DECOUVERTE SQL PAR LA PRATIQUE

UNE BASE « *JOUET* » POUR DECOUVRIR SQL

LES TABLES

Table UN

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

Table DEUX

d	e
x	8
y	4
x	1

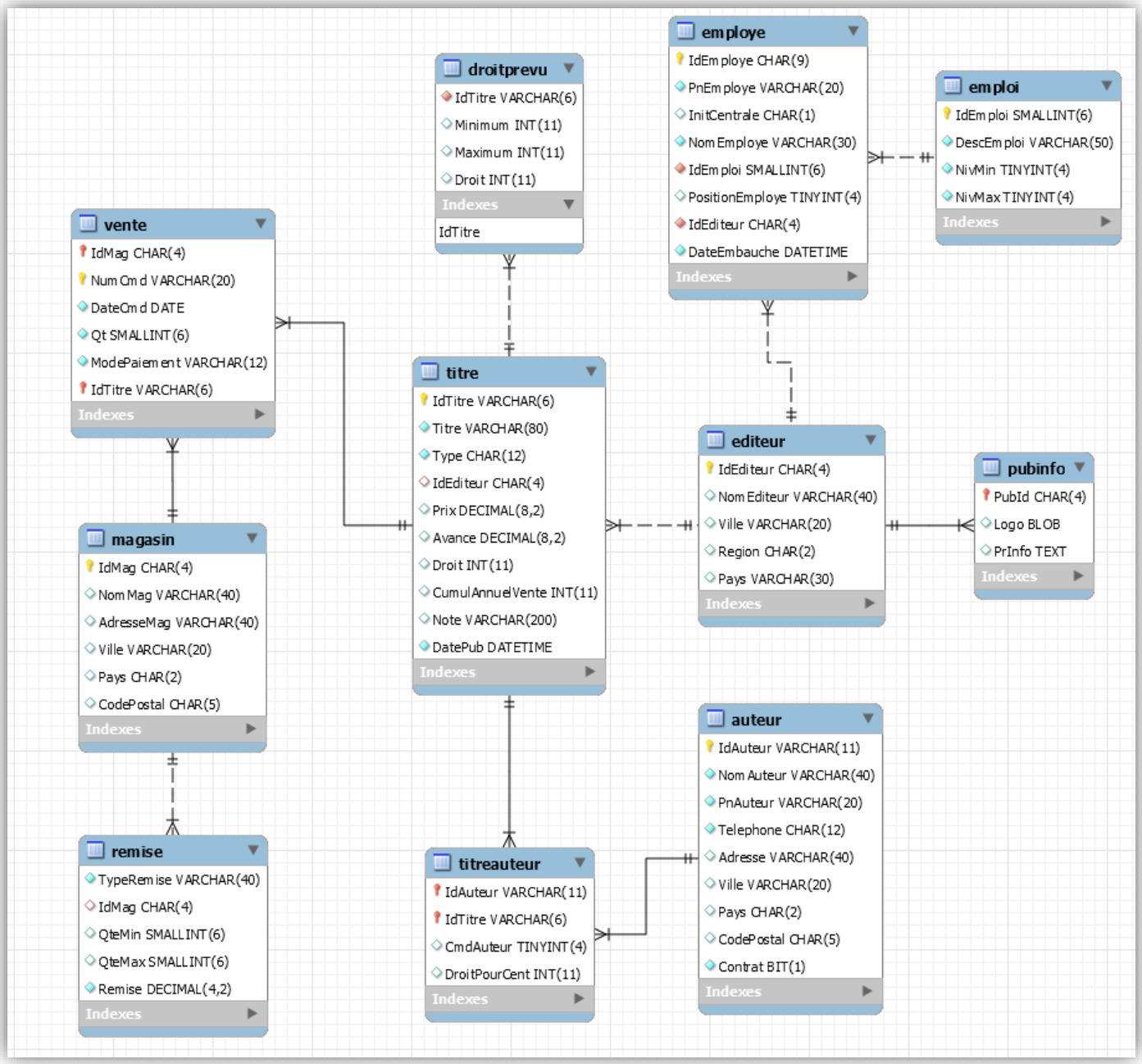
QUELQUES REQUETES

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 ;

BASE DE DONNEES « 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 **pubinfo**), 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* ***droitsprevus****).*
  - *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 editeurs

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

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

Table pubinfo

Nom_colonne	Type de données	NULL	Par défaut Check	Clé/index
pubId	*char*(4)	non		CP, ordonné., CE éditeurs(idEditeur)
logo	*image*	oui		
pr_info	*text*	oui		
idEditeur	***logo*** (1)	***info_rp*** (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 texte, 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 positionEmploye), compris entre le coefficient minimum et le coefficient maximum correspondant à leur type d'emploi (nivMin et nivMax dans la table emplois)

Nom_colonne	Type de données	NULL	Par défaut Check	Clé/index
idEmploye	*empid*	non	oui (1)	CP, non ordonné.
pnEmploye	*varchar*(20)	non		Composé, ordonné. (2)
initCentrale	*char*(1)	oui		Composé, ordonné. (2)
nomEmploye	*varchar*(30)	non		Composé, ordonné. (2)
idEmploi	*smallint*	non	1	CE emplois(idEmploi)
positionEmploye	*tinyint*	non	10	
idEditeur	*char*(4)	non	'9952'	CE éditeurs(idEditeur)
dateEmbauche	*datetime*	non	GETDATE( )	

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

2. *L'index composé, ordonné est défini sur nomEmploye, pnEmploye, initCentrale.*

Table emplois

A chaque type d'emploi, correspond un coefficient minimal (nivMin) et un coefficient maximal (nivMax)

Nom_colonne	Type de données	NULL	Par défaut Check	Clé/index
idEmploi *descEmploi* *nivMin* *nivMax*	*smallint* *varchar*(50) *tinyint* *tinyint*	non non non non	IDENTITY(1,1) oui (1) oui (2) oui (3)	CP, ordonné.

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
idAuteur	id	non		oui (1)	CP .
nomAuteur	varchar(40)	non			Composé, non ordonné (3)
pnAuteur	varchar(20)	non			Composé, non ordonné (3)
téléphone	char(12)	non	'INCONNU'		
adresse	varchar(40)	oui			
ville	varchar(20)	oui			
pays	char(2)	oui			
codePostal	char(5)	oui		oui (2)	
contrat	bit	non			

1. *La contrainte CHECK idAuteur est définie comme (idAuteur LIKE '[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9]').*
2. *La contrainte CHECK codePostal est définie comme (codePostal LIKE '[0-9][0-9][0-9][0-9][0-9]').*
3. *L'index composé non ordonné est défini sur nomAuteur, pnAuteur.*

Table titreauteur

Attention : certains auteurs peuvent être décrits dans la table Auteur 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
idAuteur *idTitre* *cmdAuteur* *droitsPourcent*	*id* *tid* *tinyint* *int*	non non oui oui		Composé CP, ordonné., (1) CE auteurs(idAuteur) (2) Composé CP, ordonné., (1) CE titres(idTitre) (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 **cumulAnnuelVente)**, et la tranche de droit d'auteur atteinte par ce livre (colonne **droits**)

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

Table droitsprevus

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
idTitre *minimum* *maximum* *droits*	*idt* *int* *int* *int*	non oui oui oui		CE titres(idTitre)

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 idMag), le numéro de commande (numCmd), le titre du livre commandé (idTitre). La table Ventes possède donc une clé primaire composée, constituée de ces trois colonnes.

Nom_colonne	Type de données	NULL	Clé/index
idMag	*char*(4)	non	Composé CP, ordonné. (1) , CE magasins(idMag)
numCmd	*varchar*(20)	non	Composé CP, ordonné. (1)
dateCmd	*datetime*	non	
qt	*smallint*	non	
modepaiements	*varchar*(12)	non	
idTitre	*idt*	non	Composé CP, ordonné., (1) CE titres(idTitre)

1. *L'index composé, clé primaire, ordonné est défini sur idMag, numCmd, idTitre.*

Table magasins

NOM_COLONNE	TYPE DE DONNEES	NULL	PAR DEFAUT CHECK	CLE/INDEX
idMag	*char*(4)	non		CP, ordonné
nomMag	*varchar*(40)	Oui		
adresseMag	*varchar*(40)	Oui		
ville	*varchar*(20)	Oui		
pays	*char*(2)	Oui		
codePostal	*char*(5)	oui		

Table remises

Les éditeurs consentent trois types de remises : des « remises client » à certains magasins privilégiés (référencés par la colonne idMag) ; 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 DEFAUT CHECK	CLE/INDEX
typeremise	varchar(40)	non		
idMag	char(40)	Oui		CE magasins(idMag)
qtémin	smallint	Oui		
qtémax	smallint	Oui		
remise	decimal	non		

CONSULTATION D'UNE BASE DE DONNEES

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 > cumulAnnuelVente * prix*

- Des intervalles (BETWEEN et NOT BETWEEN)
  WHERE cumulAnnuelVente 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 (cumulAnnuelVente > 2000 AND cumulAnnuelVente < 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 partie du résultat.*

Regrouper les titres en fonction du type, en éliminant les groupes qui contienne 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
```

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 positionEmploye) 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 6 : GROUP BY, MAX

Pour chaque Identificateur d e titre, calculer les droits prévus maximum (table droitsprevus, 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'

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*

  - *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

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.

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

nom_éditeur	titre	nom_mag	qt
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

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).

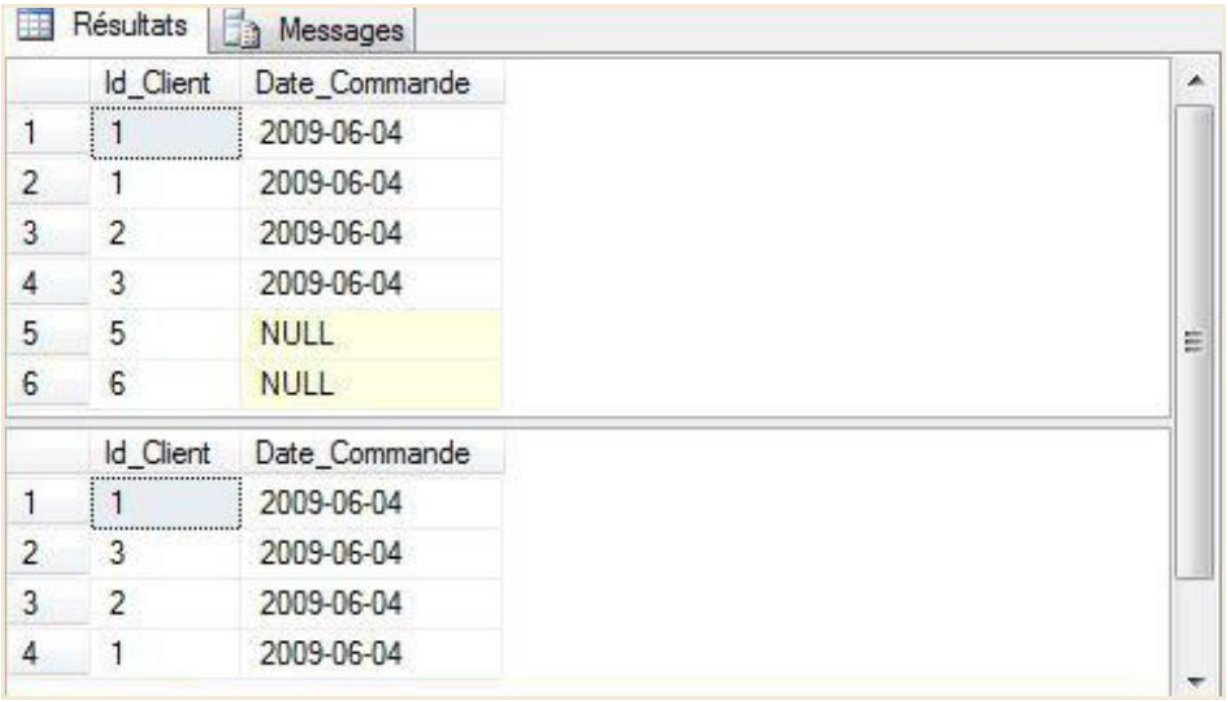
Les jointures externes

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 :


Résultats Messages

	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 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.

- *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)

- *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 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.droitsPourcent = 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 MySQL, 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. MySQL 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'
```

❖ *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 idAuteur, pnAuteur, nomAuteur et adresse,
la table nouveauxAuteurs contient les colonnes idAuteur, adresse, pnAuteur et nomAuteur.

```
INSERT auteurs (idAuteur, adresse, nomAuteur, pnAuteur)
SELECT * FROM nouveauxAuteurs
        ou
INSERT auteurs
SELECT idAuteur, pnAuteur, nomAuteur, adresse FROM
nouveauxAuteurs
```

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.idTitre et titreauteur.idTitre vous empêche d'exécuter cet effacement parce que la clé étrangère ne permettra pas que vous effaciez des titres ecore 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.