Les erreurs SQL les plus fréquentes

Date de publication : 22/02/2004

Par SQLPro

niveau : intermédiaire

Tous les débutants commetent les mêmes erreurs. Confondant par exemple la gestion de lignes de fichiers avec une table, la logique procédurale avec la logique ensembliste... Le but de cet article est de faire le point sur les erreurs les plus fréquentes, pourquoi sont-elles commises et comment y remédier.

1. Les noms d'objets SQL

J'ai donné des noms explicites à mes tables et colonnes, et je me heurte à des problèmes... Pourquoi ?

SQL est un langage qui repose sur une norme. Les noms des objets SQL ont donc une construction normative spécifique.
En particulier on ne peut faire usage que des 26 lettres de l'alphabet, des dix chiffres et du caractère "blanc souligné" (underscore). Tout autre caractère est interdit pour nommer un objet SQL (table, colonne, vue, utilisateur, contrainte...). Certains SGBDR autorisent n'importe quoi comme caractères dans le nom d'un objet... hélas !

Le nombre maximal de caractères est de 128, mais certains SGBDR ont des limites plus petites comme Oracle.
Sont donc interdits le caractère "blanc", les caractères diacritiques (accents, cédille, etc.).

En principe, **la casse** (c'est à dire majuscule et minuscule) **n'a pas d'importance** dans la nomination des objets. Cependant, certains SGBDR s'avèrent sensibles à la casse pour le nom des objets SQL dans certaines conditions. C'est la cas en particulier de SQL Server, si on le paramètre à l'installation en CASE SENSITIVE.

Il est très dangereux de donner des noms d'objets avec des blancs ou des accents. Pourquoi ? Parce que certains modules de programme ne travaillent pas sur les mêmes codes de pages. A l'heure où les bases de données sont de plus en plus intégrées à Internet, la multiplicité des interfaces de traitement, rend difficile, voire impossible l'interaction du code si les noms des objets SQL n'ont pas été définis correctement. De plus si le nom comporte des caractères illicites (ce que certains SGBDR permettent comme Access ou SQL Server) alors il faut ruser en utilisant des expressions spécifiques, et les traitements qui en découlent prennent plus de temps que si les noms avaient été définis proprement.

La norme en la matière

Bien écrire...

Voici quelques règles pour bien écrire votre code SQL...

Tout d'abord, toujours écrire votre code SQL en majuscule. Ainsi les noms des objets seront par exemple écrit :

```
CREATE TABLE CLIENT
(CLI_NUM INTEGER NOT NULL PRIMARY KEY,
 CLI_NOM VARCHAR(32))

SELECT *
FROM CLIENT
WHERE CLI_NOM LIKE 'DU%T'
```

Ouvrez n'importe quel bouquin d'informatique traitant de SQL. Vous y verrez toujours le code SQL en majuscule. C'est une habitude acquise depuis très longtemps. Elle est devenue un véritable standard de fait. Pourquoi ? Parce que lorsque l'on mélange du code client (Cobol, C, VB, Delphi, PHP, C#, Java...) et du code SQL, il est difficile de s'y retrouver si l'on ne peut facilement distinguer l'un de l'autre. Ainsi l'habitude a été prise d'écrire le code "client" en minuscule et le code "server" en majuscule. Ceci permet une distinction efficace et immédiate des éléments de code s'exécutant sur le seveur et ceux

s'exécutant localement.

Exemple :

```
With dmSys.QAdoGenericUpdate
do
begin
   if active
   then
      close;
   SQL.clear;
   SQL.add('UPDATE TS_PARAMETRES_BASE_PRB SET PRB_VALEUR = '+QuotedStr(ParaVal)
         +' WHERE PRB_NOM = '+QuotedStr(ParaName));
   execSQL;
end;
```

Ensuite, **indentez votre code SQL** ! En effet SQL peut devenir difficile à lire s'il n'est pas proprement présenté. Voici pour comparaison une même requête écrite malproprement et sa jumelle écrite avec indentation... Laquelle sauriez vous relire et comprendre plus rapidement ?

```
SELECT DISTINCT VILLE_ETP
FROM T_ENTREPOT WHERE
RAYON_RYN IN (SELECT
RAYON_RYN FROM T_ENTREPOT
WHERE RAYON_RYN NOT IN
(SELECT RAYON_RYN FROM
T_ENTREPOT WHERE
RAYON_RYN NOT IN (SELECT
RAYON_RYN  FROM T_RAYON))
) GROUP BY VILLE_ETP
HAVING COUNT (*) =
(SELECT COUNT(DISTINCT
RAYON_RYN) FROM T_RAYON)
```

```
SELECT DISTINCT VILLE_ETP
FROM   T_ENTREPOT
WHERE  RAYON_RYN IN (SELECT RAYON_RYN
                     FROM   T_ENTREPOT
                     WHERE  RAYON_RYN NOT IN (SELECT RAYON_RYN
                                              FROM   T_ENTREPOT
                                              WHERE RAYON_RYN N
RAYON_RYN
                                                          FROM
GROUP BY VILLE_ETP
HAVING COUNT (*) = (SELECT COUNT(DISTINCT RAYON_RYN)
                    FROM   T_RAYON)
```

La règle de base en la matière est :

1. toute requête doit être un bloc homogène de lignes indentées à la même origine
2. chaque nouvelle clause d'une requête doit commencer à la ligne
3. lorsqu'une clause est composée de plusieurs éléments bien distinct, les écrire sur différentes lignes
4. développer la clause FROM sous la forme d'un arbre avec une ligne pour la table et une ligne pour la clause de jointure
5. ne pas dépasser 70 à 80 caractères par ligne

Exemple :

```
SELECT ITM.ITM_ID, PRG.PRG_ID, PRG_TEXTE, TPG_LIBELLE,
       CASE
          WHEN TPG.TPG_ID = 1
               THEN ' ' +COALESCE(CAST(PRG_ORDRE AS VARCHAR(32)),'')
          ELSE ''
       END AS TEXTE,
       PRG_ORDRE

FROM   T_ITEM_ITM ITM
       LEFT OUTER JOIN T_PARAGRAPHE_PRG PRG
            ON ITM.ITM_ID = PRG.ITM_ID
            LEFT OUTER JOIN TR_TYPE_PARAGRAPHE_TPG TPG
                 ON PRG.TPG_ID = TPG.TPG_ID
       LEFT OUTER JOIN TR_STYLE_ITEM_STI STI
            ON ITM.STI_ID = STI.STI_ID

ORDER  BY ITM_BG, PRG_ORDRE
```

Ici l'arbre de jointure est :
Table T_ITEM_ITM => racine de l'arbre
Table T_PARAGRAPHE_PRG => branche depuis table racine T_ITEM_ITM
Table TR_TYPE_PARAGRAPHE_TPG => branche depuis table T_PARAGRAPHE_PRG (donc double indentation)
Table TR_STYLE_ITEM_STI => branche depuis table racine T_ITEM_ITM

Si le coeur vous en dit, n'hésitez pas à adopter un standard de codification des objets de votre base. Dans ce cas, lisez ceci : Normalisation des noms des objets des bases de données

Les noms des objets sont-ils sensibles à la casse ?

La norme SQL dit que par principe, les noms des objets SQL (nom de table, de vue, de colonne...) ne sont pas sensible à la casse. Ainsi T_Client et T_CLIENT sont un seul et même objet. Mais cette règle possède une exception. En effet, lorsque l'on utilise un identifiant de SQL comme nom, alors le nom devient sensible à la casse. Par exemple, si je veux utiliser le mot clef TYPE comme nom de colonne d'une table, il me faut l'entourer de guillemets car ce mot est réservé. Dès lors, ce nom de colonne devient sensible à la casse.

Exemple :

```
CREATE MA_TABLE
("TYPE"  INTEGER,
 "Type"  FLOAT,
 "TYpe"  CHAR(1))
```

Constitue un ordre de création de table SQL tout à fait valide !

ATTENTION : comme les noms des objets sont stockés dans des tables de la base dites "table de schéma" ou "dictionnaire" ou encore table "système", certains SGBDR dans certaines configurations sont sensibles à la casse pour les noms des objets. C'est le cas par exemple de SQL Server qui, lors de son installation peut être configuré pour être sensible à la casse...

2. Terminologie SQL

Idée reçue : "champ" et "enregistrement" n'existent pas dans les bases de données ! Pas plus que "droits"...

CHAMP: cette notion n'existe pas dans les bases de données car un champ est un élément visuel (champ opératoire du chirurgien, champ visuel du pilote, champs de saisie dans une interface de saisie...)
En base de données on parle de "**colonne**", car les tables sont visualisables sous forme de tableaux dans lesquels existent des lignes et des colonnes...

ENREGISTREMENT: cette notion n'existe pas dans les bases de données car un enregistrement suppose un fichier lu ligne par ligne, alors que (sauf exception pour les bases de type fichier) toutes les tables d'une même base sont stockées dans un seule et même fichier par des granules de stockage unitaires appelée "pages".
On parle alors de **ligne** qui est une notion virtuelle.

DROITS: les droits n'existent pas en SQL. En effet les droits supposent un mode d'accès sur un fichier. Or les bases de données et SQL gèrent différents modes qui n'ont pas forcément quelque chose à voir avec les lectures et écritures de données. Ainsi SQL permet de définir des privilèges d'utilisation sur des objets de bases de données qui ne sont ni des tables, ni des vues, ni des colonnes, ni des données, mais des éléments de conception de la base de données... C'est pourquoi parler de droits est ... absurde et réservé aux notions "systèmes" basées sur des fichiers et des ressources. Dans SQL on parle donc de **privilèges** que l'on **octroie** ou **révoque**.

3. NULL, n'est pas une valeur !

Je n'arrive pas à retrouver mes valeurs nulles.

Je fais SELECT ... WHERE MaColonne = NULL et cela ne marche pas !
NULL : cet élément du SQL n'est pas une valeur, puisque c'est justement l'absence de valeur. En fait NULL est un marqueur comme NIL en programmation qui indique qu'au bout du pointeur il n'y a pas de ressource.
Par exemple comparer une colonne à NULL n'a aucun sens. Pour traiter les marqueurs NULL il faut utiliser des prédicats spéciaux comme IS NULL, IS NOT NULL ou des fonctions particulières comme COALESCE ou NULLIF ou encore en utilisant la structure CASE.

A lire sur le sujet les opérateurs de traitement des valeurs NULL : Opérateurs de traitement des marqueurs NULL

4. CASE SENSITIVE

Une fausse bonne idée consiste à **définir une base de données** (ou certaines tables, colonnes) avec une collation qui rends les chaînes de caractères **insensible à la casse** (plus de différences entre majuscules et minuscules). C'est une hérésie que d'utiliser un tel comportement par défaut. En effet autant il est facile dans les requêtes de s'affranchir de la casse dans les comparaisons, autant il est très difficile voire impossible de rendre des éléments sensible à la casse lorsque ces derniers ont été rendus CASE INSENSITIVE ! Pour rendre vos comparaisons insensibles à la casse, il suffit de jouer avec la fonction UPPER ou LOWER présente dans tous les SGBDR.

Exemple :

```
SELECT *
FROM CLIENT
WHERE UPPER(CLI_NOM) LIKE 'DU%T'
```

Mais comment faire la distinction entre "Dupont" et "DUPONT" si votre SGBDR a été rendu insensible à la casse ?
La solution consiste, si votre SGBDR le supporte, à transtyper les colonnes en chaînes binaires :

Exemple :

```
SELECT *
FROM CLIENT
WHERE CAST(CLI_NOM AS VARBINARY(32)) =  CAST('DUPONT' AS VARBINARY(32))
```

Ce code est évidemment incommensurablement plus long à exécuter que si votre SGBDR avait été rendu sensible à la casse !

Au fait, savez-vous pourquoi on parle de casse pour distinguer les majuscules et minuscules ? Cette terminologie existe depuis des temps anciens. Des temps ou les imprimeurs avaient à leur disposition pour fabriquer les matrices d'impression, un grand "bac" divisé en de multiples cases dans lesquelles on trouvait les majuscules dans les cases des rangées supérieures et les minuscules dans les cases des rangées inférieurs. Comme l'on utilisait plus fréquemment les minuscules, on les avaient rangés en bas de manière à ce quelle soient plus accessibles à l'utilisateur. Ce grand bac, s'apelle une casse, et cette technique de rangement relève de l'ergonomie. ! Donc, en bas les minuscules en haut les majuscules, d'où les expressions bas de casse (minuscules) et haut de casse (majuscules). Bref, l'informatique n'a rien inventé en la matière...

Voir à ce sujet les problématiques de pages de caractères et collations : Une question de caractères...

5. Le dernier...

Une question qui revient souvent dans les discussions est de **retrouver la dernière ligne insérée dans une table**. Cette question n'a aucun sens dans l'univers des bases de données, car ces dernières ne possèdent pas d'ordre établie. Une table est un sac de bille. Si vous recevez mon sac de bille, avez-vous un moyen quelconque de savoir quelle a été la dernière bille que j'y ai fourrée ? Les SGBDR ont l'habitude de minimiser les

coûts de gestion des entrées/sorties de disque. Autrement dit, lorsque je supprime une ligne, celle ci n'est pas effacée, ni compactée. Elle reste un espace mort. Si une nouvelle insertion arrive, il y a fort à parier que votre SGBDR y placera la nouvelle ligne à insérer !

Exemple :

```
CREATE TABLE TEST
(MA_COLONNE VARCHAR(32))

INSERT INTO TEST VALUES ('abc')
INSERT INTO TEST VALUES ('def')
INSERT INTO TEST VALUES ('ghi')

DELETE TEST WHERE MA_COLONNE = 'def'

INSERT INTO TEST VALUES ('jkl')

SELECT * FROM TEST
```

```
MA_COLONNE
----------
abc
jkl
ghi
```

Vous vous attendiez sans doute que la ligne "jkl" soit la dernière ! Erreur, le SGBDR a récupéré la place de la ligne "def" pour y mettre jkl !

Il n'y a aucune notion d'ordre dans les SGBDR, ni ordre des lignes, ni ordre des colonnes, sauf ceux que l'utilisateur veut y voir figurer dans les données en ajoutant une colonne spécifique pour le tri.
Tout au plus, si une colonne possède une donnée dont la valeur est toujours supérieure a celles déjà insérées (par exemple une colonne auto incrémentée) alors on peut retrouver la valeur maximale (donc la dernière ligne insérée) par une requête de calcul d'agrégat comme SELECT MAX(...
Mais s'il faut avoir l'assurance de la séquence dans le temps, il convient d'ajouter à la table une colonne contenant une défintion de type TIMESTAMP (combiné date heure) avec la valeur par défaut du serveur par une fonction comme CURRENT_TIMESTAMP et cela dans la défintion de la table.

Exemple :

```
CREATE MaTable
(ID INT,
 DT TIMESTAMP DEFAULT CURRENT_TIMESTAMP)
```

Mais cette pratique est dangereuse, car lors d'insertion massives la précision de ce combiné date heure peut générer des données indentiques...
C'est pourquoi certains SGBDR ont ajouté a leur définition un type particulier tel que l'horodatage (TIMESTAMP par exemple pour MS SQL Server) assurant d'avoir toujours une colonne dont toutes les valeurs sont distinctes et croissante en fonction du moment de l'insertion.
Enfin, dans la cas de calculs de clefs auto incrémentées, l'utilisation de SELLECT MAX(...) +1 est une abération qui ne peut que conduire la base à des violations de contraintes de clefs primaires du fait de la concurrence.

A lire sur le sujet : Clefs auto-incrémentées

6. Position...

Autre problématique, ajouter une colonne à la position n, par exemple entre la colonne x et la colonne y de la table... Autre utopie bien entendu car il n'existe pas d'ordre dans les bases de données (voir ci dessus). Par d'ordre concernant les tables, ni d'ordre concernant les colonnes au sein de la table.
Quoique... !
Tout bien considéré la norme SQL présente dans les informations de schéma une colonne intitulé "ORDINAL_POSITION" qui précise la place de la colonne au sein de la liste des colonnes de la table et cette place correspondant généralement à celle définie lors de la création. Toute nouvelle colonne étant rajoutée a la position la plus haute incrémentée d'une unité. Cet ordre ordinal est utilisé notamment lors de l'insertion de données (INSERT) lorsque l'on ne précise pas la liste des colonnes. Autre particularité : la possibilité de définir les colonnes dans la clause de tri (ORDER BY) par leur ordre positionnel dans la clause SELECT et non par leur nom...

La question bête qui revient sans cesse est : **comment faire pour insérer une colonne Z exactement entre la colonne X et la colonne Y ?**
Simplement en recréant la table. Mais cette méthode, qui n'offre aucune certitude est lourde à utiliser car il faut préserver les données dans une table temporaire avant de détruire la table pour la recréer dans son architecture nouvelle.
D'ailleurs SQL n'offre aucune certitude quand à l'ordre de restitution des colonnes lorsque, au lieu de spécifier les colonnes explicitement on utiliser le joker * !

Mais si vous vous obstinez à vouloir placer votre nouvelle colonne à la position n alors, visitez la page : Changer le nom ou le type d'une colonne

7. Format de Date...

La notion de format de date n'existe pas en SQL ni dans les SGBDR. La date est stockée sous un type DATE, qui, la plupart du temps est un entier et pour TIMESTAMP (DATETIME) un réel. La partie entière étant le nombre de jours écoulés depuis une certaine date. Ainsi dans SQL Server la date "origine" est le 1er janvier 1900 :

```
SELECT CAST(0 AS DATETIME) AS DATE_TEST
```

```
DATE_TEST
-----------------------
1900-01-01 00:00:00.000
```

La partie fractionnaire compte le temps écoulé sur 24 heures. Par exemple 0,5 signifie 1er janvier 1900 à midi pour SQL Server.
Mais comment spécifier une date avec une expression littérale comme 21/01/2003 16h18m ?
Il convient d'écrire la date en littéral et de la transtyper en DATE ou DATETIME. Le trsantypage peut être implicite ou explicite :

Exemples :

```
SELECT CAST('21/01/2003 12:23' AS DATETIME) AS DATE_TEST => conversion explicite

INSERT INTO (ID, DATE_FIN) VALUES (33, '21/01/2003 12:23') => conversion implicite
car le colonne DATE_FIN est de type DATETIME
```

Mais sur quel format la conversion va t-elle opérer ?

La plupart des SGBDR reposent sur le format normatif ISO de date qui spécifie :
AAAA-MM-JJ hh:mm:ss.xxx

Exemple :

```
SELECT CAST('2003-01-21 12:33:00.000' AS DATETIME) AS DATE_TEST
```

Mais certains SGBDR proposent de pouvoir spécifier librement le format avec lequel on veut travailler. Par exemple pour SQL Server il faut positionner le flag DATEFORMAT avec le paramétrage souhaité. Ce flag peut prendre les valeurs : YMD, YDM, MDY, MYD, DYM, DMY.
Voici comment sous SQL Server on précise l'utilisation des dates ISO :

```
SET DATEFORMAT YMD
SELECT CAST('2003-01-21 12:33:00.000' AS DATETIME) AS DATE_TEST
```

Et voilà !

Pour d'autres problématiques de date, référez vous à l'article sur la gestion des plannings : Calendrier, timing et horaires en SQL...

8. Dédoublonnage

8.1. DISTINCTROW ou l'exemple parfait de l'absurdité d'Access !

Les quidams qui passent d'Access à un SGBDR qui respecte la norme SQL (car SQL est un langage fortement normalisé : 1986, 1992, 1999, 2003 !) ne comprennent pas que l'opérateur DISTINCTROW n'existe pas dans SQL... Rapellons à quoi sert ce mot clef loufoque : "DISTINCTROW" concerne les requêtes qui utilisent au moins une jointure. Il arrive que les tuples résultant apportent des données doublonnées dans le sous ensemble des colonnes de la table mère. Pour empêcher cela Access nous invite à utiliser le mot DISTINCTROW. Un exemple étant toujours plus compréhensible voici les élements de notre jeu d'essai :

Soit une table T_PERSONNE_PRS contenant des "parents" :

```
PRS_ID     PRS_NOM         PRS_DATE_NAISSANCE
---------  --------------  --------------------
1          DUPONT          21/05/1960
2          DUPONT          12/07/1972
3          DUVAL           16/11/1980
```

et une table des enfants T_ENFANT_ENF :

```
ENF_ID     PRS_ID          ENF_PRENOM
---------  --------------  --------------------
48         1               Camille
57         1               Alain
63         2               Marcel
78         3               Raoul
```

Voici ce que donne la requete qui "met à plat" les informations :

```
SELECT PRS.PRS_ID, PRS.PRS_NOM, PRS.PRS_DATE_NAISSANCE, ENF.ENF_ID, ENF.ENF_PRENOM
FROM   T_PERSONNE_PRS PRS
       INNER JOIN T_ENFANT_ENF ENF
             ON PRS.PRS_ID = ENF.PRS_ID
```

```
PRS_ID     PRS_NOM         PRS_DATE_NAISSANCE  ENF_ID      ENF_PRENOM
---------  --------------  ------------------  ----------  ----------
1          DUPONT          21/05/1960          48          Camille
1          DUPONT          21/05/1960          57          Alain
2          DUPONT          12/07/1972          63          Marcel
3          DUVAL           16/11/1980          78          Raoul
```

Voici maintenant une requête qui donne un aperçu partiel des données jointes :

```
SELECT PRS.PRS_NOM
FROM   T_PERSONNE_PRS PRS
       INNER JOIN T_ENFANT_ENF ENF
             ON PRS.PRS_ID = ENF.PRS_ID
```

```
PRS_NOM
--------------
DUPONT
DUPONT
DUPONT
DUVAL
```

Comme on le voit, le nom DUPONT apparait trois fois, ce qui n'a pas d'intérêt car il est impossible de savoir quelle ligne contenant le nom DUPONT appartient à la personne de clef 1 ou 2 ou de l'enfant de clef 48, 57 ou 63...

Avec le mot clef DISTINCT (celui de la norme SQL) , nous obtenons :

```
SELECT DISTINCT PRS.PRS_NOM
FROM   T_PERSONNE_PRS PRS
       INNER JOIN T_ENFANT_ENF ENF
             ON PRS.PRS_ID = ENF.PRS_ID
```

```
PRS_NOM
--------------
```

```
DUPONT
DUVAL
```

Les lignes redondantes ont été éliminées, et c'est tant mieux !

Mais avec le mot Access DISTINCTROW, une surprise nous attend...

```
SELECT DISTINCTROW PRS.PRS_NOM
FROM   T_PERSONNE_PRS PRS
       INNER JOIN T_ENFANT_ENF ENF
             ON PRS.PRS_ID = ENF.PRS_ID
```

```
PRS_NOM
--------------
DUPONT
DUPONT
DUVAL
```

Mais comment cette ineptie est-elle possible ??? Je vous le demande !
Simplement parce qu'il y a deux DUPONT, l'un d'identifiant 1 et l'autre d'identifiant 2 et c'est pourquoi vous vous retrouvez avec deux fois DUPONT, sans savoir à la lecture du résultat à quel DUPONT correspond l'identifiant 1 ou 2 !!!

A quoi ça sert ? A rien, sinon à embrouiller le développeur car cet opérateur viole les bases mêmes de la logique ensembliste sur laquelle repose les fondements des bases de données.

8.2. Dédoublonnage partiel

En revanche, une question qui revient souvent est de pouvoir dédoublonner partiellement un ensemble de données. Cette demande cache souvent une incompréhension globale de la logique ensembliste des bases de données et une méconnaissance de SQL en particulier.

Pour mieux comprendre cette demande, nous allons créer notre jeu de données :
Soit la table T_MACHINE_MCH et la table T_INCIDENT_ICD qui historise les incidents sur les différentes machines :

```
CREATE TABLE T_MACHINE_MCH
(MCH_ID INTEGER NOT NULL PRIMARY KEY,
 MCH_NOM VARCHAR(16))
```

```
INSERT INTO T_MACHINE_MCH
       VALUES (1, 'Avion')
INSERT INTO T_MACHINE_MCH
       VALUES (2, 'Vélo')
```

```
CREATE TABLE T_INCIDENT_ICD
(ICD_ID INTEGER NOT NULL PRIMARY KEY,
 MCH_ID INTEGER NOT NULL FOREIGN KEY
        REFERENCES T_MACHINE_MCH (MCH_ID),
 ICD_NATURE VARCHAR(16),
 ICD_DATEHEURE TIMESTAMP)
```

```
INSERT INTO T_INCIDENT_ICD
       VALUES (63, 1, 'Arrêt réacteur', '2002-11-17 21:58:23')
INSERT INTO T_INCIDENT_ICD
       VALUES (78, 1, 'Panne APU', '2002-11-17 22:03:21')
INSERT INTO T_INCIDENT_ICD
       VALUES (79, 2, 'Crevaison', '2003-02-27 09:11:57')
INSERT INTO T_INCIDENT_ICD
       VALUES (82, 2, 'Casse chaine', '2003-03-04 11:22:33')
INSERT INTO T_INCIDENT_ICD
       VALUES (87, 2, 'Roue voilée', '2003-09-01 17:14:33')
```

Notre quidam voudrait récupérer le dernier incident ayant eût lieu (dans l'ordre chronologique) sur chacune des machines, mais surtout, il voudrait récupérer l'identifiant de ce dernier incident, afin de pouvoir parfaitement identifier les données concernant ces incidents.

Voici la jointure des données entre les deux tables :

```
SELECT MCH.MCH_ID, MCH_NOM, ICD_ID, ICD_NATURE, ICD_DATEHEURE
FROM   T_MACHINE_MCH MCH
       INNER JOIN T_INCIDENT_ICD ICD
             ON MCH.MCH_ID = ICD.MCH_ID
```

```
MCH_ID      MCH_NOM          ICD_ID      ICD_NATURE
ICD_DATEHEURE
----------- ---------------- ----------- ---------------- -----------------------
1           Avion            63          Arrêt réacteur   2002-11-17 21:58:23.000
1           Avion            78          Panne APU        2002-11-17 22:03:21.000
2           Vélo             79          Crevaison        2003-02-27 09:11:57.000
2           Vélo             82          Casse chaine     2003-03-04 11:22:33.000
2           Vélo             87          Roue voilée      2003-01-09 17:14:33.000
```

Ce que notre quidam veut obtenir, c'est la réponse suivante :

```
MCH_ID      MCH_NOM          ICD_ID      ICD_NATURE
ICD_DATEHEURE
----------- ---------------- ----------- ---------------- -----------------------
1           Avion            78          Panne APU        2002-11-17 22:03:21.000
2           Vélo             82          Casse chaine     2003-03-04 11:22:33.000
```

Autrement dit il voudrait dédoublonner sur les colonnes MCH_ID et MCH_NOM en obtenant le MAX de ICD_DATEHEURE et les données de ICD_ID / ICD_NATURE...

Quelque chose comme :

```
SELECT DISTINCT(MCH.MCH_ID, MCH_NOM), ICD_ID, ICD_NATURE, MAX(ICD_DATEHEURE)
FROM   T_MACHINE_MCH MCH
       INNER JOIN T_INCIDENT_ICD ICD
             ON MCH.MCH_ID = ICD.MCH_ID
```

```
MCH_ID      MCH_NOM          ICD_ID      ICD_NATURE
ICD_DATEHEURE
----------- ---------------- ----------- ---------------- -----------------------
1           Avion            78          Panne APU        2002-11-17 22:03:21.000
2           Vélo             82          Casse chaine     2003-03-04 11:22:33.000
```

Or DISTINCT n'est pas une fonction applicable à une colonne ou un groupe de colonne particulier. DISTINCT concerne la totalité des données de la ligne résultante. Il faut donc procéder par étape :

ÉTAPE N°1 : trouver les incidents ayant le MAX de la dateHeure pour chaque machine :

```
SELECT MAX(ICD_DATEHEURE) AS MAX_DH, MCH_ID
FROM   T_INCIDENT_ICD
GROUP  BY MCH_ID
```

```
MAX_DH                                                  MCH_ID
------------------------------------------------------- ------
```

```
2002-11-17 22:03:21.000                              1
2003-03-04 11:22:33.000                              2
```

ÉTAPE N°2 : relier ce résultat avec l'incident visé à l'aide d'une sous requête corrélée sur l'identifiant de machine :

```
SELECT ICD_ID, MCH_ID, ICD_NATURE, ICD_DATEHEURE
FROM   T_INCIDENT_ICD ICD1
WHERE  ICD_DATEHEURE = (SELECT MAX(ICD_DATEHEURE)
                        FROM   T_INCIDENT_ICD ICD2
                        WHERE  ICD2.MCH_ID = ICD1.MCH_ID)
```

```
ICD_ID      MCH_ID      ICD_NATURE
ICD_DATEHEURE
----------- ----------- --------------- -----------------------
82          2           Casse chaine     2003-03-04 11:22:33.000
78          1           Panne APU        2002-11-17 22:03:21.000
```

ÉTAPE N°3 : étendre ce résultat par jointure pour obtenir les informations sur les machines :

```
SELECT ICD_ID, MCH.MCH_ID, ICD_NATURE, ICD_DATEHEURE, MCH_NOM
FROM   T_INCIDENT_ICD ICD1
       INNER JOIN T_MACHINE_MCH MCH
             ON ICD1.MCH_ID = MCH.MCH_ID
WHERE  ICD_DATEHEURE = (SELECT MAX(ICD_DATEHEURE)
                        FROM   T_INCIDENT_ICD ICD2
                        WHERE  ICD2.MCH_ID = ICD1.MCH_ID)
```

```
ICD_ID      MCH_ID      ICD_NATURE       ICD_DATEHEURE
MCH_NOM
----------- ----------- --------------- --------------------------- -------
82          2           Casse chaine     2003-03-04 11:22:33.000      Vélo
78          1           Panne APU        2002-11-17 22:03:21.000      Avion
```

Et voilà notre problème résolu.

Difficile de faire cela sans sous requêtes...

9. Cosmétique...

Je voudrais présenter mes données de telle manière. Comment faire cela avec SQL ?

Un SGBDR n'est surtout pas fait pour faire du "cosmétique" !

La cosmétique (nom commun) signifie "qui est propre aux soins de beauté"...
Non, un SGBDR ne sert pas, et ne doit jamais servir, à traiter la présentation des données. Une fois que les données sont là dans le résultat d'une requête, alors il suffit de piocher ou l'on veut pour construire telle ou telle "facture" de présentation : en ligne, en colonne, en tableau, en cube, a onglet, en relief, en hébreu et en couleur, si cela vous fait plaisir, mais pas avec un SGBDR. L'ensemble de ces traitements doivent être effectués par la couche client du serveur, c'est à dire, le programme exécutable ou un objet d'un serveur d'application par exemple. MAIS PAS EN CODE SQL !!!
Le langage SQL n'est pas fait pour cela, et le SGBDR se révélera TOUJOURS un veau si vous lui demandez de faire ce travail et tout le monde sera mécontent : vous d'abord, car a la longue le temps de traitement sera désatsreux, lui ensuite (le SGBDR) parce qu'il s'emmerdera a faire de la peinture alors que dans votre orchestre lui, c'est le piano, et vous lui demander de jouer le triangle !

L'exemple le plus classique est le suivant... Je voudrais que mes données en colonnes soient remise en ligne et tout cela dans une jolie requête !
Ben voyons...
Alors puisqu'il faut démontrer l'absurdité d'une telle chose, démontrons là...
Comme exemple nous allons nous intéresser au calcul du chiffre d'affaire par trimestre en fonction des clients.

Voici une table des commandes :

```
CREATE TABLE COMMANDE
(CMD_ID INTEGER,
 CLI_ID INTEGER,
 CMD_DATE DATE,
 CMD_MONTANT FLOAT)
```

```
INSERT INTO COMMANDE (CMD_ID, CLI_ID, CMD_DATE,     CMD_MONTANT)
              VALUES (33,     1,      '2000-10-18', 1287.22)
INSERT INTO COMMANDE (CMD_ID, CLI_ID, CMD_DATE,     CMD_MONTANT)
              VALUES (101,    1,      '2001-01-15', 7854.12)
INSERT INTO COMMANDE (CMD_ID, CLI_ID, CMD_DATE,     CMD_MONTANT)
              VALUES (102,    1,      '2001-03-10', 11474.25)
INSERT INTO COMMANDE (CMD_ID, CLI_ID, CMD_DATE,     CMD_MONTANT)
              VALUES (103,    1,      '2001-11-01', 3587.00)
INSERT INTO COMMANDE (CMD_ID, CLI_ID, CMD_DATE,     CMD_MONTANT)
              VALUES (104,    2,      '2001-02-02', 114472.89)
INSERT INTO COMMANDE (CMD_ID, CLI_ID, CMD_DATE,     CMD_MONTANT)
              VALUES (105,    2,      '2001-02-25', 858.21)
INSERT INTO COMMANDE (CMD_ID, CLI_ID, CMD_DATE,     CMD_MONTANT)
              VALUES (106,    2,      '2001-01-15', 7854.12)
INSERT INTO COMMANDE (CMD_ID, CLI_ID, CMD_DATE,     CMD_MONTANT)
              VALUES (107,    2,      '2001-05-11', 82462.05)
INSERT INTO COMMANDE (CMD_ID, CLI_ID, CMD_DATE,     CMD_MONTANT)
              VALUES (108,    2,      '2001-07-01', 61458.00)
INSERT INTO COMMANDE (CMD_ID, CLI_ID, CMD_DATE,     CMD_MONTANT)
              VALUES (109,    2,      '2001-12-01', 962.28)
```

CLI_ID étant la clef étrangère référençant le client.

Notre direction commerciale cherche à obtenir les chiffres des ventes par client et par trimestre concernant l'année 2001...
La requête peut s'écrire :

```
SELECT CLI_ID,
       CASE
          WHEN    (EXTRACT(MONTH FROM CMD_DATE) = 1)
               OR (EXTRACT(MONTH FROM CMD_DATE) = 2)
               OR (EXTRACT(MONTH FROM CMD_DATE) = 3)  THEN 1
          WHEN    (EXTRACT(MONTH FROM CMD_DATE) = 4)
               OR (EXTRACT(MONTH FROM CMD_DATE) = 5)
```

```
              OR (EXTRACT(MONTH FROM CMD_DATE) = 6)  THEN 2
          WHEN    (EXTRACT(MONTH FROM CMD_DATE) = 7)
              OR (EXTRACT(MONTH FROM CMD_DATE) = 8)
              OR (EXTRACT(MONTH FROM CMD_DATE) = 9)  THEN 3
          WHEN    (EXTRACT(MONTH FROM CMD_DATE) = 10)
              OR (EXTRACT(MONTH FROM CMD_DATE) = 11)
              OR (EXTRACT(MONTH FROM CMD_DATE) = 12) THEN 4
       END AS TRIMESTRE,
       SUM(CMD_MONTANT) AS CA
FROM COMMANDE
WHERE EXTRACT(YEAR FROM CMD_DATE) = 2001
GROUP BY CLI_ID,
          CASE
             WHEN    (EXTRACT(MONTH FROM CMD_DATE) = 1)
                  OR (EXTRACT(MONTH FROM CMD_DATE) = 2)
                  OR (EXTRACT(MONTH FROM CMD_DATE) = 3)  THEN 1
             WHEN    (EXTRACT(MONTH FROM CMD_DATE) = 4)
                  OR (EXTRACT(MONTH FROM CMD_DATE) = 5)
                  OR (EXTRACT(MONTH FROM CMD_DATE) = 6)  THEN 2
             WHEN    (EXTRACT(MONTH FROM CMD_DATE) = 7)
                  OR (EXTRACT(MONTH FROM CMD_DATE) = 8)
                  OR (EXTRACT(MONTH FROM CMD_DATE) = 9)  THEN 3
             WHEN    (EXTRACT(MONTH FROM CMD_DATE) = 10)
                  OR (EXTRACT(MONTH FROM CMD_DATE) = 11)
                  OR (EXTRACT(MONTH FROM CMD_DATE) = 12) THEN 4
          END
ORDER BY CLI_ID, TRIMESTRE
```

```
CLI_ID      TRIMESTRE   CA
----------- ----------- -----------
1           1           19328.37
1           4           3587.0
2           1           123185.22
2           2           82462.05
2           3           61458.0
2           4           962.28
```

Bien entendu, notre service commercial préférerait avoir les résultats sous cette forme :

```
CLI_ID  CA_TRIMESTRE_1  CA_TRIMESTRE_2  CA_TRIMESTRE_3  CA_TRIMESTRE_4
------  --------------  --------------  --------------  --------------
1       19328.37                                        3587.0
2       123185.22       82462.05        61458.0         962.28
```

Cela permet de mieux se rendre compte que certains trimestres sont à zéro pour certains clients, ce qui n'était pas évident dans la présentation venant de la réponse à notre première requête.
Déterminer le chiffre d'affaires par client pour un trimestre déterminé n'est pas bien difficile en SQL. Par exemple calculons-le pour le premier trimestre 2001 :

```
SELECT CLI_ID, SUM(CMD_MONTANT) AS CA_T1
FROM   COMMANDE
WHERE  CMD_DATE BETWEEN '2001-01-01' AND '2001-03-31'
GROUP  BY CLI_ID
```

```
CLI_ID      CA_T1
----------- --------------
1           19328.37
2           123185.22
```

Il n'est pas possible, du fait des groupages nécessaire à l'agrégation de caculer les autres trimestres dans la même requête. Mais il est possible de modifier la présentation pour la faire correspondre au souhait de notre service commercial :

```
SELECT CLI_ID,
       SUM(CMD_MONTANT) AS CA_T1,
       0.0 AS CA_T2,
       0.0 AS CA_T3,
       0.0 AS CA_T2
FROM   COMMANDE
WHERE  CMD_DATE BETWEEN '2001-01-01' AND '2001-03-31'
GROUP  BY CLI_ID
```

Rien ne nous empêche maintenant de répéter cette requête pour chacun des trimestres et d'effectuer une union pour aboutir à une table plus proche de la demande :

```
SELECT CLI_ID, SUM(CMD_MONTANT) AS CA_T1, 0.0 AS CA_T2, 0.0 AS CA_T3, 0.0 AS CA_T4
FROM   COMMANDE
WHERE  CMD_DATE BETWEEN '2001-01-01' AND '2001-03-31'
GROUP  BY CLI_ID
UNION
SELECT CLI_ID, 0.0 AS CA_T1, SUM(CMD_MONTANT) AS CA_T2, 0.0 AS CA_T3, 0.0 AS CA_T4
FROM   COMMANDE
WHERE  CMD_DATE BETWEEN '2001-04-01' AND '2001-06-30'
GROUP  BY CLI_ID
UNION
SELECT CLI_ID, 0.0 AS CA_T1, 0.0 AS CA_T2, SUM(CMD_MONTANT) AS CA_T3, 0.0 AS CA_T4
FROM   COMMANDE
WHERE  CMD_DATE BETWEEN '2001-07-01' AND '2001-09-30'
GROUP  BY CLI_ID
UNION
SELECT CLI_ID, 0.0 AS CA_T1, 0.0 AS CA_T2, 0.0 AS CA_T3, SUM(CMD_MONTANT) AS CA_T4
FROM   COMMANDE
WHERE  CMD_DATE BETWEEN '2001-10-01' AND '2001-12-31'
GROUP  BY CLI_ID
```

```
CLI_ID      CA_T1        CA_T2        CA_T3        CA_T4
----------- ------------ ------------ ------------ ------------
1           0.0          0.0          0.0          3587.0
1           19328.38     0.0          0.0          0.0
2           0.0          0.0          0.0          962.28
2           0.0          0.0          61458.0      0.0
2           0.0          82462.05     0.0          0.0
2           123185.22    0.0          0.0          0.0
```

Dès lors la solution saute aux yeux : il suffit de faire la somme de ces différents chiffres d'affaire trimestriels par client. Cela est possible en imbriquant la précédente requête dans une clause FROM d'un SELECT :

```
SELECT S.CLI_ID, SUM(S.CA_T1) AS CA_T1, SUM(S.CA_T2) AS CA_T2,
                 SUM(S.CA_T3) AS CA_T3, SUM(S.CA_T4) AS CA_T4
FROM (SELECT CLI_ID, SUM(CMD_MONTANT) AS CA_T1,  0.0 AS CA_T2, 0.0 AS CA_T3, 0.0 AS
CA_T4
      FROM   COMMANDE
      WHERE  CMD_DATE BETWEEN '2001-01-01' AND '2001-03-31'
```

```
      GROUP  BY CLI_ID
      UNION
      SELECT CLI_ID, 0.0 AS CA_T1, SUM(CMD_MONTANT) AS CA_T2, 0.0 AS CA_T3, 0.0 AS
CA_T4
      FROM   COMMANDE
      WHERE  CMD_DATE BETWEEN '2001-04-01' AND '2001-06-30'
      GROUP  BY CLI_ID
      UNION
      SELECT CLI_ID, 0.0 AS CA_T1, 0.0 AS CA_T2, SUM(CMD_MONTANT) AS CA_T3, 0.0 AS
CA_T4
      FROM   COMMANDE
      WHERE  CMD_DATE BETWEEN '2001-07-01' AND '2001-09-30'
      GROUP  BY CLI_ID
      UNION
      SELECT CLI_ID, 0.0 AS CA_T1, 0.0 AS CA_T2, 0.0 AS CA_T3, SUM(CMD_MONTANT) AS
CA_T4
      FROM   COMMANDE
      WHERE  CMD_DATE BETWEEN '2001-10-01' AND '2001-12-31'
      GROUP  BY CLI_ID) AS S
GROUP  BY S.CLI_ID
ORDER  BY S.CLI_ID
```

```
CLI_ID      CA_T1        CA_T2        CA_T3        CA_T4
----------- ------------ ------------ ------------ -------------
1           19328.37     0.0          0.0          3587.0
2           123185.22    82462.05     61458.0      962.28
```

C'est à dire le résultat attendu, et comme les NULL sont ignorés par les calculs d'agrégation, nous avons même de jolis zéros plus explicites que l'absence d'information !

Comme on le voit, la première requête donne les même résultats que la dernière... Seule la présentation à changée.... Mais cette dernière compte en fait 4 sous requêtes plus la requête regroupant les sous requêtes soit 5 SELECT. Quel est à votre avis la plus coûteuse des deux ?
Si vous utilisez un programme procédural pour faire cette présentation à partir de la première requête, vous divisez le temps de traitement par un facteur énorme. Je parierais entre 100 et 1000 fois plus rapide... Alors, à vous de juger !

Bémol

Toutefois il existe un bémol à une telle façon de faire. En effet, certains SGBDR (comme certains outils de développement, Delphi par exemple) proposent un ensemble de technique (ou de composant utilisant de telles techniques) dites ROLAP (Relational On Line Analytical Process).
Dans ce cas, de telle requêtes sont facile à écrire en utilisant un agrégat spécial (GROUPING) et les mots clef CUBE et ROLLUP. Mais là il ne s'agit plus de présentation mais d'analyse multidimensionnelle ! Et votre base de données doit avoir été conçue pour (étoile, flocon, nuage...).

10. Insertion multiple

Je n'arrive pas à insérer dans plusieurs tables à la fois ! Comment faire ?

Les ordres de manipulation des données (INSERT, UPDATE, DELETE) ne sont pas fait pour travailler sur plus d'une table. Pourquoi ? Parce que dans la très grande majorité des cas de relations entre entité, la cardinalité n'étant pas strictement 1:1, le SGBDR ne peut pas retrouver ses petits ! D'ailleurs, même ce cas de relation 1:1 est... impossible, sauf à se résumer à une seule et même table. En effet, considérons une table des hommes et une des femmes et la relation obligatoire mariage avec cardinalité 1:1. Cela signifie que tout homme est marié à une femme et une seule et que, de même, toute femme est marié à un homme et un seul. Exit donc les célibataires et vive le mariage dès la naissance...
Dès lors comment faire pour remplir notre table ? Si nous insérons un homme, la contrainte d'intégrité référentielle nous impose de préciser l'épouse de cet homme. D'un autre côté si nous tentons d'insérer une femme, alors il faut préciser son mari, ce qui supposerait que son mari soit déjà inséré dans la table des hommes, sans liens avec cette femme ce qui est impossible du fait de l'intégrité référentielle ! Donc, il est impossible de saisir une quelconque donné dans ce modèle.

Alors nous devons le casser et par exemple définir une relation en 0:1. Des lors nous pouvons insérer des données dans une des tables ou dans l'autre sans pour cela être freiné par la contrainte d'intégrité référentielle. Mais cette insertion ne peut se faire dans une seule table à la fois !

Le cas est encore plus frappant dans le cadre des relations 1:n ou n:m. En effet comment insérer dans une table une seule ligne et dans les autres plusieurs lignes en un seul ordre SQL ? Cela n'a pas de sens !

Alors **comment faire pour insérer dans plusieurs tables à la fois en étant sûr que toutes les insertions soient faites ?** La solution nous est fournie par SQL : cela s'apelle une transaction ! C'est pour cela qu'existe les transactions : jouer plusieurs ordres SQL, les valider tous d'un coup ou les annuler tous d'un coup.

A lire sur les transactions : les transactions