ecrire la requête principale avec ces constantes

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

- 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 )
```

2. Les requêtes renvoyant une liste d'enregistrements. Elles sont utilisées avec IN, EXIST, ANY, SOME ou encore ALL :

Exemple :

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

DECOUVERTE DU SQL PAR LA PRATIQUE

- 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
```

DECOUVERTE DU SQL PAR LA PRATIQUE

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 12 : 2 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 13 : 1 sous-requête, MAX

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

Exercice 14 : 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 15 : 1 sous-requête, MAX

Afficher les ventes par titre et magasins.

Sélectionner la plus forte de ces ventes.

Exercice 15 bis :

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

L'opérateur ensembliste 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.

MISE A JOUR D'UNE BASE DE DONNEES

- 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, ...)
```

DECOUVERTE DU SQL PAR LA PRATIQUE

- ❖ 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'
```

- ❖ 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 16

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 17

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 18 :

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.

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 référencées dans *titreauteur* : effacez en premier ces références.

Exercice 19

Détruire les lignes créées dans la table *auteur*, dans les exercices 16 et 17, afin de remettre la table dans son état initial.