LE SQL de A à Z : 1ere partie - bases de données, SQL et types de données

Date de publication : 26/08/2003

Par SQLPro (autres articles) (CV)

Niveau : débutant

Voici le premier chapitre d'un long article concernant le SQL et son implémentation dans les SGBD les plus courants (Paradox, Access, Oracle, SQL Server, Sybase...). Tout le monde connaît le SQL, mais l'avez-vous suffisamment fréquenté pour en tirer toute l'essence ? Dans cet article, nous allons voir l'historique de ce langage et ses différentes composantes. Dans les suivants, nous verrons dans l'ordre la manipulation des données à l'aide des commandes SELECT, INSERT, UPDATE, DELETE, puis la création des structures de bases de données avec CREATE, ALTER, DROP et enfin l'attribution et la révocation des droits (GRANT et REVOKE). Mais au fait, le SQL, bientôt mort ?... Pas si sûr !

Préambule

Parler du SQL de nos jours comme d'une nouveauté, serait une gageure... cependant, ne faut-il pas voir en cet indestructible langage, une tentative un peu tardive, mais souhaitable, au travers des différents middlewares disponibles, de standardisation d'un mode d'interrogation des données?
Force est de constater que même les bases de données objet et le web se mettent au SQL. Le poids du passé sans doute...

Mais alors que faire ? Squeezer SQL ou s'en accommoder?

Il y a plus d'une vingtaine d'années le COBOL était, disait-on, assuré d'une mort certaine et à court terme. Aujourd'hui le défunt est encore haletant bien que défraîchi. En sera t-il de même pour le SQL ? Tout le laisse supposer!

1. SQL un langage ?

Vous trouverez des compléments d'information sur le sujet aux pages 29 à 51 de l'ouvrage "SQL"*, collection* "*La Référence*"*, Campus Press éditeur.*

Et d'abord, SQL est-il un vrai langage ?

Si l'on doit accepter ce mot au sens informatique du terme, il semble difficile de dire oui tant SQL est loin de la structure et des possibilités d'un langage de programmation courant. Point de variable, point de procédure ou de fonction... Pourtant il s'agit bien de former des phrases qui seront compilées afin d'exécuter des traitements.
Même si SQL s'est doté au fil du temps d'extensions, comme la possibilité de paramétrer les requêtes, il y a des lacunes importantes qui sont autant de frein à sa pénétration. Par exemple SQL ne sait être récursif [1] alors que ce mode d'exécution est nécessaire pour résoudre une frange importante de problèmes, notamment ceux pour traiter les arbres ou les graphes.

En fait SQL est un langage de type " déclaratif ". On spécifie ce que l'on veut obtenir ou faire et c'est la machine qui décide comment elle doit l'exécuter. Pour certains, SQL est perçu comme un pis-aller tandis que d'autres préfèrent l'éviter ou retarder au plus l'inéluctable moment où son apparition sera incontournable.

La différence fondamentale entre les langages courants comme C ou Pascal, qui sont des langages procéduraux, réside dans le fait qu'avec ces derniers, vous indiquez l'ensemble des instructions nécessaires à traiter un problème. Vous gardez ainsi une certaine maîtrise sur le cycle d'exécution et l'enchaînement des différentes tâches d'un programme vous est parfaitement connu. En revanche, dans SQL vous n'avez, et d'ailleurs ne devez, pas avoir la moindre idée de comment la machine exécute votre demande, ni même dans quel ordre elle

décompose le traitement de la requête en différentes tâches, ni d'ailleurs même comment elle les synchronise.

Lors d'une extraction concernant plusieurs tables, le moteur relationnel pourrait parfaitement lancer plusieurs traitements en parallèles pour extraire les données de chacune des tables, puis effectuer les jointures des différentes tables résultantes en fonction de l'ordre de terminaison des extractions... C'est pourquoi, les moteurs relationnels incluent des modules d'optimisation logique et parfois statistique.

Ainsi un optimiseur logique préférera traiter en premier les clauses excluant de la réponse un maximum de données, tandis qu'un optimiseur statistique commencera par traiter les tables de plus faible volume.

Exemple :

Soit une table contenant le personnel de la société " VIVE LE SQL " et une autre contenant
les salaires mensuels desdits employés.

" VIVE LE SQL " compte 1000 employés soit autant de lignes dans la table du personnel, et
la moyenne de durée d'emploi étant de 6 ans, la table des salaires compte donc environ
72 000 lignes.

Soit la requête suivante : rechercher les employés dont le nom commence par Z ou qui sont
domiciliés à Trifouilly et qui ont eu au moins un salaire supérieur à 50 000 Francs.

* Un optimiseur logique commencera immanquablement par traiter la table des salaires
puis celle des employés.
* Un optimiseur statistique commencera par traiter les employés puis les salaires.

Nous pauvres humains, aurions immédiatement traité la table des salaires, car à ce niveau
d'émolument il ne devrait pas y avoir grand monde dans la table des salariés...

Eh bien, croyez moi ou pas... certains optimiseurs statistiques auraient eût la même appréciation !
En effet les plus performants ne se contentent pas seulement de conserver le nombre de lignes ou
le volume des données d'une table, mais aussi les moyennes, médianes, maximums et minimums de
certaines colonnes, voir même le nombre de valeurs distinctes (indice de dispersion) ainsi que
le nombre d'occurrences des valeurs les plus fréquentes, notamment pour les colonnes pourvues
d'index... C'est le cas en particulier du SGBDR INGRES.

Bref, pour conclure cette brève présentation de SQL, nous pouvons affirmer que, malgré ses défauts, et en attendant un futur langage objet pour l'interrogation des données, incluant tous les mécanismes dynamiques et bien entendu la récursivité [1], il nous faut continuer à utiliser sagement SQL, du mieux que nous pouvons, en rendant ainsi hommage a quelques-uns de ses créateurs...

2. SQL une histoire...

Nous sommes en 1970. Le docteur Codd, un chercheur d'IBM à San José, propose une nouvelle manière d'aborder le traitement automatique de l'information, se basant sur la théorie de l'algèbre relationnel (théorie des ensembles et logique des prédicats). Cette proposition est faite afin de garantir une plus grande indépendance entre la théorie et l'implémentation physique des données au sein des machines. C'est ainsi que naîtrons, vers la fin des années 70, les premières applications basées sur la proposition de Ted Codd, connues de nos jours sous l'acronyme SGBDR (signifiant Système de Gestion de Bases de Données Relationnelles).

Dans cette même période, Peter CHEN tente une approche pragmatique et conceptuelle du traitement automatique des données en proposant le modèle Entité-Association comme outil de modélisation. Nous sommes en 1976.

En parallèle, différents chercheurs travaillent à réaliser ce que seront les SGBDR d'aujourd'hui : citons entre autres l'équipe de Gene Wong à l'université de Berkeley qui entame le projet INGRES en 1972, et en 1975 propose le langage QUEL comme outil d'interrogation des données (aucun ordre de mise à jour des données n'y figure).
De même à Noël 1974 démarre chez IBM le projet qui portera le nom de System R, et donnera comme langage d'interrogation SEQUEL (Structured English QUEry Langage) en 1976.

Lors d'une conférence internationale à Stockolm (IFIP Congres), Larry Ellison, dirigeant d'une petite entreprise appelée Software Development Laboratories entend parler du projet et du langage d'interrogation et de manipulation des données. Il rentre en contact avec les chercheurs d'IBM. Quelques années après, Larry Ellison sort le premier SGBDR commercialisé et change le nom de sa société. Elle s'appellera désormais Oracle...

IBM, pour sa part, sortira une version commerciale bien après celle d'Oracle, ce qui fera dire à l'un des responsables du projet que " cela vous montre combien de temps il faut à IBM pour faire n'importe quoi... ". Finalement la première installation de System R est réalisée en 1977 chez Pratt & Whitney, et déjà les commandes du langage d'interrogation se sont étoffées de quelques judicieuses techniques préconisées par Oracle, comme la gestion des curseurs.

Au même moment apparaissent d'autres langages d'interrogation comme QBE de Zloof (IBM 1977) repris pour Paradox par Ansa Software, ou REQUEST de Fred Damerau (basé sur le langage naturel) ou encore RENDEZ VOUS (1974) de l'équipe de Ted Codd ainsi que SQUARE (Boyce 1975).

Mais le point critique du langage SEQUEL porte sur l'implémentation du ... rien ! Ou plutôt devrais-je dire du nul !!!

Finalement la première réalisation connue sous le nom de SQL (82) voit le jour après l'arrivée de DB2, une avancée significative du System R d'IBM datant de mars 1979.

On peut donc dire que le langage SQL est né en 1979, mais baptisé SQL en 1982. En définitive SQL a vingt ans, et, comme le dit la chanson ... on n'a pas tous les jours 20 ans...

Et 20 ans pour les uns ça suffit !, pour les autres ça se fête...

2.1. SQL une norme

Notons que SQL sera normalisé à quatre reprises : 1986 (SQL 86 - ANSI), 1989 (ISO et ANSI) et 1992 (SQL 2 - ISO et ANSI) et enfin 1999 (SQL:1999 - ISO) souvent apellé à tort (y compris par moi même !) SQL 3. A ce jour, aucun SGBDR n'a implémenté la totalité des spécifications de la norme actuellement en vigueur. Mais je dois dire que le simple (?) SELECT est argumenté de quelques 300 pages de spécifications syntaxiques dans le dernier document normatif...

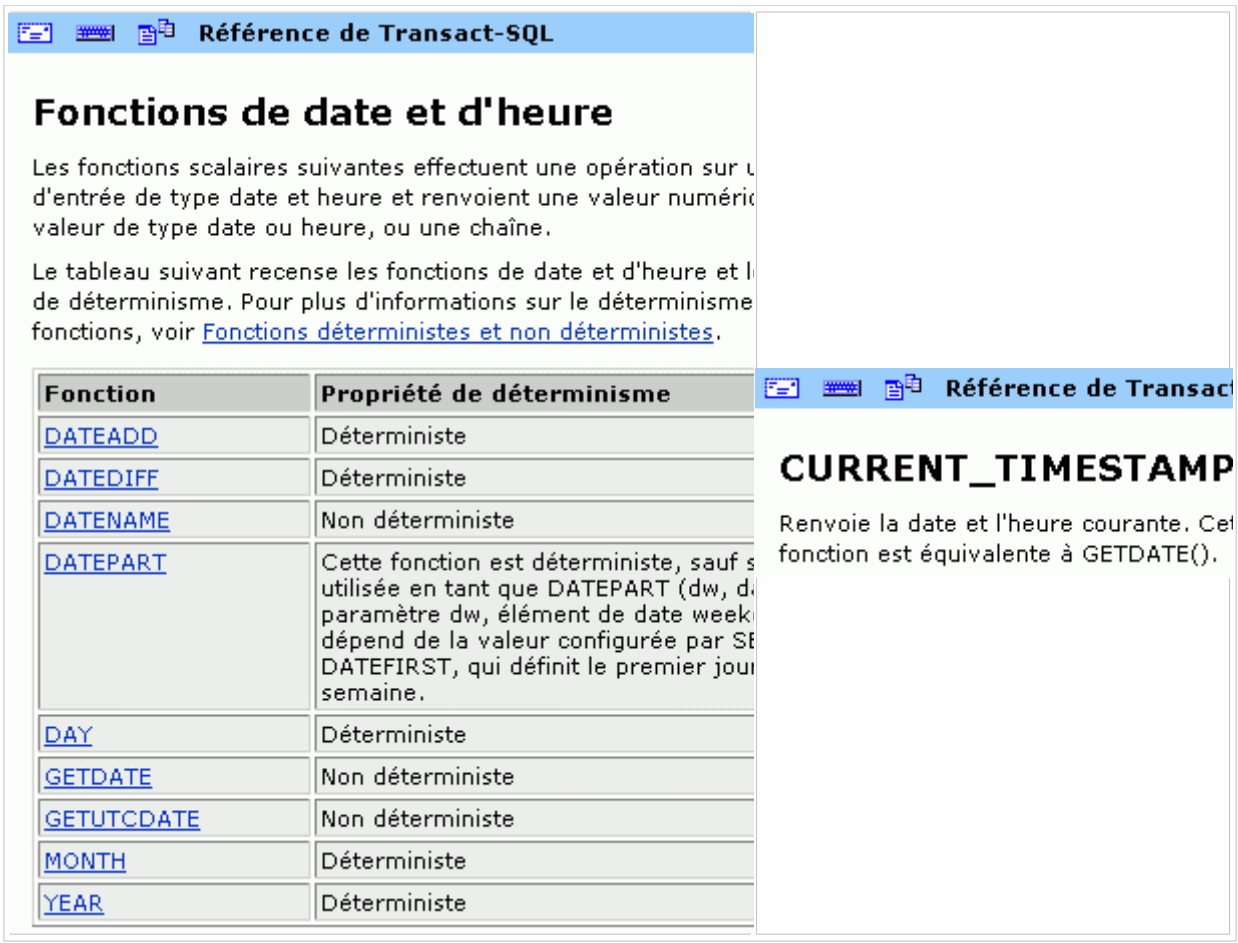
Néanmoins, la version SQL 2 (1992) est la version vers laquelle toutes les implémentations tendent. C'est pourquoi nous nous baserons sur SQL 2.

A noter, la future norme SQL:2003, sortira en juillet 2003.

2.2. SQL un standard

Même si SQL est considéré comme le standard de toutes les bases de données relationnelles et commercialisées, il n'en reste pas moins vrai que chaque éditeur tend à développer son propre dialecte, c'est à dire à rajouter des éléments hors de la norme. Soit fonctionnellement identiques mais de syntaxe différentes (le LIKE d'Access en est l'exemple le plus stupide !) soit fonctionnellement nouveau (le CONNECT BY d'Oracle pour la récursivité par exemple). Il est alors très difficile de porter une base de données SQL d'un serveur à l'autre. C'est un moindre mal si l'on a respecté au maximum la norme, mais il est de notoriété absolue que lorsque l'élément normatif est présent dans le SGBDR avec un élément spécifique ce sera toujours l'élément spécifique qui sera proposé et documenté au détriment de la norme !

Exemple, la fonction CURRENT_TIMESTAMP, bien présente dans MS SQL Server, n'est pas spécifié dans la liste des fonctions temporelle de l'aide en ligne !!!



Référence de Transact-SQL

Fonctions de date et d'heure

Les fonctions scalaires suivantes effectuent une opération sur u
d'entrée de type date et heure et renvoient une valeur numéric
valeur de type date ou heure, ou une chaîne.

Le tableau suivant recense les fonctions de date et d'heure et l
de déterminisme. Pour plus d'informations sur le déterminisme
fonctions, voir Fonctions déterministes et non déterministes.

Fonction	Propriété de déterminisme
DATEADD	Déterministe
DATEDIFF	Déterministe
DATENAME	Non déterministe
DATEPART	Cette fonction est déterministe, sauf s utilisée en tant que DATEPART (dw, d paramètre dw, élément de date week dépend de la valeur configurée par SE DATEFIRST, qui définit le premier jou semaine.
DAY	Déterministe
GETDATE	Non déterministe
GETUTCDATE	Non déterministe
MONTH	Déterministe
YEAR	Déterministe

Référence de Transac

CURRENT_TIMESTAMP

Renvoie la date et l'heure courante. Cet
fonction est équivalente à GETDATE().

De plus, le plus petit dénominateur commun entres les 4 poids lourds de l'édition que sont Oracle, Sybase (ASE), Microsoft (SQL Server) et IBM (DB2) est tel qu'il est franchement impossible de réaliser la moindre base de données compatible entre ces différentes implémentations, sauf à n'y stocker que des données numériques !!!

Pour chaque SGBDR, on parle alors de dialecte. Il y a donc le dialecte SQL d'Oracle, le dialecte SQL de Sybase...etc.

Dans un excellent article, Peter Gulutzan fait le point sur l'essentiel de ce qui est normatif chez les principaux éditeurs de SGBDR : http://www.dbazine.com/gulutzan3.html
Et fustige leur comportement, similaire à celui des constructeurs d'ordinateurs en matière d'OS UNIX incompatible et et qui maintenant pleurent à chaudes larmes devant le rouleau compresseur "Linux" !
Une autre étude, par Michael M. Gorman montre que la volonté de ces éditeurs de SGBDR et leur intérêts mercantiles à courte vue est incompatible avec le respect des standards...
A lire donc : http://www.tdan.com/i016hy01.htm

Nous avons aussi démontré qu'une même table créée sur différents SGBDR avec les mêmes données n'avait par forcément le même comportement au regard du simple SELECT du fait des jeux de caractères et collations par défaut. Or tous les SGBDR ne sont pas paramétrables à ce niveau et ceux qui le sont, ne présentent pas en général les mêmes offres en cette matière.
A lire : Une question de caractères...

Enfin, pour tous ceux qui veulent connaître la norme comparée au dialecte de leur SGBDR favori, il suffit de lire le tableau de comparaison des fonctions de SQL : Toutes les fonctions de SQL

3. Remarques préliminaires sur les SGBDR

Avant tout, nous supposerons connues les notions de "bases de données", "table", "colonne", "ligne", "clef", "index", "intégrité référentielle" et "transaction", même si ces termes, et les mécanismes qu'ils induisent seront plus amplement décrits au fur et à mesure de votre lecture.

Voici quelques points qu'il convient d'avoir à l'esprit lorsque l'on travaille sur des bases de données :

- Il existe une grande **indépendance** entre la couche abstraite que constitue le SGBDR et la couche physique que sont le ou les fichiers constituant une base de données. En l'occurrence il est déraisonnable de croire que les fichiers sont arrangés dans l'ordre des colonnes lors de la création des tables et que les données sont rangées dans l'ordre de leur insertion ou de la valeur de la clef.
- Il n'existe **pas d'ordre spécifique** pour les tables dans une base ou pour les colonnes dans une table, même si le SGBDR en donne l'apparence en renvoyant assez généralement l'ordre établi lors de la création (notamment pour les colonnes). Par conséquent les tables, colonnes, index... doivent être repérés par leur nom et uniquement par cet identifiant.
- Les **données sont systématiquement présentées sous forme de tables**, et cela quel que soit le résultat attendu. Pour autant la table constituée par la réponse n'est pas forcément une table persistante, ce qui signifie que si vous voulez conserver les données d'une requête pour en faire usage ultérieurement, il faudra créer un objet dans la base (table ou vue) afin d'y placer les données extraites.
- La logique sous-jacente aux bases de données repose sur l'algèbre relationnel lui-même basé sur la théorie des ensembles. Il convient donc de penser en terme d'ensemble et de logique et non en terme d'opération de nature atomique. *A ce sujet, il est bon de se rappeler l'usage des patates (ou diagrammes de Wen) lorsque l'on "sèche" sur une requête.*
- Du fait de l'existence d'optimiseurs, la manière d'écrire une requête à peu d'influence en général sur la qualité de son exécution. Dans un premier temps il vaut mieux se consacrer à la résolution du problème que d'essayer de savoir si la clause "bidule" est plus gourmande en ressource lors de son exécution que la clause "truc". Néanmoins l'optimisation de l'écriture des requêtes sera abordée dans un article de la série.

4. Les subdivisions du SQL

Le SQL comporte 5 grandes parties, qui permettent : **la définition des éléments d'une base de données** (tables, colonnes, clefs, index, contraintes...), **la manipulation des données** (insertion, suppression, modification, extraction...), **la gestion des droits d'accès aux données** (acquisition et révocation des droits), **la gestion des transactions** et enfin le **SQL intégré**. La plupart du temps, dans les bases de données "fichier" (dBase, Paradox...) le SQL n'existe qu'au niveau de la manipulation des données, et ce sont d'autres ordres spécifiques qu'il faudra utiliser pour créer des bases, des tables, des index ou gérer des droits d'accès.
Certains auteurs ne considèrent que 3 subdivisions incluant la gestion des transactions au sein de la manipulation des données... Cependant ce serait restreindre les fonctionnalités de certains SGBDR capable de gérer des transactions comprenant aussi des ordres SQL de type définition des données ou encore gestion des droits d'accès !

4.1. DDL : " Data Definition Language "

C'est la partie du SQL qui permet de créer des bases de données, des tables, des index, des contraintes...
Elle possède les commandes de base suivante :

CREATE, ALTER, DROP

Qui permettent respectivement de créer, modifier, supprimer un élément de la base.

4.2. DML : " Data Manipulation Language "

C'est la partie du SQL qui s'occupe de traiter les données.
Elle comporte les commandes de base suivantes :

INSERT, UPDATE, DELETE, SELECT

Qui permettent respectivement d'insérer, de modifier de supprimer et d'extraire des données.

4.3. DCL : " Data Control Language "

C'est la partie du SQL qui s'occupe de gérer les droits d'accès aux tables.
Elle comporte les commandes de base suivantes :

GRANT, REVOKE

Qui permettent respectivement d'attribuer et de révoquer des droits.

4.4. TCL : " Transaction Control Language "

C'est la partie du SQL chargé de contrôler la bonne exécution des transactions. Elle comporte les commandes de base suivantes :

SET TRANSACTION, COMMIT, ROLLBACK

Qui permettent de gérer les propriétés **ACID** des transactions.

4.5. SQL intégré : " Embedded SQL "

Il s'agit d'éléments procéduraux que l'on intégre à un langage hôte :

SET, DECLARE CURSOR, OPEN, FETCH...

Le terme "**ACID**", ne fait pas référence, loin s'en faut au LSD, mais plus basiquement aux termes suivants :

A - Atomicité : une transaction s'effectue ou pas (tout ou rien), il n'y a pas de demi-mesure. Par exemple l'augmentation des prix de 10% de tous les articles d'une table des produits ne saurait être effectuée partiellement, même si le système connaît une panne en cours d'exécution de la requête.

C - Cohérence : le résultat ou les changements induits par une transaction doivent impérativement préserver la cohérence de la base de données. Par exemple lors d'une fusion de société, la concaténation des tables des clients des différentes entités ne peut entraîner la présence de plusieurs client ayant le même identifiant. Il faudra résoudre les conflits portant sur le numéro de client avant d'opérer l'union des deux tables.

I - Isolation : les transactions sont isolées les unes des autres. Par exemple la mise à jour des prix des articles ne sera visible pour d'autres transactions que si ces dernières ont démarré après la validation de la transaction de mise à jour des données. Il n'y aura donc pas de vue partielle des données pendant toute la durée de la transaction de mise à jour.

D - Durabilité : une fois validée, une transaction doit perdurer, c'est à dire que les données sont persistantes même s'il s'ensuit une défaillance dans le système. Par exemple, dès lors qu'une transaction a été validée, comme la mise à jour des prix, les données modifiées doivent être physiquement stockées pour qu'en cas de panne, ces données soient conservées dans l'état où elles ont été spécifiées à la fin de la transaction.

Voici ce que disent, Sébastien BRAU, Christian CHANE-NAM, Louis-Laurent ANTIGNY, Gilberto DE VASCONCELOS sur les propriétés ACID d'un SGBDR :

Atomicité : si tout se passe correctement, les actions de la transaction sont toutes validées, sinon
on retourne à l'état initial.
L'unité de travail est indivisible. Une transaction ne peut être partiellement effectuée.

Cohérence : le passage de l'état initial à l'état final respecte la cohérence de la base.

Isolation : les effets de la transaction ne sont pas perceptibles tant que

celle-ci n’est pas terminée.
Une transaction n’est pas affectée par le traitement des autres transactions.

Durabilité : les effets de la transaction sont durable.

Rendre transparentes la complexité et la localisation des traitements et des données, tout en assurant
un bon niveau de performance, telles sont aujourd’hui les demandes des entreprises et les nouveaux
critères sur lesquels s’arrêtent leurs choix.

Pour assurer l'ensemble de ces fonctions de base, les SGBDR utilisent le principe de la journalisation : un fichier dit "journal", historise toutes les transactions que les utilisateurs effectuent et surtout leur état : en cours, validé ou annulée. Une mise à jour n'est réellement effectuée que si la transaction aboutit. Ainsi en cas de panne du système, une relecture du journal permet de resynchroniser la base de données pour assurer sa cohérence. De même en cas de "RollBack", les instructions de la transaction sont lues "à l'envers" afin de rétablir les données telles qu'elles devaient être à l'origine de la transaction.

5. Implémentation physique des SGBDR

Il existe à ce jour, deux types courant d'implémentation physique des SGBD relationnels. Ceux qui utilisent un service de fichiers associés à un protocole de réseau afin d'accéder aux données et ceux qui utilisent une application centralisée dite serveur de données. Nous les appellerons SGBDR "fichier" et SGBDR client/serveur (ou C/S en abrégé).

5.1. SGBDR "fichier"

Le service est très simple à réaliser : il s'agit de placer dans une unité de stockage partagée (en général un disque d'un serveur de réseau) un ou plusieurs fichiers partageables. Un programme présent sur chaque poste de travail assure l'interface pour traiter les ordres SQL ainsi que le va et vient des fichiers de données sur le réseau.
Il convient de préférer des SGBDR à forte granularité au niveau des fichiers. En effet plus il y a de fichiers pour une même base de données et moins la requête encombrera le réseau, puisque seuls les fichiers nécessaires à la requête seront véhiculés sur le réseau.

Ces SGBDR "fichier" ne proposent en général pas le contrôle des transactions, et peu fréquemment le DDL et le DCL. Ils sont, par conséquent, généralement peu ACID !

Les plus connus sont ceux qui se reposent sur le modèle XBase. Citons parmi les principaux SGBDR "fichier" : dBase, Paradox, Foxpro, BTrieve, MySQL, ... Généralement basés sur le modèle ISAM de fichiers séquentiels indexés.

Avantage : simplicité du fonctionnement, coût peu élevé voire gratuit, format des fichiers ouverts, administration quasi inexistante.

Inconvénient : faible capacité de stockage (quoique certains, comme Paradox, acceptent 2 Go de données par table !!!), encombrement du réseau, rarement de gestion des transactions, faible nombre d'utilisateurs, faible robustesse, cohérence des données moindre.

Access se distingue du lot en étant assez proche d'un serveur SQL : pour une base de données, un seul fichier et un TCL. Mais cela présente plus d'inconvénients que d'avantages : en effet pour interroger une petite table de quelques enregistrements au sein de base de données de 500 Mo, il faut rapporter sur le poste client, la totalité du fichier de la base de données... Un non sens absolu, que Microsoft pali en intimant à ses utilisateurs de passer à SQL Server dès que le nombre d'utilisateurs dépasse 10 !

5.2. SGBDR "Client/Serveur"

Le service consiste à faire tourner sur un serveur physique, un moteur qui assure une relative indépendance entre les données et les demandes de traitement de l'information venant des différentes applications : un poste client envoie à l'aide d'un protocole de réseau, un ordre SQL (une série de trames réseau), qui est exécuté, le moteur renvoie les données. De plus le SGBDR assure des fonctions de gestions d'utilisateurs de manière indépendante aux droits gérés par l'OS.

A ce niveau il convient de préférer des SGBDR C/S qui pratiquent : le verrouillage d'enregistrement plutôt que le verrouillage de page (évitez donc SQL Server...), et ceux qui tournent sur de nombreuses plates-formes système (Oracle, Sybase...). Enfin certains SGBDR sont livrés avec des outils de sauvegarde et restauration.

Les SGBDR "C/S" proposent en général la totalité des services du SQL (contrôle des transactions, DDL et DCL). Ils sont, par conséquent, pratiquement tous ACID. Enfin de plus en plus de SGBD orientés objets voient le jour. Dans ce dernier cas, ils intègrent la plupart du temps le SQL en plus d'un langage spécifique d'interrogation basé sur le concept objet (O^2, ObjectStore, Objectivity, Ontos, Poet, Versant, ORION,GEMSTONE...)
Les serveurs SQL C/S les plus connus sont : Oracle, Sybase, Informix, DB2, SQL Server, Ingres, InterBase, SQL Base...

Avantage : grande capacité de stockage, gestion de la concurrence dans un SI à grand nombre d'utilisateurs, haut niveau de paramétrage, meilleure répartition de la charge du système, indépendance vis à vis de l'OS, gestion des transactions, robustesse, cohérence des données importante. Possibilité de montée en charge très importante en fonction des types de plateformes supportées.

Inconvénient : lourdeur dans le cas de solution "monoposte", complexité du fonctionnement, coût élevé des licences, administration importante, nécessité de machines puissantes.

NOTA : Pour en savoir plus sur le sujet, lire l'étude comparative sur les SGBDR à base de fichier et ceux utilisant un moteur relationnel, intitulée Quand faut-il investir sur le client/serveur ? On y discute aussi des différents modes de verrouillage ...

6. Type de données

Dernier point que nous allons aborder dans ce premier article, les différents types de données spécifiés par SQL et leur disponibilité sur les 5 systèmes que nous avons retenus pour notre étude.

Selon la norme ISO de SQL 92

6.1. Types alphanumériques


CHARACTER (ou **CHAR**) : valeurs alpha de longueur fixe
CHARACTER VARYING (ou **VARCHAR** ou **CHAR VARYING**) : valeur alpha de longueur maximale fixée
Ces types de données sont codés sur 2 octets (EBCDIC ou ASCII) et on doit spécifier la longueur de la chaîne.

Exemple :

```
NOM_CLIENT CHAR(32)
OBSERVATIONS VARCHAR(32000)
```

NATIONAL CHARACTER (ou **NCHAR** ou **NATIONAL CHAR**) : valeurs alpha de longueur fixe
NATIONAL CHARACTER VARYING (ou **NCHAR VARYING** ou **NATIONAL CHAR VARYING**) : valeur alpha de longueur maximale fixée sur le jeu de caractère du pays
Ces types de données sont codés sur 4 octets (UNICODE) et on doit spécifier la longueur de la chaîne.

Exemple :

```
NOM_CLIENT NCHAR(32)
OBSERVATIONS NCHAR VARYING(32000)
```

Nota : la valeur maximale de la longueur est fonction du SGBDR.

6.2. Types numériques

NUMERIC (ou **DECIMAL** ou **DEC**) : nombre décimal à représentation exacte à échelle et précision facultatives
INTEGER (ou **INT**): entier long
SMALLINT : entier court
FLOAT : réel à virgule flottante dont la représentation est binaire à échelle et précision obligatoire
REAL : réel à virgule flottante dont la représentation est binaire, de faible précision
DOUBLE PRECISION : réel à virgule flottante dont la représentation est binaire, de grande précision
BIT : chaîne de bit de longueur fixe
BIT VARYING : chaîne de bit de longueur maximale

Pour les types réels NUMERIC, DECIMAL, DEC et FLOAT, on doit spécifier le nombre de chiffres significatifs et la précision des décimales après la virgule.

Exemple :

```
NUMERIC (15,2)
```

signifie que le nombre comportera au plus 15 chiffres significatifs dont deux décimales.

ATTENTION : le choix entre le type DECIMAL (représentation exacte) et le type FLOAT ou REAL (représentation binaire) doit être dicté par des considérations fonctionnelles. En effet, pour des calculs comptables il est indispensable d'utiliser le type DECIMAL exempt, dans les calculs de toute fraction parasite capable d'entraîner des erreurs d'arrondis. En fait le type DECIMAL se comporte comme un entier dans lequel la virgule n'est qu'une représentation positionnelle. En revanche pour du calcul scientifique on préférera utiliser le type FLOAT, plus rapide dans les calculs.

Exemple :

```
SELECT CAST(3.14159 AS FLOAT (16,4)) AS PI_FLT, CAST(3.14159 AS
DECIMAL (16,4)) AS PI_DEC
PI_FLT                                   PI_DEC
-------------------------------------------------- ------------------
3.1415899999999999                       3.1416
```

NOTA : on peut utiliser une notation particulière pour forcer le typage implicite. Il s'agit d'une lettre précédent la chaîne à transtyper. Les lettres autorisées, sont : N (pour Unicode), B pour binary (chaine de 0 et 1) et X pour binary (chaine hexadécimale constituées de caractères allant de 0 à F).

Exemple :

```
SELECT N'toto' AS MY_NCHAR, B'01010111' AS MY_BINARY_BIT, X'F0A1'
AS MY_BINARY_HEX
```

6.3. Types temporels

DATE : date du calendrier grégorien
TIME : temps sur 24 heures
TIMESTAMP : combiné date temps
INTERVAL : intervalle de date / temps

Rappelons que les valeurs stockées doivent avoir pour base le calendrier grégorien [2] qui est en usage depuis 1582, date à laquelle il a remplacé le calendrier julien. En matière de temps, la synchronisation s'effectue par rapport au TU ou temps universel (UTC : Universal Time Coodinated) anciennement GMT (Greenwich Mean Time) l'ensemble ayant été mis en place, lors la conférence de Washington DC en 1884, pour éviter que les chemins de chemins ne se télescopent.

ATTENTION : Le standard ISO adopté pour le SQL repose sur le format AAAA-MM-JJ. Il est ainsi valable jusqu'en l'an 9999... ou AAAA est l'année sur 4 chiffres, MM le mois sur deux chiffres, et JJ le jour. Pour l'heure le format ISO est hh:mm:ss.nnn (n étant le nombre de millisecondes)

Exemple :
1999-03-26 22:54:28.123
est le 26 mars 1999 à 22h 54m, 28s et 123 millisecondes.

Mais peu de moteurs de requêtes l'implémente de manière aussi formelle...

Le type INTERVAL est très particulier. Il est malheureusement rarement présent dans les SGBDR.
Sa définition se fait à l'aide de la syntaxe suivante :
INTERVAL précision_min TO [précision_max]
où précision_min et précision_max peuvent prendre les valeurs :
{YEAR | MONTH | DAY | HOUR | MINUTE | SECOND}
avec la condition supplémentaire suivante : précision_max ne peut être qu'une mesure temporelle plus fine que précision_min.

Exemple :
```
JOURS      INTERVAL DAY
TRIMESTRE  INTERVAL MONTH TO DAY
TACHE      INTERVAL HOUR TO SECOND
DUREE_FILM INTERVAL MINUTE
```
ATTENTION
Veuillez noter que les réceptacles de valeurs temporelles ainsi créés par le type INTERVAL sont des entiers, et que leur valeur est contrainte lorsqu'ils sont définis avec une précision maximum sur tous les éléments les composant sauf le premier. Ainsi, dans une colonne définie par le type INTERVAL MONTH TO DAY, on pourra stocker une valeur de 48 mois et 31 jours, mais pas une valeur de 48 mois et 32 jours. De même dans un INTERVAL HOUR TO SECOND la valeur en heure est illimitée, mais celle en minute et en seconde ne peut dépasser 59.

6.4. Types " BLOBS " (hors du standard SQL 2)

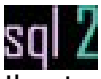
Longueur maximale prédéterminée, donnée de type binaire, texte long voire formaté, structure interprétable directement par le SGBDR ou indirectement par add-on externes (image, son, vidéo...). Attention : ne sont pas normalisés !

On trouve souvent les éléments suivants :
TEXT : suite longue de caractères de longueur indéterminé
IMAGE : stockage d'image dans un format déterminé
OLE : stockage d'objet OLE (Windows)

6.5. Autres types courants, hors norme SQL 92

BOOLEAN (ou **LOGICAL**) : curieusement le type logique (ou encore booléen) est absent de la norme. On peut en comprendre aisément les raisons... La pure logique booléenne ne saurait être respectée à cause de la possibilité offerte par SQL de gérer les valeurs nulles. On aurait donc affaire à une logique dite " 3 états " qui n'aurait plus rien de l'algèbre booléenne. La norme passe donc sous silence, et à bon escient ce problème et laisse à chaque éditeur de SGBDR le soin de concevoir ou non un booléen " à sa manière ".
On peut par exemple implémenter un tel type de données, en utilisant une colonne de type caractère longueur 1, non nul et restreint à deux valeurs (V / F ou encore T / F).
MONEY : est un sous type du type NUMERIC avec une échelle maximale et une précision de deux chiffres après la virgule.
BYTES (ou **BINARY**) : Type binaire (octets) de longueur devant être précisée. Permet par exemple le stockage d'un code barre.
AUTOINC : entier à incrément automatique par trigger.

6.6. Les domaines, ou la création de types spécifiques



Il est possible de créer de nouveau types de données à partir de types pré existants en utilisant la notion de DOMAINE.

Dans ce cas, avant d'utiliser un domaine, il faut le recenser dans la base à l'aide d'un ordre CREATE :

```
CREATE DOMAIN nom_du_domaine AS type_de_donnée
```

Exemple :

```
CREATE DOMAIN DOM_CODE_POSTAL AS CHAR(5)
```

Dès lors il ne suffira plus que d'utiliser ce type à la place de CHAR(5).

L'utilisation des domaines possède de nombreux avantages :

- ils peuvent faire l'objet de contraintes globales
- ils peuvent être modifiés (ce qui modifie le type de toutes les colonnes de table utilisant ce domaine d'un seul coup !)

Exemple :

```
ALTER DOMAINE DOM_CODE_POSTAL
ADD CONSTRAINT MIN_CP CHECK (VALUE >= '01000')
```

Dans ce cas le code postal saisie devra au minimum s'écrire 01000. On pourrait y ajouter une contrainte maximum de forme (VALUE <= '99999').

NOTA : certains SGBDR n'ont pas implémenté l'ordre CREATE DOMAIN. C'est le cas par exemple de SQL Server. Ainsi, il faut aller "trifouiller" les tables systèmes pour insérer un nouveau type de donnée à l'aide de commandes "barbares" propre au SGBDR.

Exemple (SQL Server) :

```
sp_addtype DOM_CODE_POSTAL, 'CHAR(5)', 'null'
```

Un des immenses avantages de passer par des définitions de domaines plutôt que d'utiliser directement des types de donnés est que les domaines induisent une bonne normalisation de la conception du schéma des données.

Pour ma part j'utilise souvent le jeu de domaine suivant :

```
D_KEY_INTEGER    entier
D_TRIGRAMME      char(3)
D_CODE           char(8)
D_LIBELLE_COURT  varchar(16)
D_LIBELLE        varchar(32)
D_LIBELLE_LONG   varchar(64)
D_TITRE          varchar(128)
D_DATE           date
D_TEMPS          dateTime
D_TEXTE          text
D_ADRESSE        varchar(32) /* spécifique aux adresses */
D_BOOLEEN        smallint(1) /* contraint à 0 ou 1, valeur par défaut 0 */
D_MONNAIE
D_ENTIER_COURT
D_ENTIER_LONG
D_REEL
...
```

7. Contraintes de données

Dans la plupart des SGBDR il est possible de contraindre le formatage des données à l'aide de différents mécanismes.

Parmi les contraintes les plus courantes au sein des données de la table on trouve :

- valeur minimum
- valeur maximum
- valeur par défaut
- valeur obligatoire
- valeur unique
- clef primaire
- index secondaire
- format ou modèle (par exemple 3 caractères majuscules suivi de 2 caractères numériques)
- table de référence (recopie d'une valeur d'une table dans un champ d'une autre table en sélectionnant par la clef) aussi appelé CHECK en SQL
- liste de choix

Enfin entre deux tables liées, il est souvent nécessaire de définir une contrainte de référence qui oblige un enregistrement référencé par sa clef à être présent ou détruit en même temps que l'enregistrement visé est modifié, inséré ou supprimé. Ce mécanisme est appelé **INTÉGRITÉ RÉFÉRENTIELLE**.

Exemple : Soit une base de données contenant deux tables : CLIENT et COMMANDE dotées des structures suivantes ...

CLIENT :

NO_CLIENT	INTEGER
NOM_CLIENT	CHAR(32)

COMMANDE :

REF_COMMANDE	CHAR(16)
DATE_COMMANDE	DATE
MONTANT_COMMANDE	MONEY
NO_CLIENT	INTEGER

Et dans lesquelles on trouve les valeurs suivantes :

CLIENT :

143	DUPONT
212	MARTIN
823	DUBOIS

COMMANDE :

1999-11	11/7/1999	1235.52	212
1999-12	17/7/1999	45234.63	823

1999-13	18/7/1999	5485.23	142
1999-14	21/7/1999	11542.23	212

Si l'on détruit la ligne de la table CLIENT concernant le n°212 (MARTIN) alors les factures 1999-11 et 1999-14 deviennent orphelines. Il faut donc interdire la suppression de ce client tant que des références de ce client persistent dans la table COMMANDE.
De même, le changement de la valeur de la clef NO_CLIENT ferait perdre la valeur de référence du lien entre les deux tables, à moins que la modification ne soit répercutée dans la table fille.

NOTA : on parle alors de tables en relation mère / fille ou encore maître / esclave.

8. Triggers et procédures stockées

En ce qui concerne les SGBDR en architecture client / serveur, il est courant de trouver des mécanismes de triggers (permettant d'exécuter du code en fonction d'un événement survenant dans une table) ainsi que des procédures stockées (du code pouvant être déclenché à tout moment). Dans les deux cas, c'est sur le serveur, et non dans le poste client que la procédure ou le trigger s'effectue.

L'avantage réside dans une plus grande intégrité du maniement des données et souvent d'un traitement plus rapide que si le même code trournait sur le poste client.

Ainsi le calcul d'un tarif dans une table de prestation peut dépendre de conditions parfois complexes ne pouvant être facilement exécutée à l'aide de requêtes SQL. Dans ce cas on aura recours au langage hôte du SGBDR, pour lequel on écrira une procédure permettant de calculer ce tarif à partir de différents paramètres.

Mais la plupart du temps, triggers et procédures stockées s'écrivent dans un langage propre au SGBDR (Transact SQL pour SQL Server et Sybase, PL/SQL pour Oracle, etc...). Pour un aperçu du langage Transact SQL, veuillez lire l'article "Un aperçu du langage Transact SQL".

Exemple : on désire calculer le tarif d'adhésion à une mutuelle santé pour une famille composée d'un homme né le 11/5/1950, d'une compagne née le 21/6/1965, d'un fils né le 16/3/1992, d'une fille né le 11/1/1981 et d'une grand mère à charge (ascendant) née le 21/12/1922, le futur adhérent désirant payer sa cotisation au mois.
Les bases tarifaires établies sont les suivantes :

Table "TARIF_BASE" :

TYPE	SEXE	AGE_MIN	TARIF

ADHERENT	HOMME	16	1500
ADHERENT	HOMME	65	1800
ADHERENT	FEMME	16	1400
ADHERENT	FEMME	65	1700
CONJOINT	HOMME	16	1200
CONJOINT	HOMME	65	1500
CONJOINT	FEMME	16	1100
CONJOINT	HOMME	65	1300
ENFANT	HOMME	0	400
ENFANT	HOMME	8	600
ENFANT	HOMME	14	800
ENFANT	HOMME	18	1000
ENFANT	FEMME	0	300
ENFANT	FEMME	8	500
ENFANT	FEMME	14	700
ENFANT	FEMME	18	850
ASCENDANT	HOMME	35	1200
ASCENDANT	HOMME	65	1400
ASCENDANT	FEMME	35	1100
ASCENDANT	HOMME	65	1300

Table "TARIF_MAJO"

PAIEMENT	MAJORATION
MOIS	12%
TRIMESTRE	8%
SEMESTRE	4%
ANNEE	0%

Table "TARIF_MINO"

Nb_ENFANT_MAX	MINORATION
1	10%
2	25%
3	50%
4	100%

Il est très difficile d'établir une requête permettant de trouver le bon tarif dans un tel cas. En revanche, en passant la table de paramètre suivant à une procédure :

Table "PARAMS" :

TYPE	SEXE	DATE_NAISSANCE
ADHERENT	HOMME	11/5/1950
CONJOINT	FEMME	21/6/1965
ENFANT	HOMME	16/3/1992
ENFANT	FEMME	11/1/1981
ASCENDANT	FEMME	21/12/1922

Il n'est pas très compliquée d'écrire dans un langage donné une procédure permettant de calculer ce tarif.
Une telle, procédure pourrait s'écrire dans un pseudo code proche du Pascal :

```
Procedure CalcTarif(Params Array, modePaiement string) : money

var
  unTarif   : mney
  leTarif   : money
  i,j       : smallint   ; indice de boucle
  n         : samllint   ; nombre d'enfants
; tableau multicellulaire représentant les données des tables
  BaseTarif : Array      ; table TARIF_BASE
  MajoTarif : Array      ; table TARIF_MAJO
  Minotarif : Array      ; table TARIF_MINO
endVar

leTarif := 0

for i from 1 to Params.size()
; recherche du tarif pour les enfant avec comptage du nombre d'enfants
  if Params.TYPE = "ENFANT"
  then
    for j from 1 to BaseTarif.size()
      if (BaseTarif[j].TYPE = Params[i].TYPE) and (BaseTarif[j].SEXE =
Params[i].SEXE)
      then
        if BaseTarif[j].AGE_MIN >= CalcAge(Params[i].DATE_NAISSANCE)
        then
          unTarif := BaseTarif[j].TARIF
```

```
            break
          endif
        endif
      endFor
; cumul des différents tarifs enfant trouvés
      leTarif := leTarif + unTarif
; dénombrement des enfants
      n = n+1
; recherche de la minoration pour le nombre d'enfants
      for j from MinoTarif.size() downTo 1
        if MinoTarif[j].NB_ENFANT_MAX <= n
        then
; minoration du tarif cumulé des enfants
          leTarif := leTarif * (1 - MinoTarif[j].MINORATION / 100)
          break
        endif
      endFor
  endif
endFor
for i from 1 to Params.size()
; recherche du tarif pour les autres types excepté les enfants
  if Params.TYPE = "ENFANT"
  then
    continue
  endif
  for j from 1 to BaseTarif.size()
    if (BaseTarif[j].TYPE = Params[i].TYPE) and (BaseTarif[j].SEXE =
Params[i].SEXE)
    then
      if BaseTarif[j].AGE_MIN >= CalcAge(Params[i].DATE_NAISSANCE)
      then
        unTarif := BaseTarif[j].TARIF
        break
      endif
    endif
  endFor
; cumul des différents autres tarifs trouvés
  leTarif := leTarif + unTarif
endFor
; calcul de la majoration pour le mode de paiement :
for i from 1 to MajoTarif.size()
  if modePaiement = MajoTarif[i].PAIEMENT
  then
    leTarif :=  leTarif * (1 + MajoTarif[j].MAJORATION / 100)
    break
  endif
endFor
return leTarif

endProcedure
```

9. Résumé

Voici les différentes implémentations du SQL sur quelques uns des différents moteurs relationnels que nous avons choisi d'analyser.

SGBDR	Paradox 7	Access 97	Sybase adaptive 11	SQL Server 7	Oracle 8
Nature	Service de fichier	Service de fichier	Serveur de données	Serveur de données	Serveur de données
Nb utilisateur (max / en pratique)	255 / 50	300 / 10			
Taille max de la base	illimitée	1 Go			
Taille max d'une table	2 Go (hors BLOBS)	1 Go			
Normalisation	SQL 92	SQL 89 ?	SQL 92	SQL 92	SQL 89
DDL	Oui (a)	Oui	Oui	Oui	Oui
DML	Oui	Oui	Oui	Oui	Oui
DCL	Non	Oui	Oui	Oui	Oui
TCL	Oui (b,c)	Non	Oui	Oui	Oui
DCL	Non	Oui	Oui	Oui	Oui
TCL	Oui (b,c)	Non	Oui	Oui	Oui
CHAR	limité à 255 car.	limité à 255 car.	limité à 255 car.	limité à 8000 car.	limité à 2000 car.
VARCHAR	Non	Non	Oui	Oui	limité à 4000 car.
NUMERIC	Oui, avec 15 chiffres significatifs	sous types comprenant des entiers et des réels	Oui	Oui	Oui
INTEGER	Oui	Non	Oui	Oui	Oui
SMALLINT	Oui	Non	Oui	Oui	Oui
FLOAT	Oui, en fait NUMERIC	Voir NUMERIC	Oui	Oui	Oui
DATE	Oui	Non	Non	Non	Oui
TIME	Oui	Non	Non	Non	Non

TIMESTAMP	Oui	Oui	Oui	Oui	Non
INTERVAL	Non	Non	Non	Non	Non
BIT	Non	Oui	Oui		
BOOLEAN	Oui (LOGICAL)	Oui	Non	Non	Non
MONEY	Oui	Oui	Oui	Oui	Non
BYTES	Oui	Non	Non	Non	Oui (RAW)
AUTOINC	Oui	Oui	Non (d)	Non (d)	Non (d)
BLOB	Oui (4 types différents : MEMO, MEMO FORMATE en RTF, IMAGE et BINARY) limités à 2 Go	Oui (2 types différents : MEMO, HYPERLIEN limité à 64 ko)	Oui (2 types différents TEXT IMAGE)	Oui (2 types différents IMAGE, TEXT)	Oui (7 types différents : LONG, LONG RAW, LONG VARCHAR, BFILE, BLOB, CLOB, NCLOB)
Autres types	OCTET (1 à 255), OLE	OLE, liste de choix	BIT, BINARY	BIT, BINARY, CURSOR, TINYINT, GUID	ROWID (N° d'enregistrement)
INTEGRITÉ RÉFÉRENTIELLE	Oui, stricte ou cascade (suppression et modif.)	Oui, stricte ou cascade (suppression et modif.)	Oui	Oui, pas en cascade	Oui
TRIGGERS	Non	Non	Oui	Oui, limités	BEFORE INSERT, BEFORE UPDATE, BEFORE DELETE, AFTER INSERT, AFTER UPDATE, AFTER DELETE
PROCÉDURES STOCKÉES	Non	Non	Oui, langage propriétaire Transact SQL	Oui, langage propriétaire Transact SQL	Oui, langage propriétaire PL/SQL

(a) Avec quelques limitations (par exemple les contraintes d'intégrité référentielles ne peuvent être crées par le SQL)
(b) Limité en rollback à 255 enregistrements
(c) Pas dans le SQL, mais en code du langage hôte
(d) Mais possible à l'aide de triggers ou de commandes spécifiques

10. Conclusion

En matière de SGBDR " fichier ", Paradox se révèle plus pauvre au niveau du DDL et du TCL, mais plus riche en matière de DML qu'Access. Quant aux types de données, Paradox se révèle bien plus complet qu'Access qui n'intègre même pas de champ de type " image "... Pensez que dans Access le type entier n'est même pas défini ! Enfin en matière de BLOB la plupart des SGBDR acceptent jusqu'à 2Go de données, sauf Access qui est limité à 64 Ko...
En ce qui concerne la capacité de stockage Access révèle très rapidement de nombreuses limites, comme en nombre d'utilisateurs en réseau.
En matière de contrôle des transactions Paradox est limité à 255 enregistrements en RollBack. Mais la présence de tables auxiliaires permet de dépasser ces limites sans encombre, à condition de prévoir le code à mettre en œuvre.
Différence fondamentale pour Paradox, pas de DCL. Mais cela est largement compensé par un niveau de sécurité a forte granularité qui n'est pas compatible avec le SQL normalisé. Ainsi dans Paradox on peut placer des droits au niveau des tables mais aussi de chaque champ et le moteur crypte les données dès qu'un mot de passe est défini (SQL Base de Centura permet aussi de crypter les données à l'aide des plus récents algorithmes de chiffrage)
Point très négatif pour Access dans sa catégorie : il pratique le verrouillage de pages !
Enfin les vues n'existent pas dans Access mais elles sont présentent dans Paradox sous une forme non SQL appelée " vue de requête reliés " (QBE).

En matière de serveur SQL C/S, le SGBDR Sybase se révèle très proche de SQL Server ce qui n'est pas absurde puisqu'ils sont parents. Oracle possède une bonne diversité de types mais sa conformité à la norme laisse à désirer (pas de JOIN par exemple, pauvreté des fonctions temporelles). En revanche Oracle possède un type de champ bien utile et intégré à toutes les tables, le ROWID qui donne le n° de la ligne dans la table et qui est toujours unique, même si la ligne a été supprimée. On retrouve des mécanismes similaires dans SQL Server sous le nom de GUID ou bien avec l'auto incrémentation via "identity".

---

(1) SQL 3 permet une certaine récursivité à l'aide d'une clause WITH

(2) Grégoire XIII (224eme pape), était un pape assez moderne et surtout féru de science... lui au moins aurait certainement sut que la capote ne se mettait pas à l'index ! -

1. Définitions

Terme	Définition	Exemple, remarques
Attribut	Propriété d'une entité dans un modèle correspondant généralement à une colonne dans une table de la base.	Attribut date de naissance d'une personne
Base de données	Programme assurant la cohérence et l'accès aux informations stockées dans un modèle physique de données particulier (hiérarchique, réseau, relationnel; objet...).	IDS est un SGBD de type réseau qui a été utilisé pour la conquête spatiale américaine (programme Apollo).
Base de données relationnelle	Programme assurant la cohérence et l'accès aux informations stockées dans un modèle physique de données de type relationnel.	Paradox, Access, Oracle, DB2, Ingres, Informix, Sybase, SQL Server...
Table	Ensemble de données relatives à un même concept, ou permettant de lier d'autres tables.	Table des clients, table des commandes
Colonne	Élément vertical dans une table représentant un ensemble de valeurs d'un attribut. le mot champ n'est pas approprié à la terminologie des bases de données car il suppose une notion visuelle qui n'existe pas dans une base de données relationnelle (exemple champ opératoire du chirurgien, champ visuel du pilote...).	Colonne "Nom" de la table des clients, colonne "Mode de paiement" de la table des commandes
Ligne	Élément horizontal dans une table représentant une énumération des valeurs des différents attributs (colonne) pour une même référence (identifiant). le mot enregistrement n'est pas approprié à la terminologie des bases de données car il suppose une notion physique (support magnétique de stockage) qui n'existe pas dans une base de données relationelle.	Ligne de référence N° de commande = 328 de la table commande
Clef (Clé ou Clef primaire)	Identifiant d'une table composé d'un ou plusieurs attributs ou colonne. Une clef permet de retrouver sans ambiguïté la ligne dans une table. Les valeurs d'une clef doivent être unique au sein de la table.	328 est la valeur unique de la clef n° de commande permettant de retrouver l'enregistrement 2884 du fichier commande.DB
Clef étrangère	Colonne, ou groupe de colonne représentant une clef d'une table, insérée dans une autre table afin	Le n° de client dans la table des commandes est une clef étrangère dans cette table, et

	d'assurer la relation entre les deux tables.	une clef primaire dans la table des clients
Entité	Ensemble d'éléments informatif relatif à un même concept dans un modèle. Correspond généralement à une table dans l'univers des bases de données.	Entité "Personne"
Identifiant	Autre nom du concept de clef.	Identifiant N° de commande de la table commande
Intégrité référentielle	Mécanisme assurant la cohérence des dépendances de données entre les différentes tables.	Lorsqu'une intégrité référentielle est posée sur la table des commandes relative au n° de client situé dans la table des clients, il devient impossible de supprimer les données relatives à un client si la référence de ce client est utilisée dans la table des commandes...
Contrainte	Mécanisme de contrôle de validité des données interdisant l'insertion de données violant les contraintes.	Contrainte d'intégrité référentielle, contrainte minimum et maximum, obligation de valeur (interdiction d'insertion de données indéfinies)..
Violation (de contrainte, de clef)	Erreur générée lorsque l'on tente d'insérer une ligne avec une valeur de clef déjà utilisée (violation de clef), ou pour des valeurs de colonne hors des contraintes définies (violation de contrainte).	L'insertion d'une commande avec le n° de commande 328 sera rejetée si la commande d'identifiant 328 existe déjà dans la table des commandes. L'insertion d'une commande avec un total négatif sera rejetée si la colonne total possède une contrainte de minimum à zéro.
Relation	Lien logique entre deux entités, représenté dans l'univers des SGBDR par l'insertion d'une clef étrangère ou par l'utilisation d'une table de jointure.	La relation entre les tables "client" et "commande" se fait par une table de jointure possédant comme clef, l'ensemble des colonnes relatives aux deux clefs des deux tables.
Index	Construction physique d'une structure de données relative à une ou plusieurs colonnes d'une table, permettant d'accélérer les recherches au sein d'une table.	En principe, les SGBDR construisent des index pour les colonnes définissant la clef de la table et chacune des clefs étrangères.
Transaction	Mécanisme permettant de cumuler différentes instructions qui seront vues comme une tâche unique, dont l'enchaînement est structuré, et	Par analogie, le vol d'un avion peut être vu comme une "transaction" : roulage, décollage, montée, vol de

	pour laquelle chaque instruction est indissociable.	croisière, descente, approche, atterrissage. Impossible de modifier l'ordre de ces différentes phases, impossible de supprimer l'une d'entre elles.
Ordre	Construction littérale dont la syntaxe répond aux spécificités du SQL et qui intime au moteur de la base de données d'effectuer tel ou tel travail.	Dans la norme SQL l'ordre d'insertion de données commence par le mot clef INSERT
Clause	Partie d'un ordre SQL précisant un fonctionnement particulier	La clause ORDER BY est la dernière clause de tout ordre SQL et ne doit figurer qu'une seule fois dans le SELECT, même s'il existe des requêtes imbriquées ou des requêtes ensemblistes.
Prédicat	Construction logique dont la valeur peut être évaluée à vrai ou faux.	La phrase suivante "la couleur de la voiture du président de la république est jaune" peut être évaluée logiquement à vrai ou faux.

Introduction

A quel moment faut-il passer d'un SGBD micro de type serveur de fichier comme dBase, Paradox, FoxPro ou Access, à un poids lourd du client/serveur comme Oracle, Sybase, SQL Server, Informix ou DB2 ?

Nous supposons que le lecteur du présent article maîtrise un tantinet le monde des bases de données. Nous supposons qu'il sait ce qu'est une requête (même s'il n'en a jamais écrit), ce qu'est un accès concurrentiel aux données (deux utilisateurs voulant modifier la même donnée par exemple), et qu'il est familiarisé avec la notion de table, colonne (champs) et enregistrement. De plus, nous exigeons qu'il sache le minimum au sujet des systèmes d'exploitation, à savoir ce qu'est un fichier, un octet, un poste client, un serveur et un réseau…

Posons alors la question suivante :
Quelle différence y a t-il entre un SGBD à base de fichiers et un SGBD en C/S ?

En fait, il existe deux différences fondamentales entre les deux systèmes. L'ensemble des techniques du C/S est dédié à minimiser le trafic des données sur un réseau et assurer une plus grande intégrité lors du traitement des données…

Pour y parvenir, le C/S utilise plusieurs techniques particulières, l'une concerne le mode de verrouillage lors des concurrences d'accès, d'autres les modes de traitement des données. Nous allons analyser l'ensemble de ces techniques et prendre quelques exemples concrets afin de rendre tangible nos propos.

Notons d'ores et déjà une différence fondamentale : alors que dans un système à base de fichiers, c'est chaque poste de travail qui traite localement les données, dans un SGBD C/S, tous les traitements sont, en principe, effectués sur le serveur par le biais de requêtes.

2. Le mode de verrouillage

Le mode de verrouillage est une fonction essentielle de la base de données. A ce jour, il n'existe que deux modes distincts : le verrouillage « optimiste » et le verrouillage « pessimiste ».

Dans la vie courante il est possible que plusieurs personnes lisent, regardent, écoutent, la même chose en même temps (journaux, radio, télévision…) mais il est impossible que plusieurs personnes créent la même information simultanément. Par exemple il est impossible à deux écrivains de tenir le même stylo afin d'écrire le même roman, comme il est impossible à deux speakers de parler en même temps des même choses sans que cela ne devienne cacophonique.
Bref, l'écriture de l'information est un acte solitaire, mais sa lecture peut être accomplie par de nombreuses personnes simultanément.

Dans l'univers des bases de données le problème est le même.
Il s'agit donc de prévoir un mécanisme capable de gérer un accès simultané en

lecture à tous les utilisateurs, et d'empêcher plusieurs utilisateurs de créer, de modifier ou de supprimer la même donnée.

Pour résoudre ce problème, les chercheurs en algorithmique ont inventé les modes de verrouillage pessimiste et optimiste que nous allons maintenant détailler…

2.1. Verrouillage pessimiste

Lorsque qu'un utilisateur désire modifier une donnée, le SGBD tente de poser un verrou en écriture. Soit l'opération est effective (et donc le verrou est posé), soit l'opération échoue parce qu'un autre utilisateur a déjà posé un verrou. Si l'opération est positive, pendant toute la durée de la modification, le verrou est actif pour cet utilisateur jusqu'à ce que l'utilisateur valide ses modifications ou les annule. Dans ces deux cas le verrou est libéré.

2.2. Verrouillage optimiste

Lorsque qu'un utilisateur désire modifier une donnée, le SGBD relève le numéro de version de l'enregistrement et ne pose aucun verrou. Plusieurs utilisateurs peuvent faire de même sans aucune limite. Le premier utilisateur qui valide ses modifications, incrémente le numéro de version de l'enregistrement. Tous les autres se voient refuser leur modification, car le numéro de version qu'ils ont relevé au début de la session de modification des données n'est pas le même que celui qu'ils peuvent lire à la fin de cette session.

2.3. Discussion sur ces différents modes de verrouillage

Lorsque l'on utilise la technique du verrouillage pessimiste, il est bon de ne pas « jeter » l'utilisateur dès qu'il tente de poser un verrou pour modifier les données. Dans ce cas, le SGBD effectue une boucle d'attente en tentant de poser le verrou pendant n secondes. S'il échoue, il en informe l'utilisateur et lui recommande de renouveler sa tentative un peu plus tard…
C'est comme cela que fonctionne la quasi-totalité des SGBD à base de fichiers (dBase, Paradox, FoxPro, Access…).

Il semble a priori évident que le mode de verrouillage pessimiste devrait être préféré dans tous les cas… C'est cependant oublier le principe de base du C/S : minimiser la charge du réseau…

Or effectuer une boucle d'attente qui interroge un fichier, monopolise dramatiquement le réseau. C'est pourquoi la quasi-totalité des SGBD C/S fonctionne en verrouillage optimiste, car dans ce mode, l'échange de données entre la base et le poste client est rapide et bref…

De plus si nous prenons en compte l'intégrité référentielle et que la structure de la base permet de modifier des clefs existantes en répercutant leurs valeurs dans

l’ensemble des tables filles, alors dans le cas d’un SGBD fichier ont peut arriver à obtenir un trafic réseau intense tandis que dans le cas du client serveur, l’opération sera menée à bien au sein du SGBD donc sans aucune influence sur la charge du réseau.

3. Le SQL en C/S

Il n’y a aucune différence entre un SQL disponible pour un SGBD de type fichier et un SGBD de type C/S : les ordres SQL sont aussi pauvres ou aussi riches en fonction de la qualité de l’éditeur du SGBD… En revanche il existe deux différences fondamentales en matière d’exécution des ordres SQL et de contrôle des transactions…

3.1. Exécution du SQL

En revanche, il existe une différence fondamentale en matière d’exécution de ces requêtes. En fait dans le cadre d’un SGBD de type fichier, celui-ci exécute toujours la requête de manière locale tandis que dans un SGBD C/S ce dernier exécute la requête sur le serveur.
La différence est fondamentale en matière de congestion du trafic réseau… et il ne faut pas oublier qu’à la base, que se soit en fichier ou en C/S, tous les accès aux données se font par le biais de requêtes…

Pour comprendre la différence, voici un exemple simple. Une table de nom CLIENT contient 50 000 enregistrements de 200 octets chacun. La requête est la suivante :

SELECT * FROM CLIENT WHERE (CLI_NAME like ‘%CLINTON%’)
Dans un SGBD de type fichier, voici le flot de données :

- Le PC rapatrie depuis le serveur le fichier contenant la table CLIENT (Soit au minimum 50 000 x 200 octets (sans compter les index et le reste [1])
- Le PC traite localement la requête
- Le PC affiche le résultat (soit par exemple 5 lignes de 200 octets)

Dans un SGBD de type C/S, voici le flot de données :

- Le PC envoie au serveur le fichier texte de la requête (soit environ 50 octets)
- Le serveur exécute la requête sur la table CLIENT
- Le serveur renvoie au PC les données répondant à la requête (soit nos 5 lignes de 200 octets)
- Le PC affiche le résultat

Bilan de ces opérations :
SGBD fichier : environ 10 001 000 octets ont été véhiculé sur le réseau
SGBD C/S : environ 1 050 octets ont été véhiculés sur le réseau

Dans ce cas, le rapport est de près de 1/10000 en faveur du C/S…

3.2. Le transactionnel

De plus les SGBD C/S disposent d'un mécanisme qui n'est généralement pas disponible dans les SGBD fichier, le transactionnel.

Le transactionnel permet de regrouper de multiples commandes SQL comme s'il s'agissait d'une instruction de type atomique, c'est à dire s'exécutant en totalité ou aucunement (comme la division de deux nombres sur le microprocesseur de l'ordinateur).
Cela suppose de donner un point de départ à la transaction un point d'arrivée et de contrôler si l'ensemble des commandes SQL ont été réalisées avec succès. Dans le cas ou l'une quelconque des commandes SQL n'a pas été traitée correctement, alors le point final de la transaction permet de revenir en arrière sur l'ensemble des opérations déjà effectuées. La transaction se présente alors comme une seule instruction de type 'tout ou rien'.
Cela n'est possible que sur des serveurs capables d'enregistrer l'ensemble des insertions, modifications et suppressions effectuées sur les enregistrements des tables d'une base (historisation des modifications).

Un exemple va nous montrer comment se présente une session de transaction :
Supposons que nous disposons d'une base de données centralisée sur un serveur SQL en C/S permettant de collecter les données des clients et des commandes. Supposons aussi que les commerciaux sont dotés de PC portables et que, chaque semaine, ils viennent au siège de la société afin de remplir la base centrale des nouveaux clients et des commandes qu'ils ont saisies sur leur portable.
Pour réaliser ce recollement d'information, l'informaticien a réalisé la transaction suivante :

```
BEGIN TRANSACTION
INSERT INTO CENTRAL_CLIENT (NO_CLI, NOM_CLI, ADRESSE_CLI)
  SELECT PC.NO_CLI, PC.NOM_CLI, PC.ADRESSE_CLI
  FROM PORTABLE_CLIENT PC
INSERT INTO CENTRAL_COMMANDE (NO_CLI, NO_COM, MONTANT_COM)
  SELECT PC.NO_CLI, PC.NO_COM, PC.MONTANT_COM
  FROM PORTABLE_COMMANDE PC
IF ERROR
THEN
  ROLLBACK
ELSE
  COMMIT
ENDIF
```

S'il n'avait fait qu'enchaîner les deux requêtes, il aurait été possible que les clients soit insérés sans que les commandes le soit…
Pire, le système aurait pu tenter d'insérer les commandes en n'ayant pas inséré les clients.
De plus, les deux requêtes auraient pu être interverties… Mais là tous les bons SGBD, qu'ils soient C/S ou fichier, savent gérer les intégrités référentielles et auraient empêché la moindre insertion dans la table des commandes sans avoir la référence du client préalablement insérée dans la table des clients…

4. Les triggers, spécialités du C/S

Tout le monde connaît maintenant la programmation événementielle. Son principe est simple : faire correspondre à un événement (l'ouverture d'un fichier, l'appui sur un bouton, l'arrivée d'une valeur dans un champ), le déclenchement d'une séquence de code.

Les triggers sont à la base de données ce qu'est la programmation événementielle à un environnement graphique : ils font correspondre aux événements d'un SGBD du code que tout développeur est capable d'écrire.

Prenons un exemple afin de décrire plus en avant nos propos. Supposons que nous avons une table CLIENT (parent) liée à 3 tables filles (FACT, LIGNFACT, BONLIV) et que nous voudrions l'effacement des données dans les tables filles lorsque l'utilisateur demande la suppression d'un enregistrement dans la table CLIENT.
Bien entendu, il est possible d'écrire un tel traitement dans la plupart des langages de développement possédant des outils d'accès aux bases de données, mais dans ce cas, le code s'exécute sur le poste client, et il ne faut pas oublier de l'appeler dans tous les éléments de code qui effectuent la suppression dans la table mère…

Dans une base de données C/S, avec un trigger, le traitement s'effectue directement sur le serveur et il ne nécessite aucun appel à aucune fonction d'aucune sorte. C'est en effet la base elle-même qui va se rendre compte que l'événement de suppression de la table client nécessite un traitement et va l'exécuter.

Dans un tel exemple, le code pourrait être le suivant :

```
CREATE TRIGGER DELETE_CASCADE_CLIENT (CLI_ID_CLI INTEGER) BOOLEAN
BEFORE DELETE
DELETE FROM FACT WHERE (ID_CLI = CLI_ID_CLI)
DELETE FROM LIGNFACT WHERE (ID_CLI = CLI_ID_CLI)
DELETE FROM BONLIV WHERE (ID_CLI = CLI_ID_CLI)
IF ERROR
THEN
   RETURN FALSE
ENDIF
RETURN TRUE
```

Ce trigger comportant différentes commandes SQL devra être appelé au sein d'une transaction plus globale incluant la suppression de l'enregistrement dans la table mère. Les commandes SQL de ce trigger réclament la suppression dans les tables FACT, LIGNFACT et BONLIV de tous les enregistrements faisant référence à l'identifiant du client transmis en paramètre. Si au moins, une des requêtes échoue, le trigger renvoi False à la procédure l'ayant appelée et l'ensemble des requêtes sera annulé. Sinon, si tout va bien, le trigger renvoi True et l'ensemble des modifications sera rendu effectif par une validation (COMMIT).

Ce trigger est stocké au sein même de la définition de la base de données et s'exécute, sur le serveur, à chaque fois qu'un ordre de suppression arrive dans la table CLIENT. Il n'y a donc aucun trafic sur le réseau pour effectuer ce traitement…

5. Les procédures stockées, régals du C/S

En complément aux triggers, les procédures stockées permettent de stocker un traitement (opérant en général uniquement sur les données de la base) au sein même de la base de données, qui s'exécutera, par appel de la procédure au moment voulu, sur le serveur.

Exemple : une application réclame un traitement de modification des prix de vente des produits en fonction de la variation d'un certain nombre d'indices boursiers. Cette opération ne peut être déclenchée que manuellement par le responsable du service commercial…
Le code pourrait en être le suivant :

```
CREATE PROCEDURE AUGMENTATION_PRIX
VAR
   INDICE REAL
ENDVAR
; calcul du nouvel indice d'augmentation
; moyenne de la différence entre les anciens et les nouveaux indices
SELECT AVG(DERNIER_INDICE – INDICE_PRECEDENT) AS :INDICE
   FROM INDICE_BOURSE
; teste s'il s'agit bien d'une augmentation et non d'une diminution
; autrement dit un indice positif et non négatif
IF INDICE < 0
THEN
   RETURN
ENDIF
; effectue l'augmentation
UPDATE PRODUITS
   SET PRIX_VENTE_HT = (PRIX_VENTE_HT * (1 + (:INDICE)))
IF ERROR
THEN
   RETURN FALSE
ENDIF
RETURN TRUE
```

Dès lors cette procédure peut être appelée n'importe quand, par n'importe quel poste client… Elle s'exécutera sur le serveur et ne nécessitera pratiquement aucun trafic réseau.
On peut aussi imaginer qu'elle soit couplée avec un trigger ou encore exécutée à une heure différée, par exemple la nuit afin de ne pas ralentir les accès aux données plus fréquents en général dans la journée…

6. Le journal, particularité des SGBD C/S

En plus des triggers et des procédures stockées, la plupart des SGBD C/S proposent d'historiser la « vie » des données, dans ce que l'on appelle un journal des transactions.
Celui-ci conserve une trace temporelle de toutes les insertions, suppressions, modifications intervenues dans la base de données depuis son origine.
Ce journal permet de revenir en arrière de manière partielle sur les données.
L'intérêt essentiel réside dans le fait de pouvoir, en cas de crash, revenir à un état des données satisfaisant.

Reprenons l'exemple de transaction vu au paragraphe 2.2.
Pour assurer correctement la transaction constituée des 2 requêtes de mise à jour, le SGBD effectue en séquence les opérations suivantes :

1. écrit dans le journal une étiquette permettant de repérer le début de la transaction (point de synchronisation)
2. requêtes 1 : copie les données des enregistrements avant modification dans le journal
3. requêtes 1 : copie les données des enregistrements après modification dans le journal
4. requêtes 2 : copie les données des enregistrements avant modification dans le journal
5. requêtes 2 : copie les données des enregistrements après modification dans le journal
6. écrit dans le journal une étiquette indiquant que la transaction a été un succès ou un échec
7. répercute les modifications des données dans les tables
8. écrit dans le journal une étiquette indiquant la fin de la transaction

De cette manière le système est capable en cas de panne survenant au cours d'une transaction de revenir dans un état stable et cohérent des données, afin d'assurer l'intégrité et donc l'atomicité de la transaction.
Pour revenir à cet état stable, le système lit le journal en débutant par la fin, et reprend toutes les transactions qui ont été un succès, sans que la mise à jour physique des données ait été effectuée.

7. Conclusion

Peut-on accéder aux données d'une base en s'affranchissant des requêtes ?

Un des points négatifs des systèmes C/S est que tous les traitements portant sur les données de la base doivent être, à un moment ou un autre, effectués par des requêtes. En revanche dans les systèmes à base de fichier il est souvent possible d'accéder directement aux données en s'affranchissant des requêtes, et de procéder à un traitement en utilisant par exemple une lecture séquentielle des enregistrements d'une table.

En général les SGBD de type fichiers proposent des outils (L4G, composants…) permettant de réaliser ce genre de traitement en ayant à aucun moment recours aux requêtes. Dans ce cas il est souhaitable de bien connaître le format interne de la base et ses particularités [2].
Ce type d'accès s'avère intéressant et très souple dans le cas de base de données de faible volume. En revanche il ne permet pas la mise à jour en une seule requête ou transaction de plusieurs tables… Autre avantage : il est indépendant de la qualité du SQL fourni. Dans le cas de « middleware [3] » comme le moteur BDE d'Inprise (ex Borland) il permet d'unifier les traitements quelle que soit la base de données sous-jacente et sa plate-forme. On peut ainsi, théoriquement, changer de SGBD sans avoir à réécrire une seule ligne de code de l'application…
Dans le cas des SGBD C/S il est quand même possible d'effectuer des traitements locaux dans le langage de programmation en utilisant la technique des « cursor » (curseur) : le curseur est un ordre SQL qui permet de tamponner les données d'une table en mémoire et d'accéder aux données par le biais de variables pour y effectuer un traitement.

Est-il possible de contourner le mode de verrouillage optimiste, afin de le rendre pessimiste ?

Bien entendu oui. Mais le faire systématiquement serait une hérésie, notamment dans le cas d'une base de données dont le volume des transactions et le taux de mise à jour des données serait très important : des blocages intempestifs s'y produiraient fréquemment et pourraient même conduire à la paralysie du système tout entier.
Cependant réaliser un verrouillage pessimiste est d'une grande simplicité : il suffit de créer au sein de la base de donnée une table de sémaphore contenant le nom de l'utilisateur, le nom de la table, les références à l'enregistrement a verrouiller (par exemple son 'ROW_ID' si le serveur est capable de le fournir). Avant toute mise à jour ou suppression il suffit de consulter la table de sémaphore, et si l'enregistrement n'y est pas référencé de l'y insérer, de procéder à la modification, puis de le supprimer de la table de sémaphore.
Néanmoins il faut se prémunir de plusieurs petits problèmes sous-jacents : par exemple comment supprimer les verrous devenus obsolètes ? Cela peut arriver, notamment lorsqu'un utilisateur a commencé une modification et que son poste client a été victime d'une panne matérielle le coupant du réseau…

Le SQL est-il un langage normalisé ?

Oui, bien entendu. Il existe une norme de définition du langage SQL (SQL 2 : ISO / ANSI 92). Mais chaque éditeur est libre de se reposer ou non sur la norme. Il en résulte une forte incompatibilité entre les différents éditeurs de SGBD, que l'on peut éventuellement contourner soit en se reposant uniquement sur les éléments communs aux principaux éditeurs (très limité), soit en utilisant un middleware normatif qui dans ce cas permettra d'écrire toutes les requêtes à l'aide du SQL local et assurera sa traduction dans le SQL spécifique au SGBD.
Un des autres avantages de certains middlewares est de proposer d'effectuer des requêtes hétérogènes, c'est à dire entre différentes bases de données, de différents SGBD, pouvant même tourner sous différentes plates-formes système. C'est en particulier le cas du moteur BDE d'Inprise (ex Borland).

Quel langage est utilisé pour écrire des procédures stockées et des triggers ?

Malheureusement il n'existe aucun langage commun entre les différents éditeurs. De plus l'évolution des SGBDR vers les technologies de l'objet et du web font évoluer de manière importante les structures des langages des différents éditeurs d'une version à l'autre.
Par exemple Oracle porte désormais son fameux langage PL/SQL dans le monde de l'objet et le dote de modules spécialisés (bibliothèques de fonctions) permettant par exemple d'accéder aux techniques du web ou de faire du pilotage SIG (Systèmes d'Informations Géographiques).
La norme SQL 3 tente de résoudre un certain nombre de ces problèmes.

Quel type de machine est nécessaire pour faire fonctionner un SGBDR C/S ?

Il est nécessaire de se munir d'un serveur puissant, avec un processeur plutôt spécialisé dans le traitement des données de base (texte et nombre) que le graphique ou le son…
La quantité de mémoire nécessaire est évidemment bien plus importante que dans le cadre d'un SGBD fichier (par exemple sous système d'exploitation NT, un SGBDR en C/S nécessite au minimum 128 Mo de RAM même avec un nombre très restreint d'utilisateurs). Dans les deux cas il faut avoir des disques à accès très rapide (SCSI UW par exemple) et ne pas hésiter à investir dans un contrôleur RAID éventuellement en « hot plug ».

Voici un tableau qui pourra vous aider dans votre choix de passage de l'un à l'autre :

	SGBD Fichiers	**SGBD Client/Serveur**	**C/S + moniteur transactionnel, répartition, synchronisation, clustering…**
Vitesse du réseau	Rapide de préférence (100 Mo/s)	Normal (10 Mo/s)	Normal ou rapide
Volume des données	Faible (10 – 300 Mo)	Important (250 Mo – 2 Go)	Très importants (500 Mo – 4 To)
Nombres d'utilisateurs	Faible (1 à 25)	Moyen (5 à 250)	Important (50 à 5 000)
Contraintes d'intégrité	Faibles	Fortes	Fortes
Volume des transactions, requêtes	Faible	Moyen	Fort
Tolérance aux pannes	Inexistante	Forte	Très forte
Coût d'achat, déploiement	Faible, inexistants	Importants	Très importants
Coûts d'exploitation	Inexistants	Importants	Très importants

SGBD Fichiers	**SGBD Client/Serveur**
dBase	Informix
FoxPro	Oracle

Paradox	Sybase
Access	SQL Server
MySQL	DB2
Approach	InterBase
	4 D
	PostGreSQL

La base de données d'exemple

Date de publication : 23/01/2003 , Date de mise à jour : 11/03/2009

Par SQLPro (autres articles) (CV)

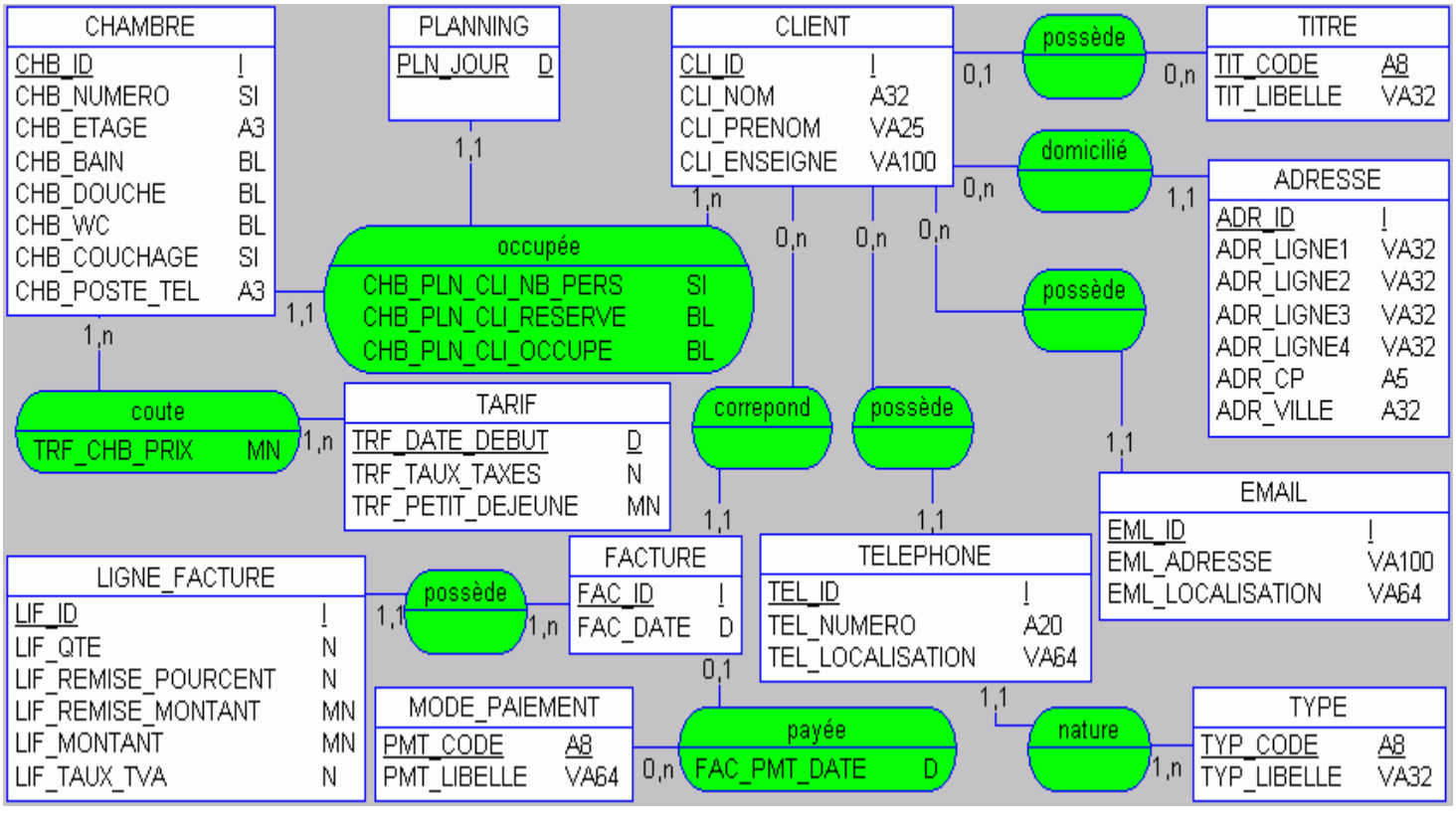
niveau : débutant

Pour les exemples de ces articles, nous avons choisi de modéliser la gestion d'un hôtel pompeusement intitulé GRAND HOTEL !...

Version hors-ligne (Miroir)

1. Le modèle conceptuel de données

Voici un MCD établi à partir de l'outil AMC (AMC*Designor ou encore Power AMC) :



Pour ceux qui ne seraient pas familiarisé avec cet outil et la méthode MERISE, voir l'article consacré à la méthode MERISE sur ce même site.

Quelques explications
La plupart des clefs sont des entiers (I) qui pourront être auto générés par exemple par un type AUTOINCREMENT (Paradox, Access) ou encore via un trigger (identity de SQL Server...). Pour certaines entités, notamment celles servant de références à la saisie (MODE_PAIEMENT, TYPE, CODE) la clef est un code. Enfin pour les entités TARIF et PLANNING, nous avons choisi une date comme clef.
Chaque entité est repérée à l'aide d'un trigramme (code de 3 lettres) qui sert de préfixe pour chaque attribut. Exemple : CHB pour CHAMBRE, LIF pour LIGNE_FACTURE, etc...
Les booléens seront représentés par des valeurs numériques 0 (faux) et 1 (vrai), chaque attribut ayant obligatoirement une valeur par défaut.

Voici les codes des différents types de données :

I	Integer (entier long)
N	Number (réel)
SI	Short Integer (entier court)
BL	Boolean (booléen)
A	Char (caractères alpha de longueur fixe)
VA	VarChar (caractères alpha longueur variable avec un maximum)
D	Date
MN	Money (monnaie)

L'association "**occupée**" permet de connaître la réservation ou l'occupation d'une chambre (une chambre peut avoir été réservée mais pas occupée), c'est pourquoi cette association possède les attributs NB_PERS (nombre de personnes : entier) RESERVE (réservée : booléen) et OCCUPE (occupe : booléen). Une chambre à une date donnée, ne peut être occupée que par un seul client. Mais un client peut occuper plusieurs chambres à la même date ou la même chambre à différentes dates, voire même plusieurs chambres à plusieurs dates...
Entité **CLIENT** : Un client peut avoir plusieurs adresses, plusieurs numéros de téléphone et plusieurs e-mail. Pour le téléphone, comme pour l'e-mail, l'attribut 'localisation' permet de savoir si le téléphone est situé au domicile, à l'entreprise, etc...
L'entité **TITRE** permet de donner un titre à une personne, parmi les valeurs 'M.' (monsieur), 'Mme.' (madame) et 'Melle.' (mademoiselle).
L'entité **TYPE** permet de connaître le type de téléphone, parmi les valeurs 'TEL' (téléphone), 'FAX' (télécopie) et 'GSM' (portable).
L'entité **MODE_PAIEMENT** permet de connaître le genre de paiement, parmi les valeurs 'ESP' (espèces), 'CHQ' (chèque), 'CB' (carte bancaire).
L'association "**payée**" intègre la date du paiement d'une facture.

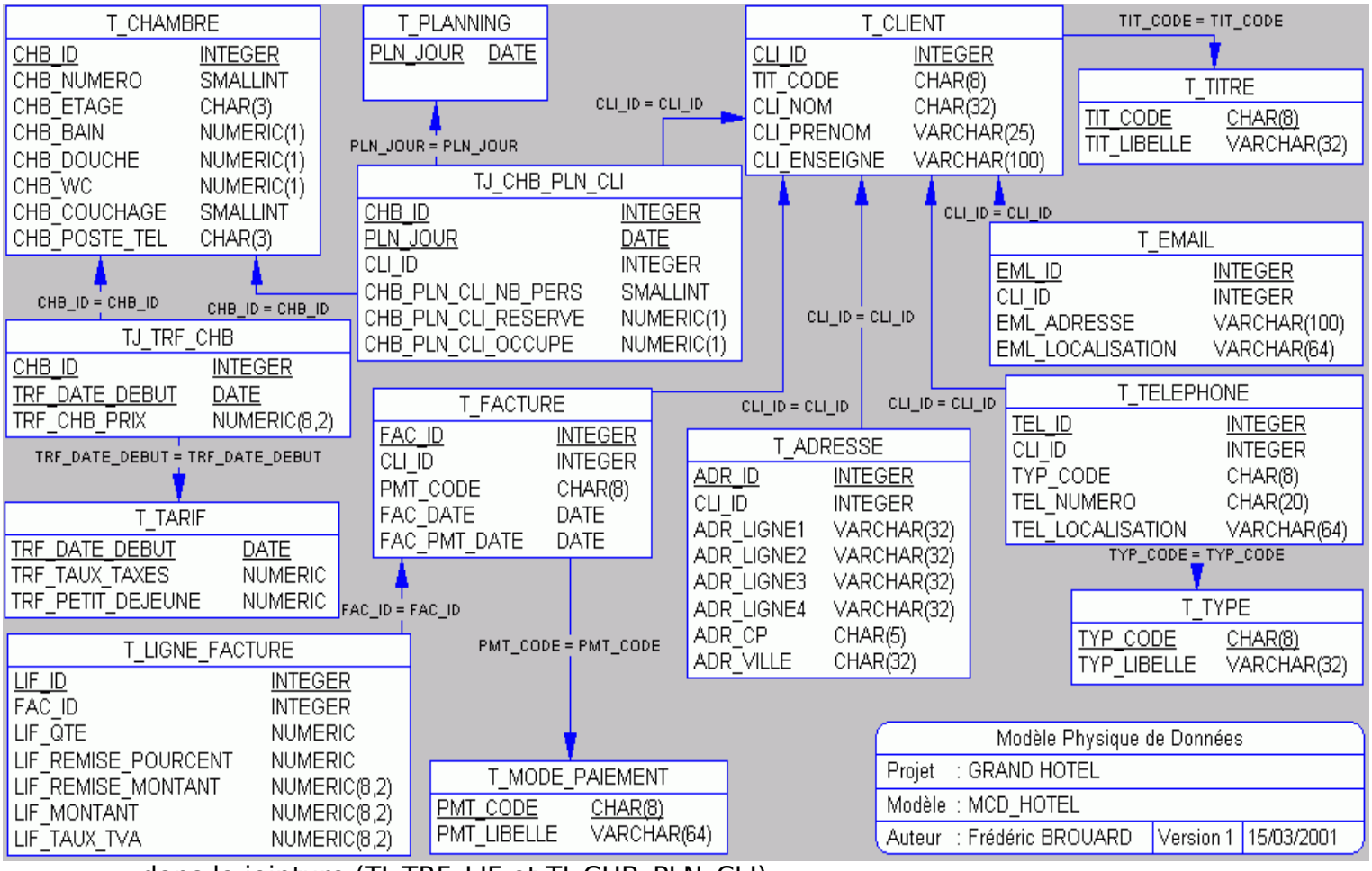
NOTA : ce modèle est incomplet. Si l'on devait faire figurer l'adresse sur la facture il faudrait choisir une adresse du client. La meilleure façon de régler le problème est de faire glisser la clef du client dans la table des adresses et d'ajouter dans la table facture l'ID de l'adresse choisie pour la facture. C'est ce que l'on apelle un "lien identifiant" qui se positionne au niveau du lien entre

l'association "domicilié" et l'entité "adresse". On rajoute alors une association entre la facture et l'adresse de cardinalité 0,1.

2. Le modèle physique

Nous avons demandé à générer un modèle basé sur le SQL ANSI de manière à pouvoir être compatible avec la plupart des SGBDR :

Vous constaterez que toutes les tables ont été préfixées avec la lettre T lorsquelles proviennent d'entités, et de TJ lorsqu'elles proviennent d'associations. Dans ce dernier cas, leur nom a été constitué des trigrammes des tables en jeu



dans la jointure (TJ_TRF_LIF et TJ_CHB_PLN_CLI).

3. Script de création de la base de données

La génération de la base de données au format SQL standard donne le code suivant :

```
--
==============================================================
------------
--   Nom de la base   :  MCD_HOTEL
--   Nom de SGBD      :  ANSI Niveau 2
--   Date de création :  16/01/2001  22:24
--   Copyright        :  Frédéric BROUARD
--
==============================================================
------------

--
==============================================================
------------
--   Table : T_CHAMBRE
--
==============================================================
------------
create table T_CHAMBRE
(
   CHB_ID             INTEGER             not null,
   CHB_NUMERO         SMALLINT            not null,
   CHB_ETAGE          CHAR(3)                  ,
   CHB_BAIN           NUMERIC(1)          not null       default 0,
   CHB_DOUCHE         NUMERIC(1)          not null       default 1,
   CHB_WC             NUMERIC(1)          not null       default 1,
   CHB_COUCHAGE       SMALLINT            not null,
   CHB_POSTE_TEL      CHAR(3)                  ,
   primary key (CHB_ID)
);

--
==============================================================
------------
--   Index : T_CHAMBRE_PK
--
==============================================================
------------
create unique index T_CHAMBRE_PK on T_CHAMBRE (CHB_ID asc);

--
==============================================================
------------
--   Table : T_TARIF
--
==============================================================
------------
create table T_TARIF
(
   TRF_DATE_DEBUT      DATE               not null,
```

```
   TRF_TAUX_TAXES       NUMERIC              not null,
   TRF_PETIT_DEJEUNE    NUMERIC(8,2)         not null,
   primary key (TRF_DATE_DEBUT)
);

--
==============================================================
------------
--   Index : T_TARIF_PK
--
==============================================================
------------
create unique index T_TARIF_PK on T_TARIF (TRF_DATE_DEBUT asc);

--
==============================================================
------------
--   Table : T_PLANNING
--
==============================================================
------------
create table T_PLANNING
(
   PLN_JOUR             DATE                 not null,
   primary key (PLN_JOUR)
);

--
==============================================================
------------
--   Index : T_PLANNING_PK
--
==============================================================
------------
create unique index T_PLANNING_PK on T_PLANNING (PLN_JOUR asc);

--
==============================================================
------------
--   Table : T_TITRE
--
==============================================================
------------
create table T_TITRE
(
   TIT_CODE             CHAR(8)              not null,
   TIT_LIBELLE          VARCHAR(32)          not null,
   primary key (TIT_CODE)
);

--
==============================================================
------------
--   Index : T_TITRE_PK
--
==============================================================
------------
create unique index T_TITRE_PK on T_TITRE (TIT_CODE asc);

--
```

```
==============================================
============
--   Table : T_TYPE
--
==============================================
============
create table T_TYPE
(
   TYP_CODE           CHAR(8)             not null,
   TYP_LIBELLE        VARCHAR(32)         not null,
   primary key (TYP_CODE)
);

--
==============================================
============
--   Index : T_TYPE_PK
--
==============================================
============
create unique index T_TYPE_PK on T_TYPE (TYP_CODE asc);

--
==============================================
============
--   Table : T_MODE_PAIEMENT
--
==============================================
============
create table T_MODE_PAIEMENT
(
   PMT_CODE           CHAR(8)             not null,
   PMT_LIBELLE        VARCHAR(64)         not null,
   primary key (PMT_CODE)
);

--
==============================================
============
--   Index : T_MODE_PAIEMENT_PK
--
==============================================
============
create unique index T_MODE_PAIEMENT_PK on T_MODE_PAIEMENT (PMT_CODE
asc);

--
==============================================
============
--   Table : T_CLIENT
--
==============================================
============
create table T_CLIENT
(
   CLI_ID             INTEGER             not null,
   TIT_CODE           CHAR(8)                    ,
   CLI_NOM            CHAR(32)            not null,
   CLI_PRENOM         VARCHAR(25)                ,
   CLI_ENSEIGNE       VARCHAR(100)               ,
```

```
   primary key (CLI_ID)
);

--
==============================================================
------------
--   Index : T_CLIENT_PK
--
==============================================================
------------
create unique index T_CLIENT_PK on T_CLIENT (CLI_ID asc);

--
==============================================================
------------
--   Index : L_CLI_TIT_FK
--
==============================================================
------------
create index L_CLI_TIT_FK on T_CLIENT (TIT_CODE asc);

--
==============================================================
------------
--   Table : T_FACTURE
--
==============================================================
------------
create table T_FACTURE
(
   FAC_ID              INTEGER              not null,
   CLI_ID              INTEGER              not null,
   PMT_CODE            CHAR(8)                      ,
   FAC_DATE            DATE                 not null,
   FAC_PMT_DATE        DATE                         ,
   primary key (FAC_ID)
);

--
==============================================================
------------
--   Index : T_FACTURE_PK
--
==============================================================
------------
create unique index T_FACTURE_PK on T_FACTURE (FAC_ID asc);

--
==============================================================
------------
--   Index : L_FAC_CLI_FK
--
==============================================================
------------
create index L_FAC_CLI_FK on T_FACTURE (CLI_ID asc);

--
==============================================================
------------
--   Index : TJ_FAC_PMT_FK
```

```
--
==============================================================
create index TJ_FAC_PMT_FK on T_FACTURE (PMT_CODE asc);

--
==============================================================
--   Table : T_ADRESSE
--
==============================================================
create table T_ADRESSE
(
   ADR_ID               INTEGER              not null,
   CLI_ID               INTEGER              not null,
   ADR_LIGNE1           VARCHAR(32)          not null,
   ADR_LIGNE2           VARCHAR(32)                  ,
   ADR_LIGNE3           VARCHAR(32)                  ,
   ADR_LIGNE4           VARCHAR(32)                  ,
   ADR_CP               CHAR(5)              not null,
   ADR_VILLE            CHAR(32)             not null,
   primary key (ADR_ID)
);

--
==============================================================
--   Index : T_ADRESSE_PK
--
==============================================================
create unique index T_ADRESSE_PK on T_ADRESSE (ADR_ID asc);

--
==============================================================
--   Index : L_ADR_CLI_FK
--
==============================================================
create index L_ADR_CLI_FK on T_ADRESSE (CLI_ID asc);

--
==============================================================
--   Table : T_TELEPHONE
--
==============================================================
create table T_TELEPHONE
(
   TEL_ID               INTEGER              not null,
   CLI_ID               INTEGER              not null,
   TYP_CODE             CHAR(8)              not null,
   TEL_NUMERO           CHAR(20)             not null,
   TEL_LOCALISATION     VARCHAR(64)                  ,
   primary key (TEL_ID)
);
```

```
--
==========================================================
============
--   Index : T_TELEPHONE_PK
--
==========================================================
============
create unique index T_TELEPHONE_PK on T_TELEPHONE (TEL_ID asc);

--
==========================================================
============
--   Index : L_TEL_CLI_FK
--
==========================================================
============
create index L_TEL_CLI_FK on T_TELEPHONE (CLI_ID asc);

--
==========================================================
============
--   Index : L_TEL_TYP_FK
--
==========================================================
============
create index L_TEL_TYP_FK on T_TELEPHONE (TYP_CODE asc);

--
==========================================================
============
--   Table : T_EMAIL
--
==========================================================
============
create table T_EMAIL
(
   EML_ID              INTEGER              not null,
   CLI_ID              INTEGER              not null,
   EML_ADRESSE         VARCHAR(100)         not null,
   EML_LOCALISATION    VARCHAR(64)                  ,
   primary key (EML_ID)
);

--
==========================================================
============
--   Index : T_EMAIL_PK
--
==========================================================
============
create unique index T_EMAIL_PK on T_EMAIL (EML_ID asc);

--
==========================================================
============
--   Index : L_EML_CLI_FK
--
==========================================================
============
create index L_EML_CLI_FK on T_EMAIL (CLI_ID asc);
```

```
--
==============================================================
------------
--   Table : T_LIGNE_FACTURE
--
==============================================================
------------
create table T_LIGNE_FACTURE
(
   LIF_ID              INTEGER              not null,
   FAC_ID              INTEGER              not null,
   LIF_QTE             NUMERIC              not null,
   LIF_REMISE_POURCENT  NUMERIC                      ,
   LIF_REMISE_MONTANT   NUMERIC(8,2)                 ,
   LIF_MONTANT         NUMERIC(8,2)         not null,
   LIF_TAUX_TVA        NUMERIC(8,2)         not null,
   primary key (LIF_ID)
);

--
==============================================================
------------
--   Index : T_LIGNE_FACTURE_PK
--
==============================================================
------------
create unique index T_LIGNE_FACTURE_PK on T_LIGNE_FACTURE (LIF_ID asc);

--
==============================================================
------------
--   Index : L_LIF_FAC_FK
--
==============================================================
------------
create index L_LIF_FAC_FK on T_LIGNE_FACTURE (FAC_ID asc);

--
==============================================================
------------
--   Table : TJ_TRF_CHB
--
==============================================================
------------
create table TJ_TRF_CHB
(
   CHB_ID              INTEGER              not null,
   TRF_DATE_DEBUT      DATE                 not null,
   TRF_CHB_PRIX        NUMERIC(8,2)         not null,
   primary key (CHB_ID, TRF_DATE_DEBUT)
);

--
==============================================================
------------
--   Index : TJ_TRF_CHB_PK
--
==============================================================
------------
```

```
create unique index TJ_TRF_CHB_PK on TJ_TRF_CHB (CHB_ID asc,
TRF_DATE_DEBUT asc);

--
==============================================================
------------
--   Index : L_CHB_TRF_FK
--
==============================================================
------------
create index L_CHB_TRF_FK on TJ_TRF_CHB (CHB_ID asc);

--
==============================================================
------------
--   Index : L_TRF_CHB_FK
--
==============================================================
------------
create index L_TRF_CHB_FK on TJ_TRF_CHB (TRF_DATE_DEBUT asc);

--
==============================================================
------------
--   Table : TJ_CHB_PLN_CLI
--
==============================================================
------------
create table TJ_CHB_PLN_CLI
(
   CHB_ID              INTEGER              not null,
   PLN_JOUR            DATE                 not null,
   CLI_ID              INTEGER              not null,
   CHB_PLN_CLI_NB_PERS  SMALLINT             not null,
   CHB_PLN_CLI_RESERVE  NUMERIC(1)           not null       default 0,
   CHB_PLN_CLI_OCCUPE   NUMERIC(1)           not null       default 1,
   primary key (CHB_ID, PLN_JOUR)
);

--
==============================================================
------------
--   Index : TJ_CHB_PLN_CLI_PK
--
==============================================================
------------
create unique index TJ_CHB_PLN_CLI_PK on TJ_CHB_PLN_CLI (CHB_ID asc,
PLN_JOUR asc, CLI_ID asc);

--
==============================================================
------------
--   Index : L_CHB_PLN_CLI_FK
--
==============================================================
------------
create index L_CHB_PLN_CLI_FK on TJ_CHB_PLN_CLI (CHB_ID asc);

--
==============================================================
```

```
============
--   Index : L_PLN_CHB_CLI_FK
--
================================================
============
create index L_PLN_CHB_CLI_FK on TJ_CHB_PLN_CLI (PLN_JOUR asc);

--
================================================
============
--   Index : L_CLI_CHB_PLN_FK
--
================================================
============
create index L_CLI_CHB_PLN_FK on TJ_CHB_PLN_CLI (CLI_ID asc);
```

Vous noterez que nous avons volontairement omis les intégrités référentielles de manière à alléger le code mais aussi pour le rendre le plus compatible possible.

Téléchargez le script SQL de création de la base de données (ANSI niveau 2)

Si vous voulez la version complète du code de génération de cette base de données, voici un tableau des différentes versions que j'ai fait générer par AMC :

ACCESS 95/97
DB2 C/S 2
DBASE 5.0 Windows
FOXPRO 5 Windows
INFORMIX SQL 7.1
INGRES 6.4
INTERBASE 4.0
ORACLE 7
PARADOX 7 Windows
SQL SERVER 6
SYBASE SQL SERVER 10

NOTA : nous n'avons pas introduit de colonne de type auto incrémenté dans les scripts de création de base de données, mais vous pouvez les modifier en y introduisant un trigger. Ne le faites pas si vous voulez pouvoir exploiter le jeu de données nécessaire aux exercices qui se trouvent dans les chapitres qui suivent.
Exemples : pour le SGBDR InterBase de Borland / Inprise, vous pouvez utiliser un générateur de nombre séquentiel utilisable par tous. Il faut donc créer autant de générateur qu'il existe dans la base de colonnes nécessitant une auto incrémentation, puis dans chacun des triggers de type BEFORE INSERT, appeler ce générateur.
Pour SQL Server de Microsoft, vous pouvez utiliser le type 'identity', mais vous devrez certainement modifier le type des colonnes des clefs étrangères dans le script de création de la base.
Pour Paradox il suffit de remplacer le type "I" par le type "+" dans les colonnes où cela s'avère nécessaire.

Exemple de trigger pour InterBase :

A la création de la base :

```
CREATE GENERATOR CLI_ID_GEN TO 2301;
```

Qui indique de réserver un espace pour stocker la valeur de l'auto incrément de nom CLI_ID_GEN et commençant par la valeur 2301.
Dans le trigger BEFORE INSERT de la table CLIENT, on utilise ce générateur pour alimenter le champs NUM_CLI :

```
CREATE TRIGGER AUTOINC_CLI FOR T_CLIENT
BEFORE INSERT AS
BEGIN
  NEW.CLI_ID = GEN_ID(CLI_ID_GEN,1);
END
```

NEW.CLI_ID est la valeur de la colonne après passage dans le trigger et AUTOINC_CLI une fonction appelant le générateur.

4. Création des données

Comme il nous faut des données pour travailler, vous trouverez ci dessous un script SQL dans lequel chaque ligne constitue une instruction d'insertion de données.
Nous avons utilisé les caractères /* et */ pour y insérer des commentaires. Si votre interpréteur SQL ne comprend pas ces instructions, vous pouvez réaliser un petit programme qui lit ce fichier ligne par ligne et lance les requêtes d'insertion sauf dans les deux cas suivants :

- la ligne est vide
- la ligne débute par /*

Téléchargez le script SQL de création du jeu d'essai :

Formats d'insertion	Exemple (23 février 2001)
Date ISO (AAAA-MM-JJ)	insert into table (colonne_date) values ('2001-02-23')
Date US ('MM/JJ/AAAA')	insert into table (colonne_date) values ('02/23/2001')
Date FR ('JJ/MM/AAAA')	insert into table (colonne_date) values ('23/02/2001')

Pour le SGBDR Paradox, vous pouvez :

1. mettre le fichier des ordres SQL d'insertion dans une colonne de table via un import de données. Par exemple dans une table Paradox de nom INSERT_EXEMPLE possédant une colonne de nom SQL_ORDER.
2. jouer le script ci dessus afin d'insérer les données :

```
var
  tc     TCursor
  svar   String
  sqlVar SQL
```

```
  db     Database
endvar

errorTrapOnWarnings(True)
db.open(...) => chemin de la base de données cible
tc.open("INSERT_EXEMPLE.db")

scan tc :
  svar = TC.SQL_ORDER
  try
    sqlVar.readFromString(svar)
    sqlVar.executeSQL(db)
  onFail
    errorShow()
    msgInfo("ORDRE SQL",sVar)
    quitLoop
  endTry
endscan

errorTrapOnWarnings(False)
endMethod
```

Les mots clefs du SQL

Date de publication : 04/09/2002 , Date de mise à jour : 12/06/2008

Par SQLPro (autres articles)

Voici quelques définitions en usage dans l'univers des SGBDR et du SQL.

Version PDF (Miroir) Version hors-ligne (Miroir)

Mots réservés du SQL

Mot réservé	SQL 92	SQL 99	SQL:2003	SQL:2008
ABS			OUI	OUI
ABSOLUTE	OUI	OUI		
ACTION	OUI	OUI		
ADD	OUI	OUI		
AFTER		OUI		
ALL	OUI	OUI	OUI	OUI
ALLOCATE	OUI	OUI	OUI	OUI
ALTER	OUI	OUI	OUI	OUI
AND	OUI	OUI	OUI	OUI
ANY	OUI	OUI	OUI	OUI
ARE	OUI	OUI	OUI	OUI
ARRAY		OUI	OUI	OUI
AS	OUI	OUI	OUI	OUI
ASC	OUI	OUI		
ASENSITIVE			OUI	OUI
ASSERTION	OUI	OUI		
ASYMMETRIC			OUI	OUI
AT	OUI	OUI	OUI	OUI

ATOMIC			OUI	OUI
AUTHORIZATION	OUI	OUI	OUI	OUI
AVG	OUI		OUI	OUI
BEFORE		OUI		
BEGIN	OUI	OUI	OUI	OUI
BETWEEN	OUI	OUI	OUI	OUI
BIGINT			OUI	OUI
BINARY		OUI	OUI	OUI
BIT	OUI	OUI		
BIT_LENGTH	OUI			
BLOB		OUI	OUI	OUI
BOOLEAN		OUI	OUI	OUI
BOTH	OUI	OUI	OUI	OUI
BREADTH		OUI		
BY	OUI	OUI	OUI	OUI
CALL		OUI	OUI	OUI
CALLED			OUI	OUI
CARDINALITY			OUI	OUI
CASCADE	OUI	OUI		
CASCADED	OUI	OUI	OUI	OUI
CASE	OUI	OUI	OUI	OUI
CAST	OUI	OUI	OUI	OUI
CATALOG	OUI	OUI		
CEIL			OUI	OUI
CEILING			OUI	OUI
CHAR	OUI	OUI	OUI	OUI
CHARACTER	OUI	OUI	OUI	OUI
CHARACTER_LENGTH	OUI		OUI	OUI
CHAR_LENGTH	OUI		OUI	OUI
CHECK	OUI	OUI	OUI	OUI
CLOB		OUI	OUI	OUI
CLOSE	OUI	OUI	OUI	OUI
COALESCE	OUI		OUI	OUI
COLLATE	OUI	OUI	OUI	OUI
COLLATION	OUI	OUI		
COLLECT			OUI	OUI
COLUMN	OUI	OUI	OUI	OUI
COMMIT	OUI	OUI	OUI	OUI

CONDITION		OUI	OUI	OUI
CONNECT	OUI	OUI	OUI	OUI
CONNECTION	OUI	OUI		
CONSTRAINT	OUI	OUI	OUI	OUI
CONSTRAINTS	OUI	OUI		
CONSTRUCTOR		OUI		
CONTINUE	OUI	OUI		
CONVERT	OUI		OUI	OUI
CORR			OUI	OUI
CORRESPONDING	OUI	OUI	OUI	OUI
COUNT			OUI	OUI
COVAR_POP			OUI	OUI
COVAR_SAMP			OUI	OUI
CREATE	OUI	OUI	OUI	OUI
CROSS	OUI	OUI	OUI	OUI
CUBE		OUI	OUI	OUI
CUME_DIST			OUI	OUI
CURRENT	OUI	OUI	OUI	OUI
CURRENT_DATE	OUI	OUI	OUI	OUI
CURRENT_DEFAULT_TRANSFORM_GROUP		OUI	OUI	OUI
CURRENT_PATH		OUI	OUI	OUI
CURRENT_ROLE		OUI	OUI	OUI
CURRENT_TIME	OUI	OUI	OUI	OUI
CURRENT_TIMESTAMP	OUI	OUI	OUI	OUI
CURRENT_TRANSFORM_GROUP_FOR_TYPE		OUI	OUI	OUI
CURRENT_USER	OUI	OUI	OUI	OUI
CURSOR	OUI	OUI	OUI	OUI
CYCLE		OUI	OUI	OUI
DATA		OUI		
DATE	OUI	OUI	OUI	OUI
DAY	OUI	OUI	OUI	OUI
DEALLOCATE	OUI	OUI	OUI	OUI
DEC	OUI	OUI	OUI	OUI
DECIMAL	OUI	OUI	OUI	OUI
DECLARE	OUI	OUI	OUI	OUI
DEFAULT	OUI	OUI	OUI	OUI

DEFERRABLE	OUI	OUI		
DEFERRED	OUI	OUI		
DELETE	OUI	OUI	OUI	OUI
DENSE_RANK			OUI	OUI
DEPTH		OUI		
DEREF		OUI	OUI	OUI
DESC	OUI	OUI		
DESCRIBE	OUI	OUI	OUI	OUI
DESCRIPTOR	OUI	OUI		
DETERMINISTIC		OUI	OUI	OUI
DIAGNOSTICS	OUI	OUI		
DISCONNECT	OUI	OUI	OUI	OUI
DISTINCT	OUI	OUI	OUI	OUI
DO		OUI		
DOMAIN	OUI	OUI		
DOUBLE	OUI	OUI	OUI	OUI
DROP	OUI	OUI	OUI	OUI
DYNAMIC		OUI	OUI	OUI
EACH		OUI	OUI	OUI
ELEMENT			OUI	OUI
ELSE	OUI	OUI	OUI	OUI
ELSEIF		OUI		
END	OUI	OUI	OUI	OUI
END-EXEC	OUI	OUI	OUI	OUI
EQUALS		OUI		
ESCAPE	OUI	OUI	OUI	OUI
EVERY			OUI	OUI
EXCEPT	OUI	OUI	OUI	OUI
EXCEPTION	OUI	OUI		
EXEC	OUI	OUI	OUI	OUI
EXECUTE	OUI	OUI	OUI	OUI
EXISTS	OUI	OUI	OUI	OUI
EXIT		OUI		
EXP			OUI	OUI
EXTERNAL	OUI	OUI	OUI	OUI
EXTRACT	OUI		OUI	OUI
FALSE	OUI	OUI	OUI	OUI
FETCH	OUI	OUI	OUI	OUI

FILTER			OUI	OUI
FIRST	OUI	OUI		
FIRST_VALUE				OUI
FLOAT	OUI	OUI	OUI	OUI
FLOOR			OUI	OUI
FOR	OUI	OUI	OUI	OUI
FOREIGN	OUI	OUI	OUI	OUI
FOUND	OUI	OUI		
FREE		OUI	OUI	OUI
FROM	OUI	OUI	OUI	OUI
FULL	OUI	OUI	OUI	OUI
FUNCTION		OUI	OUI	OUI
FUSION			OUI	OUI
GENERAL		OUI		
GET	OUI	OUI	OUI	OUI
GLOBAL	OUI	OUI	OUI	OUI
GO	OUI	OUI		
GOTO	OUI	OUI		
GRANT	OUI	OUI	OUI	OUI
GROUP	OUI	OUI	OUI	OUI
GROUPING		OUI	OUI	OUI
HANDLE		OUI		
HAVING	OUI	OUI	OUI	OUI
HOLD		OUI	OUI	OUI
HOUR	OUI	OUI	OUI	OUI
IDENTITY	OUI	OUI	OUI	OUI
IF		OUI		
IGNORE				OUI
IMMEDIATE	OUI	OUI		
IN	OUI	OUI	OUI	OUI
INDICATOR	OUI	OUI	OUI	OUI
INITIALLY	OUI	OUI		
INNER	OUI	OUI	OUI	OUI
INOUT		OUI	OUI	OUI
INPUT	OUI	OUI		
INSENSITIVE	OUI		OUI	OUI
INSERT	OUI	OUI	OUI	OUI
INT	OUI	OUI	OUI	OUI

INTEGER	OUI	OUI	OUI	OUI
INTERSECT	OUI	OUI	OUI	OUI
INTERSECTION			OUI	OUI
INTERVAL	OUI	OUI	OUI	OUI
INTO	OUI	OUI	OUI	OUI
IS	OUI	OUI	OUI	OUI
ISOLATION	OUI	OUI		
JOIN	OUI	OUI	OUI	OUI
KEY	OUI	OUI		
LAG				OUI
LANGUAGE	OUI	OUI	OUI	OUI
LARGE		OUI	OUI	OUI
LAST	OUI	OUI		
LAST_VALUE				OUI
LATERAL		OUI	OUI	OUI
LEAD				OUI
LEADING	OUI	OUI	OUI	OUI
LEAVE		OUI		
LEFT	OUI	OUI	OUI	OUI
LEVEL	OUI	OUI		
LIKE	OUI	OUI	OUI	OUI
LN			OUI	OUI
LOCAL	OUI	OUI	OUI	OUI
LOCALTIME		OUI	OUI	OUI
LOCALTIMESTAMP		OUI	OUI	OUI
LOCATOR		OUI		
LOOP		OUI		
LOWER	OUI		OUI	OUI
MAP		OUI		
MATCH	OUI	OUI	OUI	OUI
MAX	OUI		OUI	OUI
MEMBER			OUI	OUI
MERGE			OUI	OUI
METHOD		OUI	OUI	OUI
MIN	OUI		OUI	OUI
MINUTE	OUI	OUI	OUI	OUI
MOD			OUI	OUI
MODIFIES		OUI	OUI	OUI

MODULE	OUI	OUI	OUI	OUI
MONTH	OUI	OUI	OUI	OUI
MULTISET			OUI	OUI
NAMES	OUI	OUI		
NATIONAL	OUI	OUI	OUI	OUI
NATURAL	OUI	OUI	OUI	OUI
NCHAR	OUI	OUI	OUI	OUI
NCLOB		OUI	OUI	OUI
NESTING		OUI		
NEW		OUI	OUI	OUI
NEXT	OUI	OUI		
NO	OUI	OUI	OUI	OUI
NONE		OUI	OUI	OUI
NORMALIZE			OUI	OUI
NOT		OUI	OUI	OUI
NTH_VALUE				OUI
NTILE				OUI
NULL	OUI	OUI	OUI	OUI
NULLIF	OUI		OUI	OUI
NULLS				OUI
NUMERIC	OUI	OUI	OUI	OUI
OBJECT		OUI		
OCTET_LENGTH	OUI		OUI	OUI
OF	OUI	OUI	OUI	OUI
OFFSET				OUI
OLD		OUI	OUI	OUI
ON	OUI	OUI	OUI	OUI
ONLY	OUI	OUI	OUI	OUI
OPEN	OUI	OUI	OUI	OUI
OPTION	OUI	OUI		
OR	OUI	OUI	OUI	OUI
ORDER	OUI	OUI	OUI	OUI
ORDINALITY		OUI		
OUT		OUI	OUI	OUI
OUTER	OUI	OUI	OUI	OUI
OUTPUT	OUI	OUI		
OVER			OUI	OUI
OVERLAPS	OUI	OUI	OUI	OUI

OVERLAY			OUI	OUI
PAD	OUI	OUI		
PARAMETER		OUI	OUI	OUI
PARTIAL	OUI	OUI		
PARTITION			OUI	OUI
PATH		OUI		
PERCENTILE_CONT			OUI	OUI
PERCENTILE_DISC			OUI	OUI
PERCENT_RANK			OUI	OUI
POSITION	OUI		OUI	OUI
POWER			OUI	OUI
PRECISION	OUI	OUI	OUI	OUI
PREPARE	OUI	OUI	OUI	OUI
PRESERVE	OUI	OUI		
PRIMARY	OUI	OUI	OUI	OUI
PRIOR	OUI	OUI		
PRIVILEGES	OUI	OUI		
PROCEDURE	OUI	OUI	OUI	OUI
PUBLIC	OUI	OUI		
RANGE			OUI	OUI
RANK			OUI	OUI
READ	OUI	OUI		
READS		OUI	OUI	OUI
REAL	OUI	OUI	OUI	OUI
RECURSIVE		OUI	OUI	OUI
REDO		OUI		
REF		OUI	OUI	OUI
REFERENCES	OUI	OUI	OUI	OUI
REFERENCING		OUI	OUI	OUI
REGR_AVGX			OUI	OUI
REGR_AVGY			OUI	OUI
REGR_COUNT			OUI	OUI
REGR_INTERCEPT			OUI	OUI
REGR_R2			OUI	OUI
REGR_SLOPE			OUI	OUI
REGR_SXX			OUI	OUI
REGR_SXY			OUI	OUI
REGR_SYY			OUI	OUI

RELATIVE	OUI	OUI		
RELEASE		OUI	OUI	OUI
REPEATE		OUI		
RESIGNAL		OUI		
RESPECT				OUI
RESTRICT	OUI	OUI		
RESULT		OUI	OUI	OUI
RETURN		OUI	OUI	OUI
RETURNS		OUI	OUI	OUI
REVOKE	OUI	OUI	OUI	OUI
RIGHT	OUI	OUI	OUI	OUI
ROLE		OUI		
ROLLBACK	OUI	OUI	OUI	OUI
ROLLUP		OUI	OUI	OUI
ROUTINE		OUI		
ROW		OUI	OUI	OUI
ROWS	OUI	OUI	OUI	OUI
ROW_NUMBER			OUI	OUI
SAVEPOINT		OUI	OUI	OUI
SCHEMA	OUI	OUI		
SCOPE			OUI	OUI
SCROLL	OUI	OUI	OUI	OUI
SEARCH		OUI	OUI	OUI
SECOND	OUI	OUI	OUI	OUI
SECTION	OUI	OUI		
SELECT	OUI	OUI	OUI	OUI
SENSITIVE			OUI	OUI
SESSION	OUI	OUI		
SESSION_USER	OUI	OUI	OUI	OUI
SET	OUI	OUI	OUI	OUI
SETS		OUI		
SIGNAL		OUI		
SIMILAR		OUI	OUI	OUI
SIZE	OUI	OUI		
SMALLINT	OUI	OUI	OUI	OUI
SOME	OUI	OUI	OUI	OUI
SPACE	OUI	OUI		
SPECIFIC		OUI	OUI	OUI

SPECIFICTYPE		OUI	OUI	OUI
SQL	OUI	OUI	OUI	OUI
SQLCODE	OUI			
SQLERROR	OUI			
SQLEXCEPTION		OUI	OUI	OUI
SQLSTATE	OUI	OUI	OUI	OUI
SQLWARNING		OUI	OUI	OUI
SQRT			OUI	OUI
START		OUI	OUI	OUI
STATE		OUI		
STATIC		OUI	OUI	OUI
STDDEV_POP			OUI	OUI
STDDEV_SAMP			OUI	OUI
SUBMULTISET			OUI	OUI
SUBSTRING	OUI		OUI	OUI
SUM	OUI		OUI	OUI
SYMMETRIC			OUI	OUI
SYSTEM			OUI	OUI
SYSTEM_USER	OUI	OUI	OUI	OUI
TABLE	OUI	OUI	OUI	OUI
TABLESAMPLE			OUI	OUI
TEMPORARY	OUI	OUI		
THEN	OUI	OUI	OUI	OUI
TIME	OUI	OUI	OUI	OUI
TIMESTAMP	OUI	OUI	OUI	OUI
TIMEZONE_HOUR	OUI	OUI	OUI	OUI
TIMEZONE_MINUTE	OUI	OUI	OUI	OUI
TO	OUI	OUI	OUI	OUI
TRAILING	OUI	OUI	OUI	OUI
TRANSACTION	OUI	OUI		
TRANSLATE	OUI		OUI	OUI
TRANSLATION	OUI	OUI	OUI	OUI
TREAT		OUI	OUI	OUI
TRIGGER		OUI	OUI	OUI
TRIM	OUI		OUI	OUI
TRIM_ARRAY				OUI
TRUE	OUI	OUI	OUI	OUI
UESCAPE			OUI	OUI

UNDER		OUI		
UNDO		OUI		
UNION	OUI	OUI	OUI	OUI
UNIQUE	OUI	OUI	OUI	OUI
UNKNOWN	OUI	OUI	OUI	OUI
UNNEST		OUI	OUI	OUI
UNTIL		OUI		
UPDATE	OUI	OUI	OUI	OUI
UPPER	OUI		OUI	OUI
USAGE	OUI	OUI		
USER	OUI	OUI	OUI	OUI
USING	OUI	OUI	OUI	OUI
VALUE	OUI	OUI	OUI	OUI
VALUES	OUI	OUI	OUI	OUI
VARCHAR	OUI	OUI	OUI	OUI
VARYING	OUI	OUI	OUI	OUI
VAR_POP			OUI	OUI
VAR_SAMP			OUI	OUI
VIEW	OUI	OUI		
WHEN	OUI	OUI	OUI	OUI
WHENEVER	OUI	OUI	OUI	OUI
WHERE	OUI	OUI	OUI	OUI
WHILE		OUI		
WIDTH_BUCKET			OUI	OUI
WINDOW			OUI	OUI
WITH	OUI	OUI	OUI	OUI
WITHIN			OUI	OUI
WITHOUT		OUI	OUI	OUI
WORK	OUI	OUI		
WRITE	OUI	OUI		
YEAR	OUI	OUI	OUI	OUI
ZONE	OUI	OUI		