```
/*************************************************************************
====----    AUDIT QUALITÉ D'UNE BASE DE DONNÉES VIA 20 REQUÊTES SQL    ----====
*************************************************************************
* Fred. Brouard - http://sqlpro.developpez.com - www.sqlspot.com - 2015-10-05 *
*************************************************************************/

/*************************************************************************
* ATTENTION : lancez ces commandes en mode d'isolation READ UNCOMMITTED      *
*             lorsque c'(est la base de production que vous auditez.         *
*             En effet, certaines requêtes peuvent durer longtemps et bloquer *
*             des traitement de production                                   *
*************************************************************************/

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

-- Un éditeur de logiciel peut être mis face à ses reponsabilités sur le plan
juridique dans deux cas bien précis :
-- il y a malfaçon du logiciel dans ce cas, c'est la garantie dit de "vice
caché" qui s'applique.
-- il y a non respect des règles de l'art, dans ce cas une expertise est
nécessaire (au audit préalable souhaitable)
-- Le cas de malfaçon est caractérisé par des traitements faux ou des données
incorrectes
-- Dans une base de données, des traitements faux peuvent être constatées dans
les routines (procédures stockées, déclencheurs et fonctions utilisateur : UDF)
-- les données incorrectes peuvent être des données incompatibles avec la nature
de l'information, par exemple un prix négatif ou une date de début postérieure à
une date de fin
-- En ce qui concerne les règles de l'art, lire la définition dans le LAMY
juridique : http://www.lamy-expertise.fr/definition/regles-de-l-art.html
-- La jurisprudence en a consacré depuis longtemps la chose, comme par exemple
le 5/4/1993 (CA Toulouse) "Les règles de l'art désignent les procédés, savoir
faire ou tour de main, inhérents à la profession considérée".
-- Bien que les règles de l'art s'impose à défaut aux professionnel, la
vérification de leur application nécessite le pointr de vue d'un expert.
-- Lire à ce sujet, le paragraphe intitulé "2 - LA PORTEE DES REGLES DE L'ART"
dans : http://www.reds.msh-paris.fr/communication/textes/normtech.htm
-- Pour ce dernier point, voir :
http://www.marche-public.fr/Marches-publics/Definitions/Entrees/Regles-art.htm
-- En cas d'action en justice, le juge peut demander une contre expertise ou
explorer plus en profondeur via un "sapiter"
-- Définition du terme "sapiteur" :
http://www.dictionnaire-juridique.com/definition/sapiteur.php
-- On trouvera des analyses plus poussées au sujet des règles de l'art et leur
application en informatique notamment dans l'ouvrage :
-- "Aide mémoire de droit à l'usage des responsables informatique" :
http://www.amazon.fr/dp/2100570668/

-- Reste à savoir quelles sont les règles de l'art en matière de bases de
données relationnelles.
-- Pour les SGBDR comme Oracle, MS SQL Server, IBM DB2... Ces règles de l'art
reposent sur la théorie mathématique de l'algèbre relationnelle pour laquelle il
```

20 requetes pour auditer la structure de votre base de donnees.sql
existe différents documents formateurs,
-- reposant essentiellement sur la modélisation des données. Ce sont : les règles de Codd (Franck Edgar Codd est le créateur des bases de données relationnelles), le concepts de modélisation (travaux de Peter Chen) et les techniques qui vont avec : dépendances fonctionnelles et formes normales.
-- Pour la structure de la base, les règles de l'art sont formellement définies par le concept de relation et ce qu'il implique mais aussi de l'articulation des relations entre elles, reglée par ce que l'on appelle les "formes normales"
-- Au niveau de l'articulation des relations entre elles, il est nécessaire que la base soit au minimum en 3e forme normale, c'est à dire :
-- 1) Première forme normale : que les tables soient dotées d'une clef et que les colonnes comportent des données atomiques (non "sécables")
-- 2) Deuxième forme normale : qu'il n'existe pas de dépendance dans une même table entre deux colonnes non clef
-- 3) Troisième forme normale : qu'il n'existe pas de dépendance dans une même table entre une colonne clef et une colonne non clef
-- Au niveau de la forme des relations (c'est à dire des objets contenant les données) et des données qu'elles contiennent, cela doit obéir à un certain nombre de contraintes parmi lesquelles :
-- Des contraintes de domaines (plage de valeur des colonnes. Par exemple un pourcentage de réduction ne peut dépasser l'intervalle 1 à 100, un prix ne peut être négatif...)
-- Des contraintes de validation (par exemple qu'une date de debut et de fin de contrat soit chronologiquement ordonné...)
-- Des contraintes d'intégrité référentielles (qu'une table fille maintienne ses liens avec la table mère. Par exemple un client et ses factures).
-- Des contraintes plus généralistes, appelées "Assertions" qui portent sur plusieurs tables et qui découlent généralement des règles d'entreprise ou des contraintes du modèle (complétude, exclusivité, partition...) et qui sont réalisées via des déclencheurs.
-- Le but final d'un modèle de données repose essentiellement sur 3 règles d'or :
-- 1) pas de NULL (les informations à recueillir doivent être toutes connuées à un moment ou un autre)
-- 2) pas de redondance (une information doit apparaître à un seul endroit de la base)
-- 3) la mise à jour d'une information ne doit pas impacter plus d'une ligne

```
/*****************************************************
* DECELER DES TRAITEMENTS POTENTIELLEMENT INCORRECTS *
*****************************************************/

-- 1) Routines présentant un niveau d'isolation READ UNCOMMITTED
WITH
R AS
(SELECT ROUTINE_SCHEMA, ROUTINE_NAME, ROUTINE_TYPE,
        REPLACE(REPLACE(REPLACE(REPLACE(ROUTINE_DEFINITION, ' ' , ''), '  ',
'') , CHAR(10), ''), CHAR(13), '') AS ROUTINE_DEFINITION
 FROM   INFORMATION_SCHEMA.ROUTINES
 UNION ALL
 SELECT s.name, o.name, 'TRIGGER',
```

```
        REPLACE(REPLACE(REPLACE(REPLACE(m.definition, ' ' , ''), '  ', ''),
CHAR(10), ''), CHAR(13), '')
 FROM   sys.triggers AS d
        JOIN sys.sql_modules AS m
             ON d.object_id = m.object_id
        JOIN sys.objects AS o
             ON d.object_id = o.object_id
        JOIN sys.schemas AS s
             ON o.schema_id = s.schema_id
 UNION  ALL
 SELECT TABLE_SCHEMA, TABLE_NAME, 'VIEW',
        REPLACE(REPLACE(REPLACE(REPLACE(VIEW_DEFINITION, ' ' , ''), '  ', ''),
CHAR(10), ''), CHAR(13), '')
 FROM   INFORMATION_SCHEMA.VIEWS
)
SELECT ROUTINE_SCHEMA, ROUTINE_NAME
FROM   R
WHERE  ROUTINE_DEFINITION LIKE '%SETTRANSACTIONISOLATIONLEVELREADUNCOMMITTED%'
COLLATE French_CI_AS
OR     ROUTINE_DEFINITION LIKE '%(NOLOCK)%' COLLATE French_CI_AS;
--> Le niveau d'isolaltion READ UNCOMMITTED ou le "hint" NOLOCK entraine une
lecture inconsistante des données :
--  saut des lignes verrouillées et lecture multiples d'autres lignes.
--  À moins que les requêtes ne soient destinées à présenter des statistiques
grossières, ceci constitue une malfaçon du logiciel
--  car cela se traduit systématiquement par la manipulation de données fausses
et par conséquent des traitements erronés
--  Les données impactées sont hélas assez difficile à retrouver


-- 2) Déclencheurs potentiellement incomplet (présence de variables dans le
code)
WITH D AS
(
 SELECT o.name AS TRIGGER_NAME, s.name AS TABLE_SCHEMA, t.name AS TABLE_NAME,
COUNT(*) OVER() AS N,
        REPLACE(REPLACE(REPLACE(REPLACE(m.definition, ' ' , ''), '  ', ''),
CHAR(10), ''), CHAR(13), '') AS ROUTINE_DEFINITION
 FROM   sys.triggers AS d
        JOIN sys.sql_modules AS m
             ON d.object_id = m.object_id
        JOIN sys.objects AS o
             ON d.object_id = o.object_id
        JOIN sys.objects AS t
             ON d.parent_id = t.object_id
        JOIN sys.schemas AS s
             ON t.schema_id = s.schema_id
)
SELECT *
FROM   D
WHERE  ROUTINE_DEFINITION LIKE '%DECLARE%@%' COLLATE French_CI_AS;
-- Un déclencheur qui manipule les données transitoire via des variables ne peut
```

```
généralement traiter qu'une seule ligne
-- Or les déclencheur SQL Server sont ensemblistes et par conséquent ne se
déclenche qu'une seule fois quelque soit le nombre de lignes
-- de la commande SQL qui a généré l'événement (INSERT, UPDATE, DELETE).
-- Dès lors le cas est grand que le traitement porte sur des données fausses.
C'est généralement un signe de malfaçon du logiciel


-- 3) Taux de présence de contrainte d'intégrité référentielle (FOREIGN KEYs)
WITH T AS
(
SELECT CAST(COUNT(*) AS FLOAT) AS NOMBRE_TABLE,
       (SELECT COUNT(*)
        FROM   INFORMATION_SCHEMA.TABLE_CONSTRAINTS
        WHERE  CONSTRAINT_TYPE = 'FOREIGN KEY') AS NOMBRE_CONTRAINTE_IR
FROM   INFORMATION_SCHEMA.TABLES
WHERE  TABLE_TYPE = 'BASE TABLE'
)
SELECT NOMBRE_TABLE, NOMBRE_CONTRAINTE_IR,
       CAST((NOMBRE_CONTRAINTE_IR / NOMBRE_TABLE) * 100.0 AS DECIMAL(16,2)) AS
POURCENTAGE
FROM   T;
--> Un faible taux de présence de contrainte de type FOREIGN KEY (intégrité
référentielle) constitue une double problématique :
--  1) la présence potentielle de lignes orphelines conduisant à des traitement
erronés. C'est encore un cas de malfaçon du logiciel.
--  2) l'impossibilité pour l'optimiseur de simplifier certains plan d'exécution
de requête en se basant sur les contraintes FOREIGN KEY.
--  Il n'est pas possible pour une application de simuler une contrainte SQL
portant sur un ensemble de données sauf à ce que la base ne comporte jamais plus
d'un seul utilisateur simultané.

--> si présence des contraintes, vérifier qu'elles ont des index sous-jacents

-- 4) Absence d'utilisation de certains types de données
WITH T AS (
SELECT CASE
          WHEN DATA_TYPE IN ('char', 'varchar', 'nchar', 'nvarchar') THEN 'ALFA'
                  WHEN DATA_TYPE IN ('date', 'time', 'datetime', 'datetime2',
'datetimeoffset') THEN 'TEMPS'
                  WHEN DATA_TYPE IN ('float', 'real', 'decimal', 'numeric',
'int', 'bigint', 'smallint', 'tinyint') THEN 'NOMBRE'
                  ELSE 'AUTRES'
          END AS FAMILLE_TYPE, COUNT(*) AS NOMBRE
FROM   INFORMATION_SCHEMA.COLUMNS AS C
       INNER JOIN INFORMATION_SCHEMA.TABLES AS T
                ON C.TABLE_SCHEMA = T.TABLE_SCHEMA
                        AND C.TABLE_NAME = T.TABLE_NAME
WHERE  T.TABLE_TYPE = 'BASE TABLE'
GROUP  BY
       CASE
          WHEN DATA_TYPE IN ('char', 'varchar', 'nchar', 'nvarchar') THEN 'ALFA'
```

```
                WHEN DATA_TYPE IN ('date', 'time', 'datetime', 'datetime2',
'datetimeoffset') THEN 'TEMPS'
                WHEN DATA_TYPE IN ('float', 'real', 'decimal', 'numeric',
'int', 'bigint', 'smallint', 'tinyint') THEN 'NOMBRE'
                ELSE 'AUTRES'
          END)
SELECT FAMILLE_TYPE, NOMBRE, SUM(NOMBRE) OVER() AS TOTAL,
       CAST( 100.0 *NOMBRE / SUM(NOMBRE) OVER() AS DECIMAL(5,2)) AS POURCENT
FROM   T;
--> Dans les standards, une base de données normalement constituée devrait avoir
au moins 10% de chacun des types de données (exceptés "AUTRES").
--  En cas d'absence de l'un des types "ALFA", "NOMBRE", "TEMPS" il est probable
que l'on ait utilisé des types innapropriés, par exemple stocké des dates dans
des types littéraux,
--  ce qui entraines des problématique de performances et l'écritue de requêtes
complexes pour certains traitement.
--  Dans ce cas c'est un cas de malfaçon.

-- 4 bis) Utilisation abusive du type VARCHAR ou NVARCHAR
WITH T AS (
SELECT CASE
          WHEN DATA_TYPE IN ('char', 'nchar') THEN 'FIXE'
          WHEN DATA_TYPE IN ('varchar', 'nvarchar') THEN 'VARIABLE'
       END AS FORME,
       CASE
          WHEN DATA_TYPE IN ('char', 'varchar') THEN 'ASCII'
          WHEN DATA_TYPE IN ('nchar', 'nvarchar') THEN 'UNICODE'
       END AS ENCODAGE,
       CASE
           WHEN COALESCE(NULLIF(COALESCE(CHARACTER_MAXIMUM_LENGTH, -1), -1),
2147483647) <= 8 THEN '1 - PETIT'
           WHEN COALESCE(NULLIF(COALESCE(CHARACTER_MAXIMUM_LENGTH, -1), -1),
2147483647) <= 64 THEN '2 - MOYEN'
           WHEN COALESCE(NULLIF(COALESCE(CHARACTER_MAXIMUM_LENGTH, -1), -1),
2147483647) <= 256 THEN '3 - GRAND'
           WHEN COALESCE(NULLIF(COALESCE(CHARACTER_MAXIMUM_LENGTH, -1), -1),
2147483647) <= 1024 THEN '4 - ENORME'
           ELSE '9 - GIGANTESQUE'
       END AS CLASSE
FROM   INFORMATION_SCHEMA.COLUMNS AS C
       INNER JOIN INFORMATION_SCHEMA.TABLES AS T
                  ON C.TABLE_SCHEMA = T.TABLE_SCHEMA
                         AND C.TABLE_NAME = T.TABLE_NAME
WHERE  T.TABLE_TYPE = 'BASE TABLE'
AND    DATA_TYPE LIKE '%char'),
U AS (
SELECT DISTINCT ENCODAGE, FORME, CLASSE,
       COUNT(*) OVER(PARTITION BY FORME, ENCODAGE, CLASSE) AS NOMBRE,
       COUNT(*) OVER(PARTITION BY ENCODAGE) AS NB_ENCODAGE,
       COUNT(*) OVER(PARTITION BY FORME) AS NB_FORME,
       COUNT(*) OVER() AS NB_TOTAL
FROM   T)
```

```
       20 requetes pour auditer la structure de votre base de donnees.sql
SELECT FORME, ENCODAGE, CLASSE, NB_TOTAL,
       CAST(100.0 * NOMBRE / NB_TOTAL AS DECIMAL(5,2)) AS POURCENTAGE,
       CAST(100.0 * NB_ENCODAGE / NB_TOTAL  AS DECIMAL(5,2)) AS
POURCENT_ENCODAGE,
       CAST(100.0 * NB_FORME / NB_TOTAL  AS DECIMAL(5,2)) AS POURCENT_FORME
FROM   U
ORDER BY 1, 2, 3;
-- Le VARCHAR doit être bien employé sinon il cause des problèmes de
performances :
-- de petits [n]varchar (longueur inférieure à 8) sont absurde car le varchar
coute systématiquement 2 octets de plus pour le stockage (ces deux octets étant
utilisés pour stocker la longueur réelle de la chaine de caractères)
-- régulièrement mis à jour par UPDATE, le [n]varchar fragmente table et index
très rapidement et de manière élevée
-- l'espace mémoire occupé par un [n]varchar lors de certains calcul est
nécessairement réaligné à 1/2 de sa valeur maximale. En cas de surestimation de
la taille limite du [n]varchar, ceci obérera la mémoire au détriment du cache


-- 4 ter) types divergents pour colonnes de même nom
SELECT T.TABLE_SCHEMA, T.TABLE_NAME, C.COLUMN_NAME,
       CASE DATA_TYPE
           WHEN DATA_TYPE LIKE '%char' THEN '(' + CAST(CHARACTER_MAXIMUM_LENGTH
AS VARCHAR(16)) + ')'
           WHEN DATA_TYPE LIKE 'datetime%' THEN ''
           WHEN DATA_TYPE LIKE 'datetime%' THEN ''


FROM   INFORMATION_SCHEMA.COLUMNS AS C
       INNER JOIN  INFORMATION_SCHEMA.TABLES AS T
             ON C.TABLE_SCHEMA = T.TABLE_SCHEMA
             AND C.TABLE_NAME = T.TABLE_NAME
WHERE  TABLE_TYPE = 'BASE TABLE'



-- 5) Absence de contrainte CHECK sur chronologie lorsque les noms de colonnes
supposent des bornes ordonnées
WITH T AS (
SELECT 'debut' AS D, 'fin' AS F
UNION ALL
SELECT 'deb' AS D, 'fin' AS F
UNION ALL
SELECT 'avant' AS D, 'apres' AS F
UNION ALL
SELECT 'begin' AS D, 'end' AS F
UNION ALL
SELECT 'first' AS D, 'last' AS F
UNION ALL
SELECT 'premier' AS D, 'dernier' AS F
UNION ALL
SELECT 'demarrage' AS D, 'arret' AS F
```

```
UNION ALL
SELECT 'start' AS D, 'stop' AS F
UNION ALL
SELECT 'start' AS D, 'end' AS F
)
SELECT C1.TABLE_SCHEMA, C1.TABLE_NAME, C1.COLUMN_NAME AS COLONNE_DEBUT,
C2.COLUMN_NAME AS COLONNE_FIN,
       'SELECT * FROM ' + C1.TABLE_SCHEMA + '.' +  C1.TABLE_NAME + ' WHERE ' +
C1.COLUMN_NAME + ' > ' + C2.COLUMN_NAME + ';' AS REQUETE_TEST
FROM   INFORMATION_SCHEMA.COLUMNS AS C1
       INNER JOIN INFORMATION_SCHEMA.COLUMNS AS C2
                 ON C1.TABLE_SCHEMA = C2.TABLE_SCHEMA
                         AND C1.TABLE_NAME = C2.TABLE_NAME
                         AND C1.DATA_TYPE = C2.DATA_TYPE
       INNER JOIN INFORMATION_SCHEMA.TABLES AS TT
                 ON C1.TABLE_SCHEMA = TT.TABLE_SCHEMA
                         AND C1.TABLE_NAME = TT.TABLE_NAME
       CROSS JOIN T
WHERE  TT.TABLE_TYPE = 'BASE TABLE'
  AND  REPLACE(C1.COLUMN_NAME, D, '') =  REPLACE(C2.COLUMN_NAME, F, '') COLLATE
French_CI_AI
  AND  C1.COLUMN_NAME <> C2.COLUMN_NAME
  AND  NOT EXISTS(SELECT *
                  FROM   INFORMATION_SCHEMA.CHECK_CONSTRAINTS
                  WHERE  CHECK_CLAUSE LIKE '%' + C1.COLUMN_NAME + '%<%' +
C2.COLUMN_NAME + '%'  COLLATE French_CI_AI);
--> Lorsque des colonnes doivent contenir des données ordonnées (par exemple
chronologique) commme "début" et "fin",
--  l'absence de contrainte CHECK pour vérification d'ordonnancement permet la
saisie de données innopinées.
--  Rappelons que, de part le fonctionnement ensembliste des bases de données,
il n'est pas possible de simuler de telles contraintes au niveau applicatif
--  sauf à ce que l'application ne soit utilisée simultanément par un seul
utilisateur en lecture comme en écriture.
--  À l'évidence l'absence de telles contraintes indique une forte probabilité
de malfaçon de la base.
--  De surcroit, cela entraine de mauvaises performances pour les requêtes. En
effet SQL Server est doté d'un optimiseur sémantique (en plus de l'optimiseur
statistique),
--  et celui-ci se base sur les contraintes pour simplifier les plans
d'exécution des requêtes.


/*********************************************
* DECELER DES ERREURS DE STRUCTURE DE LA BASE *
*********************************************/

-- 6) Tables sans clef primaires
SELECT TABLE_SCHEMA, TABLE_NAME
FROM   INFORMATION_SCHEMA.TABLES
WHERE  TABLE_TYPE = 'BASE TABLE'
EXCEPT
```

```
       20 requetes pour auditer la structure de votre base de donnees.sql
SELECT TABLE_SCHEMA, TABLE_NAME
FROM   INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE  CONSTRAINT_TYPE = 'PRIMARY KEY';
--> Ces tables posent des problèmes de performance dans tous les SGBD
relationnel.
--  Par définition une table doit avoir une clef (cf. Algèbre Relationnelle).
--  Sans clef, le moteur relationnel aura de moins bonnes performances, y
compris pour les mises à jours (INSERT, UPDATE, DELETE).

-- 7) Groupe de colonnes répétitives
WITH T AS
(
SELECT DISTINCT C.TABLE_SCHEMA, C.TABLE_NAME, C.COLUMN_NAME, C.DATA_TYPE,
       REPLACE(
              REPLACE(
                     REPLACE(
                            REPLACE(
                                   REPLACE(
                                      REPLACE(
                                             REPLACE(
                                                    REPLACE(
                                                           REPLACE(

REPLACE(C.COLUMN_NAME
                                                                                  , '0',
''')
                                                                                     ,
'1', '')

     , '2', '')

     , '3', '')

           , '4', '')

                  , '5', '')

                     , '6', '')

                           , '7', '')

                                 , '8', '')

                                       , '9', '')
              AS COLUMN_NAME_RAW
FROM   INFORMATION_SCHEMA.COLUMNS AS C
       INNER JOIN INFORMATION_SCHEMA.TABLES AS T
             ON C.TABLE_NAME = T.TABLE_NAME
                              AND C.TABLE_SCHEMA = T.TABLE_SCHEMA
WHERE  T.TABLE_TYPE = 'BASE TABLE')
SELECT DISTINCT C1.TABLE_SCHEMA, C1.TABLE_NAME, C1.COLUMN_NAME_RAW,
C2.COLUMN_NAME,
```

```
       20 requetes pour auditer la structure de votre base de donnees.sql
       'Tableau de colonne' AS ANOMALIE
FROM   T AS C1
       INNER JOIN T AS C2
             ON C1.COLUMN_NAME_RAW = C2.COLUMN_NAME_RAW COLLATE French_CI_AI
                 AND C1.COLUMN_NAME <> C2.COLUMN_NAME COLLATE French_CI_AI
                                AND C1.TABLE_SCHEMA + C1.TABLE_NAME
                                    > C2.TABLE_SCHEMA + C2.TABLE_NAME
                 AND C1.DATA_TYPE = C2.DATA_TYPE
UNION ALL
SELECT DISTINCT C1.TABLE_SCHEMA, C1.TABLE_NAME, C1.COLUMN_NAME_RAW,
C2.COLUMN_NAME,
       'Famille de colonne' AS ANOMALIE
FROM   T AS C1
       INNER JOIN T AS C2
             ON C1.COLUMN_NAME LIKE C2.COLUMN_NAME +'%' COLLATE French_CI_AI
                 AND C1.COLUMN_NAME <> C2.COLUMN_NAME COLLATE French_CI_AI
                                AND C1.TABLE_SCHEMA = C2.TABLE_SCHEMA
                                AND C1.TABLE_NAME = C2.TABLE_NAME
                 AND C1.DATA_TYPE = C2.DATA_TYPE
ORDER BY 5 DESC, 1, 2, 3;
--> En régle général des colonnes dd'une même table dont les noms sont
répétitifs pour partie montre un défaut de modélisation tel qu'un viol de la
première forme normale par extension.
--  Ce peut être, un groupe de colonne dont les noms sont suffixés par un nombre
(Tableau de colonne). Exemple : Telephone1, Telephone2, Telephone3...
--  Ou encore, un groupe de colonne considérant un même sous ensemble (Famille
de colonne). Exemple : Telephone, TelephoneMobile, TelephonePro...
--  Dans les deux cas, ces informations auraient dû être stockées dans une autre
table.


-- 8) Tables obèses ayant plus de 20 colonnes
SELECT  T.TABLE_SCHEMA, T.TABLE_NAME, COUNT(*)
FROM    INFORMATION_SCHEMA.COLUMNS AS C
        JOIN INFORMATION_SCHEMA.TABLES AS T
             ON C.TABLE_SCHEMA = T.TABLE_SCHEMA
                AND C.TABLE_NAME = T.TABLE_NAME
WHERE   T.TABLE_TYPE = 'BASE TABLE'
GROUP  BY T.TABLE_SCHEMA, T.TABLE_NAME
HAVING COUNT(*) > 20
ORDER BY 3 DESC;
--> Une modélisation corecte conduit à de nombreuses petites tables généralement
rarement de plus de 20 colonnes.
--  Plus une table possède un grand nombre de colonne, plus les performances
seront mauvaises et cela pour de nombreuses raisons :
--  1) une table obèse contiendra généralement beucoup de NULL, c'est à dire des
abcsences de valeur dont il faut quand même prévoir l'espace de stockage pour
rien. Or plus le volume augmmente plus les performances baissent.
--  2) une table obèse est généralement plus utilisée qu'une table "mince". De
ce fait elle est plus souvent et plus longtemps verrouillée ce qui provoque plus
d'attente pour la moindre mise à jour et se traduit par de la contention, des
blocages, voire des verrous mortels.
```

```
--  3) une table obèse est difficilement optimisable. Le nombre d'index candidat
pour optimiser une talle table est un calcul d'arrangement mathématique du
nombre de colonne (factorielle). En pratique il est irréaliste et souvent
impossible d'optimiser une telle table.
--  En général, les tables obèses sont le fait d'un modèle de données non
respectueux des règles de l'art et en particulier le viol des 3 premières formes
normales


-- 9) Absence de contraintes de domaine ou de validation (CHECK)
WITH T AS
(
SELECT CAST(COUNT(*) AS FLOAT) AS NOMBRE_COLONNES,
       (SELECT COUNT(*)
        FROM   INFORMATION_SCHEMA.CHECK_CONSTRAINTS) AS NOMBRE_CONTRAINTES_CHECK
FROM   INFORMATION_SCHEMA.COLUMNS AS C
       JOIN INFORMATION_SCHEMA.TABLES AS T
            ON C.TABLE_SCHEMA = T.TABLE_SCHEMA
               AND C.TABLE_NAME = T.TABLE_NAME
WHERE  TABLE_TYPE = 'BASE TABLE'
)
SELECT NOMBRE_COLONNES, NOMBRE_CONTRAINTES_CHECK,
       CAST((NOMBRE_CONTRAINTES_CHECK / NOMBRE_COLONNES) * 100.0 AS
DECIMAL(16,2)) AS POURCENTAGE
FROM   T;
--> Sans contrainte de validation il est possible de sasiir des données
erronées.
--  Rappelons que, de part le fonctionnement ensembliste des bases de données,
il n'est pas possible de simuler de telles contraintes au niveau applicatif
--  sauf à ce que l'application ne soit utilisée simultanément par un seul
utilisateur en lecture comme en écriture.
--  À l'évidence l'absence de telles contraintes indique une forte probabilité
de malfaçon de la base.
--  De surcroit, cela entraine de mauvaises performances pour les requêtes. En
effet SQL Server est doté d'un optimiseur sémantique (en plus de l'optimiseur
statistique),
--  et celui-ci se base sur les contraintes pour simplifier les plans
d'exécution des requêtes.


-- 10) Table ayant des types de données obsolètes (text, ntext, image)
SELECT TABLE_SCHEMA, TABLE_NAME, COUNT(*) AS NOMBRE
FROM   INFORMATION_SCHEMA.COLUMNS
WHERE  DATA_TYPE IN ('image', 'text', 'ntext')
GROUP  BY TABLE_SCHEMA, TABLE_NAME;
--> Les types de données text, ntext, image sont considérés comme obsolète
depuis la version 2005 de SQL Server.
--  Microsoft recommande de les changer car ils pourront ne plus fonctionner
dans une future version
--  La présence de tels types de données montre à l'évidence une malfaçon du
logiciel
```

```
-- 10 bis) Tables ayant des types déconseillés (timestamp, datetime,
sql_variant, smalldatetime)
SELECT TABLE_SCHEMA, TABLE_NAME, COUNT(*) AS NOMBRE
FROM   INFORMATION_SCHEMA.COLUMNS
WHERE  DATA_TYPE IN ('timestamp', 'datetime', 'sql_variant', 'smalldatetime')
GROUP  BY TABLE_SCHEMA, TABLE_NAME;
--> Les types de données timestamp, datetime, sql_variant, smalldatetime sont
considérés comme déconseillés depuis les versions 2005 (timestamp, sql_variant)
et 2008 (datetime, smalldatetime) de SQL Server.
--  Microsoft conseille de les changer car ils pourront ne plus fonctionner dans
une future version


-- 10 ter : liste exhaustive
SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, DATA_TYPE
FROM   INFORMATION_SCHEMA.COLUMNS
WHERE  DATA_TYPE IN ('image', 'text', 'ntext', 'timestamp', 'datetime',
'sql_variant', 'smalldatetime');

/*********************************************
* DECELER DES PROBLEMES POTENTIELS DE SECURITE *
*********************************************/

-- 11 ) Routines présentant un potentiel d'injection de code
WITH
R AS
(SELECT ROUTINE_SCHEMA, ROUTINE_NAME, ROUTINE_TYPE,
        REPLACE(REPLACE(REPLACE(REPLACE(ROUTINE_DEFINITION, ' ' , ''), '  ',
''), CHAR(10), ''), CHAR(13), '') AS ROUTINE_DEFINITION
 FROM   INFORMATION_SCHEMA.ROUTINES
 WHERE  ROUTINE_TYPE = 'PROCEDURE'
 UNION ALL
 SELECT s.name, o.name, 'TRIGGER',
        REPLACE(REPLACE(REPLACE(REPLACE(m.definition, ' ' , ''), '  ', ''),
CHAR(10), ''), CHAR(13), '')
 FROM   sys.triggers AS d
        JOIN sys.sql_modules AS m
             ON d.object_id = m.object_id
        JOIN sys.objects AS o
             ON d.object_id = o.object_id
        JOIN sys.schemas AS s
             ON o.schema_id = s.schema_id)
SELECT ROUTINE_SCHEMA, ROUTINE_NAME
FROM   R
WHERE  ROUTINE_DEFINITION LIKE '%EXEC(@%)%' COLLATE French_CI_AS
OR     ROUTINE_DEFINITION LIKE '%EXEC(''%'')%' COLLATE French_CI_AS
OR     ROUTINE_DEFINITION LIKE '%sp_executesql%' COLLATE French_CI_AS;
--> L'utilisation du SQL dynamique (construction de requêtes par raboutage de
chaines de caractères) est sujet à des attaques par injection de code
--  Comme il n'est pas toujours possible de s'en passer du fait de la complexité
des requêtes et des performances attendues, il faut néanmoins se protéger par
```

```
       20 requetes pour auditer la structure de votre base de donnees.sql
des test préalables rarement mis en place en pratique.
--  Il est donc nécessaire de vérifier pour les routines listées qu'elle ne
devraient pas être sujettes à de telles attaques.


/************************************************
* DECELER DES PROBLEMES POTENTIELS DE PERFORMANCES *
************************************************/

-- 12) Tables sans index cluster
SELECT OBJECT_SCHEMA_NAME(object_id) AS TABLE_SCHEMA, OBJECT_NAME(object_id) AS
TABLE_NAME
FROM   sys.indexes
WHERE  index_id = 0;
--> SQL Server est spécialiement conçu pour fonctionner avec des tables
organisées en index CLUSTERED.
--  L'absence d'index CLUSTERED dans une table donne de moins bonne performances
en général


-- 13) Tables ayant des clefs primaires de plus de 3 colonnes
SELECT KCU.TABLE_SCHEMA, KCU.TABLE_NAME, COUNT(*)
FROM   INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS KCU
       JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS TC
                ON KCU.CONSTRAINT_SCHEMA = TC.CONSTRAINT_SCHEMA
                        AND KCU. CONSTRAINT_NAME = TC.CONSTRAINT_NAME
WHERE  TC.CONSTRAINT_TYPE = 'PRIMARY KEY'
GROUP  BY KCU.TABLE_SCHEMA, KCU.TABLE_NAME
HAVING COUNT(*) > 3
ORDER BY 3 DESC;
-- SQL Server dispose d'un optimiseur dont la majeure partie est basée sur des
statistiques.
-- Les statistiques d'optimisation collectées dans un index (et une clef
primaire est indexées) ne concerne que la première colonne.
-- Des statistiques complémentaire sont généralement disponible pour les autres
colonnes.
-- Cependant l'estimation de cardinalité d'une combinaison de colonne, nécessite
un calcul de statistiques corrélées qui perd rapidement de son acuité au fur et
à mesure de l'ajout des colonnes dans une clef.
-- Au dela de 3 colonnes dans une clef, la précision de cette estimation est
tellement flou que le plan d'exécution calculé peut devenir critique et en
particulier si la dispersion des valeurs de la première colonne est faible.
-- L'optimiseur est donc induit en erreur, les jointures sont mal évaluées et le
coût de l'opération de jointure devient critique.
-- Au final, les performances peuvent être catastrophiques (utilisation
d'algorithmes totalement contre performant)


 -- 14) Tables dont les lignes dépassent potentiellement la taille des pages ###
WITH TT AS (
-- prévoir les types 2008 DATE, DATETIME2 et
SELECT T.TABLE_SCHEMA, T.TABLE_NAME,
```

```
       SUM(COALESCE(NULLIF(CHARACTER_OCTET_LENGTH, -1), 0) +
       CASE WHEN DATA_TYPE = 'bigint' THEN 8.0
            WHEN DATA_TYPE = 'int' THEN 4.0
            WHEN DATA_TYPE = 'smallint' THEN 2.0
            WHEN DATA_TYPE = 'tinyint' THEN 1.0
            WHEN DATA_TYPE = 'uniqueidentifier' THEN 16.0
            WHEN DATA_TYPE IN ('numeric', 'decimal', 'money', 'smallmoney')
                 THEN CASE WHEN NUMERIC_PRECISION BETWEEN 1 AND 9 THEN 5.0
                           WHEN NUMERIC_PRECISION BETWEEN 10 AND 19 THEN 9.0
                           WHEN NUMERIC_PRECISION BETWEEN 20 AND 28 THEN 13.0
                           WHEN NUMERIC_PRECISION BETWEEN 29 AND 38 THEN 17.0
                      END
            WHEN DATA_TYPE = 'float'
                 THEN CASE WHEN NUMERIC_PRECISION BETWEEN 1 AND 24 THEN 4.0
                           WHEN NUMERIC_PRECISION BETWEEN 25 AND 53 THEN 8.0
                      END
            WHEN DATA_TYPE = 'real' THEN 4.0
            WHEN DATA_TYPE = 'date' THEN 3.0
                        WHEN DATA_TYPE = 'datetime' THEN 8.0
                        WHEN DATA_TYPE = 'datetime2'
                             THEN CASE WHEN DATETIME_PRECISION BETWEEN 0 AND 2
THEN 6.0
                           WHEN DATETIME_PRECISION BETWEEN 3 AND 4 THEN 7.0
                           WHEN DATETIME_PRECISION BETWEEN 5 AND 7 THEN 8.0
                                        END
                        WHEN DATA_TYPE = 'datetimeoffset' THEN 10.0
                        WHEN DATA_TYPE = 'time'
                             THEN CASE WHEN DATETIME_PRECISION BETWEEN 0 AND 2
THEN 3.0
                           WHEN DATETIME_PRECISION BETWEEN 3 AND 4 THEN 4.0
                           WHEN DATETIME_PRECISION BETWEEN 5 AND 7 THEN 5.0
                                        END
            WHEN DATA_TYPE = 'uniqueidentifier' THEN 16.0
            WHEN DATA_TYPE = 'smalldatetime' THEN 4.0
            WHEN DATA_TYPE = 'bit' THEN 0.125
            ELSE 0.0
       END) AS SIZE_O_MAX
FROM   INFORMATION_SCHEMA.TABLES AS T
       INNER JOIN INFORMATION_SCHEMA.COLUMNS AS C
             ON T.TABLE_SCHEMA = C.TABLE_SCHEMA
                AND T.TABLE_NAME = C.TABLE_NAME
WHERE  TABLE_TYPE = 'BASE TABLE'
GROUP  BY T.TABLE_SCHEMA, T.TABLE_NAME
)
SELECT *
FROM   TT
WHERE  SIZE_O_MAX > 8060
ORDER  BY SIZE_O_MAX DESC;
--> Les taille des pages (structure de donées basique stockant les ligens des
tables et index) de SQL Server est fixé à 8 Ko
--  et peut recevoir au plus 8060 octets de données. En principe, une ligne
devrait ne pas dépasser la taille d'une page,
```

```
       20 requetes pour auditer la structure de votre base de donnees.sql
--  sauf si elle contient une ou plusieurs colonnes de type "LOBs"
(VARBINARY(max), VARCHAR(max), NVARCHAR(max)).
--  Au delà de cette limite, la table présentera une fragmenttation irréfragable
et donc systématiquement des problèmes de performances.


-- 15) Tables ayant des clefs trop longues
WITH T AS (
-- prévoir les types 2008 DATE, DATETIME2 et
SELECT TC.TABLE_SCHEMA, TC.TABLE_NAME,
       SUM(
       CASE WHEN DATA_TYPE = 'bigint' THEN 8
            WHEN DATA_TYPE = 'int' THEN 4
            WHEN DATA_TYPE = 'smallint' THEN 2
            WHEN DATA_TYPE = 'tinyint' THEN 1
            WHEN DATA_TYPE IN ('numeric', 'decimal', 'money', 'smallmoney')
                 THEN CASE WHEN NUMERIC_PRECISION BETWEEN 1 AND 9 THEN 5
                           WHEN NUMERIC_PRECISION BETWEEN 10 AND 19 THEN 9
                           WHEN NUMERIC_PRECISION BETWEEN 20 AND 28 THEN 13
                           WHEN NUMERIC_PRECISION BETWEEN 29 AND 38 THEN 17
                      END
            WHEN DATA_TYPE = 'float'
                 THEN CASE WHEN NUMERIC_PRECISION BETWEEN 1 AND 24 THEN 4
                           WHEN NUMERIC_PRECISION BETWEEN 25 AND 53 THEN 8
                      END
            WHEN DATA_TYPE = 'real' THEN 4
            WHEN DATA_TYPE = 'date' THEN 3
                        WHEN DATA_TYPE = 'datetime' THEN 8
                        WHEN DATA_TYPE = 'datetime2'
                             THEN CASE WHEN DATETIME_PRECISION BETWEEN 0 AND 2
THEN 6
                                WHEN DATETIME_PRECISION BETWEEN 3 AND 4 THEN 7
                                WHEN DATETIME_PRECISION BETWEEN 5 AND 7 THEN 8
                                                END
                        WHEN DATA_TYPE = 'datetimeoffset' THEN 10
                        WHEN DATA_TYPE = 'time'
                             THEN CASE WHEN DATETIME_PRECISION BETWEEN 0 AND 2
THEN 3
                                WHEN DATETIME_PRECISION BETWEEN 3 AND 4 THEN 4
                                WHEN DATETIME_PRECISION BETWEEN 5 AND 7 THEN 5
                                                END
            WHEN DATA_TYPE = 'uniqueidentifier' THEN 16.0
            WHEN DATA_TYPE = 'smalldatetime' THEN 4
            WHEN DATA_TYPE IN ('varchar', 'nvarchar') THEN 2
            ELSE CHARACTER_OCTET_LENGTH
       END) AS SIZE_O_MIN,
       SUM(
       COALESCE(CHARACTER_OCTET_LENGTH,
       CASE WHEN DATA_TYPE = 'bigint' THEN 8
            WHEN DATA_TYPE = 'int' THEN 4
            WHEN DATA_TYPE = 'smallint' THEN 2
            WHEN DATA_TYPE = 'tinyint' THEN 1
```

```
            WHEN DATA_TYPE IN ('numeric', 'decimal', 'money', 'smallmoney')
                 THEN CASE WHEN NUMERIC_PRECISION BETWEEN 1 AND 9 THEN 5
                           WHEN NUMERIC_PRECISION BETWEEN 10 AND 19 THEN 9
                           WHEN NUMERIC_PRECISION BETWEEN 20 AND 28 THEN 13
                           WHEN NUMERIC_PRECISION BETWEEN 29 AND 38 THEN 17
                      END
            WHEN DATA_TYPE = 'float'
                 THEN CASE WHEN NUMERIC_PRECISION BETWEEN 1 AND 24 THEN 4
                           WHEN NUMERIC_PRECISION BETWEEN 25 AND 53 THEN 8
                      END
            WHEN DATA_TYPE = 'real' THEN 4
            WHEN DATA_TYPE = 'date' THEN 3
                        WHEN DATA_TYPE = 'datetime' THEN 8
                        WHEN DATA_TYPE = 'datetime2'
                             THEN CASE WHEN DATETIME_PRECISION BETWEEN 0 AND 2
THEN 6
                                WHEN DATETIME_PRECISION BETWEEN 3 AND 4 THEN 7
                                WHEN DATETIME_PRECISION BETWEEN 5 AND 7 THEN 8
                                                  END
                        WHEN DATA_TYPE = 'datetimeoffset' THEN 10
                        WHEN DATA_TYPE = 'time'
                             THEN CASE WHEN DATETIME_PRECISION BETWEEN 0 AND 2
THEN 3
                                WHEN DATETIME_PRECISION BETWEEN 3 AND 4 THEN 4
                                WHEN DATETIME_PRECISION BETWEEN 5 AND 7 THEN 5
                                                  END
            WHEN DATA_TYPE = 'uniqueidentifier' THEN 16.0
            WHEN DATA_TYPE = 'smalldatetime' THEN 4
            WHEN DATA_TYPE IN ('varchar', 'nvarchar') THEN 2
       END)) AS SIZE_O_MAX
FROM   INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS TC
       INNER JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS KCU
             ON TC.CONSTRAINT_SCHEMA = KCU.CONSTRAINT_SCHEMA
                AND TC.CONSTRAINT_NAME = KCU.CONSTRAINT_NAME
       INNER JOIN INFORMATION_SCHEMA.COLUMNS AS C
             ON TC.TABLE_SCHEMA = C.TABLE_SCHEMA
                AND TC.TABLE_NAME = C.TABLE_NAME
                AND KCU.COLUMN_NAME = C.COLUMN_NAME
WHERE   CONSTRAINT_TYPE = 'PRIMARY KEY'
GROUP  BY TC.TABLE_SCHEMA, TC.TABLE_NAME)
SELECT *
FROM   T
WHERE  SIZE_O_MAX > 8
ORDER  BY SIZE_O_MAX DESC;
--> Des clefs de longue taille necessitent de nombreuses lectures pour les CPU,
dès que ces clefs dépassant la taille du mot du processeur (8 cotets dans un OS
64 bits)
--  Or les clefs servent de recherche et de jointure pour une très grande
majorité de requêtes.
--  Micosoft recommabnde d'utiliser assez systématiquement des entiers (INT ou
BIGINT) auto incrémentés pour servir de clef.
--  Des clefs obèses provoque systématiquement des problèmes de performances.
```

```
-- 16) Manque d'index
WITH
T1 AS (
SELECT SUM(CASE WHEN index_id <= 1 THEN 1 ELSE 0 END) AS NOMBRE_TABLE,
       SUM(CASE WHEN index_id >  1 THEN 1 ELSE 0 END) AS NOMBRE_INDEX
FROM   sys.indexes),
T2 AS (
SELECT CAST(ROUND(SUM(au.used_pages) / 128.0, 0) AS BIGINT) AS BASE_MO,
                CAST(ROUND(SUM(
                   CASE
                            WHEN it.internal_type IN (202, 204, 211, 212, 213,
214, 215, 216)
                               THEN 0
                                 WHEN au.type <> 1
                                    THEN au.used_pages
                                 WHEN p.index_id < 2
                                    THEN au.data_pages
                                 ELSE 0
                          END) / 128.0, 0) AS BIGINT) AS TABLES_MO
FROM   sys.partitions AS p
       INNER JOIN sys.allocation_units au
             ON p.partition_id = au.container_id
                 LEFT OUTER JOIN sys.internal_tables AS it
                       ON p.object_id = it.object_id)
SELECT NOMBRE_TABLE, NOMBRE_INDEX,
       CAST(1.0 * NOMBRE_INDEX / NOMBRE_TABLE AS DECIMAL(16,2)) AS RATIO_INDEX,
       BASE_MO,
       TABLES_MO,
       BASE_MO - TABLES_MO AS INDEX_MO,
       CAST(100.0 * CAST( BASE_MO - TABLES_MO AS FLOAT)
            / BASE_MO AS DECIMAL(5,2)) AS INDEX_POURCENT
FROM   T1
       CROSS JOIN T2;
--> Les ratio constatés ordinairement sur le nombre d'index par table ou le
volume des index par rapport à la base tournent autour de 3 et 33%
--  Un éloignement considérable en plus ou en moins de ces valeurs est souvent
l'indication d'une base mal modélisée, sur ou sous indexées
--  et entraîne par conséquent des problèmes de performances : trop d'index
pénalise les mises à jours (INSERT, UDDATE, DELETE)
--  et pas assez d'index pénalise les lectures (SELECT), mais aussi certaines
écritures (UPDATE, DELETE).


-- 17) Index dupliqués / couvrant
WITH
-- sous requête CTE donnant les index avec leurs colonnes
T0 AS (SELECT ic.object_id, index_id, c.column_id, key_ordinal,
              CASE is_descending_key
                 WHEN '0' THEN 'ASC'
                 WHEN '1' THEN 'DESC'
```

```
              END AS sens, c.name AS column_name,
              ROW_NUMBER() OVER(PARTITION BY ic.object_id, index_id ORDER BY
key_ordinal DESC) AS N,
              is_included_column
       FROM   sys.index_columns AS ic
              INNER JOIN sys.columns AS c
                    ON ic.object_id = c.object_id
                       AND ic.column_id = c.column_id
       WHERE  key_ordinal > 0
         AND  index_id > 0),
-- sous requête CTE récursive composant les clefs des index sous forme
algébrique et littérale
T1 AS (SELECT object_id, index_id, column_id, key_ordinal, N,
              CASE WHEN is_included_column = 0 THEN CAST(column_name AS
VARCHAR(MAX)) + ' ' + sens ELSE '' END AS COMP_LITTERALE,
              CASE WHEN is_included_column = 0 THEN CAST(column_id AS
VARCHAR(MAX)) + SUBSTRING(sens, 1, 1) ELSE '' END AS COMP_MATH,
              MAX(N) OVER(PARTITION BY object_id, index_id) AS CMAX,
              CASE WHEN is_included_column = 1 THEN CAST(column_name AS
VARCHAR(MAX)) ELSE '' END AS COLONNES_INCLUSES
       FROM   T0
       WHERE  key_ordinal = 1
       UNION  ALL
       SELECT T0.object_id, T0.index_id, T0.column_id, T0.key_ordinal, T0.N,
              COMP_LITTERALE  +
              CASE WHEN is_included_column = 0 THEN  ', ' + CAST(T0.column_name
AS VARCHAR(MAX)) + ' ' + T0.sens ELSE '' END,
              COMP_MATH +
              CASE WHEN is_included_column = 0 THEN CAST(T0.column_id AS
VARCHAR(MAX)) + SUBSTRING(T0.sens, 1, 1) ELSE '' END,
              T1.CMAX, COLONNES_INCLUSES + CASE WHEN is_included_column = 1 THEN
', ' + CAST(column_name AS VARCHAR(MAX)) ELSE '' END
       FROM   T0
              INNER JOIN T1
                    ON T0.object_id = T1.object_id
                       AND T0.index_id = T1.index_id
                       AND T0.key_ordinal = T1.key_ordinal + 1),
-- sous requête CTE de dédoublonnage
T2 AS (SELECT object_id, index_id, COMP_LITTERALE, COMP_MATH, CMAX,
COLONNES_INCLUSES
       FROM   T1
       WHERE  N = 1),
-- sous requête sélectionnant les anomalies
T4 AS (SELECT T2.object_id, T2.index_id,
              T3.index_id AS index_id_anomalie,
              T2.COMP_LITTERALE AS CLEF_INDEX,
              T3.COMP_LITTERALE AS CLEF_INDEX_ANORMAL,
              T2.COLONNES_INCLUSES, T3.COLONNES_INCLUSES AS
COLONNES_INCLUSES_ANORMAL,
              CASE
                 WHEN T2.COMP_MATH = T3.COMP_MATH
                   THEN 'DOUBLONS'
```

```
                WHEN T2.COMP_MATH LIKE T3.COMP_MATH +'%'
                  THEN 'INCLUS'
             END AS ANOMALIE,
             ABS(T2.CMAX - T3.CMAX) AS DISTANCE
      FROM   T2
             INNER JOIN T2 AS T3
            ON T2.object_id = T3.object_id
               AND T2.index_id <> T3.index_id
               AND T2.COMP_MATH LIKE T3.COMP_MATH +'%')
-- Requête finale rajoutant les informations manquantes
SELECT T4.*,
       s.name +'.' + o.name AS NOM_TABLE,
       i1.name AS NOM_INDEX,
       i2.name AS NOM_INDEX_ANORMAL
       , i1.filter_definition AS FILTRE_INDEX
       , i2.filter_definition AS FILTRE_INDEX_ANORMAL
FROM   T4
       INNER JOIN sys.objects AS o
             ON T4.object_id = o.object_id
             INNER JOIN sys.schemas AS s
                   ON o.schema_id = s.schema_id
       INNER JOIN sys.indexes AS i1
             ON T4.object_id = i1.object_id
                AND T4.index_id = i1.index_id
       INNER JOIN sys.indexes AS i2
             ON T4.object_id = i2.object_id
                AND T4.index_id_anomalie = i2.index_id
WHERE  o."type" IN ('U', 'V')
  AND  CLEF_INDEX < CLEF_INDEX_ANORMAL
   OR  T4.index_id < index_id_anomalie
ORDER  BY ANOMALIE, NOM_TABLE, NOM_INDEX;
--> Cette requête recherche les index en double (DOUBLONS) ou les index inclus
l'un dans l'autre (INCLUS) sans tenir compte d'un éventuel filtre (clause WHERE)
ou de l'inclusion de colonne complémentaires (clause INCLUDE)
--  Un index redondant cause des problèmes de perfoemance et est à supprimer
systématiquement
--  Un index inclus doitn être étudié plus en détail pour juger de on
opportunité ou non. Dans ce dernier cas, le supprimer.


-- 18) Utilisation potentiellement abusive des curseurs
WITH T AS (
SELECT s.name AS OBJECT_SCHEMA, o.name AS OBJECT_NAME,
       o.type_desc  AS OBJECT_TYPE, m.definition,
           COUNT(*) OVER() AS NOMBRE_ROUTINES
FROM   sys.sql_modules AS m
       INNER JOIN sys.objects AS o
             ON m.object_id = o.object_id
       INNER JOIN sys.schemas AS s
                 On o.schema_id = s.schema_id
WHERE  o.type NOT IN ('IF', 'V'))
SELECT OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE,
```

```
       20 requetes pour auditer la structure de votre base de donnees.sql
       NOMBRE_ROUTINES, COUNT(*) OVER() AS NOMBRE_AVEC_CURSEUR,
           CAST(100.0 * COUNT(*) OVER() / NOMBRE_ROUTINES AS DECIMAL(5,2)) AS
POURCENTAGE
FROM   T
WHERE  definition LIKE '%CURSOR%OPEN%FETCH%' COLLATE French_CI_AS
ORDER BY 3, 1, 2;
--> Un nombre important de routine ayant des curseurs est généralement un signe
de mauvais développement.
--  Un curseur ne peut pas être optimisé contrairement à une requête.
--  Or en pratique un très grand nombre de curseurs peuvent être avantageusement
remplacés par des requêtes.
--  Un fort taux de présence de curseur est un signe de contre performances


--> 18 bis) curseurs abusant potentiellement des ressources
WITH T AS (
SELECT s.name AS OBJECT_SCHEMA, o.name AS OBJECT_NAME,
       o.type_desc  AS OBJECT_TYPE, m.definition,
           COUNT(*) OVER() AS NOMBRE_ROUTINES_CURSEUR
FROM   sys.sql_modules AS m
       INNER JOIN sys.objects AS o
             ON m.object_id = o.object_id
       INNER JOIN sys.schemas AS s
                 On o.schema_id = s.schema_id
WHERE  o.type NOT IN ('IF', 'V')
AND    definition LIKE '%CURSOR%OPEN%FETCH%' COLLATE French_CI_AS)
SELECT OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE,
       NOMBRE_ROUTINES_CURSEUR, COUNT(*) OVER() AS
NOMBRE_CURSEUR_CONTRE_PERFORMANT,
           CAST(100.0 * COUNT(*) OVER() / NOMBRE_ROUTINES_CURSEUR AS
DECIMAL(5,2)) AS POURCENTAGE
FROM   T
WHERE  NOT (definition LIKE '%CURSOR%LOCAL%FOR' COLLATE French_CI_AS OR
            definition LIKE '%CURSOR%FORWARD_ONLY%FOR' COLLATE French_CI_AS OR
            definition LIKE '%CURSOR%STATIC%FOR' COLLATE French_CI_AS OR
            definition LIKE '%CURSOR%READ_ONLY%FOR' COLLATE French_CI_AS)
ORDER BY 3, 1, 2                        ;
--> Les curseurs ouverts avec les valeurs par défaut de la norme SQL maximise
l'utilisation des ressources et travaillent sur des données "vivantes".
--  La plupart du temps, cela n'est pas nécessaire. En particulier on peut
mettre les options suivantes
--  LOCAL : le curseur n'est pas utilisable par d'autres procédure que celle qui
l'a ouvert. À défaut un curseur est visibale par toutes les procédures appelées.
--  FORWARD_ONLY : le curseur ne peut qu'avancer à la ligne suivante. À défaut,
le curseur peut avancer ou reculer de manière relative ou directe.
--  STATIC : le curseur effectue une copie de travail des données à manipuler. À
défaut, le curseur travaille sur des données vivantes et les bloque. Les
changements de données fait par les utilisateurs concurrents sont reflétés.
--  READ_ONLY : le curseur ne permet pas de modifier les données sous-jacentes
présente dans les tables. À défaut, onj peut faire un UPDATE des tables à
travers le curseur via le filtre WHERE CURRENT OF...
--  Ces 4 options devraient être systématiques sauf exception pour toute
```

```
ouverture de curseurs.
--  À défaut les ressources nécessaires pour la manipulation de curseurs ouverts
dans de telles mauvaises conditions lorsqu'elles ne sont pas nécessaires, posent
des problèmes de performances, induisent des problèmes de blocage
--  et peuvent condure a des traitements erronés (ouvertures de 2 curseurs ayant
le même nom dans 2 procédures différentes appelées l'une depuis l'autre).


-- 19) Utilisation abusive de tables temporaires ou de variable table
WITH T AS (
SELECT s.name AS OBJECT_SCHEMA, o.name AS OBJECT_NAME,
       o.type_desc  AS OBJECT_TYPE, m.definition,
           COUNT(*) OVER() AS NOMBRE_ROUTINES
FROM   sys.sql_modules AS m
       INNER JOIN sys.objects AS o
             ON m.object_id = o.object_id
       INNER JOIN sys.schemas AS s
                 On o.schema_id = s.schema_id
WHERE  o.type NOT IN ('IF', 'V'))
SELECT OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE,
       NOMBRE_ROUTINES, COUNT(*) OVER() AS NOMBRE_AVEC_TABLE_TEMPORAIRE,
           CAST(100.0 * COUNT(*) OVER() / NOMBRE_ROUTINES AS DECIMAL(5,2)) AS
POURCENTAGE
FROM   T
WHERE  REPLACE(REPLACE(REPLACE(REPLACE(definition, ' ' , ''), '  ', ''),
CHAR(10), ''), CHAR(13), '')
       LIKE '%CREATETABLE#%' COLLATE French_CI_AS
   OR   REPLACE(REPLACE(REPLACE(REPLACE(definition, ' ' , ''), '  ', ''),
CHAR(10), ''), CHAR(13), '')
       LIKE '%DECLARE@%TABLE%(' COLLATE French_CI_AS
ORDER BY 3, 1, 2;
--> Le recours aux tables temporaires (ou bien aux variables tables) devrait
être l'exception, SQL Server décidant lui même s'il est opportun, dans les
requêtes complexes,
--  d'utiliser ou non de telles tables afin de stocker des résultats
intermédaires s'ils s'avèrent trop volumineux pour figurer en mémoire.
--  Un fort taux d'utilisation de tables temporaires pose de multiples problèmes
de performances.


-- 20) Absence de SET NOCOUNT ON dans les routines
WITH T AS (
SELECT s.name AS OBJECT_SCHEMA, o.name AS OBJECT_NAME,
       o.type_desc  AS OBJECT_TYPE, m.definition,
           COUNT(*) OVER() AS NOMBRE_ROUTINES
FROM   sys.sql_modules AS m
       INNER JOIN sys.objects AS o
             ON m.object_id = o.object_id
       INNER JOIN sys.schemas AS s
                 On o.schema_id = s.schema_id
WHERE  o.type NOT IN ('IF', 'V', 'TF', 'FN', 'D', 'R'))
SELECT OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE,
```

```
       20 requetes pour auditer la structure de votre base de donnees.sql
       NOMBRE_ROUTINES, COUNT(*) OVER() AS NOMBRE_AVEC_TABLE_TEMPORAIRE,
           CAST(100.0 * COUNT(*) OVER() / NOMBRE_ROUTINES AS DECIMAL(5,2)) AS
POURCENTAGE
FROM   T
WHERE  REPLACE(REPLACE(REPLACE(REPLACE(definition, ' ' , ''), '  ', ''),
CHAR(10), ''), CHAR(13), '')
       NOT LIKE '%SETNOCOUNTON%' COLLATE French_CI_AS
ORDER BY 3, 1, 2;
--> Le paramétrage SET NOCOUNT ON évite d'envoyer en permancence et pour chaque
requête (SELECT, INSERT, UPDATE, DELETE) un message précisant le nombre de
lignes impactées.
--  Ce type de message est très rarement traité par les applications clientes
mais constitue des trames d'octets qui voyagent entre le serveur et les
applications clientes.
--  En pratique toutes les routines (procédures stockées et déclencheurs)
devraient commencer par cette directive (SET NOCOUNT ON;) afin d'interdire ces
messages intempestifs.
--  Notez que le template de créaton des routines proposée par SSMS (l'IHM
associées à SQL Server pour les développeurs) propose systématiquement cette
directive.
```