OFPPT/DRPS/ISGI LAAYOUNE

FILIERE : TECHNIQUES DE DEVELOPPEMENT INFORMATIQUE

TDI2

MODULE : SYSTEME DE GESTION DE BASE DE DONNES I

Table des matières

I. LES SYSTEMES DE GESTION DES BASE DE DONNEES 10
1) PRINCIPES DE FONCTIONNEMENT 10
II. SQL server 2008 comme exemple d'un SGBDR 13
A. Composants SQL Server 13
I. Pourquoi gérer les droits ? 21
II. Gestion des droits dans le processus de développement 22
A. Notions de base 22
B. Gestion des droits : principes 22
III. Implémentation d'une stratégie de sécurité 23
A. Choix du mode d'authentification : 23
IV. Gestion des rôles 24
A. Qu'est-ce qu'un rôle exactement ? 24
B. Gestion des droits : règles 28
C. Notions supplémentaires Login/User 29
D. Les Schémas 30
V. Gestion des permissions 33
A. Droit d'utilisation d'instructions 34
B. Autorisations 34
A. Retirer privilèges 36
B. Interdire l'utilisation d'un privilège 37
C. Gestion des permissions sur les objets 37
I. Notion de base de données 43
A. Définition 43
B. Conception d'une base de données 43
C. Introduction au Modèle Relationnel 44
II. Différentes Opérations appliquées sur les relations 48
A. Opération PROJECTION 48
B. Opération PROJECTION EN langage SQL 50
C. Opération RESTRICTION 50
D. Opération JOINTURE (équi-jointure) 52
E. Opération JOINTURE (équi-jointure) 54

I. Les opérations ensemblistes 57
A. Opération UNION 58
B. Opération UNION avec transact SQL 59
C. Opération INTERSECTION 59
D. Opération INTERSECTION en langage SQL 60
E. Opération DIFFERENCE 62
F. Opération DIFFERENCE avec langage SQL 63
G. Opération PRODUIT CARTESIEN 65
H. Opération Division 66
I. Opération TRI 67
I. Les types de données 71
II. Création des tables 72
A. Les tables temporaires : 73
I. Les contraintes de colonnes (verticales) 75
A. Obligatoire ([NOT] NULL) 76
I. Mise en œuvre des contraintes 77
A. Valeur par défaut (DEFAULT) 77
B. Clef (PRIMARY KEY) 78
C. Unicité (UNIQUE) 79
D. CREATION D'INDEXS 80
E. Validation (CHECK) 82
II. Intégrité référentielle (FOREIGN KEY / REFERENCES) 83
III. Les contraintes de table 85
A. Clef multicolonne (PRIMARY KEY) 85
B. Unicité globale (UNIQUE) 86
C. Validation de ligne (CHECK) 87
IV. Intégrité référentielle de table (FOREIGN KEY / REFERENCES) 87
V. Suppression d'une table 89
VI. MODIFIER UN TABLEAU 89
A. Ajout d'une nouvelle colonne 89
B. Suppression d'une colonne 90
C. Modification du type de données d'une colonne 90
D. Ajout d'une colonne avec une contrainte 91
E. Ajout d'une contrainte CHECK non vérifiée à une colonne existante 92
F. Ajout d'une contrainte DEFAULT à une colonne existante 92
G. Ajout de plusieurs colonnes avec des contraintes 93
I. Langage de manipulation des données 95
A. Insertion 95

B. Modification 96
Suppression 98
I. Interrogations 101
A. Syntaxe générale 101
B. Clause SELECT 101
C. Clause FROM 102
D. Clause WHERE 103
E. Opérateurs logiques AND et OR 104
II. Sous-interrogation 105
A. Sous-interrogation à une ligne et une colonne 105
B. Sous-interrogation ramenant plusieurs lignes 107
C. Les Prédicats : ALL, DISTINCT, DISTINCTROW, TOP 108
D. Sous-interrogation synchronisée ou bien corrélées 111
E. Sous-interrogation ramenant plusieurs colonnes 112
F. Clause EXISTS 113
G. Division avec la clause EXISTS 113
III. Fonctions de groupes 116
Clause GROUP BY 117
Clause HAVING 119
IV. Fonctions 120
A. Fonctions arithmétiques 120
B. Fonctions chaîne de caractères 120
C. Les fonctions des dates : 122
I. Initialisation validation ou annulation de transaction 124
A. Détection des erreurs 126
B. Gestion des erreurs 128

GUIDE PEDAGOGIQUE

SYSTÈME DE GESTION DE BASES DE DONNÉES I

Code : TDI-17 Durée : 75 Heures

OBJECTIF OPÉRATIONNEL

COMPÉTENCE

Créer et exploiter des bases de données.

PRÉSENTATION :

Ce module de compétence particulière constitue un préalable pour l'enseignement des modules : "Système de gestion de bases de données II", "Programmation Client/Serveur"" et "Programmation de sites web dynamiques". Il permet au stagiaire de manipuler une base de données en utilisant le langage SQL.

DESCRIPTION :

L'objectif de ce module vise à ce que le stagiaire soit capable d'alimenter une base de données relationnelle et d'en extraire les données avec le langage de requêtes SQL. Les requêtes doivent être écrites en langage SQL dans l'un des utilitaires du système de gestion de base de données en mode console.

Pour les travaux pratiques, utiliser un SGBDR puissant tel que : Oracle ou SQL Server 2008.

CONTEXTE D'ENSEIGNEMENT

STRATEGIES D'ENSEIGNEMENT

Un cours théorique sur les principes de base du modèle relationnel. Des exercices et études de cas pratiques permettant au stagiaire de manipuler des bases de données relationnelles représentant des systèmes d'information variés.

ACTIVITES D'APPRENTISSAGE

Exercices et travaux pratiques permettant aux stagiaires de :

- Concevoir une base de données.
- Écrire des requêtes SQL pour manipuler une base de données.
- Assurer la sécurité des données.

EVALUATION

- Individuellement.
- Travail effectué à l'aide :
- d'un poste informatique ;
- d'un Système de Gestion de Base de Données Relationnel dans notre cas SQL server 2008
- d'un utilitaire d'interface pour introduire et exécuter les requêtes SQL.

  - **Travail effectué à partir :**

- d'études de cas et mises en situation ;
- de sources de référence ;
- des consignes du formateur.

MATERIEL ET EQUIPEMENT

Matériel :

- Un système d'exploitation supportant le SGBD utilisé.
- Un système de gestion de bases de données relationnel.
- La documentation et l'aide en ligne du SGBD choisi.
- Notes de cours.

Équipement :

- Un poste informatique.

PRÉCISIONS ET PREALABLES ÉLÉMENTS DE CONTENU

1. Établir un modèle conceptuel et logique représentant un système d'information.

2. Maîtriser les opérations de base du modèle relationnel.

• Les principes du modèle relationnel.

• Opérations ensemblistes :

- projection ;
- restriction ;
- différence ;
- intersection ;
- union.

• Opérations spécifiques :

- produit cartésien ;
- division ;

- jointure ;
- agrégation.

• Représentation des requêtes en utilisant les arbres algébriques.

A. Traduire les opérations de l'algèbre Relationnel en requêtes SQL.

- Définition du formalisme d'une requête de consultation de données, ordre Select.
- Expressions et fonctions du SGBD.
- Opérateurs de Projection, Restriction, union, intersection.
- Extraction de données en provenance de plusieurs tables : Jointure (equi-jointures, jointures externes, auto-jointures).
- Statistiques sur les données en utilisant les fonctions de groupe.
- Sous interrogations et sous interrogations synchronisées.
- Représentation des données de manière hiérarchique.

3. Connaître l'environnement d'un SGBDR.

- Présentation du système de gestion de base de données utilisé.
- Outil d'interface du SGBD permettant d'exécuter les requêtes.

B. Exploiter l'environnement duSGBDR pour interroger une base de données.

- Construction d'une base de données.
- Écriture et exécution des requêtes SQL.
- Correction des erreurs.

4. Connaître les différents types de données manipulés par le SGBD.
5. Connaître l'importance de clé primaire dans une relation.
6. Connaître l'importance des contraintes d'intégrité référentielle dans la garantie de la cohérence et L'intégrité de données.

- Différents types de données.
- Règles de nomination des objets.
- Importance des contraintes d'intégrité référentielle dans la garantie de la cohérence et l'intégrité de données.

C. Exploiter les commandes de description de données.

• Formalisme d'une requête de description de données pour :

- utiliser CREATE TABLE ;
- définir des contraintes d'intégrité au niveau colonne et table : clé primaire, unique, contrainte d'intégrité référentielle, contrainte de domaine

PRÉCISIONS ET PREALABLES ÉLÉMENTS DE CONTENU

- (CHECK) ;
- utiliser DROP TABLE ;
- utiliser ALTER TABLE ;

D. Exploiter les commandes de manipulation des données.

• Formalisme d'une requête de manipulation de données pour :

- insérer des données dans les tables existantes ;
- insérer les données en utilisant un SELECT ;
- respecter les contraintes au moment de l'insertion ;
- modifier les données en utilisant UPDATE ;

- supprimer les données en utilisant DELETE;

7. Expliquer le rôle des transactions dans les applications client/serveur et dans un contexte multiutilisateur.

- Le modèle Client/Serveur.
- Principe des systèmes transactionnels.

E. Gérer les transactions.

- Formalisme d'une requête de création de transaction.
- Verrouillage des données lors de l'exécution des commandes INSERT UPDATE DELETE.
- Fin des transactions : Commit, RollBack.

8. Définir le rôle d'autres objets de la base de données.

- Rôle des accélérateurs.
- Rôle des vues utilisateurs pour la sécurité et la simplification de l'écriture des requêtes.

Avantage des séquences dans la génération des clés primaires.

F. Utiliser les différents types d'objets sur une base de données.

- Création de séquences pour générer des valeurs de clés primaires.
- Création de vues, et expliquer leur rôle dans la sécurité et la simplification de manipulation de données.
- Optimisation des accès aux données en créant des indexes.

9. Connaître les fonctionnalités offertes par le SGBD pour la sécurité des données.

- Fonctionnalités de sécurité offertes par le SGBD utilisé.

G. Sécuriser les données.

- Création des utilisateurs et des rôles.
- Définition des privilèges système et objet.
- Commandes de gestion des privilèges GRANT et REVOKE.

H. Utiliser le dictionnaire de données.

- Exploitation des vues de dictionnaire.
- Vérification des objets et des privilèges sur les objets.

CHAPITRE : 1

*Leçon1 : Système de gestion des bases **de** données*

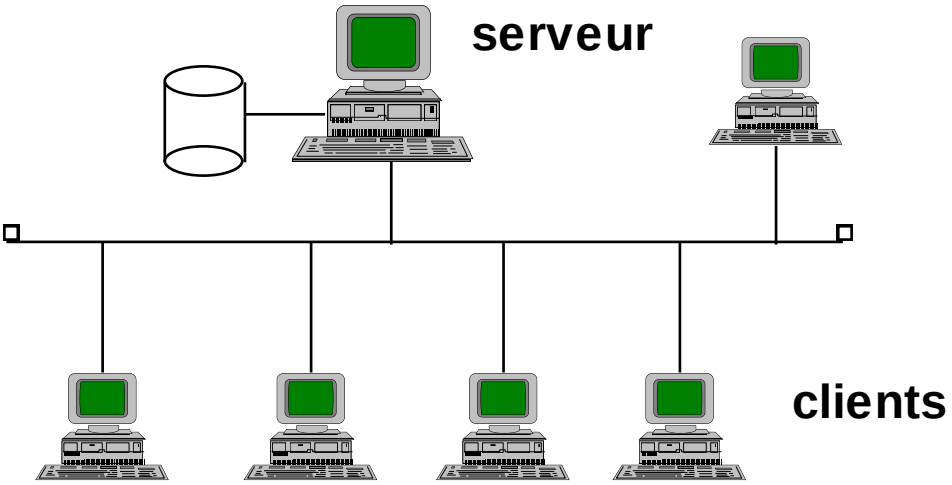
Objectifs: vous saurez à même d'effectuer les tâches suivantes

- Principes de fonctionnement d'un SGBDR.
- objectifs principaux d'un SGBD
- quelques SGBDR connus et utilisés
- SQL Server 2008 comme exemple

I. LES SYSTEMES DE GESTION DES BASE DE DONNEES

1) PRINCIPES DE FONCTIONNEMENT

La gestion et l'accès à une base de données sont assurés par un ensemble de programmes qui constituent le Système de gestion de base de données (SGBD). Un SGBD doit permettre l'ajout, la modification et la recherche de données. Un système de gestion de bases de données héberge généralement plusieurs bases de données, qui sont destinées à des logiciels ou des thématiques différentes.

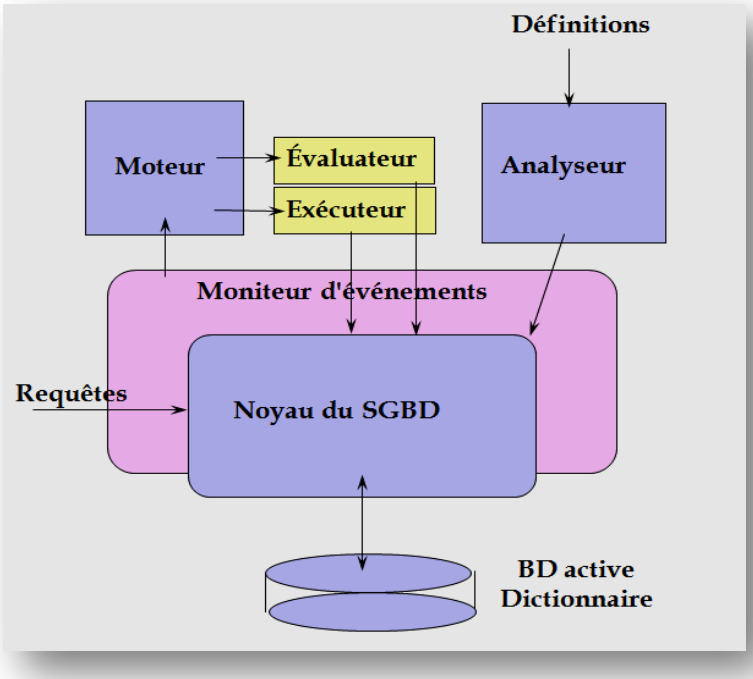


Actuellement, la plupart des SGBD fonctionnent selon un mode client/serveur. Le serveur (sous entendu la machine qui stocke les données) reçoit des requêtes de plusieurs clients et ceci de manière concurrente. Le serveur analyse la requête, la traite et retourne le résultat au client. Le modèle client/serveur est assez souvent implémenté au moyen de l'interface des sockets (voir le cours de réseau) ; le réseau étant Internet.

Une variante de ce modèle est le modèle ASP (Application Service Provider). Dans ce modèle, le client s'adresse à un mandataire (broker) qui le met en relation avec un SGBD capable de résoudre la requête. La requête est ensuite directement envoyée au SGBD sélectionné qui résout et retourne le résultat directement au client.

Quelque soit le modèle, un des problèmes fondamentaux à prendre en compte est la cohérence des données. Par exemple, dans un environnement où plusieurs utilisateurs peuvent accéder concurremment à une colonne d'une table par exemple pour la lire ou pour l'écrire, il faut s'accorder sur la politique d'écriture. Cette politique peut être : les lectures concurrentes sont autorisées mais dès qu'il y a une écriture dans une colonne, l'ensemble de la colonne est envoyée aux autres utilisateurs layant lue pour quelle soit rafraîchie.

A. Objectifs



Des objectifs principaux ont été fixés aux SGBD dès l'origine de ceux-ci et ce, afin de résoudre les problèmes causés par la démarche classique. Ces objectifs sont les suivants :

a) Indépendance physique :

La façon dont les données sont définies doit être indépendante des structures de stockage utilisées.

b) Indépendance logique :

Un même ensemble de données peut être vu différemment par des utilisateurs différents. Toutes ces visions personnelles des données doivent être intégrées dans une vision globale.

c) Accès aux données :

L'accès aux données se fait par l'intermédiaire d'un Langage de Manipulation de Données (LMD). Il est crucial que ce langage permette d'obtenir des réponses aux requêtes en un temps « raisonnable ». Le LMD doit donc être optimisé, minimiser le nombre d'accès disques, et tout cela de façon totalement transparente pour l'utilisateur.

d) Administration centralisée des données (intégration) :

Toutes les données doivent être centralisées dans un réservoir unique commun à toutes les applications. En effet, des visions différentes des données (entre autres) se résolvent plus facilement si les données sont administrées de façon centralisée.

e) Non redondance des données :

Afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base.

f) Cohérence des données :

Les données sont soumises à un certain nombre de contraintes d'intégrité qui définissent un état cohérent de la base. Elles doivent pouvoir être exprimées simplement et vérifiées

automatiquement à chaque insertion, modification ou suppression des données. Les contraintes d'intégrité sont décrites dans le Langage de Description de Données (LDD).

g) Partage des données :

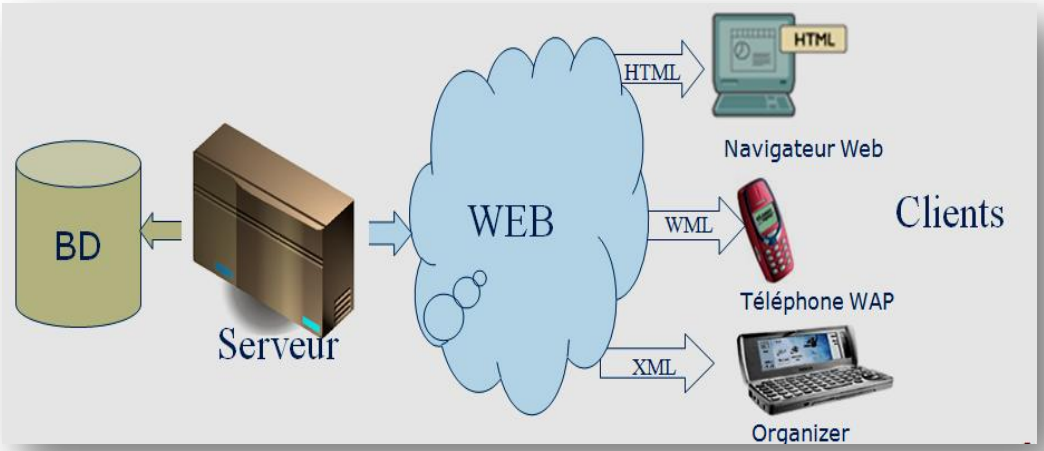
Il s'agit de permettre à plusieurs utilisateurs d'accéder aux mêmes données au même moment de manière transparente. Si ce problème est simple à résoudre quand il s'agit uniquement d'interrogations, cela ne lest plus quand il s'agit de modifications dans un contexte multiutilisateurs car il faut : permettre à deux (ou plus) utilisateurs de modifier la même donnée « en même temps » et assurer un résultat d'interrogation cohérent pour un utilisateur consultant une table pendant qu'un autre la modifie.

h) Sécurité des données :

Les données doivent pouvoir être protégées contre les accès non autorisés. Pour cela, il faut pouvoir associer à chaque utilisateur des droits d'accès aux données.

i) Résistance aux pannes :

Que se passe-t-il si une panne survient au milieu dune modification, si certains fichiers contenant les données deviennent illisibles ? Il faut pouvoir récupérer une base dans un état « sain ». Ainsi, après une panne intervenant au milieu dune modification deux solutions sont possibles : soit récupérer les données dans l'état dans lequel elles étaient avant la modification, soit terminer l'opération interrompue.



B. Quelques SGBD connus et utilisés

Il existe de nombreux systèmes de gestion de bases de données, en voici une liste non exhaustive :

PostgreSQL: http://www.postgresql.org / dans le domaine public ;

MySQL : http://www.mysql.org / dans le domaine public ;

Oracle : http://www.oracle.com / de Oracle Corporation ;

IBM DB2 : http://www-306.ibm.com/software/data/db2

Microsoft SQL : http://www.microsoft.com/sql

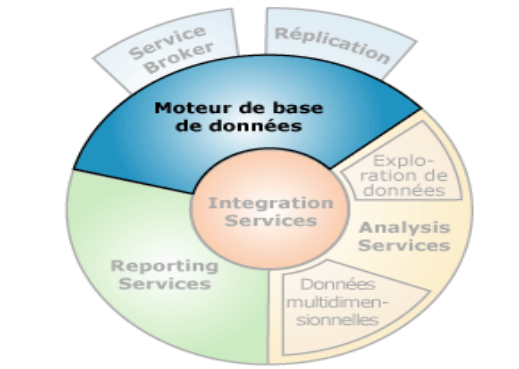
Sybase : http://www.sybase.com/linux

II. SQL server 2008 comme exemple d'un SGBDR

Dans Microsoft SQL Server 2008, les composants suivants proposent des fonctionnalités nouvelles ou améliorées. De plus, d'autres technologies offre des fonctionnalités qui s'intègrent étroitement à SQL Server 2008.

A. Composants SQL Server

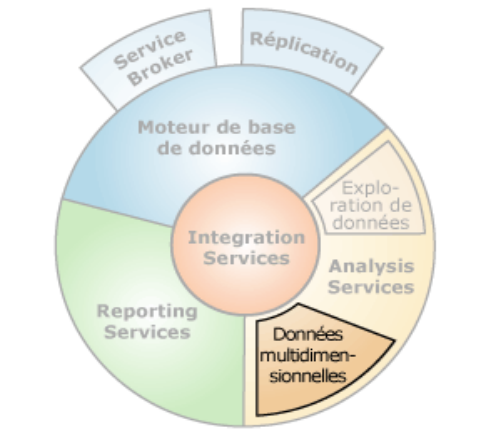
- **moteur de base de données**



Le Moteur de base de données est le service central qui permet de stocker, traiter et sécuriser les données. Grâce au moteur de base de données, il est possible de contrôler les accès et de traiter rapidement les transactions pour répondre aux besoins des applications consommatrices de données les plus exigeantes de votre entreprise.

Utilisez le Moteur de base de données pour créer des bases de données relationnelles pour le traitement de transactions en ligne ou des données de traitement analytique en ligne (OLAP). Ces opérations comprennent la création de tables pour le stockage des données, ainsi que les objets de base de données tels que les index, les vues et les procédures stockées pour l'affichage, la gestion et la sécurisation des données. Vous pouvez utiliser SQL Server Management Studio pour gérer les objets de base de données et Générateur de profils SQL Server pour capturer des événements serveur.

- **Analysis Services - Base de données multidimensionnelle**

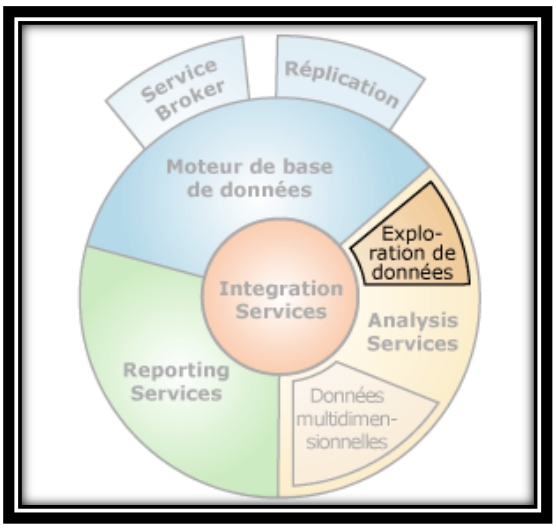


Microsoft SQL Server Analysis Services Les données multidimensionnelles vous permettent de concevoir, de créer et de manager des structures multidimensionnelles contenant des données de détail et agrégées depuis plusieurs sources de données, telles que les bases de données relationnelles, dans un modèle logique unifié et unique, pris en charge par les calculs intégrés.

Analysis Services Les données multidimensionnelles permettent une analyse rapide, intuitive et verticale de grandes quantités de données construites sur ce modèle de données unifié, disponible aux utilisateurs dans plusieurs langages et devises.

Analysis Services Les données multidimensionnelles fonctionnent avec les entrepôts de données, les mini-Data Warehouse, les bases de données de production et les magasins des données opérationnelles, en prenant en charge à la fois l'analyse des données d'historique et en temps réel.

- **Analysis Services - Exploration de données**



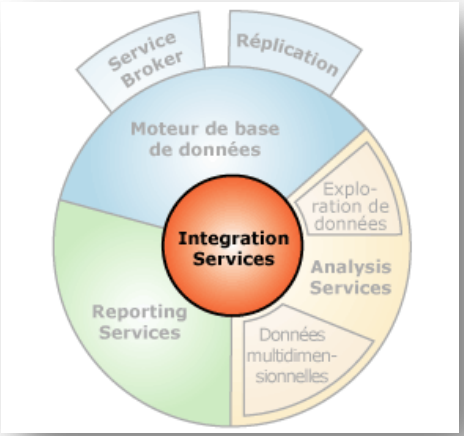
Microsoft SQL Server Analysis Services contient les fonctionnalités et les outils dont vous avez besoin pour créer des solutions d'exploration de données complexes.

- Un jeu d'algorithmes d'exploration de données standard.
- Le Concepteur d'exploration de données qui permet de créer, gérer et explorer des modèles d'exploration de données, puis de créer des prédictions à partir de ces modèles.

Le langage DMX (Data Mining Extensions) que vous pouvez utiliser pour gérer des modèles d'exploration de données et créer des requêtes de prédiction complexes.

Vous pouvez utiliser une combinaison de ces fonctionnalités et de ces outils pour dégager les tendances et les motifs présents dans vos données, et vous appuyer sur ces informations pour prendre des décisions réfléchies à propos de problèmes professionnels complexes.

- **Integration Services**



Microsoft Intégration Services est une plateforme qui permet de créer des solutions de transformation de données et d'intégration de données au niveau de l'entreprise. Intégration Services vous permet de résoudre des problèmes professionnels complexes en copiant ou en téléchargeant des fichiers, en envoyant des messages électroniques en réponse à des événements, en mettant à jour des entrepôts de données, en nettoyant et en explorant des données et en gérant des données et des objets SQL Server.

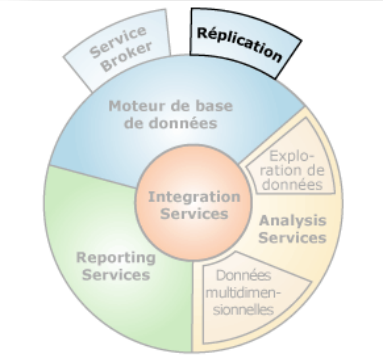
Les packages peuvent fonctionner en mode autonome ou de concert avec d'autres packages en réponse à des besoins professionnels complexes. Intégration Services peut extraire et transformer des données à partir d'un éventail de sources, tels que des fichiers de données XML, des fichiers plats et des sources de données relationnelles, puis charger les données dans une ou plusieurs destinations.

Intégration Services inclut un ensemble riche de tâches et de transformations intégrées, des outils pour construire des packages et le service Intégration Services permettant d'exécuter et de gérer des packages. Vous pouvez faire appel aux outils graphiques Intégration Services pour créer des

solutions sans écrire une seule ligne de code. Vous pouvez également programmer le modèle objet Intégration Services étendu pour créer des packages par programme et des tâches personnalisées de code et d'autres objets de package.

Réplication

La réplication repose sur un ensemble de technologies qui permettent de copier et de distribuer des données et des objets de base de données d'une base de données vers une autre, puis de synchroniser ces bases de données afin de préserver leur cohérence. Avec la réplication, vous pouvez distribuer des données en différents emplacements et à des utilisateurs distants ou mobiles sur des réseaux locaux et étendus, des connexions d'accès à distance, des connexions sans fil, et Internet.

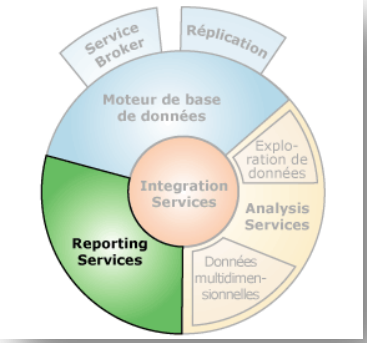


La réplication transactionnelle est généralement utilisée dans des scénarios serveur à serveur qui nécessitent un débit élevé, notamment pour l'amélioration de l'évolutivité et de la disponibilité, l'entrepôt de données et la création de rapports, l'intégration de données depuis plusieurs sites, l'intégration de données hétérogènes et le déchargement du traitement par lots. La réplication de fusion est conçue essentiellement pour les applications mobiles ou les applications de serveur distribuées contenant des conflits de données possibles. Les scénarios courants incluent l'échange de données avec des utilisateurs mobiles, les applications de point de vente aux consommateurs (POS, Consumer Point of Sale) et l'intégration des données à partir de plusieurs sites. La réplication de capture instantanée est utilisée pour fournir le jeu des données initiales pour la réplication transactionnelle et de fusion ; elle peut s'utiliser également lorsque des actualisations complètes des données sont nécessaires. Avec ces trois types de réplication, SQL Server fournit un système souple et puissant de synchronisation des données dans votre entreprise.

Outre la réplication, dans SQL Server 2008, vous pouvez synchroniser des bases de données à l'aide de Microsoft Sync Framework et de Sync Services for ADO.NET. Sync Services for ADO.NET fournit une API intuitive et flexible que vous pouvez utiliser pour générer des applications qui ciblent des scénarios de collaboration et hors connexion. Pour obtenir une vue d'ensemble de Sync Services for ADO.NET, consultez Microsoft Sync Framework. Pour obtenir une documentation complète, consultez le site Web MSDN.

Reporting Services

Microsoft SQL Server 2008 Reporting Services (SSRS) fournit une gamme complète d'outils et de services prêts à l'emploi pour vous aider à créer, déployer et gérer des rapports pour votre organisation, ainsi que des fonctions de programmation pour vous permettre d'étendre et de personnaliser vos fonctionnalités de création de rapports.



SQL Server 2008 Reporting Services (SSRS) est une plateforme serveur qui fournit des fonctionnalités complètes de création de rapports pour différentes sources de données. Reporting Services inclut un jeu complet d'outils que vous pouvez utiliser pour créer, gérer et remettre des rapports, et des interfaces de programmation d'application (API) qui permettent aux développeurs d'intégrer ou d'étendre le traitement des rapports et des données dans les applications personnalisées. Les outils Reporting Services fonctionnent au sein de l'environnement Microsoft Visual Studio et sont totalement intégrés aux outils et composants de SQL Server.

Avec Reporting Services, vous pouvez créer des rapports de type interactif, tabulaire, graphique ou libre à partir de sources de données XML, relationnelles et multidimensionnelles.

Vous pouvez publier des rapports, planifier le traitement de rapports ou accéder à des rapports à la demande. Reporting Services vous permet également de créer des rapports ad hoc basés sur des modèles prédéfinis, et d'explorer des données de manière interactive dans le modèle. Vous pouvez sélectionner divers formats d'affichage, exporter des rapports vers d'autres applications et vous abonner à des rapports publiés.

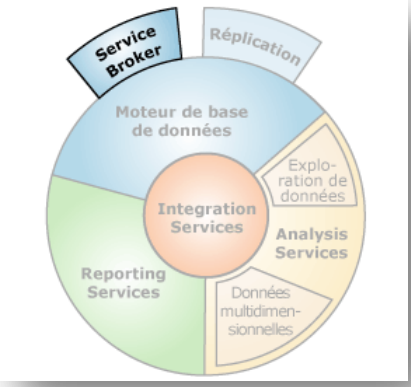
Les rapports que vous créez peuvent être consultés par le biais d'une connexion Internet ou en tant qu'application Microsoft Windows ou site SharePoint. Reporting Services fournit la clé de vos données de gestion.

Pour plus d'informations sur les autres composants, outils et ressources de SQL Server, consultez la Documentation en ligne de SQL Server

Service Broker

SQL Server Service Broker fournit la prise en charge native du Moteur de base de données SQL Server pour les applications de messagerie et de mise en file d'attente.

Cette opération permet aux développeurs de créer des applications perfectionnées qui utilisent des composants du Moteur de base de données pour communiquer entre des bases de données disparates. Les développeurs peuvent utiliser Service Broker pour créer facilement des applications fiables et distribuées.



Les développeurs d'applications qui utilisent Service Broker peuvent distribuer les charges de données sur plusieurs bases de données sans développer des mécanismes de messagerie et de communication complexes. Il est ainsi possible de réduire le travail de développement et de test puisque Service Broker gère les chemins de communication dans le contexte d'une conversation. Les performances sont aussi meilleures.

Par exemple, les sites Web qui prennent en charge des bases de données frontales peuvent enregistrer des informations et mettre les tâches intensives en file d'attente dans des bases de données dorsales. Service Broker veille à ce que toutes les tâches soient gérées dans le contexte des transactions pour garantir la fiabilité et la cohérence technique.

Résumé de la leçon

- SGBD fonctionnent selon un mode client/serveur
- Un SGBDR reçoit des requêtes de plusieurs clients et ceci de manière concurrente
- SQL server est constitué de plusieurs composants
  - Moteur de base de données
  - Analyse service, service repport,
  - intégration service

Révision de la leçon

1. Citer les avantages d'un SGBDR
2. Faite une recherche sur les SGBDR connus sur le marché
3. Quelle sont les attributions d'un gestionnaire de base de données

Travaux Dirigés

1. Faite une recherche sur l'internet pour faire un rapport sur les avantages de SQL server

2008 et les nouveautés qui il a apporté par apport à SQLserver 2000

Chapitre 2

Maîtrise des Concepts de base de sécurité

Chapitre 2

Leçon 1 : Maîtrise des Concepts de base de sécurité

Objectifs: vous serez à même d'effectuer les tâches suivantes :

- Maîtrise des Concepts de base de sécurité
- Choisir entre deux modes d'authentification
- Gérer les identifiants SQL server
- Gérer les rôles définis de serveur

I. Pourquoi gérer les droits ?

La gestion des droits d'une base de données est un domaine assez vaste et il existe de nombreux articles sur le sujet:

- **sécurité :** se protéger contre les attaques

Les réseaux informatiques sont de plus en plus souvent la cible d'attaques. Si, malgré les protections mises en place, un individu arrive à s'introduire dans votre base de données (en volant un mot de passe ou en se faisant passer pour une autre personne), il faut que son rayon d'action soit le plus limité possible.

- **protection :** empêcher les utilisateurs d'effectuer certaines actions

L'erreur est humaine, c'est un fait, et il peut arriver que des utilisateurs parfaitement habile à utiliser la base de données modifient par erreur certaines données qui devraient normalement être protégées. Une gestion correcte des droits permet de se prémunir contre ce genre de désagrément en empêchant l'exécution de certaines tâches

- **confidentialité :** restreindre l'accès à certaines données

Il est inconcevable que l'ensemble des salariés d'une entreprise aient accès aux données concernant,

par exemple, les salaires. La gestion des droits permet de restreindre la visibilité de certaines données (seules les données strictement nécessaires doivent être accessibles)

II. Gestion des droits dans le processus de développement

En fait, gérer les droits est finalement très simple si on s'y prend suffisamment tôt.

A chaque création d'un nouvel objet dans la base de données (ex: une table), il suffit de suivre les étapes suivantes :

1. on créé le nouvel objet
2. on fait la liste des rôles qui ont besoin d'accéder à cet objet et on affecte les droits en conséquence.
3. on teste si les personnes concernées ont accès à l'objet (ex: peuvent manipuler les éléments de la table Si une exception est levée à ce moment là, il manque des droits

Au cours de ce processus, de nouveaux profils d'utilisateurs peuvent apparaître (des utilisateurs doivent avoir des droits particuliers, que les autres n'ont pas), d'où la nécessité de définir de nouveaux **rôles** (et d'attribuer ces rôles à différents utilisateurs).

Dans la section suivante, nous expliquerons plus en détails ces différentes notions (rôles, utilisateurs...)

A. Notions de base

Pour accéder à une base de données, un **utilisateur** utilise une **connexion**. Un utilisateur peut être soit une personne physique, soit une application (script, batch).

Une base contient de nombreux objets (tables, vues, procédures stockées, fonctions…). Pour entreprendre certaines actions sur ces objets (consulter, exécuter, modifier…) l'utilisateur doit avoir les privilèges (aussi appelés droits) nécessaires. L'utilisateur peut obtenir ces droits de manière directe ou indirecte.

Le principe de base est donc finalement très simple. La mise en place peut être un peu plus complexe, comme nous allons le voir.

B. Gestion des droits : principes

Dans cette section, nous allons comparer 2 implémentations pour la gestion des droits. Pour que les exemples soient plus parlant, nous allons considérer une base de données avec :

- 10 utilisateurs aux droits différents suivant leur rôle
- 60 tables, vues, procédures stockées…

Pour prendre conscience des problèmes que peut représenter une mauvaise gestion des droits, nous allons imaginer l'implémentation suivante, très simple (mais très naïve) :

- pour chaque objet (table, vue, procédure stockée) on affecte les droits de manière individuelle à chaque utilisateur.

Cette implémentation posera les problèmes suivants :

- à chaque fois que l'on ajoute un objet dans la base, il faut affecter les droits pour chacun des 10 utilisateurs
- à chaque ajout d'un utilisateur, il faudra affecter les droits sur chacun des 60 objets de la base de données

En résumé, avec une implémentation aussi naïve, la gestion des droits est loin d'être aisée.

III. Implémentation d'une stratégie de sécurité

A. Choix du mode d'authentification :

SQL Server 2008 propose deux modes pour l'authentification des accès aux ressources de base de données : l'authentification Windows et l'authentification en mode mixte

Authentification Windows :

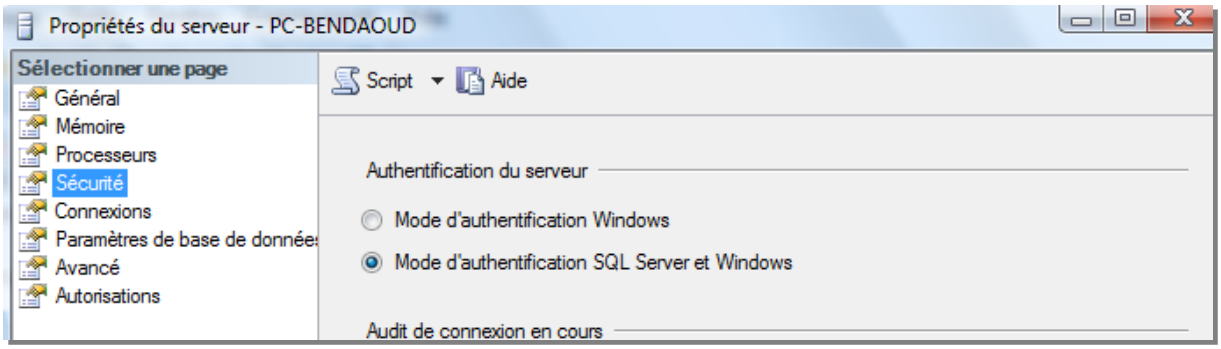
Seuls les utilisateurs Windows authentifié peuvent obtenir l'accès à l'instance SQL server. Vous devez ajouter un identifiant Windows a chaque utilisateur ou groupe Windows qui doit accéder a une instance SQL server c'est le mode d'authentification préconisé et c'est le mode par défaut, il est recommander car il permet de tirer profit de toutes les stratégies de sécurité centralisées de votre domaine active directory.

Authentification mixte :

dans ce mode, tant les identifiants de Windows que les identifiants SQL server (dont aucun n'est associé à un utilisateur du système d'exploitation) peuvent accéder a l'instance SQL server.

Comment choisir le mode d'authentification ?

Par l'intermédiaire de propriétés de votre instance SQL server :



IV. Gestion des rôles

Considérons l'implémentation suivante :

- on définit la notion de rôle (= groupe d'utilisateurs partageant les mêmes droits)
- un utilisateur appartient à un ou plusieurs rôle(s)
- pour chaque objet de la base, on affecte les droits aux différents rôles. Tous les membres d'un rôle donné hériteront des droits du rôle

Avec cette implémentation :

- à chaque fois que l'on ajoute un objet dans la base, il suffit d'affecter les droits à 1 rôle pour que tous les utilisateurs du rôle bénéficient des droits d'accès
- lorsqu'on crée un nouvel utilisateur, il suffit de l'ajouter à 1 groupe d'utilisateurs (rôle) pour qu'il bénéficie de tous les droits du rôle
- pour changer les droits d'un utilisateur, il suffit de changer son appartenance aux différents rôles

La gestion des droits devient alors nettement plus facile !

A. Qu'est-ce qu'un rôle exactement ?

Un **rôle**, c'est un ensemble de responsabilités.

Dans une entreprise, les employés ont diverses responsabilités. Chacune de ces responsabilités s'accompagne d'un certain nombre de tâches et l'entreprise doit fournir à ses employés les moyens nécessaires pour accomplir leur mission. Par ailleurs les employés peuvent avoir plusieurs responsabilités et donc cumuler les tâches.

En base de données, le principe est similaire : un rôle représente un ensemble de responsabilités au sein de l'application. Chacune de ces responsabilités s'accompagne d'un certain nombre de tâches (sous la forme de procédures stockées, par exemple). Pour accomplir ces différentes tâches, les membres d'un rôle doivent pouvoir accéder à différentes données (au travers de vues, procédures stockées, etc.), ce qui implique d'avoir les droits nécessaires pour accéder à ces données.

Lors de l'analyse, on doit donc identifier:

- les utilisateurs (users)
- les responsabilités (rôles)
- les tâches à effectuer et les moyens d'accès aux données (procédures stockées, vue, tables...)

Rôle en SQL server 2008

1. Rôle définis de serveur :

SQl server dispose d'un ensemble de rôles prédéfinis :

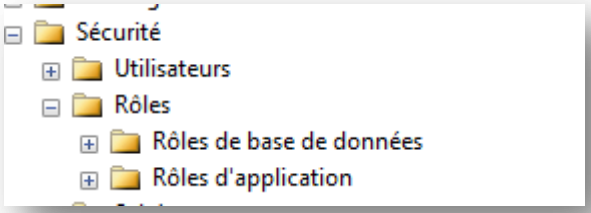
Member name	Description
BulkAdmin	Executer l'instruction BULKINSERT.
DBCreator	Créer et modifier des base de données
DiskAdmin	Gérer les fichiers sur le disque
ProcessAdmin	Gérer les processus qui s'executent dans une instance SQL server
SecurityAdmin	Gérer les identifiants de serveur
ServerAdmin	Configurer des réglages de portée serveur
SetupAdmin	Ajouter et supprimer des serveurs liés et executer certaines procedures stockées comme sp_serveroption.
SysAdmin	Effectuer toutes activités sur SQL Server.les privilèges de ce rôle comprennent toutes ceux des autres rôles

La syntaxe fondamentale d'ajout d'un identifiant à un rôle prédéfini et comme suite :

syntaxe :

```
  EXEC   master..sp_addsrvrolemember   @loginame   =   N'cn1',
@rolename = N'setupadmin'
  GO
```

2. Rôles définis de base de données :

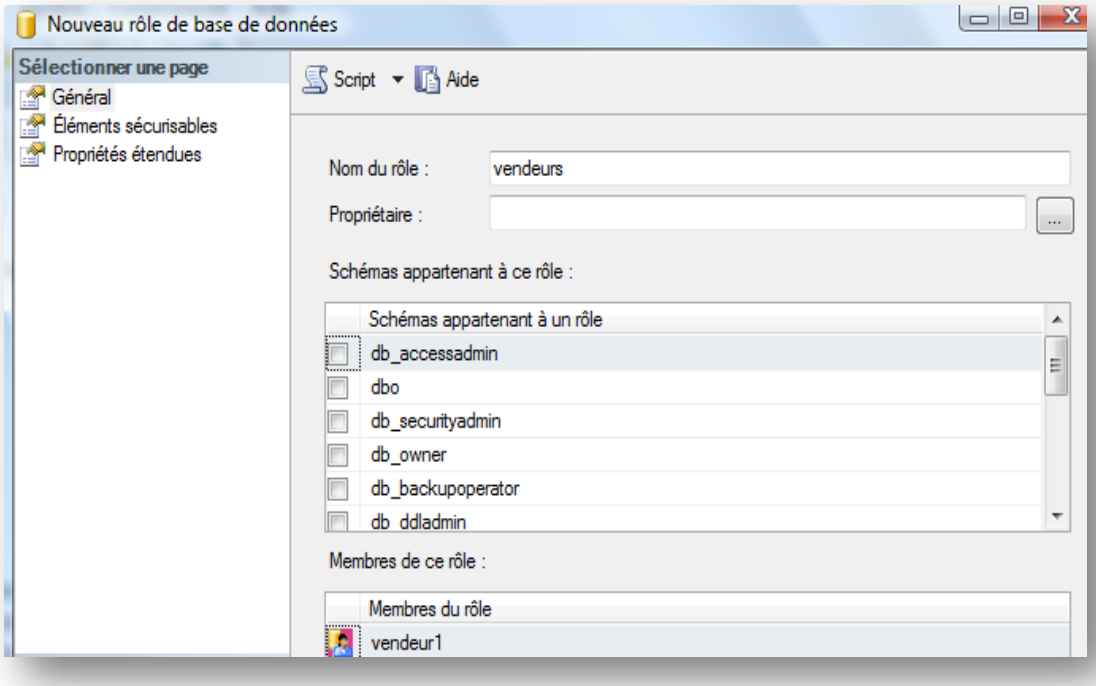


3. Rôle de base de données

Il existe des rôles prédéfinis de base de données SQL server Qui sont :

Nom du rôle au niveau de la base de données	Description
db_owner	Les membres du rôle de base de données fixe **db_owner** peuvent effectuer toutes les activités de configuration et de maintenance sur la base de données et peuvent également supprimer la base de données.
db_securityadmin	Les membres du rôle de base de données fixe **db_securityadmin** peuvent modifier l'appartenance au rôle et gérer les autorisations. L'ajout d'entités à ce rôle pourrait activer une élévation de privilèges involontaire.
db_accessadmin	Les membres du rôle de base de données fixe **db_accessadmin** peuvent ajouter ou supprimer l'accès à la base de données des connexions Windows, des groupes Windows et des connexions SQL Server.
db_backupoperator	Les membres du rôle de base de données fixe **db_backupoperator** peuvent sauvegarder la base de données.
db_ddladmin	Les membres du rôle de base de données fixe **db_ddladmin** peuvent exécuter n'importe quelle commande DDL (Data Definition Language) dans une base de données.
db_datawriter	Les membres du rôle de base de données fixe **db_datawriter** peuvent ajouter, supprimer et modifier des données dans toutes les tables utilisateur.
db_datareader	Les membres du rôle de base de données fixe **db_datareader** peuvent lire toutes les données de toutes les tables utilisateur.
db_denydatawriter	Les membres du rôle de base de données fixe **db_denydatawriter** ne peuvent ajouter, modifier ou supprimer aucune donnée des tables utilisateur d'une base de données.
db_denydatareader	Les membres du rôle de base de données fixe **db_denydatareader** ne peuvent lire aucune donnée des tables utilisateur d'une base de données.

L'administrateur de serveur peut définir ses propres rôles par Exemple (crée un rôle vendeurs qui contient tous les vendeurs)



1. Rôle d'application

Le rôle d'application est un rôle qui limite l'accès utilisateur à la base de données via des applications spécifiques. Les rôles d'application ne possèdent pas d'utilisateurs, si bien que la liste **Membres du rôle** n'est pas affichée lorsque l'option **Rôle d'application** est sélectionnée.

Transact SQL :

syntaxe :

```
CREATE APPLICATION ROLE application_role_name
    WITH PASSWORD = 'password' [ ,
DEFAULT_SCHEMA = schema_name ]
```

Exemple :

```
USE [vente] GOCREATE APPLICATION ROLE [roleApp2] WITH
DEFAULT_SCHEMA = [dbo], PASSWORD = N'azerty'
GO
```

Pour active les autorisations associées à un rôle d'application dans la base de données active, il faut exécuter la procédure **`sp_setapprole`**

syntaxe :

```
sp_setapprole [ @rolename = ] 'role',
        [ @password = ] { encrypt N'password' }
      |
        'password' [ , [ @encrypt = ] { 'none' | 'odbc' } ]
        [ , [ @fCreateCookie = ] true | false ]
    [ , [ @cookie = ] @cookie OUTPUT ]
```

Exemple :

```
exec sp_setapprole roleapp1,'azerty'
```

B. Gestion des droits : règles

Concernant la gestion des droits, quelques règles sont à respecter :

- **Identifier les rôles**

Il faut identifier les différents rôles selon les différents utilisateurs de l'application

- **Donner le minimum de droits**

Pour chaque rôle, on ne doit fournir que les droits nécessaires et suffisants à l'exécution des différentes tâches.

- **Interdire l'accès direct aux tables**

Les tables sont le support de données et leur contenu ne devrait pas être accessible directement. Par exemple, une table "salarié" peut contenir des informations personnelles qui ne doivent être accessibles qu'à un petit groupe d'individus. C'est pourquoi on accède au contenu d'une table au moyen de vues, procédures stockées, fonctions, etc. Ceci permet également de spécifier si l'accès aux données se fait en lecture seule ou si elle autorise les modifications.

En résumé, pour accéder au contenu d'une table on crée une vue (ou une procédure stockée, ou une fonction) pour laquelle on affecte les droits aux différents rôles contenant plusieurs utilisateurs. Ceci permet un meilleur contrôle des accès.

Gérer les droits le plus tôt possible

Plus on gère les droits de manière précoce, plus cette gestion est aisée car l'ajout de droits se fait au fur et à mesure du développement, et non de manière hâtive à la fin.

Au début du développement, quelques rôles sont clairement identifiés et d'autres seront ajoutés par la suite. Idem pour les utilisateurs. A chaque ajout de fonctionnalité dans le programme, on assigne les droits nécessaires pour son exécution (si les droits ne sont pas suffisants, on s'en rend très vite compte : une exception est levée). De cette façon, on gère les droits très facilement et avec un effort réduit.

Dans la suite de cet article, nous allons découvrir comment, dans le cas d'une application .Net au développement bien avancé, identifier les appels aux objets de la base de données (vues, procédures stockées…) en vue d'assigner les droits.

C. Notions supplémentaires Login/User

En SQL Server, on distingue d'un part la notion de **login** et d'autre part la notion de **user**.

Le **login**, c'est ce qui permet de se connecter à un serveur SQL Server.

Cependant un même serveur peut accueillir plusieurs bases de données et dans chacune de ces bases on définit différents **users** pour la gestion des droits. Il est ensuite nécessaire de faire le lien entre les logins et les users (ce que nous verrons par la suite)

Les transats SQL de création login /user :

	`CREATE LOGIN log11 with password ='azerty'`	**Créer une connexion log11 avec mot de passe 'azerty'**
	`create login [PC-BENDAOUD\Administrateur] from windows`	**Créer une connexion a partir d'un compte utilisateur**
	`alter login log11 with password='123'`	**Modifier le mot de passe de la connexion log11**
	`alter login log11 disable`	**Pour desactiver la connexion log11**
	`alter login log11 enable`	**Pour activer la connexion log11**
	`drop login log11`	**Supprimer la connexion log11**

Transact sql gestion utilisateurs (users)

	`create user alisalem for login log11`	**Créer l'utilisateur alisalem a partir de la connexion log11**
	`create role locataires`	**Pour créer un rôle de base de données**

exec sp_addrolememberlocataires,alisalem	**Pour ajouter un utilisateur a un groupe**
select * from sys.database_principals	Pour afficher les informations des utilisateurs
exec sp_who	liste des utilisateurs actuellement connectés

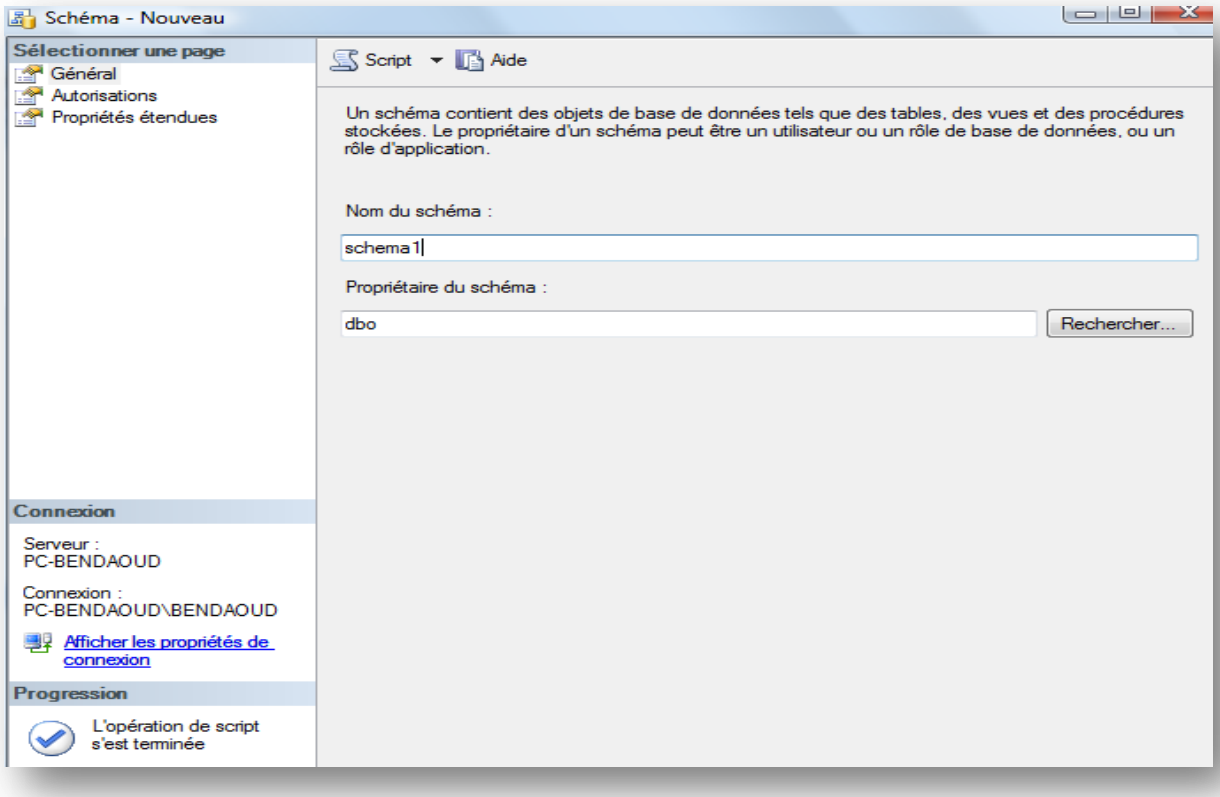
D. Les Schémas

L'objectif des schémas est de dissocier les utilisateurs de base de données des objets qu'ils vont être amenés à crée, toutefois, les objets ne sont pas laissés tels, ils sont regroupés logiquement en schema.il est ainsi possible de définir un schéma comme un ensemble logique d'objets à l'intérieur d'une base de données.

Les schémas facilitent le partage d'information entre plusieurs utilisateurs sans pour autant perdre au niveau de la securité.par exemple si plusieurs utilisateurs travaillent ensemble sur un même projet.ils vont tous se connecter en utilisant leur propre connexion et utilisateur de base de données, ce qui ne les empêche pas de travailler sur le même schéma et de partager ainsi les tables, vues, procédures, fonctions qui sont définies sur la base dans le cadre du projet.

Création d'un schéma :

Pour créer un schéma de base de données, il faut se positionner sur la base de données concernée puis développer le nœud sécurité et se positionner sur le nœud schéma , sélectionner Nouveau schéma.



syntaxe :

```
USE [AdventureWorksDW2008]
GO
CREATE SCHEMA [schema1] AUTHORIZATION [dbo]
GO
```

Révision de la leçon

1. Lesquelles des affirmations suivantes relatives aux schemas de base de données sont elles vraies ?
   A. Les schémas de base de données définissent le catalogue de base de données
   B. Les schémas regroupent les objets de base de données
   C. Les schémas regroupent des bases de données
   D. Les schémas définissent le catalogue des tables
2. Lesquelles des instructions suivante permettent-elles de créer un utilisateur de base de données nommé Ali associer a l'identifiant Ali
   a. Create user Ali from ali
   b. Create user Ali for login Ali
   c. Create user Ali for sql_login Ali
   d. Create user Ali
3. Lesquelles des affirmations suivantes relatives aux rôles de base de données, sont elle vraies ? choisissez tous les réponses pertinentes
   a. Il est possible d'imbriquer des rôles de base de données
   b. Les rôles de base de données sont prédéfinis

c. Vous pouvez ajouter de nouveaux rôles de base de données
d. Vous pouvez ajouter des rôles serveur prédéfinis à des rôles de base de données

Travaux pratiques

Réalisez les travaux pratiques se trouvant dans le dossier document

TP1
TP2

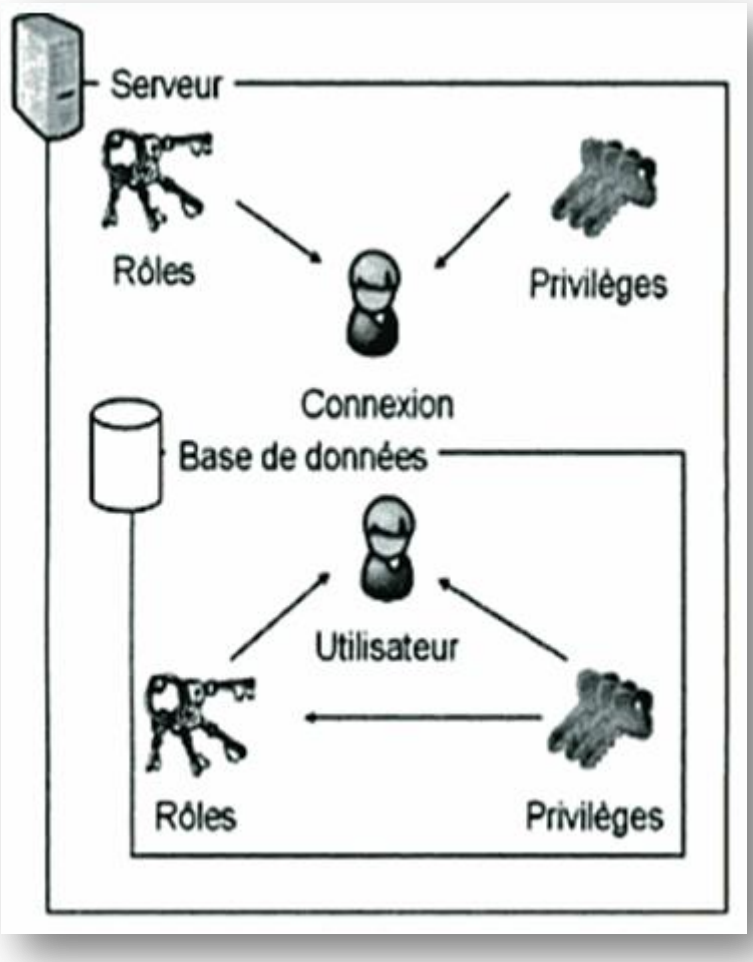
Chapitre 2

Leçon 2 : gestion des droits et permissions

Objectifs: vous serez à même d'effectuer les tâches suivantes

- Gérer les permissions
- Accorder un privilège
- Retirer privilège
- Interdire l'utilisation d'un privilège

V. Gestion des permissions



Tous les utilisateurs de base de données, y compris guest, appartiennent au groupe public.

Les droits qui vont être détaillés ci-dessous peuvent bien sûr être accordés directement à public

Les droits sont organisés de façon hiérarchique par rapport aux éléments sécurisables du serveur

Il est possible de gérer l'attribution de privilèges au niveau du serveur, de la base de données, du schéma ou bien directement de l'objet. Ainsi, les privilèges peuvent être accordés soit à un utilisateur de base de données, soit à une connexion.

SQL Server gère les privilèges avec trois types de mots clés :

- **GRANT**
- **REVOKE**
- **DENY**

C'est –à-dire d'un privilège peut être accordé(GRANT), ou bien retiré (REVOKE) s'il a été accordé. L'instruction DENY permet d'interdire l'utilisation d'un privilège particulier même si le privilège en question a été accordé soit directement soit par l'intermédiaire d'un rôle

A. Droit d'utilisation d'instructions

Les droits d'utilisation des instructions sql pour créer de nouveaux objets au sein de la base sont des autorisations pour réaliser l'exécution de certains ordres SQL.un utilisateur qui dispose de tels droits est capable par exemple de créer ses propres tables, ses procédures….. L'accord de ces droits peut être dangereux et comme pour tous les droits, doivent être accordés uniquement lorsque cela est nécessaire.

Les droits principaux d'instructions disponibles sont :

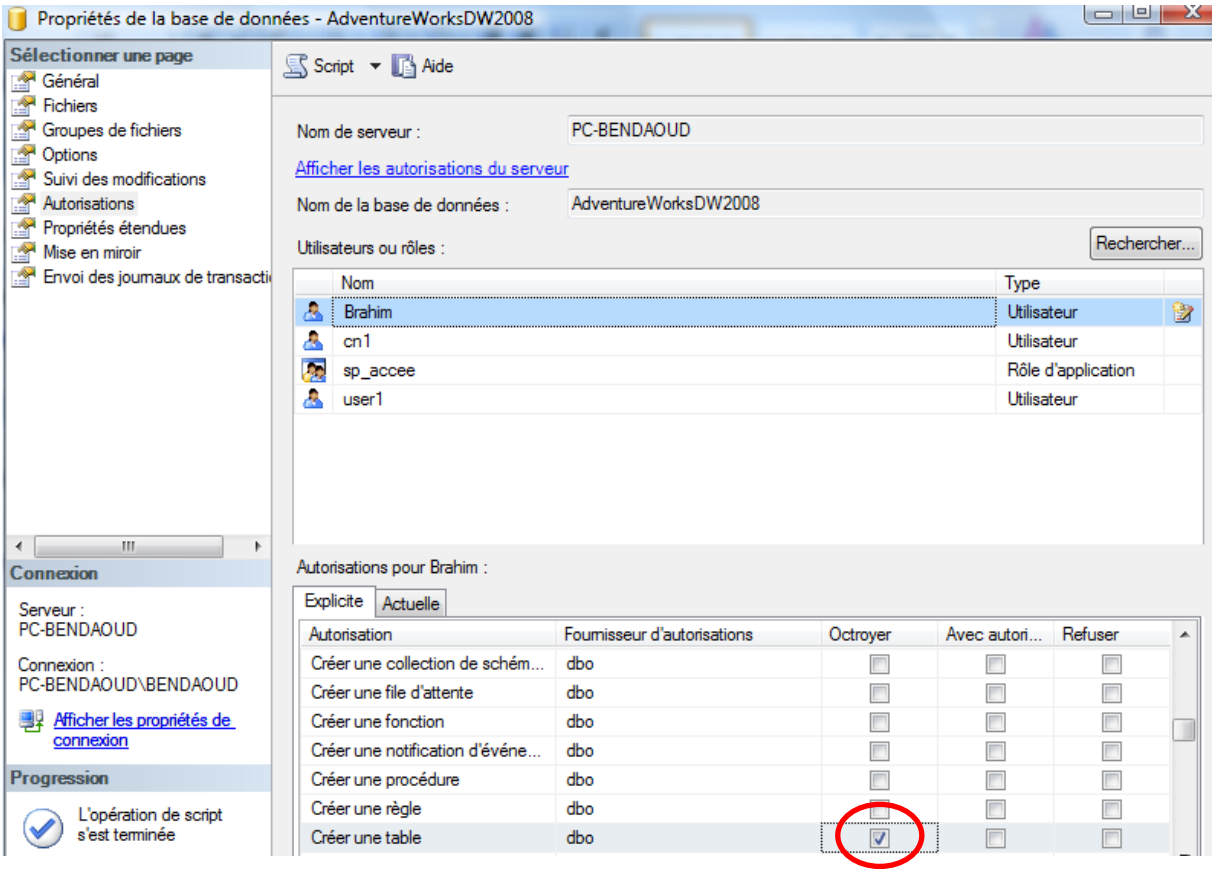
- Create database
- Create table
- Create procedure
- create function
- create table
- backup database
- create view

B. Autorisations

Ces droits sont administrés au niveau de la base de données par l'intermédiaire de la fenêtre propriétés

Exemple :

Le privilège create table est accordé à l'utilisateur de base de données Brahim par l'intermédiaire de la boite de propriétés de la base :



Exemple :

```
use [AdventureWorksDW2008]
GO
GRANT CREATE TABLE TO [Brahim]
GO
```

L'accord de privilège s'effectué en utilisant l'instruction GRANT dont la syntaxe est détaillée ci-dessous

Syntaxe :

GRANT permission[,……] TO utilisateur[,……..] [WITH GRANT OPTION]

Nom de la ou les permissions concernées par cette autorisation. Il est également possible d'utiliser le mot clé ALL à la place de citer explicitement la ou les permissions accordées. Toutefois ce terme ALL ne permet pas d'accorder des privilèges d'exécution de toutes les instructions mais simplement sur les instructions pour créer des bases de données, des tables des procédures, des fonctions, des vues ainsi que d'effectuer des sauvegardes de la base et du journal.

Utilisateur :

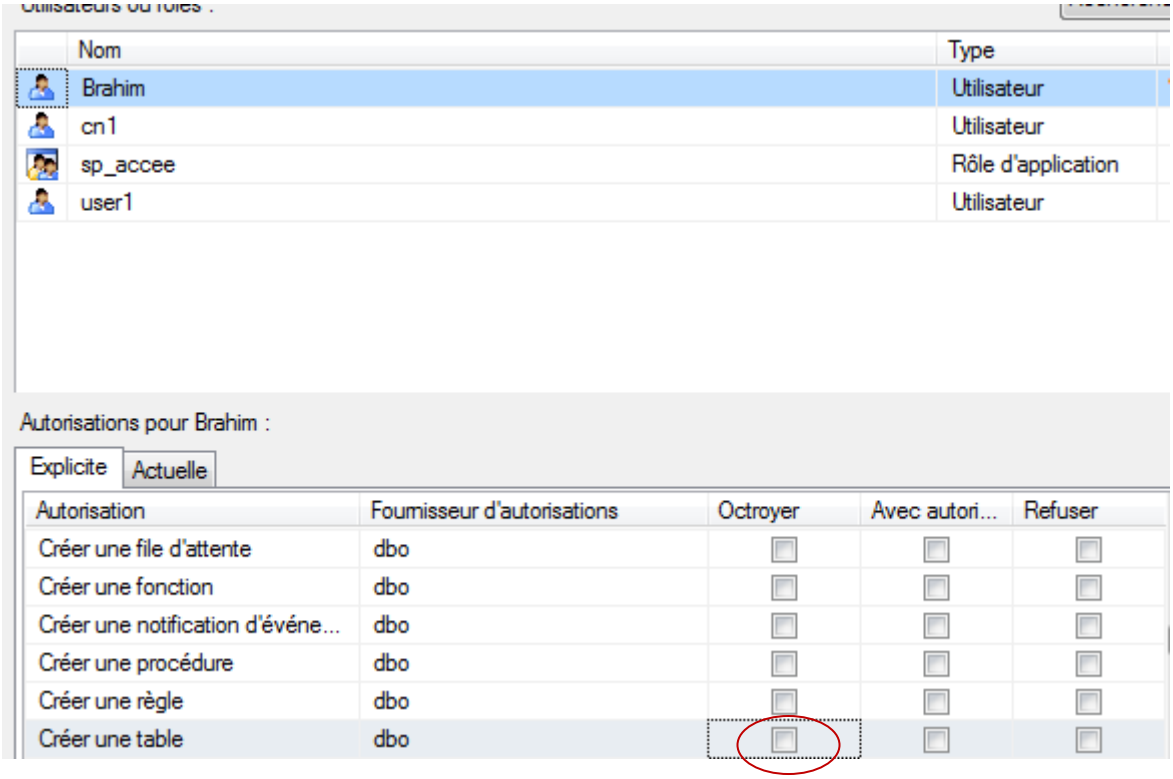
Nom d'utilisateur ou des utilisateurs de base de données qui reçoivent les permissions

WITH GRANT OPTION :

Si la permission est reçue avec ce privilège, alors l'utilisateur peut accorder la permission à d'autres utilisateurs de base de données

A. Retirer privilèges

Il est possible de retirer un privilège qui a été accordé à une entité de sécurité.si le privilège n'a pas été accordé à l'entité de sécurité, l'instruction est sans effet.



Exemple :

```
use [AdventureWorksDW2008]
GO
REVOKE CREATE TABLE TO [Brahim]
GO
```

Syntaxe :

```
REVOKE [GRANT OPTION FOR] permission[,….]
FROM utilisateur[,…….]
[CASCADE]
```

B. Interdire l'utilisation d'un privilège

L'instruction DENY permet d'interdire à un utilisateur l'utilisation d'un privilège, même s'il en reçoit la permission soit directement, soit par son appartenance à un groupe.

Exemple :

```
use [AdventureWorksDW2008]
GO
DENY CREATE TABLE TO [Brahim]
GO
```

C. Gestion des permissions sur les objets

Syntaxe :

GRANT <droits> **ON** <objet>
TO <usagers>
[**WITH GRANT OPTION**]

Les privilèges sont les clauses qui peuvent être autorisées/retirées à un utilisateur. Les principales sont:

- *DELETE*: privilège de supprimer les données d'une table
- *INSERT*: privilège d'ajouter des données à une table
- *SELECT*: privilège d'accéder aux données d'une table
- *UPDATE*: privilège de mettre à jour les données d'une table

IMPORTANT :

Qui peut accorder/retirer des permissions?

L'unique personne pouvant accorder ou retirer des droits sur un élément (table, vue ou index) est la personne qui l'a créée. Toutefois, il lui est possible de transmettre ce droit d'accorder/retirer des droits, auquel cas la personne recevant cet "honneur" aura le droit de transmettre ce "pouvoir" sur ces éléments

1. L'ACCORD DES PERMISSIONS

L'attribution de permissions

La clause *GRANT* permet d'attribuer des permissions à un ou plusieurs utilisateurs sur un ou plusieurs éléments de la base de données. La syntaxe de cette clause est la suivante:

```
  GRANT Liste_de_permissions ON Liste_d_objets TO
Liste_d_utilisateurs
  [WITH GRANT OPTION];
```

L'option *WITH GRANT OPTION* permet de définir si l'utilisateur peut lui-même accorder à un autre utilisateur les permissions qu'on lui accorde sur les éléments

Afin d'éviter à avoir à saisir l'ensemble des utilisateurs dans le cas d'une autorisation collective ou bien de citer l'ensemble des permissions il est possible d'utiliser des mots clés:

- Le mot clé *PUBLIC* en lieu et place de la liste d'utilisateurs permet d'accorder les privilèges sur le ou les objets à l'ensemble des utilisateurs
- Le mot clé *ALL* en lieu et place de la liste de permissions permet d'accorder tous les privilèges aux utilisateurs présents dans la liste

En précisant entre parenthèses un nom de colonne pour un privilège, il est possible de limiter le privilège à la colonne (ou la liste de colonnes) entre parenthèses, par exemple:

GRANT UPDATE(Nom,Prenom)
ON Etudiants
TO ALI,FATIMA,AHMED
WITH GRANT OPTION;

L'option *WITH GRANT OPTION* autorise donc plusieurs utilisateurs à accorder des permissions à un même utilisateur, il y a donc des règles à respecter lors du retraits des permissions à un utilisateur...

2. RETIRER DES PERMISSIONS

La révocation de permissions

La clause *REVOKE* permet de retirer des permissions à un ou plusieurs utilisateurs sur un ou plusieurs éléments de la base de données. La syntaxe de cette clause est la suivante:

Syntaxe :

```
REVOKE
[GRANT OPTION FOR] Liste_de_permissions
ON Liste_d_objets
FROM Liste_d_utilisateurs;
```

L'option *GRANT OPTION FOR* permet de supprimer le droit d'un utilisateur à accorder des permissions à un autre utilisateur.

Afin d'éviter à avoir à saisir l'ensemble des utilisateurs dans le cas d'une autorisation collective ou bien de citer l'ensemble des permissions il est possible d'utiliser des mots clés:

- Le mot clé *PUBLIC* en lieu et place de la liste d'utilisateurs permet de retirer les privilèges sur le ou les objets à l'ensemble des utilisateurs
- Le mot clé *ALL* en lieu et place de la liste de permissions permet de retirer tous les privilèges aux utilisateurs présents dans la liste

En précisant entre parenthèses un nom de colonne pour un privilège, il est possible de limiter la restriction de privilège à la colonne (ou la liste de colonnes) entre parenthèses, par exemple:

```
REVOKE
[GRANT OPTION FOR] UPDATE(Nom,Prenom)
ON Etudiants
FROM PUBLIC
```

L'attribution et la révocation de droits pose deux problème:

- lorsque l'on retire un droit à un utilisateur, il faut que ce droit soit retiré aux utilisateurs auxquels il a accordé le droit
- un utilisateur peut avoir reçu un droit de plusieurs utilisateurs

Il s'agit donc de retirer les droits des utilisateurs l'ayant obtenu de quelqu'un qui ne l'a plus en prenant en compte le fait qu'il peut l'avoir de plusieurs personnes simultanément...

La clause *REVOKE* étant implé:mentée différemment selon les SGBDR, il s'agit de consulter la documentation de celui-ci...

RETIRER LES PERMISSION

Syntaxe :

```
Simplified syntax for DENY
DENY { ALL [ PRIVILEGES ] }
      | permission [ ( column [ ,...n ] ) ] [
,...n ]
      [ ON [ class :: ] securable ] TO
principal [ ,...n ]
        [ CASCADE] [ AS principal ]
```

CONCLUSION :

les moyens pour faciliter la gestion des droits sont :

- définition de rôles
- ajout des droits, création des rôles au fur et à mesure du développement
- vérification que pour un rôle donné, on a bien les droits nécessaires et suffisants à l'exécution des différentes tâches

Résumé de la leçon

- SQL server gère les privilèges avec les trois instruction GRANT,REVOKE et DENY
- Les droits d'utilisation des instructions sql pour créer de nouveaux objets au sein de la base sont des autorisations pour réaliser l'exécution de certains ordres SQL.un utilisateur qui dispose de tels droits est capable par exemple de créer ses propres tables, ses procédures
-

Travaux Dirigés

Question cours :

1. Pourquoi utilise t- on les rôles en SQL server et données un exemple ?
2. Quel est la différence entre rôle de base de données et rôle définit par utilisateur ?
3. Quelles sont les caractéristiques du rôle public ?
4. Quel est la différence entre connexion et utilisateur (user)

Sécurité :

Une société de vente en ligne, possède une application en ligne, et serveur local sur lequel se trouve un SGBDR (SQL server 2008)

Les utilisateurs de cette application sont les clients qui peuvent visualiser les prix des produits et ajouter et modifier sur la table client

Les vendeurs qui peuvent ajouter des commendes, lignes commande et visualiser la table produit et client

Créer un script de transat SQL qui permet de gérer ces permissions

TRAVAIL A FAIRE VOIR RESSSOURCE

TRAVAUX PRATIQUES VOIR :

- TP1
- TP2

CHAPITRE 3

Leçon1 : Notion de base de données

Objectifs : vous serez à même d'effectuer les tâches suivantes :

- Notion de base d'une base de données
- Introduction au modèle relationnel
- Différentes opérations appliquées sur des relations
- Opération projection
- Opération restriction
- Jointure(equi jointure)

I. Notion de base de données

A. Définition

Plus précisément, on appelle base de données un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin den faciliter l'exploitation (ajout, mise à jour, recherche de données).

La gestion et l'accès à une base de données sont assurés par un ensemble de programmes qui constituent le *Système de gestion de base de données* (SGBD).

Un SGBD est caractérisé par le modèle de description des données qu'il supporte (hiérarchique, réseau, relationnel, objet). Les données sont décrites sous la forme de ce modèle, grâce à un Langage de Description des Données (LDD). Cette description est appelée *schéma.*

Une fois la base de données spécifiée, on peut y insérer des données, les récupérer, les modifier et les détruire. C'est ce qu'on appelle manipuler les données. Les données peuvent être manipulées non seulement par un Langage spécifique de Manipulation des Données (LMD) mais aussi par des langages de programmation classiques.

B. Conception d'une base de données

La conception et l'utilisation de bases de données relationnelles sur micro-ordinateurs n'est pas un domaine réservé aux informaticiens. C'est en tout cas ce que pensent beaucoup d'utilisateurs en voyant ce type de logiciel intégré aux suites bureautiques les plus connues (Microsoft Office Access).

Cependant la maîtrise d'un SGBDR micro (Système de Gestion de Bases de Données Relationnelles) est loin d'être aussi facile à acquérir que celle d'un logiciel de traitement de texte ou d'un tableur.

Plusieurs étapes sont nécessaires à la mise en place d'une base de données, dès lors que l'on a précisément défini ses besoins (ce qui n'est déjà pas chose facile !) : **la création de la structure de la base** sous forme de tables (tableaux de données) reliées entre elles par des données clés, **la conception des requêtes** qui permettront d'extraire ou de mettre à jour les informations qu'elle contient, **la conception de l'interface** homme-machine (écrans et états) qui rendra plus conviviale la saisie et la restitution des informations.

Le degré de difficulté dans la conception de l'interface varie beaucoup selon le logiciel utilisé qui est d'ailleurs le plus souvent différent du SGBDR.

La conception de la structure de la base de données, si elle est un peu complexe à appréhender, peut nécessiter, en amont, l'utilisation d'**outils de modélisation conceptuels** de type **entités-associations** (Modèle Conceptuel des Données de la méthode MERISE ou diagramme de classes du langage UML). Mais, même dans les cas les plus simples il faut obligatoirement connaître les concepts du **Modèle Relationnel**, sans quoi un utilisateur non averti pourra toujours arriver à créer une structure inadaptée et sera vite bloqué dans la conception des requêtes.

Il s'agit ici, d'étudier les principaux opérateurs de l'algèbre relationnelle servant de base à l'élaboration et à l'analyse (plan d'exécution) des requêtes.

Bon nombre d'utilisateurs qui voient les matériels informatiques et les logiciels changer tous les trois mois, seraient surpris d'apprendre que l'algèbre relationnelle a été définie par Codd en 1970.

Elle est à l'origine du langage SQL (Structured Query Language) d'IBM, langage d'interrogation et de manipulation de tous les SGBDR actuels (Oracle, PostgreSQL, MySQL, MS SQLServer, MS Access et tous les autres).

Une bonne maîtrise de l'algèbre relationnelle permet de concevoir n'importe quelle requête aussi complexe soit elle avant de la mettre en œuvre à l'aide du langage SQL.

Parmi les opérations de l'algèbre relationnelle, on dispose d'opérations classiques sur les ensembles (union, intersection, différence, produit cartésien) puis d'opérations propres (projection, sélection, jointure, division).

Sont également exposées ici des **opérations de calcul, de regroupement**, de comptage et de tri, non définies à l'origine par Codd mais très utiles.

Tous les opérateurs sont présentés à l'aide d'exemples clairs. Pris séparément, ils sont faciles à appréhender. La rédaction de requêtes (combinaison d'opérateurs) est illustrée par des exercices concrets.

Le langage SQL n'est abordé que dans le cadre des opérations évoquées ci-dessus. Seule l'instruction SELECT et ses multiples aspects sont donc présentés.

C. Introduction au Modèle Relationnel

L'exemple suivant, relatif à la gestion simplifiée des étapes du Tour de France 97, va nous servir à introduire le vocabulaire lié au modèle relationnel.

CodeEquipe	NomEquipe	DirecteurSportif
BAN	BANESTO	Eusebio UNZUE
COF	COFIDIS	Cyrille GUIMARD
CSO	CASINO	Vincent LAVENU
FDJ	LA FRANCAISE DES JEUX	Marc MADIOT
FES	FESTINA	Bruno ROUSSEL
GAN	GAN	Roger LEGEAY
ONC	O.N.C.E.	Manolo SAIZ
TEL	TELEKOM	Walter GODEFROOT

...	...	...

NuméroCoureur	NomCoureur	CodeEquipe	CodePays
8	ULLRICH Jan	TEL	ALL
31	JALABERT Laurent	ONC	FRA
61	ROMINGER Tony	COF	SUI
91	BOARDMAN Chris	GAN	G-B
114	CIPOLLINI Mario	SAE	ITA
151	OLANO Abraham	BAN	ESP
...	...	...	...

NuméroEtape	DateEtape	VilleDépart	VilleArrivée	NbKm
1	06-jul-97	ROUEN	FORGES-LES-EAUX	192
2	07-jul-97	ST-VALERY-EN-CAUX	VIRE	262
3	08-jul-97	VIRE	PLUMELEC	224
...	...	...	...	...

NuméroCoureur	NuméroEtape	TempsRéalisé
8	3	04:54:33
8	1	04:48:21
8	2	06:27:47
31	3	04:54:33
31	1	04:48:37
31	2	06:27:47
61	1	04:48:24
61	2	06:27:47
91	3	04:54:33
91	1	04:48:19
91	2	06:27:47
114	3	04:54:44
114	1	04:48:09
114	2	06:27:47
151	3	04:54:33
151	1	04:48:29
151	2	06:27:47
...	...	...

CodePays	NomPays
ALL	ALLEMAGNE
AUT	AUTRICHE
BEL	BELGIQUE
DAN	DANEMARK
ESP	ESPAGNE
FRA	FRANCE
G-B	GRANDE BRETAGNE
ITA	ITALIE
P-B	PAYS-BAS
RUS	RUSSIE
SUI	SUISSE
…	…

- Comme nous pouvons le constater, le modèle relationnel est un modèle d'organisation des données sous forme de Tables (Tableaux de valeurs) ou chaque Table représente une Relation, au sens mathématique d'**Ensemble**.

C'est ainsi que dans l'exemple présenté, figurent l'ensemble des Equipes, des Coureurs, des Etapes, des Temps réalisés par les coureurs à chacune des étapes, et enfin l'ensemble des pays.

- Les colonnes des tables s'appellent des **attributs** et les lignes des **n-uplets** (où n est le degré de la relation, c'est à dire le nombre d'attributs de la relation). Un attribut ne prend qu'une seule valeur pour chaque n-uplet. L'ordre des lignes et des colonnes n'a pas d'importance.
- Chaque table doit avoir une **clé primaire** constituée par un ensemble minimum d'attributs permettant de distinguer chaque n-uplet de la Relation par rapport à tous les autres. Chaque ensemble de valeurs formant la clé primaire d'un n-uplet est donc **unique** au sein d'une table.

C'est ainsi que dans la table COUREURS, chaque coureur a un NuméroCoureur différent.

Dans certains cas, plusieurs clés primaires sont possibles pour une seule table. On parle alors de clés candidates. Il faut alors en choisir une comme clé primaire.

- Les liens sémantiques (ou règles de gestion sur les données) existants entre les ensembles sont réalisés par l'intermédiaire de clés étrangères faisant elles-mêmes référence à des clés primaires d'autres tables.

C'est ainsi que dans la table COUREURS, la clé étrangère CodeEquipe (faisant référence à la clé primaire de même nom dans la table EQUIPES) traduit les deux règles de gestion suivantes :

- Un COUREUR appartient à une EQUIPE

- Une EQUIPE est composée de plusieurs COUREURS

Lien de type plusieur_plusieur
Il existe deux grands types de liens : **Un - Plusieurs** (comme le précédent) et **Plusieurs - Plusieurs**. La réalisation de ce dernier type de liens, un peu plus complexe, passe par l'utilisation d'une table intermédiaire dont la clé primaire est formée des clés étrangères des tables qu'elle relie.

C'est ainsi que la table des TEMPS réalisés à chaque étape par chacun des coureurs exprime les deux règles de gestion suivantes :

- Un COUREUR participe à plusieurs ETAPES
- Une ETAPE fait participer plusieurs COUREURS

Le modèle relationnel est le plus souvent décrit sous la forme suivante, les clés primaires étant soulignées et les clés étrangères marquées par un signe distinctif (ici par * ou bien #).

EQUIPES (CodeEquipe, NomEquipe, DirecteurSportif)

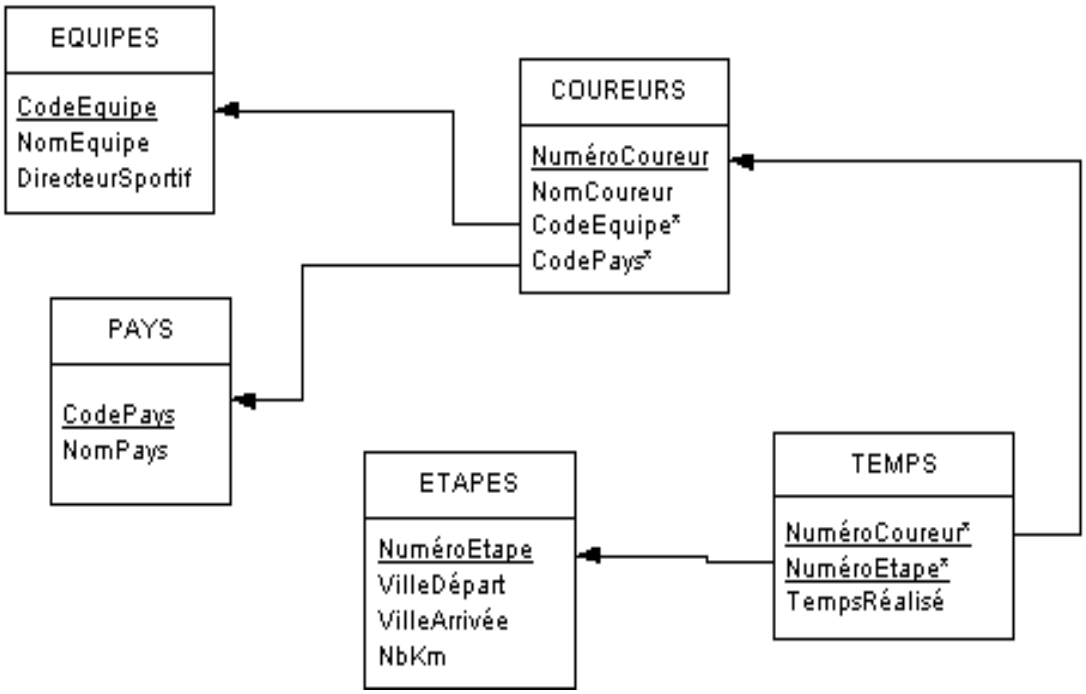
COUREURS (NuméroCoureur, NomCoureur, CodeEquipe*, CodePays*)

ETAPES (NuméroEtape, VilleDépart, VilleArrivée, NbKm)

TEMPS (NuméroCoureur*, NuméroEtape*, TempsRéalisé)

PAYS (CodePays, NomPays)

On peut aussi le représenter sous forme graphique, de manière à mieux visualiser et interpréter les liens :



- **Un COUREUR appartient à une EQUIPE**
- **Une EQUIPE est composée de plusieurs COUREURS**
- **Un COUREUR est originaire d'un PAYS**
- **Un PAYS est représenté par plusieurs COUREURS**
- **Un COUREUR participe à plusieurs ETAPES**
- **Une ETAPE fait participer plusieurs COUREURS**

conclusion

Dans le cadre d'un projet d'informatisation, la conception d'une base de données relationnelle passe d'abord par l'identification des objets de gestion (Coureurs, Etapes, …) et des règles de gestion du domaine modélisé (interviews des utilisateurs, étude des documents manipulés, des fichiers existants, …). Une fois énoncées et validées, ces règles nous conduisent automatiquement à la structure du modèle relationnel correspondant.

II. Différentes Opérations appliquées sur les relations

A. Opération PROJECTION

Syntaxe :
Formalisme : R = **PROJECTION (**R1, liste des attributs**)**

Exemple : coureurs

NuméroCoureur	NomCoureur	CodeEquipe	CodePays
8	ULLRICH Jan	TEL	ALL
31	JALABERT Laurent	ONC	FRA
61	ROMINGER Tony	COF	FRA
91	BOARDMAN Chris	GAN	G-B
114	CIPOLLINI Mario	SAE	ITA
151	OLANO Abraham	BAN	ALL
...	...	...	...

CodePays
ALL
FRA
B-G
ITA

R1 = PROJECTION (coureurs, codePays)

Cet opérateur ne porte que sur 1 relation.

Il permet de ne retenir que certains attributs spécifiés d'une relation.

On obtient tous les n-uplets de la relation ***à l'exception des doublons***.

B. Opération PROJECTION EN langage SQL

Syntaxe :

SELECT DISTINCT liste d'attributs **FROM** table **;**

SELECT liste d'attributs **FROM** table **;**

Exemple :

SELECT DISTINCT codePays FROM Coureurs ;

SELECT DISTINCT NomCoureur, codePays FROM Coureurs;

C. Opération RESTRICTION

Syntaxe :

Formalisme : R = **SELECTION (**R1, condition**)**

Exemple : R3 = SELECTION (Coureurs, codePays = "FRA")

NuméroCoureur	NomCoureur	CodeEquipe	CodePays
31	JALABERT Laurent	ONC	FRA
61	ROMINGER Tony	COF	FRA

Cet opérateur porte sur une relation.

Il permet de ne retenir que les n-uplets répondant à une condition exprimée à l'aide des opérateurs arithmétiques (=, >, <, >=, <=, <>) ou logiques de base (ET, OU, NON).

Tous les attributs de la relation sont conservés.

Un attribut peut ne pas avoir été renseigné pour certains n-uplets. Si une condition de sélection doit en tenir compte, on indiquera simplement : nomattribut "non renseigné

SELECT * FROM table **WHERE** condition **;**

Exemple :

SELECT * FROM Coureurs WHERE CodePays='FRA';

La condition de sélection exprimée derrière la clause WHERE peut être spécifiée à l'aide :

- des opérateurs de comparaison : =, >, <, <=, >=, <>
- des opérateurs logiques : AND, OR, NOT
- des opérateurs : IN, BETWEEN, LIKE, IS, ALL

Autres exemples :

Soit la table ETUDIANT(N°Etudiant, Nom, Age, CodePostal, Ville)

```
SELECT *
FROM ETUDIANT
WHERE Age IN (19, 20, 21, 22, 23) ;
```

Affiche la lise des étudiant qui ont un âgé appartenant a l'ensemble (19,20,21,22,23)

```
SELECT *
FROM ETUDIANT
WHERE Age BETWEEN 19 AND 23 ;
```

La liste des étudiant qui ont un âge compris entre 19 et 23

```
SELECT *
FROM ETUDIANT
WHERE CodePostal LIKE '42%' ;     // sous Access : LIKE "42*"
```

Affiche la liste des étudiants qui ont un codePoste qui commence par 42

```
SELECT *
FROM ETUDIANT
WHERE CodePostal LIKE '42___' ;   // sous Access : LIKE "42???"
```

Affiche la liste des étudiants qui ont un codePoste qui commence par 42 suivis par 3 caractéres

```
SELECT *
FROM ETUDIANT
WHERE Ville IS NULL ;
```

// Etudiants pour lesquels la ville n'est pas renseignée

```
SELECT *
FROM ETUDIANT
WHERE Ville IS NOT NULL ;
```

// Etudiants pour lesquels la ville est renseignée

```
SELECT *
FROM ETUDIANT
WHERE Age >= ALL (SELECT Age FROM ETUDIANT) ;
```

//Affiche la liste des étudiant les plus âgés

D. Opération JOINTURE (équi-jointure)

Formalisme : R = **JOINTURE (**R1, R2, condition d'égalité entre attributs**)**

Exemple :

PRODUIT DETAIL_COMMANDE

CodePrd	Libellé	Prix unitaire		N°cde	CodePrd	quantité

590A	HD 1,6 Go	1615	97001	590A	2
588J	Scanner HP	1700	97002	515J	1
515J	LBP 660	1820	97003	515J	3

Syntaxe :

R = JOINTURE (PRODUIT, DETAIL_COMMANDE,Produit.CodePrd=Détail_Commande.CodePrd)

A.CodePrd	Libellé	Prix unitaire	N°cde	B.CodePrd	quantité
590A	HD 1,6 Go	1615	97001	590A	2
515J	LBP 660	1820	97002	515J	1
515J	LBP 660	1820	97003	515J	3

- Cet opérateur porte sur 2 relations qui doivent avoir au moins un attribut défini dans le même domaine (ensemble des valeurs permises pour un attribut).
- La condition de jointure peut porter sur l'égalité d'un ou de plusieurs attributs définis dans le même domaine (mais n'ayant pas forcément le même nom).
- Les n-uplets de la relation résultat sont formés par la concaténation des n-uplets des relations d'origine qui vérifient la condition de jointure.

REMARQUE :

Des jointures plus complexes que l'équijointure peuvent être réalisées en généralisant l'usage de la condition de jointure à d'autres critères de comparaison que l'égalité (<,>, <=,>=, <>)

E. Opération JOINTURE (équi-jointure)

En SQL, il est possible d'enchaîner plusieurs jointures dans la même instruction SELECT.

- En SQL de base :

Syntaxe :

```
SELECT * FROM table1, table2, table3, ...
WHERE table1.attribut1=table2.attribut1 AND table2.attribut2=table3.attribut2
AND ...;
```

Exemple :

```
SELECT * FROM Produit, Détail_Commande
WHERE Produit.CodePrd=Détail_Commande.CodePrd ;
```

ou en utilisant des alias pour les noms des tables :

Exemple :

```
SELECT * FROM Produit A, Détail_Commande B
WHERE A.CodePrd=B.CodePrd ;
```

- Avec la clause **INNER JOIN** (jointure dite interne) à partir du SQL2, supportée aujourd'hui par tous les SGBDR :

Syntaxe :

```
SELECT *  FROM table1 INNER JOIN table2 ON table1.attribut1=table2.attribut1
INNER JOIN table3 ON table2.attribut2=table3.attribut3... ;
```

Le mot clé INNER est facultatif sur la plupart des SGBDR (sauf MS Access).

Cette notation rend plus lisible la requête en distinguant clairement les conditions de jointures, derrière ON, et les éventuelles conditions de sélection ou restriction, derrière WHERE. De plus, l'oubli d'un ON (et donc de la condition de jointure) empêchera l'exécution de la requête,

alors qu'avec l'ancienne notation, l'oubli d'une condition de jointure derrière WHERE, n'empêche pas l'exécution de la requête, produisant alors un bien coûteux produit cartésien entre les tables !

Le même exemple que précédemment en utilisant aussi les alias :

Exemple :

```
SELECT *
FROM Produit A INNER JOIN Détail_Commande B ON A.CodePrd=B.CodePrd ;
```

- En SQL2, outre la jointure classique (dite jointure interne), apparaissent les **jointures externes**.

On retiendra notamment les jointures externes Gauche (**LEFT OUTER JOIN**) et Droite (**RIGHT OUTER JOIN**).

Dans le cas d'une jointure externe gauche A->B, toute les lignes de la table A sont incluses même s'il ne leur correspond pas de ligne dans la table B.

Sur l'exemple précédent :

Exemple :

```
SELECT *
FROM Produit A LEFT OUTER JOIN Détail_Commande B ON A.CodePrd=B.CodePrd ;
```

Le résultat renvoyé est le suivant :

A.CodePrd	Libellé	Prix unitaire	N°cde	B.CodePrd	quantité
590A	HD 1,6 Go	1615	97001	590A	2
588J	**Scanner HP**	**1700**	**NULL**	**NULL**	**NULL**
515J	LBP 660	1820	97002	515J	1
515J	LBP 660	1820	97003	515J	3

Tous les produits apparaissent même si certains n'ont pas fait l'objet de commande (exemple : 588J). Les colonnes manquantes sont alors complétées par des valeurs NULL.

Travaux pratiques

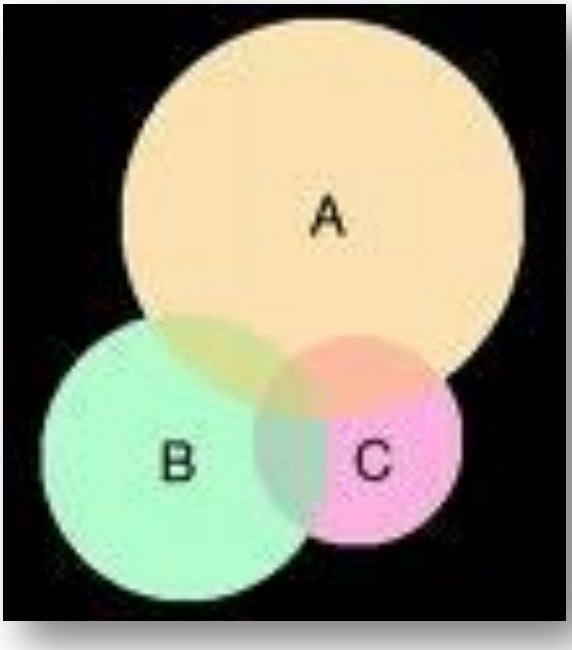
DEFINIR LES JOINTURES AVEC SSMS
COMMENT SAUVEGARDER UNE BASE DE DONNEES

Chapitre 3 :

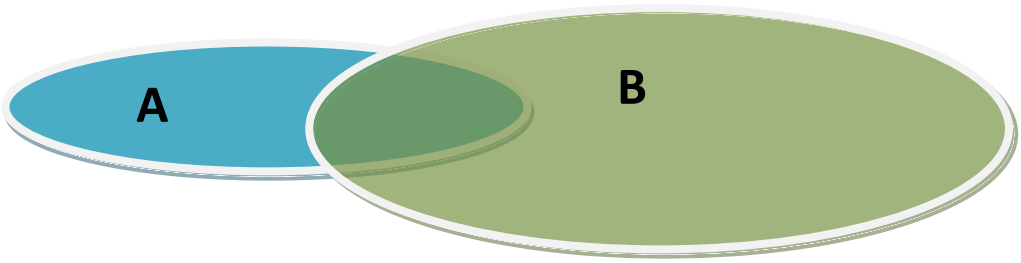
Leçon2 : les opérations ensemblistes

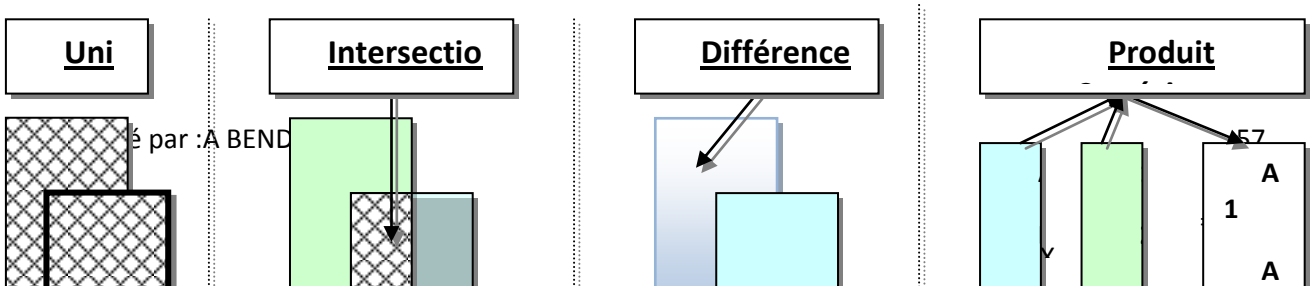
Objectifs : vous serez à même d'effectuer les tâches suivantes

- Opération union
- Opération intersection
- Opération Différence
- Opération produit cartésien
- Opération division
- Opération tri



I. Les opérations ensemblistes





A. Opération UNION

Formalisme : R = **UNION (**R1**,** R2**)**

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

E2

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

On désire obtenir l'ensemble des enseignants élus au CA ou représentants syndicaux.

R1 = UNION (E1, E2)

n°enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN

5	BERTRAND
6	MICHEL

- Cet opérateur porte sur deux relations qui doivent avoir le même (schema) nombre d'attributs définis dans le même domaine (ensemble des valeurs permises pour un attribut). On parle de relations ayant le même schéma.
- La relation résultat possède les attributs des relations d'origine et les n-uplets de chacune, avec élimination des doublons éventuels

B. Opération UNION avec transact SQL

Syntaxe :

```
SELECT liste d'attributs FROM table1
UNION
SELECT liste d'attributs FROM table 2 ;
```

Exemple :

```
SELECT n°enseignant, NomEnseignant FROM E1
UNION
SELECT n°enseignant, NomEnseignant FROM E2 ;
```

C. Opération INTERSECTION

Formalisme : R = **INTERSECTION (**R1**,** R2**)**

Exemple :

E1 : Enseignants élus au CA

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN

E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

5	BERTRAND

On désire connaître les enseignants du CA qui sont des représentants syndicaux.

R2 = INTERSECTION (E1, E2)

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN

- Cet opérateur porte sur deux relations de même schéma.
- La relation résultat possède les attributs des relations d'origine et les n-uplets communs à chacune.

D. Opération INTERSECTION en langage SQL

- En SQL de base, plusieurs possibilités :

Syntaxe :

SELECT attribut1, attribut2, ... **FROM** table1
WHERE attribut1 **IN (SELECT** attribut1 **FROM** table2**) ;**

SELECT attribut1, attribut2, ... **FROM** table1
WHERE EXISTS (SELECT * **FROM** table2 **WHERE**
table1.attribut1=table2.attribut1**)**

SELECT attribut1, attribut2, ... **FROM** table1
WHERE attribut1 **= ANY (SELECT** attribut1 **FROM** table2**) ;**

- ou avec l'opérateur INTERSECT (SQL2) :

Syntaxe :

SELECT attribut1**,** attribut2**,** ... **FROM** table1
INTERSECT
SELECT attribut1**,** attribut2**,** ... **FROM** table2 **;**

- ou avec une équi-jointure :

Syntaxe :

SELECT attribut1, attribut2, ... **FROM** table1 **INNER JOIN** table2 **ON** table1.attribut1 **=** table2.attribut1 **;**

Exemple :

SELECT n°enseignant, NomEnseignant FROM E1
WHERE n°enseignant IN (SELECT n°enseignant FROM E2) ;

ou

Exemple :

```
SELECT n°enseignant, NomEnseignant FROM E1
INTERSECT
SELECT n°enseignant, NomEnseignant FROM E2 ;
```

E. Opération DIFFERENCE

Formalisme : R = **DIFFERENCE (**R1, R2**)**

Exemple :

E1 : Enseignants élus au CA

n° enseignant	nom_enseignant
1	DUPONT
3	DURAND
4	MARTIN
5	BERTRAND

E2 : Enseignants représentants syndicaux

n°enseignant	nom_enseignant
1	DUPONT
4	MARTIN
6	MICHEL

On désire obtenir la liste des enseignants du CA qui ne sont pas des représentants syndicaux.

R3 = DIFFERENCE (E1, E2)

n°enseignant	nom_enseignant
3	DURAND
5	BERTRAND

- Cet opérateur porte sur deux relations de même schéma.
- La relation résultat possède les attributs des relations d'origine et les n-uplets de la première relation qui n'appartiennent pas à la deuxième.
- Attention ! DIFFERENCE (R1, R2) ne donne pas le même résultat que DIFFERENCE (R2, R1)

F. Opération DIFFERENCE avec langage SQL

- En SQL de base, plusieurs possibilités :

Syntaxe :
SELECT attribut1, attribut2, ... **FROM** table1 **WHERE** attribut1 **NOT IN (SELECT** attribut1 **FROM** table2**) ;**

Syntaxe :
SELECT attribut1, attribut2, ... **FROM** table1 **WHERE NOT EXISTS (SELECT** * **FROM** table2 **WHERE** table1.attribut1=table2.attribut1**) ;**

Syntaxe :
SELECT attribut1, attribut2, ... **FROM** table1 **WHERE** attribut1 **<> ALL (SELECT** attribut1 **FROM** table2**) ;**

- ou avec l'opérateur EXCEPT (SQL2) :

Syntaxe :
SELECT attribut1, attribut2, ... **FROM** table1 **EXCEPT** **SELECT** attribut1, attribut2, ... **FROM** table2 **;**

- ou encore, avec la jointure externe (SQL2),

si par exemple vous utilisez une version de MySQL qui ne dispose ni du EXCEPT, ni de la possiblité de SELECT imbriqués :

Syntaxe :

SELECT table1.attribut1, table1.attribut2,...
FROM table1 **LEFT JOIN** table2 **ON** table1.attribut1 = table2.attribut1
WHERE table2.attribut1 **IS NULL ;**

Exemple :

```
SELECT n°enseignant, NomEnseignant FROM E1
WHERE n°enseignant NOT IN (SELECT n°enseignant FROM E2) ;
```

Ou

Exemple :

```
SELECT n°enseignant, NomEnseignant FROM E1
EXCEPT
SELECT n°enseignant, NomEnseignant FROM E2 ;
```

ou encore

Exemple :

```
SELECT E1.n°enseignant, E1.NomEnseignant
FROM E1 LEFT JOIN E2 ON E1.n°enseignant = E2.n°enseignant
WHERE E2.n°enseignant IS NULL ;
```

Pour mieux comprendre cette dernière version, voici le résultat renvoyé par la jointure externe gauche entre E1 et E2 :

E1.n°enseignant	E1.NomEnseignant	E2.n°enseignant	E2.NomEnseignant
1	DUPONT	1	DUPONT

3	DURAND	NULL	NULL
4	MARTIN	4	MARTIN
5	BERTRAND	NULL	NULL

G. Opération PRODUIT CARTESIEN

Formalisme : R = **PRODUIT (**R1**,** R2**)**

Exemple :

Etudiants

n°étudiant	nom
101	DUPONT
102	MARTIN

Epreuves

libellé épreuve	coefficient
Informatique	2
Mathématiques	3
Gestion financière	5

Examen = PRODUIT (Etudiants, Epreuves)

n°étudiant	nom	libellé épreuve	coefficient
101	DUPONT	Informatique	2
101	DUPONT	Mathématiques	3
101	DUPONT	Gestion financière	5
102	MARTIN	Informatique	2
102	MARTIN	Mathématiques	3
102	MARTIN	Gestion financière	5

- Cet opérateur porte sur deux relations.
- La relation résultat possède les attributs de chacune des relations d'origine et ses n-uplets sont formés par la concaténation de chaque n-uplet de la première relation avec l'ensemble des n-uplets de la deuxième.

1. Opération PRODUIT CARTESIEN langage SQL

Syntaxe :

```
SELECT * FROM table1, table2 ;
```

Remarque : pour avoir un produit cartésien on ne met pas la clause de jointure

Exemple :

```
SELECT * FROM Etudiants, Epreuves ;
```

Pour SQL server 2008 on peut utiliser CROSS JOIN

Syntaxe :

```
SELECT    dbo.EMPLOYE.IdEmp, dbo.EMPLOYE.Nom, dbo.EMPLOYE.salaire,
dbo.DEPARTEMENT.NomDep
FROM      dbo.DEPARTEMENT CROSS JOIN
dbo.EMPLOYE
```

H. Opération Division

La division s'effectue sur deux tables (Dividende, Diviseur) possédant des colonnes à champ commun. Elle permet de répondre à la question suivante : quels sont tous les éléments d'une table qui sont associés à tous les éléments d'une autres table. (Exprime **« Pour tous les »**)

Exemple :

Quelles sont les commandes concernant tous les articles ?

Ligne de COM

Numcmd	Codart
1080	CS30
1050	CS10

Article

Codart
CS10
CS20

1070	CS10
1050	CS20
1070	CS20
1021	CS10
1050	CS30
1021	CS20
1070	CS30

CS30

Ligne de Com **DIV** Article

Numcmd
1050
1070

Réponse :

```
Select Numcmd from LC  as  LC1 Where
    Not exists (Select Codart from article Where
        Not exists ( Select * from LC  LC2 Where
                LC1.Numcmd= LC2.Numcmd
            And   LC1.Codart= LC2.Codart ) )
```

I. Opération TRI

Cette opération permet de faire le tri du jeu d'enregistrement qui est retourné par select

Syntaxe :

```
SELECT attribut1, attribut2, attribut3, ...
FROM table
ORDER BY attribut1 ASC, attribut2 DESC, ... ;
```

- ASC : par ordre croissant (Ascending)
- DESC : par ordre décroissant (Descending)

Exemple :

```
SELECT * FROM EMPLOYE ORDER BY NON DESC
```

100	**MOUHAMMED**	**CHEF**	**9000**	**10**
104	**MED FADEL**	**DIRECTEUR**	**12000**	**12**
101	**FATIMA**	**SECRETAIRE**	**7000**	**10**
102	**BRAHIM**	**DIRECTEUR**	**9900**	**11**
103	**AHMED SALEM**	**CADRE**	**11000**	**12**

Remarque : par défaut le tri se fait par ordre croissant si l'on ne précise pas ASC ou DESC.

1. Exercice d'application

Exercice d'application N°1 :

Soit le modèle relationnel suivant relatif à une base de données sur des représentations musicales :

REPRESENTATION (n°représentation, titre_représentation, lieu)
MUSICIEN (nom, n°représentation*)
PROGRAMMER (date, n°représentation*, tarif)

Remarque : les clés primaires sont soulignées et les clés étrangères sont marquées par *

Questions :

1 - Donner la liste des titres des représentations.

2 - Donner la liste des titres des représentations ayant lieu à l'opéra Bastille.

3 - Donner la liste des noms des musiciens et des titres des représentations

auxquelles ils participent.

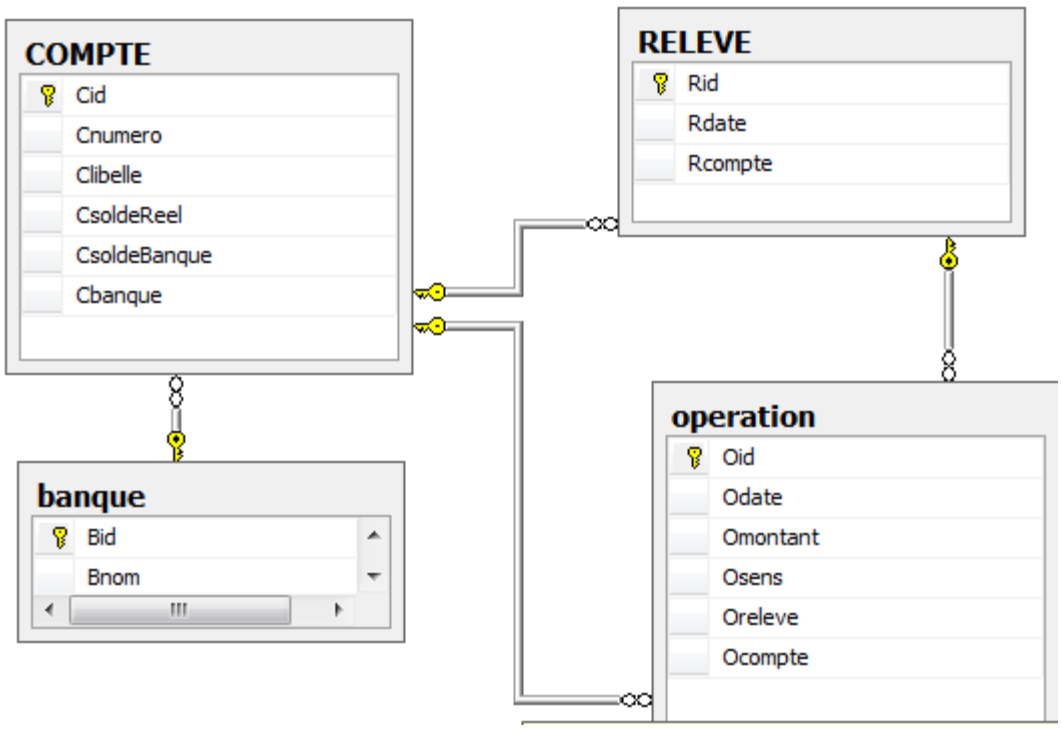
4 - Donner la liste des titres des représentations, les lieux et les tarifs pour la journée du 14/09/96.

Chapitre 4

Leçon 1 : Langage de définition des données

Objectifs: A la fin de cette leçon, vous saurez

- ***Créer des tables***
- ***Définir les types de données***
- ***Les tableaux temporaires***



I. Les types de données

Un type de données limite le type des données qui peuvent être stockées dans une colonne et, dans certains cas, limite la plage des valeurs acceptables de la colonne. Le type de donnée choisi pour une colonne est la plus importante des décisions que vous prenez quant a votre base de données.si vous choisissez un type de données trop restrictif, les applications ne pourront pas stocker des données qu'elles sont censées traiter, ce qui mène à un travail de conception énorme, en revanche si le type de données est trop laxiste, vous consommez trop d'espace mémoire

Il existe sept catégories de types de données en sql server

- Numérique exact (stock des numérique précis avec ou sans décimal)
- Numérique approximatif (stock des valeurs numérique avec ou sans décimal)
- Monétaire (stock des valeurs avec des décimales précision 4 décimales)
- Date &heure (stock des informations de date et/ou d'heure et permet de contraintes chronologique refus d'un 30 février)
- Caractère (stock des valeurs fondées sur des caractères de longueur variable)
- Binaire (stock des données binaires 0 ou1)
- But spécial (type de données complexe comme XML)

TYPE	Member name	Description
numérique exect	**BigInt**	A 64-bit signed integer.
	Decimal	A fixed precision and fixed scale numeric value between -1038 -1 and +1038 -1.
	Int	A 32-bit signed integer.
	Numeric	A fixed precision and fixed scale numeric value between -1038 -1 and +1038 -1.
	SmallInt	A 16-bit signed integer.
	TinyInt	An 8-bit unsigned integer.
numériq ue approximati fs	**Float**	An 8-byte floating point number within the range of -1.79E +308 through 1.79E +308.
	Real	A 4-bit floating point number within the range of -3.40E +38 through 3.40E +38.
monétaires	**Money**	A Decimal system object value that specifies a currency value ranging from -263 (or -922,337,203,685,477.5808) to 2 63 -1 (or +922,337,203,685,477.5807) with an accuracy of 1 in 10,000 of a currency unit.
	SmallMoney	A Decimal system object value that specifies a currency value ranging from -214,748.3648 to +214,748.3647 with an accuracy of 1 in 10,000 of a currency unit.
date/heure	**DateTime**	A DateTime system object value that specifies a date and time between January 1, 1753 and December 31, 9999 to an accuracy of 3.33 milliseconds.
	SmallDateTim e	A DataTime system object value that specifies a date and time between January 1, 1900 and June 6, 2079 to an accuracy of one minute.
	Date	Date object represents any valid Gregorian calendar date between '0001-01-01' CE and '9999-12-31' CE.
	DateTime2	DateTime2 is considered an extension of the existing DATETIME object with a large date range and large default fractional precision. Values that represent any valid Gregorian calendar date between '0001-01-01' CE and '9999-12-31' CE combined with any valid time of day based on a 24-hour clock.
	DateTimeOffs et	DateTimeOffset returns valid Gregorian calendar date between `0001-01-01' and '9999-12-31' with any valid time of day based on a 24 hour format between '00:00:00' and max '23:59:49.9999999'. Included in the DateTimeOffset is a time zone offset that must be between '-14:00' and '+14:00'.

	Time	Time object returns values for any valid time of day based on a 24 hour clock between '00:00:00' and max '23:59:59:9999999'.
	Timestamp	An automatically generated byte array value, which is guaranteed to be unique within a database.
caractères	**Char**	A fixed-length byte array of non-Unicode (256 code page) characters ranging between 1 and 8,000 characters.
	NText	A variable-length byte array of Unicode data with a maximum length of 230 - 1 (or 1,073,741,823) characters.
	NVarChar	A variable-length byte array of Unicode characters ranging between 1 and 2^63 characters.
	NVarCharMax	The NVARCHAR(MAX) type.
	Text	A variable-length byte array of non-Unicode (256 code page) data with a maximum length of 231 -1 (or 2,147,483,647) characters.
	VarChar	A variable-length byte array of non-Unicode (256 cod epage) characters ranging between 1 and 2^64 characters.
	VarCharMax	A VARCHAR(MAX) type.
	NChar	A fixed-length byte array of Unicode characters ranging between 1 and 4,000 characters.
binaire	**Binary**	A fixed-length byte array ranging between 1 and 8,000 bytes.
	Image	A variable-length byte array ranging from 0 to 231 -1 (or 2,147,483,647) bytes.
	VarBinary	A variable-length byte array ranging between 1 and 2^64 bytes.
	VarBinaryMax	A VARBINARY(MAX) type.
spécialisés	**Bit**	An unsigned bit value that can be 0, 1, or a null reference.
	Geography	Geography spatial type represents data in a round-earth coordinate system. The SQL Server geography data type stores ellipsoidal (round-earth) data, such as GPS latitude and longitude coordinates..
	Geometry	Geography spatial type represents data in a Euclidean (flat) coordinate system.
	Variant	A special data type that can contain numeric, string, binary, date data, and the SQL Server values Empty and Null. This data type is assumed if no other type is declared.
	Xml	An XML data type.

II. Création des tables

L'ordre CREATE TABLE permet de créer une table en définissant le nom et le type de caque colonnes de la table .

Syntaxe :

```
CREATE TABLE nom_table
(colonne1      type1,
Colonne2       type2,
……..
……..
……..)
```

nom_table est le nom de la tablecolonne1,colonne2….sont les noms des colonnes type1,type2….sont les types des données, qui seront contenu dans les colonnes.

On peut ajouter après la description d'une colonne l'option NOT NULL qui interdira que cette colonne contienne une valeur null.on peut aussi ajouter des contraintes d'intégrités.

Exemple :

```
CREATE TABLE Article
(ref  char(10) NOT NULL ,
prix  numeric(18,2),
dateMaj  Datetime  )
```

A. Les tables temporaires :

Un tableau temporaire, est une structure temporaire de table .elle peut être globale ou locale et peut être créée par n'importe quel utilisateur. Toutes les tables temporaires sont créées dans la base de données tempdb.

Une table temporaire n'est visible que pour l'utilisateur qui l'a créée et seulement a l'intérieur de la connexion qui a été employé pour la création de la table. En revanche les tables temporaires sont détruites automatiquement lorsque la connexion à laquelle elles sont associées est fermée .

Pour créer une table temporaire locale il faut précéder le nom de la table par un #

Syntaxe :

```
Create table #ligne_comme
(code int, datecom datetime,codeCl  int)
```

Alors que , les tables temporaires globales sont visible pour tous les utilisateurs de l'instance SQL elles sont détruites lorsque la dernière connexion est fermée.

Syntaxe :

```
Create table  ##ligne_comme
(code int, datecom datetime, codeCl  int)
```

Résumé de la leçon

- Les tables, l'élément de construction fondamental de toutes base de données
- Pour procurer à une table la structure nécessaire, vous devez choisir pour les colonnes entre les types de données numeriques,texte, date/heure et binaires afin de stocker correctement les données
- Une fois une table est définie, vous devez accorder des permissions sur cette table pour permettre aux utilisateurs de récupérer et de manipuler des données

Travaux pratiques
COMMENT CREER UN SCRIPT DE CREAATION D'UNE BASE DE DONNEES SOUS SQL SERVER 2008
METHODE PEDAGOGIQUE : UTILISER UN DIDACTICIEL

VOIR VIDEO

Chapitre 4

Leçon 2 : mise en œuvre des contraintes

Objectifs: A la fin de cette leçon, vous saurez

- ***Créer des contraintes***
  - ✓ ***NULL / NOT NULL***
  - ✓ ***Default***
  - ✓ ***Primary key***
  - ✓ ***Foreign key***
  - ✓ ***Check***
  - ✓ ***unique***
- ***Modifier un tableau (Alter table)***

I. Les contraintes de colonnes (verticales)

Une colonne peut donc recevoir les contraintes suivantes :

- **NULL / NOT NULL** : précise si une valeur doit obligatoirement être saisie dans la colonne ou non
- **DEFAULT** : valeur par défaut qui est placée dans la colonne lors des insertions et de certaines opérations particulières, lorsque l'on a pas donné de valeur explicite à la colonne
- **COLLATE** : précise la séquence de collation, c'est à dire l'ordre des caractères pour le tri et les éventuelles confusions possible (minuscules/majuscules, caractères diacritiques distinct ou non).
- **PRIMARY KEY** : précise si la colonne est la clef de la table. ATTENTION : nécessite que la colonne soit NOT NULL
- **UNIQUE** : les valeurs de la colonne doivent être unique ou NULL, c'est à dire qu'à l'exception du marqueur NULL, il ne doit jamais y avoir plus d'une fois la même valeur (pas de doublon)
- **CHECK** : permet de préciser un prédicat qui acceptera la valeur s'il est évalué à vrai
- **FOREIGN KEY** : permet, pour les valeurs de la colonne, de faire référence à des valeurs préexistantes dans une colonne d'une autre table. Ce mécanisme s'appelle intégrité référentielle

	REMARQUE : dans un tableau ces contraintes peuvent être placées plusieurs fois, à l'exception de la contrainte de clef PRIMARY KEY qui ne peut être créée qu' une seule fois .

Lorsqu'au cours d'un ordre SQL d'insertion, de modification ou de suppression, une contrainte n'est pas vérifiée on dit qu'il y a "**violation**" de la contrainte et les effets de l'ordre SQL sont totalement annulé (ROLLBACK).

A. Obligatoire ([NOT] NULL)

On peut rendre la saisie d'une colonne obligatoire en apposant le mot clef NOT NULL. Dans ce cas, il ne sera jamais possible de faire en sorte que la colonne soit vide. Autrement dit, la colonne devra toujours être renseignée lors des ordres d'insertion INSERT et de modification UPDATE. Si l'on désire que la colonne puisse ne pas être renseignée (donc accepter les marqueurs NULL), il n'est pas nécessaire de préciser le mot clef NULL, mais il est courant qu'on le fasse par facilité de lecture.

Exemple :

```
CREATE TABLE T_PERSONNE1
(PRS_ID              INTEGER     NOT NULL
 PRS_NOM             VARCHAR(32) NOT NULL,
 PRS_PRENOM          VARCHAR(32) NULL,
 PRS_DATE_NAISSANCE DATE)
```

Crée une table dont les colonnes PRS_ID et PRS_NOM doivent obligatoirement être renseignés.

Exemple : insertion et modification acceptées

```
INSERT INTO T_PERSONNE1 VALUES (1, 'DUPONT', NULL, NULL)
INSERT INTO T_PERSONNE1 (PRS_ID, PRS_NOM) VALUES (2,
'DURAND')
```

Exemple : insertion et modification refusées

```
   INSERT INTO T_PERSONNE1 VALUES (3, NULL, 'Marcel', NULL)
   INSERT INTO T_PERSONNE1 (PRS_ID, PRS_PRENOM) VALUES (4,
'Jean')
```

IMPORTANT :

les colonnes concourantes à la définition d'une clef de table doivent impérativement posséder une contrainte NOT NULL.

I. Mise en œuvre des contraintes

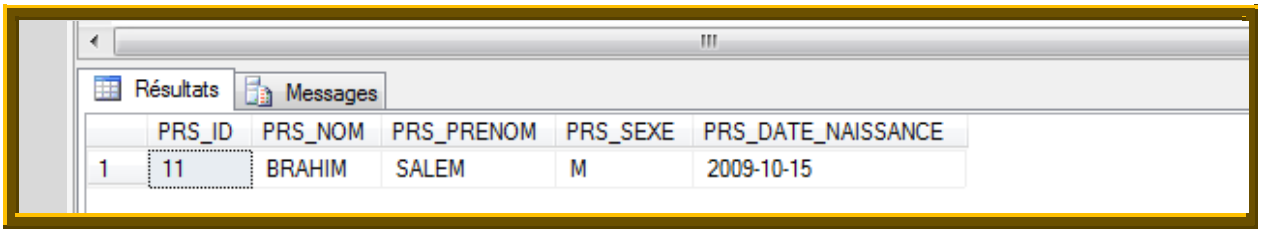
A. Valeur par défaut (DEFAULT)

La contrainte DEFAULT permet de préciser une valeur qui sera automatiquement insérée en l'absence de précision d'une valeur explicite dans un ordre d'insertion. Certains autres ordres SQL, comme la gestion de l'intégrité référentielle peuvent faire référence à cette valeur par défaut. Seule une valeur explicite, un marqueur NULL ou la valeur retournée par les fonctions suivantes sont acceptées : CURRENT_DATE, CURRENT_TIME[(p)], CURRENT_TIMESTAMP[(p)], LOCALTIME[(p)], LOCALTIMESTAMP[(p)], USER, CURRENT_USER, SESSION_USER, SYSTEM_USER.

Exemple :

```
   CREATE TABLE T_PERSONNE2
   (PRS_ID              INTEGER,
PRS_NOM             VARCHAR(32),
   PRS_PRENOM          VARCHAR(32),
PRS_SEXE            CHAR(1)      DEFAULT 'M',
 PRS_DATE_NAISSANCE DATE          DEFAULT getDATE())
Go
insert into T_PERSONNE2 (PRS_ID,PRS_NOM,PRS_PRENOM  )  values
(11,'BRAHIM','SALEM')
```

```
   select * from T_PERSONNE2
```



Résultats | Messages

	PRS_ID	PRS_NOM	PRS_PRENOM	PRS_SEXE	PRS_DATE_NAISSANCE
1	11	BRAHIM	SALEM	M	2009-10-15

B. Clef (PRIMARY KEY)

Toute table doit être munie d'une clef (souvent appelé à tort clef primaire en opposition à clef étrangère...). Et toujours selon la théorie des bases de données, une clef doit impérativement toujours être pourvue d'une valeur ! (sinon à quoi servirait une clef en l'absence de serrure ?). Lorsque la clef porte sur une seule colonne il est possible de donner à cette colonne la contrainte PRIMARY KEY.

Nous avons vu que la contrainte PRIMARY KEY peut être posée sur une colonne (contrainte verticale) ou sur plusieurs colonnes en contrainte de ligne (horizontale). Si nous choisissons de la poser en contrainte de colonne, alors une seule colonne de la table peut en bénéficier.

Exemple :

```
CREATE TABLE T_PERSONNE5
(PRS_ID  INTEGER     NOT NULL PRIMARY KEY,
 PRS_NOM      VARCHAR(32),
 PRS_PRENOM   VARCHAR(32))
```

La contrainte PRIMARY KEY assure qu'il n'y aura aucune valeur redondante (doublon) dans la colonne. La contrainte complémentaire NOT NULL assure qu'il y aura toujours une valeur. Toute tentative d'insérer une valeur préexistante de la colonne se soldera par une violation de contrainte de clef. Voici par exemple le message généré par SQL Server dans ce cas :

```
(1 ligne(s) affectée(s))
Msg 2627, Niveau 14, État 1, Ligne 2
Violation de la contrainte PRIMARY KEY
'PK__T_PERSON__218D9B381B0907CE'. Impossible d'insérer une clé
en double dans l'objet 'dbo.T_PERSONNE5'.
L'instruction a été arrêtée.
```

	REMARQUE : il est d'usage de placer la colonne clef en tête de la description de la table pour des fins de lisibilité.

C. Unicité (UNIQUE)

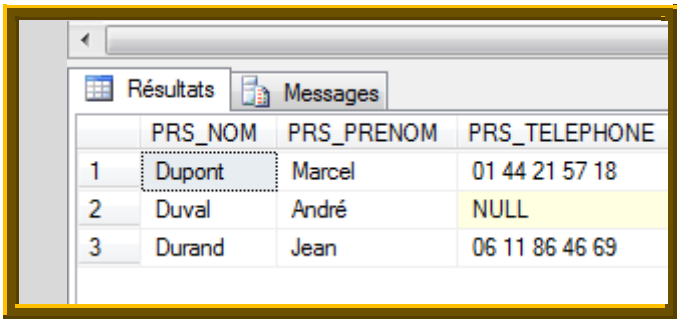
La contrainte d'unicité exige que toutes les valeurs explicites contenues dans la colonne soient uniques au sein de la table. En revanche, la colonne peut ne pas être renseignée. En effet, souvenez vous que les marqueurs NULL se propagent dans les calculs et donc comparaison d'un marqueur NULL à un ensemble de valeurs est impossible et se solde par le renvoi d'un marqueur UNKNOW à la place des valeurs TRUE ou FALSE attendue.

Exemple :

```
  CREATE TABLE T_PERSONNE7
  (PRS_NOM             VARCHAR(32),
   PRS_PRENOM          VARCHAR(32),
   PRS_TELEPHONE       CHAR(14)      UNIQUE)

  INSERT INTO T_PERSONNE7 VALUES ('Dupont', 'Marcel',  '01 44 21 57
18')
  INSERT INTO T_PERSONNE7 VALUES ('Duval', 'André',    NULL)
  INSERT INTO T_PERSONNE7 VALUES ('Durand', 'Jean',     '06 11 86 46
69')
  INSERT INTO T_PERSONNE7 (PRS_NOM, PRS_PRENOM) VALUES ('Dubois',
'Claude')
  INSERT INTO T_PERSONNE7 VALUES ('Dugland', 'Alfred', '06 11 86 46
69')

  SELECT *
  FROM T_PERSONNE7
```

Message d'erreur :

```
  Msg 2627, Niveau 14, État 1, Ligne 1

  Violation de la contrainte UNIQUE KEY 'UQ__T_PERSON__DD14909F1ED998B2'.
Impossible d'insérer une clé en double dans l'objet 'dbo.T_PERSONNE7'.

  L'instruction a été arrêtée
```



Dans cet exemple Dugland n'a pas été inséré car son numéro de téléphone est identique à Durand.

D. CREATION D'INDEXS

À l'instar de l'index d'un livre, l'index d'une base de données vous permet de retrouver rapidement des informations dans une table ou une vue indexée. Un index se compose de clés créées à partir d'une ou plusieurs colonnes dans la table ou la vue et de pointeurs qui mappent sur l'emplacement de stockage des données spécifiées. Un index bien conçu améliore de manière significative les performances des requêtes et des applications de base de données. Un index peut réduire la quantité de données qui doivent être lues par une requête pour retourner un ensemble de résultats. Les index peuvent aussi imposer l'unicité des lignes d'une table, garantissant ainsi l'intégrité des données de la table.

Les rubriques de cette section fournissent des informations qui vous aident à comprendre, à concevoir, à mettre en œuvre et à optimiser des index.

1. CONCEPTION D'INDEXS

L'engorgement des applications de base de données est souvent imputable à des index mal conçus ou en nombre insuffisant. La conception d'index efficaces est primordiale pour le bon fonctionnement des bases de données et des applications. Le choix d'index adaptés à une base de données et à sa charge de travail est une opération complexe qui vise à trouver un compromis entre vitesse des requêtes et coûts de mise à jour. Les index étroits, c'est-à-dire les index ne comportant que quelques colonnes dans la clé d'index, requièrent moins d'espace disque et de besoins de maintenance. En revanche, les index larges couvrent plus de requêtes. Vous devrez éventuellement essayer plusieurs conceptions différentes avant de trouver l'index le plus performant. Il est possible d'ajouter, de modifier et de supprimer des index sans affecter le schéma de la base de données ou la conception des applications. Par conséquent, n'hésitez à faire des essais avec différents index.

Dans la majorité des cas, l'optimiseur de requête de SQL Server choisit de manière fiable l'index le plus efficace. La stratégie globale de création d'index consiste à fournir à l'optimiseur de requête une sélection variée d'index et à se fier à lui pour faire le bon choix. Ce procédé permet de réduire le temps d'analyse et produit de bons résultats dans bon nombre de cas. Pour déterminer quels sont les index qu'utilise l'optimiseur de requête dans le cas d'une requête donnée, sélectionnez **Inclure le plan d'exécution réel** dans le menu **Requête** de SQL Server Management Studio. Pour plus d'informations, consultez Procédure : afficher un plan d'exécution réel.

L'utilisation d'index n'est pas forcément synonyme de bonnes performances, et inversement, de bonnes performances ne sauraient être nécessairement attribuables à l'utilisation d'index efficaces. Si l'utilisation d'un index contribuait toujours à produire les meilleurs résultats, le travail de

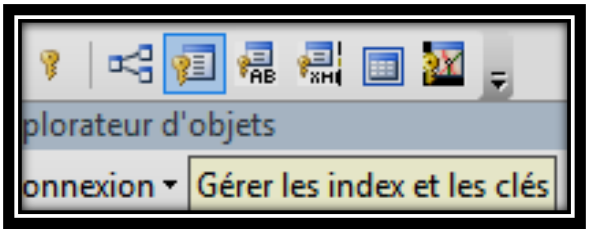
l'optimiseur de requête en serait simplifié. En réalité, le choix d'un index inapproprié peut aboutir à des performances moins que satisfaisantes. La tâche de l'optimiseur de requête est donc de ne sélectionner un index, ou une combinaison d'index, que dans les cas où cette sélection est susceptible d'améliorer les performances et d'éviter la récupération par index si elle doit les détériorer.

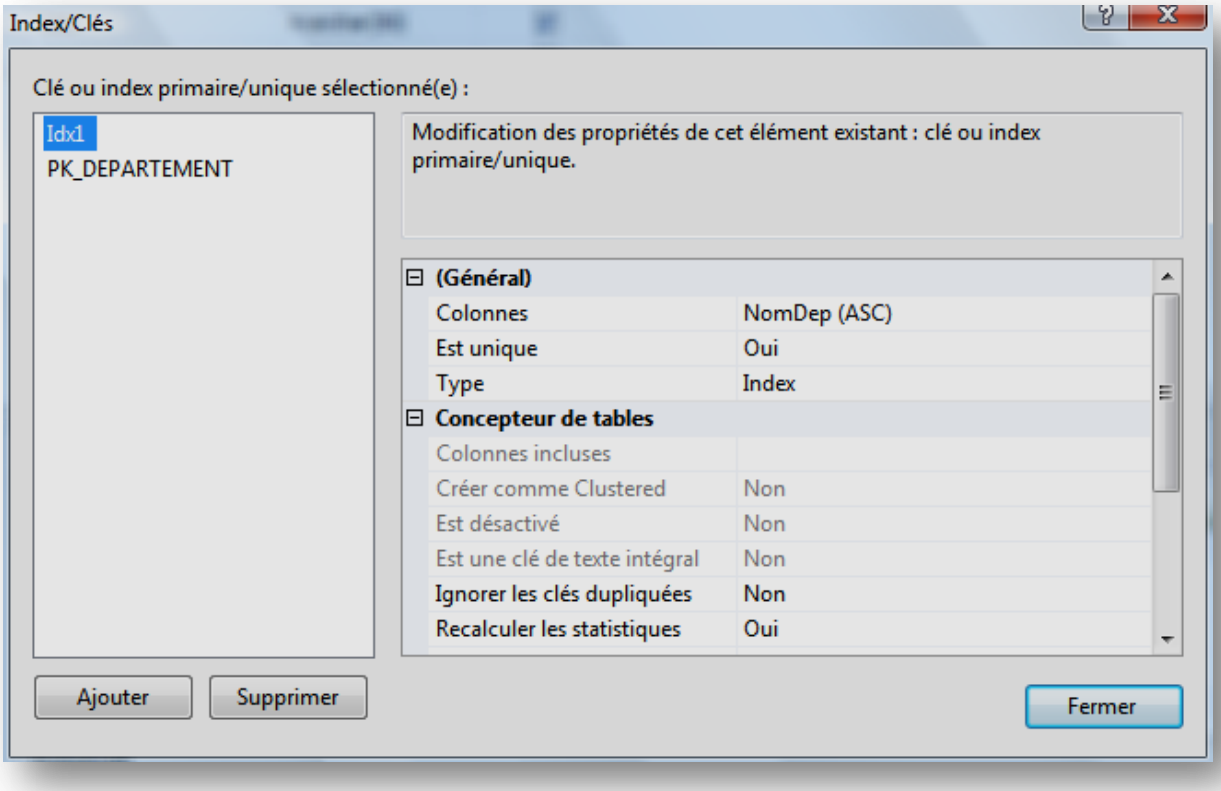
Syntaxe :

```
CREATE UNIQUE NONCLUSTERED INDEX Idx1
ON dbo.DEPARTEMENT(NomDep);
```

Comme on peut utiliser le concepteur SSMS SQL SERVER MANAGER SERVICE

Ouvrir le tableau en mode création puis cliquer sur





Fenêtre pour ajouter un index idx1

2. Directives relatives aux bases de données

- La définition de nombreux index sur une table affecte les performances des instructions INSERT, UPDATE, DELETE et MERGE , car à mesure que les données de la table changent, tous les index doivent être mis à jour en conséquence.
- Évitez que les tables mises à jour ne soient trop abondamment indexées et faites en sorte que les index soient étroits, c'est-à-dire qu'ils comprennent le moins de colonnes possible

E. Validation (CHECK)

La contrainte CHECK de validation est celle qui offre le plus de possibilité. En contre partie son exécution est très coûteuse. Elle permet de définir un prédicat complexe, basé sur une comparaison pouvant contenir une requête de type SELECT. Pour valider la contrainte, le prédicat doit être évalué à TRUE ou UNKNOWN (présence de NULL).

syntaxe :

```
CHECK ( prédicat )
```

Où prédicat peut contenir le mot clef VALUE pour faire référence à la colonne pour laquelle la contrainte est définie

Exemple :

```
CREATE TABLE T_PERSONNE9
(PRS_ID         INTEGER     CHECK (PRS_ID > 0),
 PRS_NOM        VARCHAR(32) ,
 PRS_PRENOM     VARCHAR(32) ,
 PRS_SEXE       CHAR(1)   CHECK (PRS_SEXE IN ('M', 'F')),
 PRS_TELEPHONE       CHAR(14)    )
```

- La colonne PRS_ID ne peut avoir de valeurs inférieures à 0.
- La colonne PRS_SEXE peut avoir exclusivement les valeurs M ou F.

ATTENTION : la longueur du prédicat d'une contrainte CHECK (en nombre de caractères) peut être limitée. Il faut en effet pouvoir stocker cette contrainte dans le dictionnaire des informations de la base et ce dernier n'est pas illimité.

II. Intégrité référentielle (FOREIGN KEY / REFERENCES)

La contrainte de type **FOREIGN KEY** permet de mettre en place une intégrité référentielle entre une (ou plusieurs) colonnes d'une table et la (ou les) colonne composant la clef d'une autre table afin d'assurer les relations existantes et joindre les tables dans le requête selon le modèle relationnel que l'on a défini.
Le but de l'intégrité référentielle est de maintenir les liens entre les tables quelque soit les modifications engendrées sur les données dans l'une ou l'autre table.

Cette contrainte dans sa syntaxe complète est assez complexe et c'est pourquoi nous allons dans ce paragraphe donner une syntaxe très simplifié à des fins didactiques :

syntaxe :

```
FOREIGN KEY REFERENCES table (colonne)
```

ATTENTION :

la colonne spécifiée comme référence doit être une colonne clef

Exemple :

```
   CREATE TABLE T_FACTURE1
   (FTC_ID              INTEGER,
    PRS_ID              INTEGER      FOREIGN KEY REFERENCES
T_PERSONNE5 (PRS_ID) ,
    FCT_DATE            DATE,
    FCT_MONTANT         DECIMAL(16,2))
```

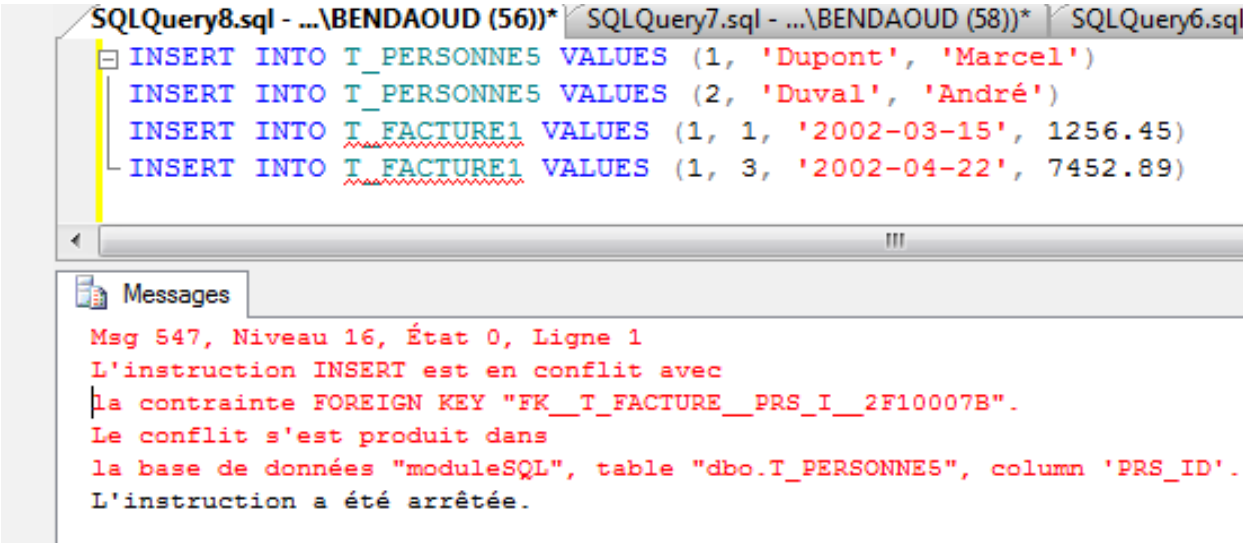
La table T_FACTURE1 est liée à la table T_PERSONNE5 et ce lien se fait entre la clef étrangère PRS_ID de la table T_FACTURE1 et la clef de la table T_PERSONNE5 qui s'intitule aussi PRS_ID.

	IMPORTANT : il est très important que les noms des colonnes de jointure soit les mêmes dans les différentes tables (notamment à cause du NATURAL JOIN), mais cela n'est pas obligatoire.

Dès lors toute tentative d'insertion d'une facture dont la référence de client est inexistante se soldera par un échec. De même toute tentative de supprimer un client pour lequel les données d'une ou de plusieurs factures sont présente se soldera par un arrêt sans effet de l'ordre SQL.

Examinons maintenant comment le SGBDR réagit pour assurer la cohérence de la base lors d'opérations tenant de briser les liens d'intégrité référentielle :

Exemple :



REMARQUE :

Comme on le voit, le mécanisme d'intégrité référentielle est un élément indispensable au maintient des relations entre tables. Un SGBD qui en est dépourvu ne peut pas prétendre à gérer le relationnel. En particulier MySQL ne peut en aucun cas prétendre être une base de données relationnelle !

III. Les contraintes de table

Une table peut être pourvue des contraintes de ligne suivante :

- **PRIMARY KEY** : précise que la ou les colonnes composent la clef de la table. ATTENTION : nécessite que chaque colonne concourrant à la clef soit NOT NULL.
- **UNIQUE** : les valeurs de la ou les colonnes doivent être unique ou NULL, c'est à dire qu'à l'exception du marqueur NULL, il ne doit jamais y avoir plus d'une fois la même valeur (pas de doublon) au sein de l'ensemble de données formé par les valeurs des différentes colonnes composant la contrainte.
- **CHECK** : permet de préciser un prédicat validant différentes colonnes de la table et qui accepterons les valeurs s'il est évalué à vrai.
- **FOREIGN KEY** : permet, pour les valeurs de la ou les colonnes, de faire référence à des valeurs préexitantes dans une ou plusieurs colonnes d'une autre table. Ce mécanisme s'apelle intégrité référentielle.

Comme dans le cas des contraintes de colonne, lorsqu'au cours d'un ordre SQL d'insertion, de modification ou de suppression, une contrainte n'est pas vérifiée on dit qu'il y a "violation" de la contrainte et les effets de l'ordre SQL sont totalement annulé (ROLLBACK).

A. Clef multicolonne (PRIMARY KEY)

La clef d'une table peut être composée de plusieurs colonnes. Dans ce cas la syntaxe est :

Syntaxe :

```
CONSTRAINT nom_contrainte PRIMARY KEY (liste_colonne)
```

Exemple :

clef primaire sur PRS_NOM / PRS_PRENOM

```
CREATE TABLE T_PERSONNE9
(PRS_NOM            VARCHAR(32) NOT NULL,
 PRS_PRENOM         VARCHAR(32) NOT NULL,
 PRS_TELEPHONE      CHAR(14),
CONSTRAINT PK_PRS PRIMARY KEY (PRS_NOM, PRS_PRENOM))
```

B. Unicité globale (UNIQUE)

Un contrainte d'unicité peut être portée sur plusieurs colonnes. Dans ce cas chaque n-uplets de valeurs explicite doit être différents.

Syntaxe :

```
CONSTRAINT nom_contrainte UNIQUE (liste_colonne)
```

Exemple :

```
CREATE TABLE T_PERSONNE10
 (PRS_ID              INTEGER,
  PRS_NOM             VARCHAR(32),
  PRS_PRENOM          VARCHAR(32),
 CONSTRAINT UNI_NOM_PRENOM UNIQUE (PRS_NOM, PRS_PRENOM))
insert into T_PERSONNE10 values(10,'mouhammed','Ali')
insert into T_PERSONNE10 values(11,'Fatima','zahra')
insert into T_PERSONNE10 values(12,'mouhammed','Ali')
```

Messages

```
(1 ligne(s) affectée(s))

(1 ligne(s) affectée(s))
Msg 2627, Niveau 14, État 1, Ligne 3
Violation de la contrainte UNIQUE KEY 'UNI_NOM_PRENOM'. Impossible d'insérer une clé en double dans l'objet
L'instruction a été arrêtée.
```

REMARQUE :

certains SGBDR comme MS SQL Server refuse de voir la présence de plusieurs marqueurs NULL dans la cas d'une contrainte d'unicité. D'autres comme InterBase refusent une contrainte d'unicité dépourvue d'une contrainte NOT NULL...

ATTENTION :
vous ne pouvez pas définir une contrainte d'unicité sur des colonnes de type BLOB

C. Validation de ligne (CHECK)

La contrainte CHECK permet d'effectuer un contrôle de validation multicolonne au sein de la table.

Sa syntaxe est :

Syntaxe :

```
CONSTRAINT nom_contrainte CHECK ( prédicat )
```

Exemple :
vérification de présence d'information dans au moins une colonne crédit ou débit de la table compte :

```
CREATE TABLE T_COMPTE
(CPT_ID              INTEGER,
 CPT_DATE            DATE,
 CPT_CREDIT          DECIMAL (16,2),
 CPT_DEBIT           DECIMAL (16,2),
 CLI_ID              INTEGER,
 CONSTRAINT CHK_OPERATION CHECK((CPT_CREDIT >= 0 AND CPT_DEBIT IS
NULL) OR (CPT_DEBIT >= 0 AND CPT_CREDITIS NULL)))
```

Toute tentative d'insérer une ligne avec des valeurs non renseignées pour les colonnes débit et crédit, ou bien avec des valeurs négative se soldera par un refus.

IV. Intégrité référentielle de table (FOREIGN KEY / REFERENCES)

Comme dans la cas d'une contrainte référentielle de colonne, il est possible de placer une contrainte d'intégrité portant sur plusieurs colonne. Ceci est d'autant plus important qu'il n'est pas rare de trouver des tables dont la clef est composée de plusieurs colonnes. La syntaxe est la suivante :

Syntaxe :

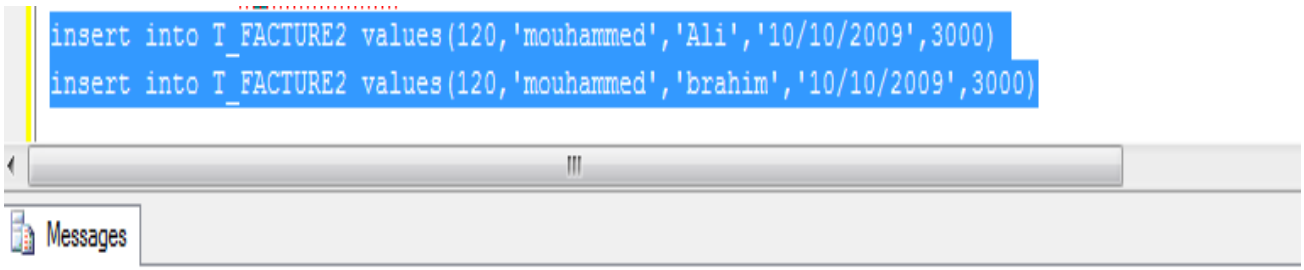
```
CONSTRAINT nom_contrainte FOREIGN KEY (liste_colonne) REFERENCES
nom_table_ref (liste_colonne_ref)
```

Exemple :

```
   CREATE TABLE T_FACTURE2
   (FTC_ID              INTEGER,
    PRS_NOM             VARCHAR(32),
    PRS_PRENOM          VARCHAR(32),
    FCT_DATE            DATE,
    FCT_MONTANT         DECIMAL(16,2),
    CONSTRAINT   FK_FCT_PRS   FOREIGN   KEY   (PRS_NOM,
PRS_PRENOM)    REFERENCES    T_PERSONNE9    (PRS_NOM,
PRS_PRENOM))
```

La table T_FACTURE2 est liée à la table T_PERSONNE9 et ce lien se fait entre la clef étrangère composite PRS_NOM / PRS_PRENOM de la table T_FACTURE2 et la clef de la table T_PERSONNE9 elle même composée des colonnes PRS_NOM / PRS_PRENOM.

Examinons maintenant comment le SGBDR réagit pour assurer la cohérence de la base lors d'opérations tenant de briser les liens d'intégrité référentielle :




```
(1 ligne(s) affectée(s))
Msg 547, Niveau 16, État 0, Ligne 2
L'instruction INSERT est en conflit avec la contrainte FOREIGN KEY "FK_FCT_PRS". Le conflit s'est produit da
L'instruction a été arrêtée.
```

La personne Mouhammed Brahim n'existe pas dans la table T_PESONNE10

V. Suppression d'une table

Syntaxe :

```
DROP TABLE MATABLE1
```

Exemple :

```
CREATE TABLE MATABLE1(column_a INT) ;
GO
DROP TABLE MATABLE1;
```

VI. MODIFIER UN TABLEAU

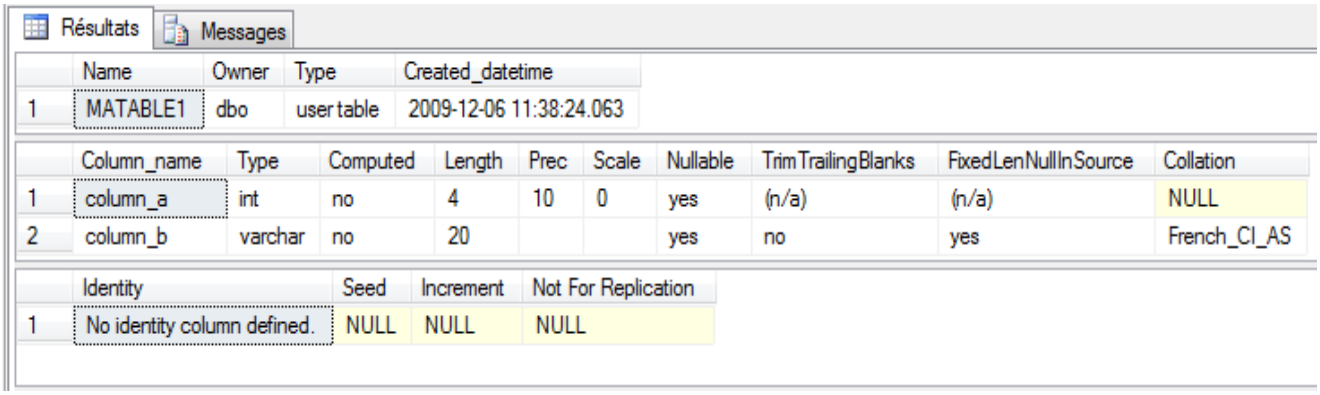
Modifie la définition d'une table en changeant, en ajoutant ou en supprimant des colonnes et des contraintes, en réaffectant des partitions, en désactivant ou en activant des contraintes et des déclencheurs.

A. Ajout d'une nouvelle colonne

L'exemple suivant ajoute une colonne qui accepte les valeurs NULL et pour laquelle aucune valeur n'est spécifiée via une définition DEFAULT. Dans la nouvelle colonne, chaque ligne aura la valeur NULL

Exemple :

```
CREATE TABLE MATABLE1(column_a INT) ;
GO
ALTER TABLE MATABLE1 ADD column_b
VARCHAR(20) NULL ;
GO
EXEC sp_help MATABLE1;
GO
DROP TABLE MATABLE1;
GO
```



Résultats | Messages

	Name	Owner	Type	Created_datetime
1	MATABLE1	dbo	user table	2009-12-06 11:38:24.063

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	column_a	int	no	4	10	0	yes	(n/a)	(n/a)	NULL
2	column_b	varchar	no	20			yes	no	yes	French_CI_AS

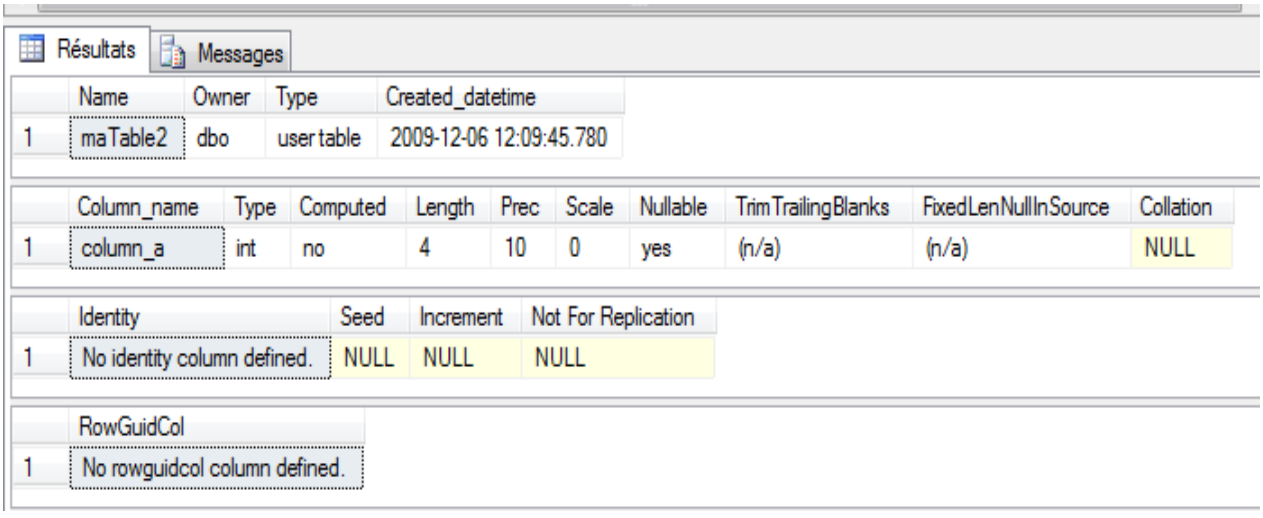
	Identity	Seed	Increment	Not For Replication
1	No identity column defined.	NULL	NULL	NULL

B. Suppression d'une colonne

L'exemple suivant supprime une colonne dans une table.

Exemple :

```
CREATE TABLE maTable2 (column_a INT, column_b VARCHAR(20) NULL) ;
GO
ALTER TABLE maTable2 DROP COLUMN column_b ;
GO
/*pour afficher le schema de la lable maTable2 après modification */
   EXEC sp_help maTable2 GO;
```



Résultats | Messages

	Name	Owner	Type	Created_datetime
1	maTable2	dbo	user table	2009-12-06 12:09:45.780

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	column_a	int	no	4	10	0	yes	(n/a)	(n/a)	NULL

	Identity	Seed	Increment	Not For Replication
1	No identity column defined.	NULL	NULL	NULL

	RowGuidCol
1	No rowguidcol column defined.

C. Modification du type de données d'une colonne

L'exemple suivant modifie le type d'une colonne d'une table de `INT` en `DECIMAL`.

Exemple :

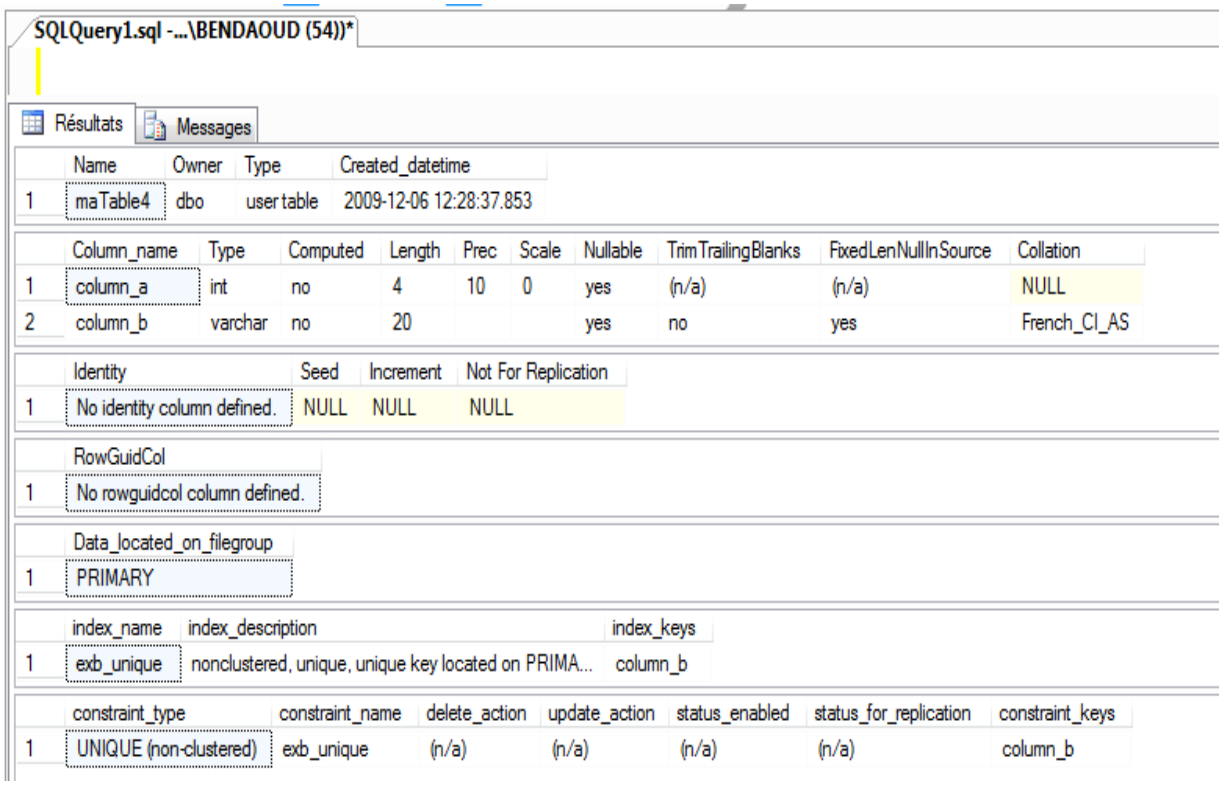
```
CREATE TABLE maTable3 (column_a INT ) ;
GO
INSERT INTO maTable3 (column_a) VALUES (10) ;
GO
ALTER TABLE maTable3 ALTER COLUMN column_a DECIMAL (5, 2) ;
--pour afficher le schema aprés modification du type de
column_a
EXEC sp_help maTable3
-- pour supprimer le tableau aprés verification
DROP TABLE maTable3 ;
   GO
```

D. Ajout d'une colonne avec une contrainte

L'exemple suivant ajoute une nouvelle colonne avec une contrainte UNIQUE.

Exemple :

```
CREATE TABLE maTable4 (column_a INT) ;
GO
ALTER TABLE maTable4  ADD column_b VARCHAR(20) NULL
    CONSTRAINT exb_unique UNIQUE ;
GO
--pour afficher le schema de la table maTable4
EXEC sp_help maTable4  ;
GO
--pour supprimer la table d'exemple
DROP TABLE maTable4  ;
   GO
```



E. Ajout d'une contrainte CHECK non vérifiée à une colonne existante

L'exemple suivant ajoute une contrainte à une colonne existante de la table. La colonne comporte une valeur qui ne respecte pas la contrainte. Par conséquent, **WITH NOCHECK** empêche la validation de la contrainte sur les lignes existantes, et permet l'ajout de la contrainte

Exemple :

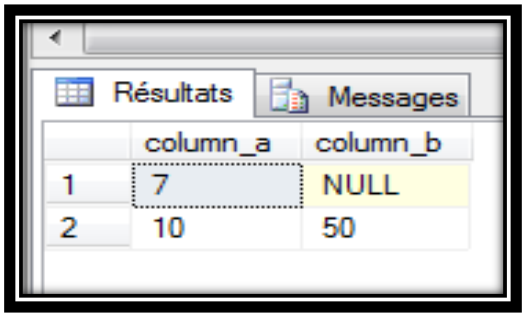
```
CREATE TABLE maTable5 ( column_a INT) ;
GO
INSERT INTO maTable5 VALUES (-1) ;
GO
ALTER TABLE maTable5 WITH NOCHECK
ADD CONSTRAINT exd_check CHECK (column_a > 1) ;
GO
--pour afficher le schema de la table maTable5
EXEC sp_help maTable5 ;
GO
--pour supprimer la table d'exemple
DROP TABLE maTable5 ;
  GO
```

F. Ajout d'une contrainte DEFAULT à une colonne existante

L'exemple suivant crée une table de deux colonnes et insère une valeur dans la première ; l'autre colonne conserve la valeur NULL. Une contrainte DEFAULT est alors ajoutée à la deuxième colonne. Pour vérifier que la valeur par défaut est appliquée, une autre valeur est insérée dans la première colonne et la table fait l'objet d'une requête

Exemple :

```
CREATE TABLE maTable6 ( column_a INT, column_b INT) ;
GO
INSERT INTO maTable6 (column_a)VALUES ( 7 ) ;
GO
ALTER TABLE maTable6
ADD CONSTRAINT col_b_def
DEFAULT 50 FOR column_b ;
GO
INSERT INTO maTable6 (column_a) VALUES ( 10 ) ;
  select * from maTable6
```



G. Ajout de plusieurs colonnes avec des contraintes

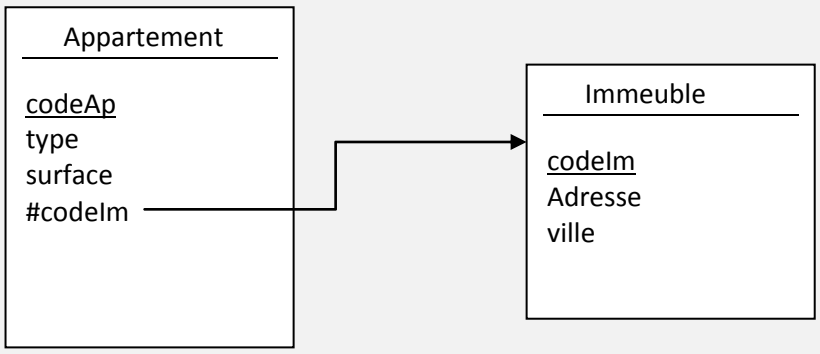
L'exemple suivant ajoute plusieurs colonnes avec des contraintes définies. La première colonne a la propriété `IDENTITY`. Chaque ligne de la table a de nouvelles valeurs incrémentielles dans la colonne d'identité.

Exemple :

```
CREATE TABLE maTable7 ( column_a INT CONSTRAINT column_a_un UNIQUE) ;
GO
ALTER TABLE maTable7 ADD

-- Add a PRIMARY KEY identity column.
column_b INT IDENTITY
CONSTRAINT column_b_pk PRIMARY KEY,

-- Add a column that references another column in the same table.
column_c INT NULL
CONSTRAINT column_c_fk
REFERENCES maTable7(column_a),

-- Add a column with a constraint to enforce that
-- nonnull data is in a valid telephone number format.
column_d VARCHAR(16) NULL
CONSTRAINT column_d_chk
CHECK
(column_d LIKE '[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]' OR
column_d LIKE
'([0-9][0-9][0-9]) [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]'),

-- Add a nonnull column with a default.
column_e DECIMAL(3,3)
CONSTRAINT column_e_default
DEFAULT .081 ;
GO
EXEC sp_help maTable7 ;
GO
DROP TABLE maTable7 ;
GO
```

Travaux Dirigés

Soit la base de données suivante :



1. Donner le code SQL qui permet de créer la table Appartement sans oublier la contrainte de la clé étrangère
2. Ajouter une contrainte a la table Appartement qui permet de contrôler le type (le type prend les valeurs 'F2','F3','F4','F5 ')

CHAPITRE 5

Leçon 1 : langage de manipulation des données

Objectifs :vous serez à même d'effectuer les tâches suivantes

- L'instruction INSERT ajout
- L'instruction UPDATE modification
- L'instruction DELETE suppression

I. Langage de manipulation des données

Le langage de manipulation de données (LMD) est le langage permettant de modifier les informations contenues dans la base.
Il existe trois commandes SQL permettant d'effectuer les trois types de Modification des données :

- INSERT ajout de lignes
- UPDATE mise à jour de lignes
- DELETE suppression de lignes

Ces trois commandes travaillent sur la base telle qu'elle était au début de l'exécution de la commande. Les modifications effectuées par les autres utilisateurs entre le début et la fin de l'exécution ne sont pas prises en compte (même pour les transactions validées).

A. Insertion

Syntaxe :

```
INSERT INTO table (col1,..., coln )
VALUES (val1,...,valn )
```

OU

Syntaxe :

```
INSERT INTO table (col1,..., coln )
SELECT ...
```

table est le nom de la table sur laquelle porte l'insertion. col1,..., coln est la liste des noms des colonnes pour lesquelles on donne une valeur. Cette liste est optionnelle. Si elle est omise, ORACLE prendra par défaut l'ensemble des colonnes de la table dans l'ordre où elles ont été données lors de la création de la table. Si une liste de colonnes est spécifiée, les colonnes ne figurant pas dans la liste auront la valeur NULL.

Exemple :

a) INSERT INTO dept VALUES (10, 'FINANCES', 'PARIS')

b) INSERT INTO dept (lieu, nomd, dept) VALUES ('GRENOBLE', 'RECHERCHE', 20)

La deuxième forme avec la clause SELECT permet d'insérer dans une table des lignes provenant d'une table de la base. Le SELECT a la même syntaxe qu'un SELECT normal.

Exemple :

Enregistrer la participation de MARTIN au groupe de projet numéro10 :

```
INSERT INTO PARTICIPATION (MATR, CODEP)
SELECT MATR, 10 FROM EMP
WHERE  NOME= 'MARTIN' ;
```

B. Modification

La commande **UPDATE** permet de modifier les valeurs d'un ou plusieurs champs, dans une ou plusieurs lignes existantes d'une table.

Syntaxe :

```
UPDATE table
SET col1 = exp1, col2 = exp2, ...
WHERE prédicat
```

OU

Syntaxe :

```
UPDATE table
SET (col1, col2,...) = (SELECT ...)
WHERE prédicat
```

Table est le nom de la table mise à jour ; col1, col2, ... sont les noms des colonnes qui seront modifiées ; exp1, exp2,... sont des expressions. Elles peuvent aussi être un ordre SELECT renvoyant les valeurs attribuées aux colonnes (deuxième variante de la syntaxe).

Les valeurs de col1, col2... sont mises à jour dans toutes les lignes satisfaisant le prédicat. La clause WHERE est facultative. Si elle est absente, toutes les lignes sont mises à jour.

Le prédicat peut contenir des sous-interrogations.

Exemple :
Faire passer MARTIN dans le département 10 :

```
UPDATE EMP SET DEPT = 10
    WHERE NOME = 'MARTIN'
```

Exemple : Augmenter de 10 % les commerciaux :

```
UPDATE EMP SET SAL = SAL * 1.1
  WHERE POSTE = 'COMMERCIAL'
```

Exemple :
Donner à CLEMENT un salaire 10 % au dessus de la moyenne des salaires des secrétaires

```
UPDATE EMP SET SAL = (SELECT AVG(SAL) * 1.10
FROM EMP   WHERE POSTE = 'SECRETAIRE')
          WHERE NOME = 'CLEMENT'
```

On remarquera que la moyenne des salaires sera calculée pour les valeurs qu'avaient les salaires au début de l'exécution de la commande UPDATE et que les modifications effectuées sur la base pendant l'exécution de cette commande ne seront pas prises en compte

Exemple :
Enlever (plus exactement, mettre à la valeur 'NULL') la commission de MARTIN :

```
UPDATE  EMP
SET COMM = NULL
WHERE NOME = 'MARTIN'
```

Suppression

L'ordre DELETE permet de supprimer des lignes d'une table.

Syntaxe :

```
DELETE FROM table
WHERE prédicat
```

La clause WHERE indique quelles lignes doivent être supprimées.

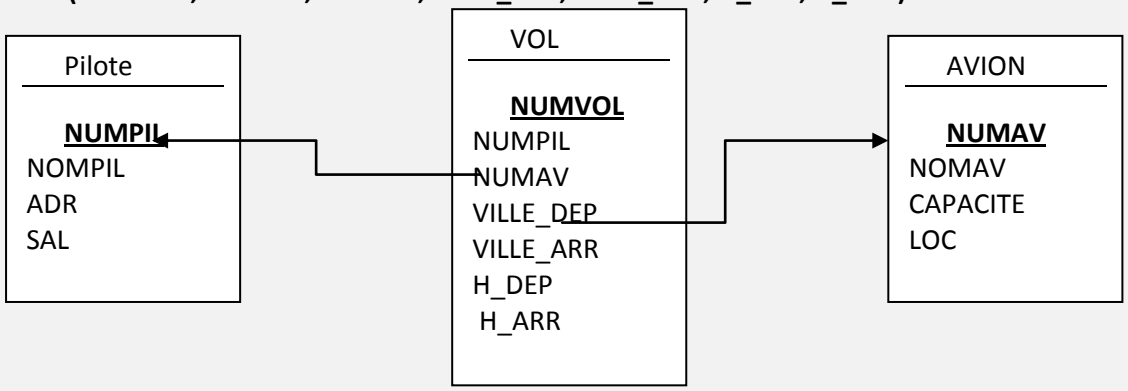
ATTENTION :
cette clause est facultative ; si elle n'est pas précisée, TOUTES LES LIGNES DE LA TABLE SONT SUPPRIMEES (heureusement qu'il existe ROLLBACK!). Le prédicat peut contenir des sous-interrogations

Exemple :

```
DELETE FROM dept WHERE dept = 10
```

Travaux Dirigés

Soit la base de données VOLS suivante :
PILOTE (NUMPIL, NOMPIL, ADR, SAL)
AVION (NUMAV, NOMAV, CAPACITE, LOC)
VOL (NUMVOL, NUMPIL, NUMAV, VILLE_DEP, VILLE_ARR, H_DEP, H_ARR)



NUMPIL: clé de PILOTE, nombre entier
NOMPIL: nom du pilote, chaîne de caractères
ADR: ville de la résidence du pilote, chaîne de caractères
SAL: salaire du pilote, nombre entier

NUMAV: cl´e de AVION, nombre entier
CAPACITE: nombre de places d'un avion, nombre entier
LOC: ville de l'aéroport d'attache de l'avion, chaîne de caractères
NUMVOL: cl´e de VOL, nombre entier
VILLE_DEP: ville de départ du vol, chaîne de caractères
VILLE_ARR: ville d'arrivée du vol, chaîne de caractères
H_DEP: heure de départ du vol, nombre entier entre 0 et 23
H_ARR: heure d'arrivée du vol, nombre entier entre 0 et 23

Question de cours :

1. En quoi consiste la notion de mappage dans QSL Server 2008

Implémentation de la base de données

1. Donner la transat SQL qui permet de créer la table vol
2. Modifier la table vol en ajoutant les contraintes (clé étrangères)
3. Modifier la table vol en ajoutant les contraintes qui contrôlent la saisie des données H_DEP H_ARR (comprise entre 0 et 23)
4. Donner le code sql qui permet de créer une connexion « operateur » et un utilisateur « Mohammed » dans la base et lui donne le droit total sur la table vol

Questions interroger une base de données:

1. Liste des pilotes dont les noms commencent par un « A »
2. Quels est le salaire minimum d'un pilote conduisant un vol Bordeaux-Marseille?
3. Dans quelle ville le salaire moyen des pilotes qui y r´résident est-il maximum?
4. Liste des pilotes qui résident dans la LOC des avions qui pilotent
5. Quel est l'ensemble des (NUMAV,V_DEP,V_ARR) de VOL tel qu'on ne trouve pas (NUMAV,V_ARR,V_DEP) dans VOL

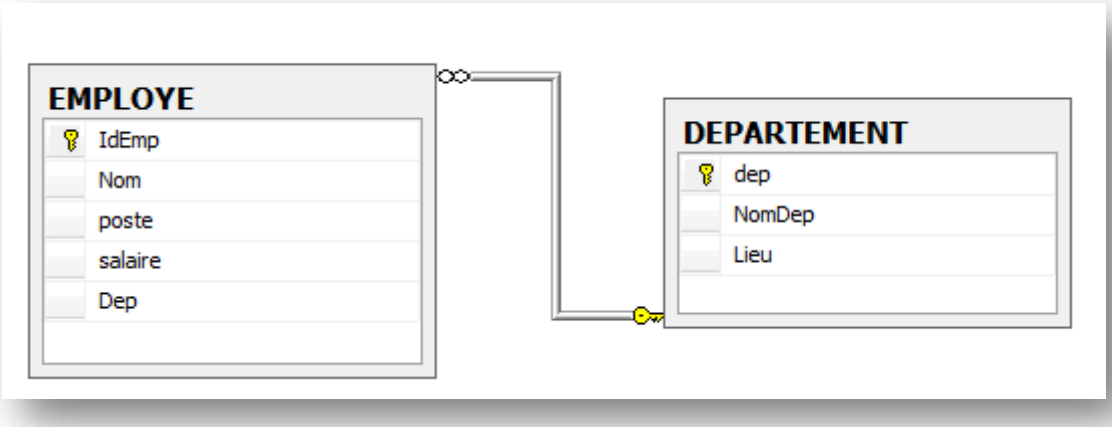
Chapitre 5

Leçon 2 : Interroger une base de données

Objectifs : vous serez à même d'effectuer les tâches suivantes

- Interroger une table
- La clause select
- La clause from
- La clause where
- Les operateurs logiques and /or
- Sous interrogation retournant ramenant une ligne et une colonne
- Sous interrogation retournant ramenant plusieurs lignes
- Prédicat ALL /ANY/exists
- Fonctions de groupes
- Division avec EXISTS
- Clause having
- Fonction arithmétique/et chaîne de caractères

Remarque : pour les exemples on va utiliser la base de données suivante :



I. Interrogations

A. Syntaxe générale

L'ordre SELECT possède six clauses différentes, dont seules les deux premières sont obligatoires. Elles sont données ci-dessous, dans l'ordre dans lequel elles doivent apparaître, quand elles sont utilisées :

Syntaxe :

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

B. Clause SELECT

Cette clause permet d'indiquer quelles colonnes, ou quelles expressions doivent être retournées par l'interrogation.

Syntaxe :

```
SELECT [DISTINCT] *
Ou
SELECT [DISTINCT] exp1 [[AS] nom1 ], exp2 [[AS] nom2 ], .....
```

exp1, exp2, … sont des expressions, nom1, nom2, … sont des noms facultatifs de 30 caractères maximum, donnés aux expressions. Chacun de ces noms est inséré derrière l'expression, séparé de cette dernière par un blanc ou par le mot clé AS (optionnel) ; il constituera le titre de la colonne dans l'affichage du résultat de la sélection. Ces noms ne peuvent être utilisés dans les autres clauses (where par exemple).

Le symbole * signifie que toutes les colonnes de la table sont sélectionnées.

Le mot clé facultatif DISTINCT ajouté derrière l'ordre SELECT permet d'éliminer les duplications : si, dans le résultat, plusieurs lignes sont identiques, Une seule sera conservée.

Exemple :

- SELECT * FROM DEPT
- SELECT DISTINCT POSTE FROM EMP
- SELECT NOME, SAL + NVL(COMM,0) AS ‘Salaire ‘ FROM EMP

La requête suivante va provoquer une erreur car on utilise le nom Salaire dans la clause where :

SELECT NOME, SAL + NVL(COMM,0) AS Salaire FROM EMP

WHERE Salaire > 1000

Si le nom contient des séparateurs (espace, caractère spécial), ou s'il est

Identique à un mot réservé SQL (exemple : DATE), il doit être mis entre crochet.

Exemple :

- SELECT NOME, SAL + NVL(COMM,0) as 'Salaire Total' FROM EMP

Le nom complet d'une colonne d'une table est le nom de la table suivi d'un Point et du nom de la colonne. Par exemple : **EMP.MATR, EMP.DEPT, DEPT.DEPT**

Le nom de la table peut être omis quand il n'y a pas d'ambiguïté. Il doit être précisé s'il y a une ambiguïté, ce qui peut arriver quand on fait une sélection sur plusieurs tables à la fois et que celles-ci contiennent des colonnes qui ont le même nom

C. Clause FROM

La clause FROM donne la liste des tables participant à l'interrogation. Il est possible de lancer des interrogations utilisant plusieurs tables à la fois.

Syntaxe :

FROM table1 as [synonyme1] , table2 as [synonyme2] , ……

synonyme1, synonyme2,… sont des synonymes attribués facultativement aux tables pour le temps de la sélection. On utilise cette possibilité pour lever certaines ambiguïtés, quand la même table est utilisée de plusieurs façons différentes dans une même interrogation (voir les exemples)

Quand on a donné un synonyme à une table dans une requête, elle n'est plus reconnue sous son nom d'origine dans cette requête. Le nom complet d'une table est celui de son créateur (celui du nom du schéma suivi d'un point et du nom de la table. Par défaut, le nom du créateur est celui de l'utilisateur en cours. Ainsi, on peut se dispenser de préciser ce nom quand on travaille sur ses propres tables. Mais il faut le préciser dès que l'on se sert de la table d'un autre utilisateur.

Pour obtenir la liste des employés avec le pourcentage de leur salaire par rapport au total des salaires, il fallait auparavant utiliser une vue. Il est maintenant possible d'avoir cette liste avec une seule instruction SELECT :

Exemple :

```
select nome, sal, sal/total*100
from emp, (select sum(sal) as total from emp)
```

D. Clause WHERE

La clause WHERE permet de spécifier quelles sont les lignes à sélectionner dans une table ou dans le produit cartésien de plusieurs tables. Elle est suivie d'un prédicat (expression logique ayant la valeur vrai ou faux) qui sera évalué pour chaque ligne. Les lignes pour lesquelles le prédicat est vrai seront sélectionnées. La clause where est étudiée ici pour la commande SELECT. Elle peut se rencontrer aussi dans les commandes UPDATE et DELETE avec la même

Clause WHERE simple

Syntaxe :
WHERE prédicat

Un prédicat simple est la comparaison de deux expressions ou plus au moyen d'un

Opérateur logique :

```
WHERE exp1 = exp2
WHERE exp1 != exp2
WHERE exp1 < exp2
WHERE exp1 > exp2
WHERE exp1 <= exp2
WHERE exp1 >= exp2
WHERE exp1 BETWEEN exp2 AND exp3
WHERE exp1 LIKE exp2
WHERE exp1 NOT LIKE exp2
WHERE exp1 IN (exp2, exp3,...)
WHERE exp1 NOT IN (exp2, exp3,...)
WHERE exp IS NULL
WHERE exp IS NOT NULL
```

Les trois types d'expressions (arithmétiques, caractères, ou dates) peuvent être comparées au moyen des opérateurs d'égalité ou d'ordre (=, !=, <, >, <=,>=) : pour les types date, la relation d'ordre est l'ordre chronologique ; pour les types caractères, la relation d'ordre est l'ordre lexicographique. Il faut ajouter à ces opérateurs classiques les opérateurs suivants BETWEEN, IN, LIKE, IS NULL :

exp1 BETWEEN exp2 AND exp3 est vrai si exp1 est compris entre exp2 et exp3, bornes incluses.

exp1 IN (exp2 , exp3...) est vrai si exp1 est égale à l'une des expressions de la liste entre parenthèses.

Exp1 LIKE exp2 teste l'égalité de deux chaînes en tenant compte des caractères jokers dans

La 2ème chaîne : ‘‘_’’ remplace 1 caractère exactement ‘’%’’ remplace une chaîne de caractères de longueur quelconque, y compris de longueur nulle

Le fonctionnement est le même que celui des caractères joker ? et * pour le shell sous Unix. Ainsi l'expression 'MARTIN' LIKE '_AR%' sera vraie.

L'opérateur IS NULL permet de tester la valeur NULL : exp IS [NOT] NULL est vrai si l'expression a la valeur NULL (ou l'inverse avec NOT).

E. Opérateurs logiques AND et OR

Les opérateurs logiques AND et OR peuvent être utilisés pour combiner plusieurs prédicats (l'opérateur AND est prioritaire par rapport à l'opérateur OR). Des parenthèses peuvent être utilisées pour imposer une priorité dans l'évaluation du prédicat, ou simplement pour rendre plus claire l'expression logique. L'opérateur NOT placé devant un prédicat en inverse le sens.

Exemple : Sélectionner les employés du département 30 ayant un salaire supérieur à 1500 frs.

```
SELECT NOME FROM EMP
WHERE DEPT = 30 AND SAL > 1500
```

Exemple : Afficher une liste comprenant les employés du département 30 dont le salaire est supérieur à 11000 Frs et (attention, à la traduction par OR) les employés qui ne touchent pas de commission.

```
SELECT nome FROM emp
WHERE dept = 30 AND sal > 11000 OR comm IS NULL
```

II. Sous-interrogation

Une caractéristique puissante de SQL est la possibilité qu'un prédicat employé dans une clause WHERE (expression à droite d'un opérateur de comparaison) comporte un SELECT emboîté.

Par exemple, la sélection des employés ayant même poste que MARTIN peut s'écrire en joignant la table EMP avec elle-même :

Exemple :

```
SELECT EMP.NOME
FROM EMP JOIN EMP MARTIN ON EMP.POSTE = MARTIN.POSTE
WHERE MARTIN.NOME = 'MARTIN'
```

mais on peut aussi la formuler au moyen d'une sous-interrogation :

Exemple :

```
SELECT NOME FROM EMP
WHERE POSTE = (SELECT POSTE
FROM EMP
WHERE NOME = 'MARTIN')
```

Les sections suivantes exposent les divers aspects de ces sous-interrogations.

A. Sous-interrogation à une ligne et une colonne

Dans ce cas, le SELECT imbriqué équivaut à une valeur.

où op est un des opérateurs = != < > <= >= exp est toute expression légale.

Exemple :

Liste des employés travaillant dans le même département que MERCIER

Syntaxe :

```
WHERE exp op (SELECT ...)
```

Exemple :

```
SELECT NOME FROM EMP
```

```
WHERE DEPT = (SELECT DEPT FROM EMP
```

Exemple :

```
WHERE NOME = 'MERCIER')
```

Un SELECT peut comporter plusieurs sous-interrogations, soit imbriquées, soit au même niveau dans différents prédicats combinés par des ANDou des OR.

Exemple : Liste des employés du département 10 ayant même poste que quelqu'un du département VENTES :

```
SELECT NOME, POSTE FROM EMP
WHERE DEPT = 10
AND POSTE IN
(SELECT POSTE FROM EMP
WHERE DEPT = (SELECT DEPT
FROM DEPT
WHERE NOMD = 'VENTES'))
```

Exemple : Liste des employés ayant même poste que MERCIER ou un salaire supérieur à CHATEL :

```
SELECT NOME, POSTE, SAL FROM EMP
WHERE POSTE = (SELECT POSTE FROM EMP
WHERE NOME = 'MERCIER')
OR SAL > (SELECT SAL FROM EMP WHERE NOME = 'CHATEL')
```

Jointures et sous-interrogations peuvent se combiner.

Exemple :

```
SELECT NOME, POSTE
FROM EMP JOIN DEPT ON EMP.DEPT = DEPT.DEPT
WHERE LIEU = 'LYON'
AND POSTE = (SELECT POSTE FROM EMP
WHERE NOME = 'FREMONT')
```

On peut aussi plus simplement utiliser la jointure naturelle puisque les

noms des colonnes de jointures sont les mêmes :

Exemple : Liste des employés travaillant à LYON et ayant même poste que FREMONT.

```
SELECT NOME, POSTE
FROM EMP NATURAL JOIN DEPT
WHERE LIEU = 'LYON'
AND POSTE = (SELECT POSTE FROM EMP
WHERE NOME = 'FREMONT')
```

ATTENTION :

une sous-interrogation à une seule ligne doit ramener une
Seule ligne ; dans le cas où plusieurs lignes, ou pas de ligne du tout seraient ramenées, un message d'erreur sera affiché et l'interrogation sera abandonnée.

B. Sous-interrogation ramenant plusieurs lignes

Une sous-interrogation peut ramener plusieurs lignes à condition que l'opérateur de comparaison admette à sa droite un ensemble de valeurs.

Les opérateurs permettant de comparer une valeur à un ensemble de valeurs sont :

l'opérateur IN les opérateurs obtenus en ajoutant ANY ou ALL à la suite des opérateurs

de comparaison classique =, !=, <, >, <=, >=.ANY : la comparaison sera vraie si elle est vraie pour au moins un élément de l'ensemble (elle est donc fausse si l'ensemble est vide).

ALL : la comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble (elle est vraie si l'ensemble est vide).

Exemple :

- WHERE exp op ANY (SELECT ...)
- WHERE exp op ALL (SELECT ...)
- WHERE exp IN (SELECT ...)
- WHERE exp NOT IN (SELECT ...)

où op est un des opérateurs =, !=, <, >, <=, >=.

Exemple :
Liste des employés gagnant plus que tous les employés du département 30 :

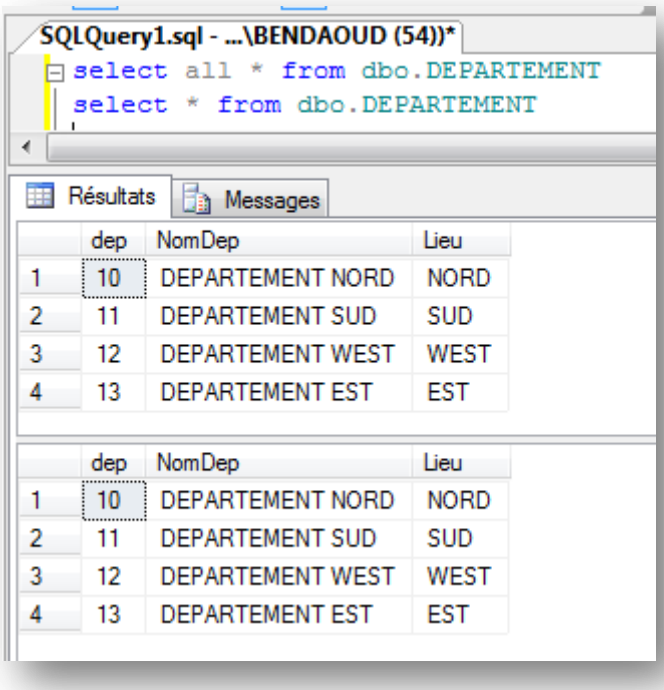
```
SELECT NOME, SAL FROM EMP
WHERE SAL > ALL (SELECT SAL FROM EMP
WHERE DEPT=30)
```

REMARQUE :
L'opérateur IN est équivalent à = ANY, et l'opérateur NOT IN est équivalent à != ALL.

C. Les Prédicats : ALL, DISTINCT, DISTINCTROW, TOP

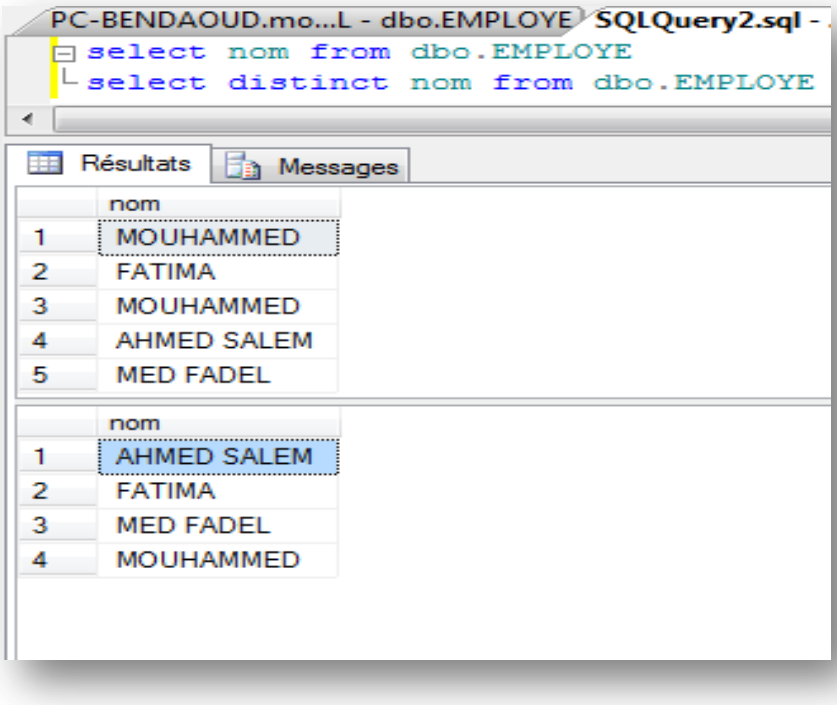
1. Le prédicat ALL

Si vous n'incluez aucun prédicat, le moteur de base de données Microsoft Jet sélectionne tous les enregistrements qui remplissent les conditions de l'instruction SQL. Les deux exemples suivants sont équivalents et renvoient tous les enregistrements de la table Employés :



2. Le prédicat DSTINCT

Omet tous les enregistrements pour lesquels les champs sélectionnés contiennent des données en double. Ainsi, pour être incluses dans les résultats de la requête, les valeurs de chaque champ répertorié dans l'instruction SELECT doivent être uniques. Par exemple, plusieurs employés répertoriés dans une table Employés peuvent avoir le même nom. Si deux enregistrements contiennent Durand dans le champ "Nom", l'instruction SQL suivante ne renvoie alors qu'un seul de ces enregistrements :



3. Predicat DISTINCTROW

Omet les données sur la base des enregistrements complets en double, et pas seulement de champs en double. Par exemple, vous pouvez créer une requête qui joint les tables Clients et Commandes à l'aide du champ "Code client". La table Clients

ne contient aucun doublon dans le champ "Code client", mais la table Commandes en contient car chaque client passe plusieurs commandes. L'instruction SQL suivante montre comment utiliser DISTINCTROW pour produire une liste de sociétés qui ont passé au moins une commande, sans afficher le détail de ces commandes :

Exemple :

```
SELECT DISTINCTROW Société
FROM Clients INNER JOIN Commandes
ON Clients.[Code client]= Commandes.[Code client]
ORDER BY Société;
```

Si vous omettez DISTINCTROW, cette requête produit plusieurs lignes pour chaque société ayant passé plusieurs commandes. DISTINCTROW n'a d'effet que si vous sélectionnez des champs dans seulement certaines des tables utilisées dans la requête. DISTINCTROW est ignoré si votre requête n'inclut qu'une seule table ou si vous sélectionnez les champs de toutes les tables.

4. TOP n [PERCENT]

Renvoie un certain nombre d'enregistrements situés au début ou à la fin d'une plage spécifiée par une clause ORDER BY. Supposons que vous souhaitiez obtenir les noms des 25 premiers étudiants de la promotion 1996 :

Exemple :

```
SELECT  TOP 25 Nom, Prénom
FROM Etudiants
WHERE Promotion = 1996
ORDER BY Moyenne DESC;
```

Si vous n'incluez pas la clause ORDER BY, la requête renverra une série de 25 enregistrements choisis arbitrairement parmi ceux de la table Students qui remplissent les conditions de la clause WHERE. Le prédicat TOP n'effectue pas de choix entre des valeurs égales. Dans l'exemple précédent, si, parmi les meilleurs résultats obtenus, le vingt-cinquième et le vingt-sixième ont obtenu la même moyenne, la requête renvoie 26 enregistrements. Vous pouvez également utiliser le mot réservé PERCENT pour renvoyer un certain pourcentage des premiers ou derniers enregistrements d'une plage spécifiée par la clause ORDER BY. Supposons qu'au lieu des 25 meilleurs étudiants, vous souhaitiez sélectionner 10 pour cent de la promotion :

Exemple :

```
    SELECT TOP 10 PERCENT
Nom, Prénom
FROM Etudiants
WHERE Promotion = 1994
ORDER BY Moyenne ASC;
```

Le prédicat ASC donne des valeurs croissantes. La valeur qui suit TOP doit être un entier non signé. TOP n'affecte pas les possibilités de mise à jour de la requête

D. Sous-interrogation synchronisée ou bien corrélées

Il est possible de synchroniser une sous-interrogation avec l'interrogation principale.

Dans les exemples précédents, la sous-interrogation pouvait être évaluée d'abord, puis le résultat utilisé pour exécuter l'interrogation principale. SQL sait également traiter une sous-interrogation faisant référence à une colonne de la table de l'interrogation principale.

Le traitement dans ce cas est plus complexe car il faut évaluer la sous interrogation pour chaque ligne de l'interrogation principale.

Exemple :

Liste des employés ne travaillant pas dans le même département que leur supérieur.

```
SELECT NOME FROM EMP  AS  E
WHERE DEPT != (SELECT DEPT FROM EMP
WHERE MATR = E.SUP)
```

Il a fallu renommer la table EMP de l'interrogation principale pour pouvoir la référencer dans la sous-interrogation.

E. Sous-interrogation ramenant plusieurs colonnes

Il est possible de comparer le résultat d'un SELECT ramenant plusieurs colonnes à une liste des colonnes. La liste de colonnes figurera entre parenthèses à gauche de l'opérateur de comparaison.

Avec une seule ligne sélectionnée :

Syntaxe :

```
WHERE (exp, exp,...) op (SELECT ...)
Avec plusieurs lignes sélectionnées :

WHERE (exp, exp,...) op ANY (SELECT ...)
WHERE (exp, exp,...) op ALL (SELECT ...)
WHERE (exp, exp,...) IN (SELECT ...)
```

où op est un des opérateurs = ou <> Les expressions _gurant dans la liste entre parenthèses seront comparées à celles qui sont ramenées par le SELECT.

Exemple :

Employés ayant même poste et même salaire que MERCIER :

```
SELECT NOME, POSTE, SAL FROM EMP
WHERE (POSTE, SAL) =
(SELECT POSTE, SAL FROM EMP
WHERE NOME = 'MERCIER')
```

On peut utiliser ce type de sous-interrogation pour retrouver les lignes qui correspondent à des optima sur certains critères pour des regroupements de lignes (voir dernier exemple des exemples

F. Clause EXISTS

La clause EXISTS est suivie d'une sous-interrogation entre parenthèses, et prend la valeur vrai s'il existe au moins une ligne satisfaisant les conditions de la sous-interrogation.

Exemple :

```
SELECT NOMD FROM DEPT
WHERE EXISTS (SELECT NULL FROM EMP
WHERE DEPT = DEPT.DEPT AND SAL > 10000);
```

Cette interrogation liste le nom des départements qui ont au moins un employé ayant plus de 10.000 comme salaire ; pour chaque ligne de DEPT la sous-interrogation synchronisée est exécutée et si au moins

une ligne est trouvée dans la table EMP, EXISTS prend la valeur vrai et la ligne de DEPT satisfait les critères de l'interrogation.

Souvent on peut utiliser IN à la place de la clause EXISTS. Essayez sur l'exemple précédent.

REMARQUE :

Il faut se méfier lorsque l'on utilise EXISTS en présence de valeurs NULL. Si on veut par exemple les employés qui ont la plus grande commission par la requête suivante,

```
select nome from emp e1
where not exists
(select matr from emp
where comm > e1.comm)
```

on aura en plus dans la liste tous les employés qui ont une commission NULL.

G. Division avec la clause EXISTS

NOT EXISTS permet de spécifier des prédicats où le mot « tous » intervient dans un sens comparable à celui de l'exemple. Elle permet d'obtenir la division de deux relations.

On rappelle que la division de R par S sur l'attribut B (notée R ÷S)

Est define par:

$$D = \{a \in R[A] / \forall b \in S, (a,b) \in R\} = \{a \in R[A] / \nexists b \in S, (a,b) \notin R\}$$

Faisons une traduction « mot à mot » de cette dernière définition en langage SQL

Exemple : 1

```
select A from R R1
where not exists
(select C from S
where not exists
(select A, B from R
where A = R1.A and B = S.C))
```

En fait, on peut remplacer les colonnes des selects placés derrière des « not Exists » par ce que l'on veut, puisque seule l'existence ou non d'une ligne compte. On peut écrire par exemple :

Exemple : 2

```
select A from R R1
where not exists
(select null from S
where not exists
(select null from R
where A = R1.A and B = S.C))
```

On arrive souvent à optimiser ce type de select en utilisant les spécificités du cas, le plus souvent en simplifiant le select externe en remplaçant une jointure de tables par une seule table.

La réponse à la question « Quels sont les départements qui participent à

tous les projets ?_ est fourni par R _Dept S où R = (PARTICIPATION

JN{Matr} EMP) [Dept, CodeP] (_JN{Matr}_ indique une jointure naturelle

sur l'attribut Matr) et S = PROJET [CodeP]

Il reste à faire la traduction _mot à mot_ en SQL :

Exemple : 3

```
SELECT DEPT
FROM PARTICIPATION NATURAL JOIN EMP E1
WHERE NOT EXISTS
(SELECT CODEP FROM PROJET
WHERE NOT EXISTS
(SELECT DEPT, CODEP
FROM PARTICIPATION NATURAL JOIN EMP
WHERE DEPT = E1.DEPT
AND CODEP = PROJET.CODEP))
```

Remarque 4.10

REMARQUE : 1

Il faudrait ajouter DISTINCT dans le premier select pour éviter les doublons. Sur ce cas particulier on voit qu'il est inutile de travailler sur la jointure de PARTICIPATION et de EMP pour le SELECT externe. On peut travailler sur la table DEPT. Il en est de même sur tous les cas où la table « R» est une jointure. D'après cette remarque, le SELECT précédent devient :

```
SELECT DEPT FROM DEPT
WHERE NOT EXISTS
(SELECT CODEP FROM PROJET
WHERE NOT EXISTS
(SELECT DEPT, CODEP
FROM PARTICIPATION NATURAL JOIN EMP
WHERE DEPT = DEPT.DEPT
AND CODEP = PROJET.CODEP))
```

REMARQUE : 2

Dans le cas où il est certain que la table dividende (celle qui est divisée ne contient dans la colonne qui sert pour la division que des valeurs qui existent dans la table diviseur, on peut exprimer la division en utilisant les regroupements (étudiés dans la prochaine section) et en comptant les lignes regroupées. Pour l'exemple des départements qui participent à tous les projets, on obtient :

```
select dept
from emp natural join participation
group by dept
```

```
having count(distinct codeP) =
(select count(distinct codeP) from projet)
```

Traduction : si le nombre des codeP associés à un département donné est égal au nombre des tous les codeP possibles, ça signifie que ce département est associé à tous les départements. Ici on a bien le résultat cherché car les codeP du select sont nécessairement des codeP de la table des projets (clé étrangère de PARTICIPATION qui référence la clé primaire de la table PROJET). Si un ensemble A est inclus dans un ensemble B et si A a le même nombre d'éléments que B, c'est que A =B.

Mais il ne faut pas oublier que dans des requêtes complexes les données qui interviennent dans les divisions peuvent provenir de requêtes emboîtées. Il n'y a alors pas nécessairement de contraintes de référence comme dans l'exemple traité ici (contrainte qui impose que codeP doit nécessairement correspondre à un codeP dans la table Projet). Si on n'a pas A _ B, le fait que A et B aient le même nombre d'éléments ne signifie pas que A = B.

S'il peut y avoir dans la colonne qui sert pour la division des valeur qui n'existent pas dans la table diviseur, la requête est légèrement plus complexe :

```
select dept
from emp natural join participation
where codeP in
(select codeP from projet)
group by dept
having count(distinct codeP) =
(select count(distinct codeP) from projet)
```

III. Fonctions de groupes

Les fonctions de groupes peuvent apparaître dans le Select ou le Having ; ce sont les fonctions suivantes :

AVG	moyenne
SUM	somme
MIN	plus petite des valeurs
MAX	plus grande des valeurs
VARIANCE	variance
STDDEV	écart type (déviation standard)
COUNT(*)	nombre de lignes

COUNT(col) nombre de valeurs non nulles de la colonne COUNT(DISTINCT col) nombre de valeurs non nulles di_érentes

Exemple :

```
(a) SELECT COUNT(*) FROM EMP
(b) SELECT SUM(COMM) FROM EMP WHERE DEPT = 10
```

Les valeurs NULL sont ignorées par les fonctions de groupe. Ainsi, SUM(col) est la somme des valeurs qui ne sont pas égales à NULL de la colonne 'col'.

De même, AVG est la somme des valeurs non NULL divisée par le nombre de valeurs non NULL.

Il faut remarquer qu'à un niveau de profondeur (relativement aux sous interrogations), d'un SELECT, les fonctions de groupe et les colonnes doivent être toutes du même niveau de regroupement. Par exemple, si on veut le nom et le salaire des employés qui gagnent le plus dans l'entreprise, la requête suivante provoquera une erreur :

ATTENTION :

```
SELECT NOME, SAL FROM EMP
WHERE SAL = MAX(SAL)
```

Il faut une sous-interrogation car MAX(SAL) n'est pas au même niveau de regroupement que le simple SAL :

```
SELECT NOME, SAL FROM EMP
WHERE SAL = (SELECT MAX(SAL) FROM EMP)
```

Clause GROUP BY

Il est possible de subdiviser la table en groupes, chaque groupe étant l'ensemble des lignes ayant une valeur commune.

Syntaxe :

```
GROUP BY exp1, exp2,...
```

groupe en une seule ligne toutes les lignes pour lesquelles exp1, exp2,... ont la même valeur. Cette clause se place juste après la clause WHERE, ou après la clause FROM si la clause WHERE n'existe pas.

Des lignes peuvent être éliminées avant que le groupe ne soit formé grâce à la clause WHERE.

Exemple :

```
(a) SELECT DEPT, COUNT(*) FROM EMP
GROUP BY DEPT

(b) SELECT DEPT, COUNT(*) FROM EMP
WHERE POSTE = 'SECRETAIRE'
GROUP BY DEPT

(c) SELECT DEPT, POSTE, COUNT(*) FROM EMP
GROUP BY DEPT, POSTE
(d) SELECT NOME, DEPT FROM EMP
WHERE (DEPT, SAL) IN
(SELECT DEPT, MAX(SAL) FROM EMP
GROUP BY DEPT)
```

RESTRICTION :

Une expression d'un SELECT avec clause GROUP BY ne peut évidemment que correspondre à une caractéristique de groupe. SQL n'est pas très « intelligent » pour comprendre ce qu'est une caractéristique de groupe ; une expression du SELECT ne peut être que :

- soit une fonction de groupe,
- soit une expression figurant dans le GROUP BY.

L'ordre suivant est invalide car NOMD n'est pas une expression du GROUP BY :

```
SELECT NOMD, SUM(SAL)
FROM EMP NATURAL JOIN DEPT
GROUP BY DEPT
```

Il faut, soit se contenter du numéro de département au lieu du nom :

```
SELECT DEPT, SUM(SAL)
FROM EMP NATURAL JOIN DEPT
GROUP BY DEPT
```

Soit modifier le GROUP BY pour avoir le nom du département :

```
SELECT NOMD, SUM(SAL)
FROM EMP NATURAL JOIN DEPT
GROUP BY NOMD
```

Clause HAVING

HAVING prédicat Sert à préciser quels groupes doivent être sélectionnés.

Elle se place après la clause GROUP BY.

Le prédicat suit la même syntaxe que celui de la clause WHERE. Cependant,

il ne peut porter que sur des caractéristiques de groupe : fonction de

Groupe ou expression figurant dans la clause GROUP BY.

Exemple :

```
SELECT DEPT, COUNT (*)
FROM EMP
WHERE POSTE = 'SECRETAIRE'
GROUP BY DEPT HAVING COUNT(*) > 1
```

On peut évidemment combiner toutes les clauses, des jointures et des Sous-interrogations. La requête suivante donne le nom du département (et son nombre de secrétaires) qui a le plus de secrétaires :

```
SELECT NOMD Département, COUNT(*)  as "Nombre de secrétaires"
FROM EMP NATURAL JOIN DEPT
WHERE POSTE = 'SECRETAIRE'
GROUP BY NOMD HAVING COUNT(*) =
(SELECT MAX(COUNT(*)) FROM EMP
WHERE POSTE = 'SECRETAIRE'
GROUP BY DEPT)
```

On remarquera que la dernière sous-interrogation est indispensable car MAX (COUNT(*)) n'est pas au même niveau de regroupement que les autres Expressions du premier SELECT.

1. Exercice d'application

1- La requête qui affiche le salaire moyen par département
2- La requête qui affiche les employés qui ont un salaire supérieur au salaire moyen de leur département
3- La requête qui affiche les départements qui ont pour salaire moyen supérieur au salaire moyen de tous les employés

2. Reponses

1- Select nom_dep ,avg(salaire) as 'SalMoy' from employe Emp INNER JOIN Departement Dep
ON Emp.CodeDep=Dep.CodeDep
GROUP BY nom_dep

2- Select nom, salaire from Employe e1
where
e1.salaire >=(select AVG(salaire) as 'SalMoy'from Employe e2
where e2.CodeDep=e1.CodeDep
GROUP BY CodeDep)

IV. Fonctions

Nous allons décrire ci-dessous les principales fonctions disponibles dansOracle. Il faut remarquer que ces fonctions ne sont pas standardisées et ne Sont pas toutes disponibles dans les autres SGBD; elles peuvent aussi avoir Une syntaxe différente, ou même un autre nom.

A. Fonctions arithmétiques

ABS(n)	valeur absolue de n
MOD (n1, n2)	n1 modulo n2
POWER (n, e)	n à la puissance e
ROUND (n [, p])	arrondit n à la précision p (0 par défaut)
SIGN(n)	-1 si n<0, 0 si n=0, 1 si n>0
SQRT(n)	racine carrée de n
TRUNC (n [, p])	tronque n à la précision p (0 par défaut)
CONVERT	Convertit une expression d'un type de données en un autre.

Exemple : Calcul du salaire journalier

```
SELECT NOME, ROUND (SAL/22, 2) FROM EMP
```

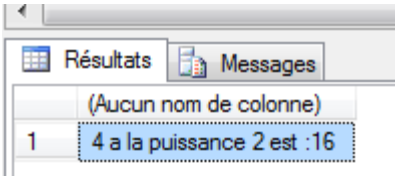
B. Fonctions chaîne de caractères

Concaténation :

il est possible de concaténer des chaînes avec l'opérateur + ;

Exemple :

```
select '4 a la puissance 2 est :' + convert(nvarchar(10),POWER(2,4))
```



LENGTH (chaîne) :

Prend comme valeur la longueur de la chaîne.

SUBSTRING :

SUBSTRING (chaîne, position [, longueur])

SUBSTRING('salut',2,3) retourne alu

Extrait de la chaîne chaîne une sous-chaîne de longueur « longueur » commençant En position « position » de la chaîne. Le paramètre longueur est facultatif : par défaut, la sous-chaîne va jusqu'à l'extrémité de la chaîne.

UPPER :

UPPER (chaîne) convertit les minuscules en majuscules

LOWER :

LOWER (chaîne) convertit les majuscules en minuscules

LTRIM :

LTRIM (chaîne)

Renvoie une chaîne de caractères après avoir supprimé les espaces de début.

select LTRIM(' salut') retoutne 'salut'

RTRIM :

RTRIM (chaîne)

Renvoie une chaîne de caractères après la suppression des espaces de fin

select RTRIM('salut') retourne 'salut'

LEFT :

Retourne la partie de gauche d'une chaîne de caractères avec le nombre spécifié de caractères.

RIGHT :

Renvoie la partie d'une expression de caractères qui commence et se situe à droite d'une position de caractère spécifiée à partir de la droite.

REPLACE :

Renvoie une expression de caractères après le remplacement d'une chaîne de caractères située dans l'expression par une autre chaîne de caractères ou une chaîne vide

REPLACE('06:67:87:67/103',':','.') retourne O6.67.87.67.10

C. Les fonctions des dates :

DATEADD :

Renvoie une nouvelle valeur DT_DBTIMESTAMP après l'ajout d'un nombre qui représente un intervalle de date ou d'heure à la partie de date spécifiée d'une date. Le paramètre numérique doit

Exemple : L'exemple suivant ajoute un mois à la date actuelle.

```
DATEADD("Month", 1,GETDATE())
```

Exemple : L'exemple suivant ajoute 21 jours aux dates de la colonne ModifiedDate

```
DATEADD("day", 21, ModifiedDate)
```

DATEDIFF :

Renvoie le nombre de limites de date et d'heure traversées entre deux dates données. Le paramètre datepart identifie quelles limites de date et d'heure il faut comparer.

Le tableau suivant décrit les parties de date et les abréviations reconnues par l'évaluateur d'expression.

Partie de date	Abréviations
Année	yy, yyyy

Trimestre	qq, q

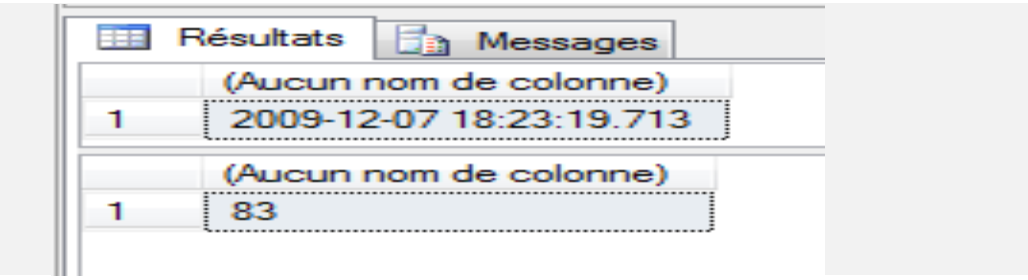
Exemple : L'exemple suivant calcule le nombre de jours entre deux littéraux de date. la fonction renvoie 7

```
  select DATEDIFF(dd, '1/8/2003', '8/8/2003')
```

Mois	mm, m
Jour de l'année	dy, y
Jour	dd, d
Semaine	wk, ww
Jour de la semaine	dw, w
Heure	Hh
Minute	mi, n
Seconde	ss, s
Milliseconde	Ms

Exemple :

```
La difference entre la date '8/1/2003' ET LA DATE actuelle
07/12/2009
select GETDATE()
  SELECT DATEDIFF(mm, '8/1/2003',GETDATE())
```



CHAPITRE 6

Leçon 1 : Transaction SQL

Objectifs : vous serez à même d'effectuer les tâches suivantes

- Commencer, valider ou annuler des transactions
- Gérer les erreurs par programmation

I. Initialisation validation ou annulation de transaction

Lorsque vous modifiez des données dans une base de données, une des choses les plus importantes à prendre en compte par les développeurs est la façon de s'assurer que ces données restent dans un état cohérent. un état cohérent signifie qu'à tout moment toutes les données de la base de données doivent être correctes.les données incorrectes doivent être supprimées ou mieux encore, jamais insérées.

Les transactions sont le mécanisme primaire par lequel vous pouvez par programmation veiller à la cohérence des données. Lorsque vous débutez une transaction, toute modification de données effectuée n'est, par défaut, visible que par votre connexion.les autres connexions ne peuvent pas voir votre modification. Elles doivent attendre que la transaction soit validée (la modification est inscrite dans la base de données) ou annulée

, auquel cas les données retrouvent leur aspect antérieur au début de la transaction

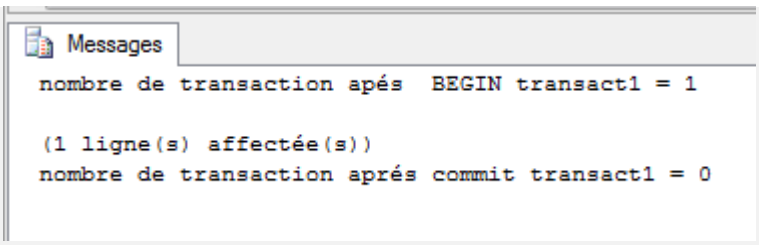
Le processus fondamental à employer lors du travail avec les transactions est le suivant :

1. Initiez une transaction à l'aide de la commande BEGIN TRANSACTION
2. Une transaction initiée par BEGIN TRANSACTION va enregistrer toutes les modifications de données accomplies par votre connexion : insertion, mise à jour et /ou suppression
3. La transaction ne prend fin que lorsque vous la validez ou que vous l'annulez

Vous validez une transaction et enregistrer les modifications à l'aide de la commande COMMIT TRANSACTION ou l'annulez à l'aide de la commande ROLLBACK TRANSACTION.si a tout moment après le début de la transaction, vous détectez un problème, ROLLBACK transaction permet de revenir aux données originales.

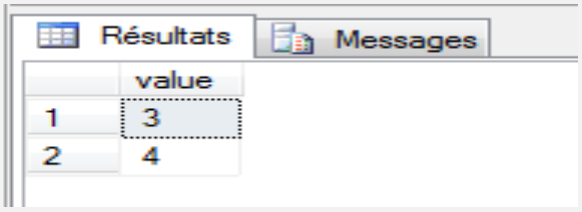
Exemple :

```
USE AdventureWorksDW2008;
GO
IF OBJECT_ID(N'TestTran',N'U') IS NOT NULL
    DROP TABLE TestTran;
GO
CREATE TABLE TestTran (Cola INT PRIMARY KEY, Colb
CHAR(3));
GO
-- This statement sets @@TRANCOUNT to 1.
BEGIN TRANSACTION transact1;
GO
PRINT N'nombre de transaction apés  BEGIN transact1 = '
    + CAST(@@TRANCOUNT AS NVARCHAR(10));
GO
INSERT INTO TestTran VALUES (1, 'aaa');
GO

COMMIT TRANSACTION transact1;
PRINT N'nombre de transaction aprés commit transact1 = '
    + CAST(@@TRANCOUNT AS NVARCHAR(10));
  GO
```



Exemple : Utilisation de ROLLBACK TRANSACTION

```
USE TempDB;
GO
CREATE TABLE ValueTable ([value] int)
GO


--debut de la procedure Transaction1,
--insertion de deux lignes

BEGIN TRANSACTION Transaction1;
       INSERT INTO ValueTable VALUES(1);
       INSERT INTO ValueTable VALUES(2);
ROLLBACK TRANSACTION Transaction1;

INSERT INTO ValueTable VALUES(3);
INSERT INTO ValueTable VALUES(4);

SELECT * FROM ValueTable

DROP TABLE ValueTable
```



A. Détection des erreurs

Nous allons donc ajouter une partie de gestion d'erreurs à notre code. Pour récupérer une erreur, il suffit d'employer la variable prédéfini @@ERROR qui contient l'erreur pour la dernière requête effectuée. Nous allons donc déclarer une variable @errors pour stocker les différentes erreurs récupérées lors de l'exécution :

```
  BEGIN TRANSACTION changement_etat_civil --On démarre une transaction et
on lui donne un nom

  DECLARE @errors INT --On déclare une variable qui sera destiné à
accueillir nos erreurs
  DECLARE @ID_INSERTION NUMERIC(19,0) -- On déclare une variable
numérique destinée à contenir l'id inséré

  UPDATE T_TEXTES SET TEXTE='Divorcé(e)' WHERE ID = '1'
  SET @error = @error + @@error
  UPDATE T_TEXTES SET TEXTE='IT:Divorcé(e)' WHERE ID = '2'
  SET @error = @error + @@error

  UPDATE T_TEXTES SET TEXTE='Marié(e)' WHERE ID = '3'
  SET @error = @error + @@error

INSERT INTO T_VALEUR_PARAMETRE (version, PARAM_CODE, OFFICE, PARAMETRE)
  VALUES (GetDate() 'etatCivil.values.pacs.value', '6', '99')
  SET @errors = @errors + @@ERROR --On additionne l'erreur liée à la
dernière requête SQL dans notre variable

  SET @errors = @errors + @@ERROR --On additionne l'erreur liée à la
dernière requête SQL dans notre variable
  SET @ID_INSERTION = @@identity --On récupère la valeur du dernier id
inséré et on le stocke dans notre variable

  INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
  VALUES('Lié(e) par un partenariat enregistré','fr', @ID_INSERTION)

  SET @errors = @errors + @@ERROR --On additionne l'erreur liée à la
dernière requête SQL dans notre variable

  INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
  VALUES('IT:Lié(e) par un partenariat enregistré','it', @ID_INSERTION)

  SET @errors = @errors + @@ERROR --On additionne l'erreur liée à la
dernière requête SQL dans notre variable

  INSERT INTO T_TEXTES (TEXTE, LANGUE, VALEUR_PARAMETRE)
  VALUES('Verbunden durch eine eingetragene Partnerschaft','de',
@ID_INSERTION)

  SET @errors = @errors + @@ERROR --On additionne l'erreur liée à la
dernière requête SQL dans notre variable

  COMMIT TRANSACTION changement_etat_civil --On commit cette transaction,
c'est à dire qu'on valide ses modifications
```

Vous pouvez aussi afficher les erreurs avec :

```
  PRINT 'Statut de l'erreur : ' + CAST(@errors AS VARCHAR(10)) --On
affiche le statut de l'erreur casté sous forme de caractère
```

Juste avant le commit.

B. Gestion des erreurs

Bon, c'est bien beau, vous me direz, on voit les erreurs, mais en cas d'erreurs notre base est toujours incohérente. Bien, ca prouve que vous suivez. Une astuce pour cela est de déclarer notre variable @errors à 1, comme ça, si à la fin de l'exécution des requêtes, elle n'est plus à un, on sait qu'il y a eu une erreur et on peut faire quelque chose.

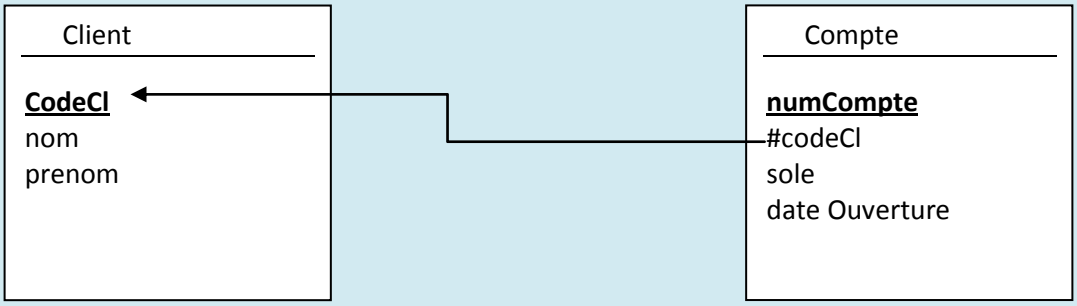
```
DECLARE @errors INT

SET @errors = 0 --On déclare notre variable à 0

--Vos requêtes
```

On va maintenant contrôler s'il y a eu ou non des erreurs et s'il y en a eu, on va annuler toutes les opérations faites dans la transaction en utilisant la commande ROLLBACK TRANSACTION :

```
IF @errors = 0 --Si errors est égale à 0, donc s'il n'y a eu aucune
erreur
        COMMIT TRANSACTION changement_etat_civil -- On commit la
transaction
ELSE --S'il y a eu des erreurs
        ROLLBACK TRANSACTION changement_etat_civil --On annule tous les
changements de cette transaction
```

Travaux pratiques

Soit la base de données suivante :



On veut transférer un montant d'un compte C1 a un autre compte C2
C à dire que le compte C1 sera débité d'un montant M1 et ce même montant sera crédité
Au compte C2 autrement dit on va soustraire le montant M1 le solde du compte C1
Et ajouter M1 au solde du compte C2 ces deux actions doivent constituer une seule transaction pour avoir la cohérence des données.
Faire une conception transactionnelle pour répondre a ce besoin

Liste des références bibliographiques.

Ouvrage	***Auteur***	***Edition***
Implémentation et maintenance de SQL server	Microsoft Press	
Langage SQL	richard Grin version 5.4.5 du polycopié	
http://webtic.free.fr/sql/		
http://georges.gardarin.free.fr/		
http://baptiste-wicht.developpez.com/tutoriel/ms-sql/securiser		