Calcul de médiane statistique avec SQL

Exemples avec MS SQL Server 2005

par Frédéric Brouard, alias SQLpro
MVP SQL Server
Expert langage SQL, SGBDR, modélisation de données

Auteur de :
- SQLpro http://sqlpro.developpez.com/
- "SQL", coll. Synthex, avec C. Soutou, Pearson Education 2005
- "SQL" coll. Développement, Campus Press 2001

Enseignant aux Arts & Métiers et à l'ISEN Toulon

Le calcul de la médiane statistique est un problème bien plus ardu qu'il n'y parait avec SQL et relève de nombreux pièges. Voici une étude complète sur le sujet afin de rendre compréhensible différents cas de figure.

1 - Définition de la MÉDIANE

Le terme de médiane vient du latin medius qui signifie "au milieu".

En Statistique, la médiane est la valeur qui permet de partager une série numérique — c'est à dire une suite de valeurs *ordonnées* — en deux parties égales en nombre d'éléments.

C'est une caractéristique particulière d'une série numérique, qu'il ne faut pas confondre avec la moyenne même si ces deux valeurs peuvent être voisines, voire égales dans le cas de distributions équilibrées.
Il est d'ailleurs important de faire la distinction entre la moyenne et la médiane. Par exemple le salaire médian est souvent plus proche de

l'impression qu'ont les individus de la notion de moyenne. En effet si nous analysons une population constituée par 9 salariés rétribués au salaire minimum garanti (smig) et un dixième gagnant 11 fois ce smig alors la moyenne sera de 2 fois le smig et la médiane égale au smig... En fait, comme les hauts salaires sont souvent très supérieurs aux bas salaires, le salaire moyen est généralement supérieur au salaire médian.

2 - La MÉDIANE et SQL

Très curieusement SQL n'a pas formalisé dans la norme le calcul de la médiane. Il y a là plusieurs pièges. Et je dois le dire, bon nombre de papiers sur le sujet et de recettes toutes faites en SQL pour calculer une médiane tombent dans le panneau...

Alors pour comprendre comment calculer sans faille une médiane, suivez moi dans les méandres de SQL, à la recherche de la formule magique !

Nous supposerons que le calcul de la médiane est faite sur des données numériques. Mais cela n'a pas grande importance. En effet la médiane d'une liste de prénoms est toute aussi calculable. Il suffit de considérer l'ordre que l'on veut, généralement alphabétique. Mais pour des données littérales la calcule de la médiane peut aussi être basé sur le nombre de caractères, le poids cumulés des caractères, etc...

NOTA : pour nos exemples nous avons travaillé avec une seule et même table dont la définition est la suivante :

```
CREATE TABLE T_STATISTIQUES_STT
(STT_ID       INT,
 STT_VALEUR   FLOAT NOT NULL);
```

Nous chercherons la médiane de la colonne STT_VALEUR.

3 - Première tentative

Supposons que le jeu des données à calculées sont toutes différentes (il n'y a pas de doublon) et en nombre impaires :

Voici le jeu de données que j'ai utilisé pour ce cas :

```
INSERT INTO T_STATISTIQUES_STT VALUES (1, 22.0);
INSERT INTO T_STATISTIQUES_STT VALUES (2, 27.5);
INSERT INTO T_STATISTIQUES_STT VALUES (3, 22.5);
INSERT INTO T_STATISTIQUES_STT VALUES (4, 24.0);
INSERT INTO T_STATISTIQUES_STT VALUES (5, 23.0);
```

Alors en ordonnant notre panel de 5 lignes par valeur croissante de la colonne STT_VALEUR, notre médiane saute aux yeux. C'est la valeur correspondante à la ligne 3, la ligne du milieu...

```
SELECT *
FROM   T_STATISTIQUES_STT
ORDER  BY STT_VALEUR;

STT_ID      STT_VALEUR
----------- -------------
1           22
3           22,5

5           23            <-- la médiane est là !
4           24

2           27,5
```

En fait, pour débusquer cette ligne, nous pourrions considérer la chose sous l'angle suivant :

1. Les lignes au dessus de la valeur
2. La ligne contenant la médiane
3. Les lignes en dessous de la valeur

Le point 1 comme le point 3 n'est pas difficile à réaliser... Cela peut s'écrire :

```
SELECT *
FROM   T_STATISTIQUES_STT SOU
WHERE  STT_VALEUR < [la valeur cherchée] :

SELECT *
FROM   T_STATISTIQUES_STT SUR
WHERE  STT_VALEUR > [la valeur cherchée] :
```

Mais il nous faudrait déjà connaître la valeur de cette médiane.
Néanmoins, constatons que ces deux requêtes donnent deux ensembles de n lignes, les n lignes au dessus de la médiane et les n lignes en dessous.

Si nous sommes capables d'éliminer ces 2 * n lignes, alors la médiane apparaît...

En fait ce n'est pas compliqué, il suffit d'équilibrer les deux requêtes, de la manière suivante :

```
ABS((SELECT COUNT(*)
     FROM   T_STATISTIQUES_STT SOU
     WHERE  STT_VALEUR < [la valeur cherchée]) -
    (SELECT COUNT(*)
     FROM   T_STATISTIQUES_STT SUR
     WHERE  STT_VALEUR > [la valeur cherchée])) <= 1
```

Il ne nous reste plus qu'à encapsuler cela dans une requête et le tour est joué !

```
SELECT *
FROM   T_STATISTIQUES_STT STT
WHERE  ABS((SELECT COUNT(*)
            FROM   T_STATISTIQUES_STT SOU
            WHERE  SOU.STT_VALEUR < STT.STT_VALEUR) -
           (SELECT COUNT(*)
            FROM   T_STATISTIQUES_STT SUR
            WHERE  SUR.STT_VALEUR > STT.STT_VALEUR)) <= 1
```

Solution 1

Et le résultat saute aux yeux :

```
STT_ID      STT_VALEUR
----------- ----------------------
5           23
```

Voici donc notre **première requête de calcul de la médiane**.

Que se passe t-il si nous ajoutons une ligne à cette ensemble ? La requête va-t-elle donner encore la bonne solution ?

Rajoutons la ligne suivante :

```
INSERT INTO T_STATISTIQUES_STT VALUES (6, 23.5)
```

Notre table contient maintenant :

```
STT_ID      STT_VALEUR
----------- ----------------------
1           22
3           22,5

5           23                    <-- médiane 1 \
                                                 Moyenne : 23,25
6           23,5                  <-- médiane 2 /

4           24
2           27,5
```

Et notre solution 1 donne :

```
STT_ID      STT_VALEUR
----------- ----------------------
5           23
6           23,5
```

4 - Seconde tentative

Supposons que le jeu des données à calculées sont toutes différentes (il n'y a pas de doublon) et en nombre paires, comme c'est le cas maintenant, il suffit de rajouter le calcul de la moyenne à cet édifice.

```
SELECT AVG(STT_VALEUR) AS MEDIANE
FROM   (SELECT STT_VALEUR
        FROM   T_STATISTIQUES_STT STT
        WHERE  ABS((SELECT COUNT(*)
                    FROM   T_STATISTIQUES_STT SOU
                    WHERE  SOU.STT_VALEUR < STT.STT_VALEUR) -
                   (SELECT COUNT(*)
                    FROM   T_STATISTIQUES_STT SUR
                    WHERE  SUR.STT_VALEUR > STT.STT_VALEUR)) <= 1 ) AS T
```

Solution 2

Et notre solution 2 donne :

```
MEDIANE
----------------------
23,25
```

Voici donc notre **seconde requête de calcul de la médiane**.

Mais nous avons à chaque fois interdit les doublons. Que se passe t-il si notre jeu de données est distendu vers le haut ou le bas. Bref si une grande accumulation de valeurs stagne dans la fourchette basse et qu'il n'y a que peu de valeur en fourchette haute ?
Essayons de rajouter quelques lignes à notre table :

```
INSERT INTO T_STATISTIQUES_STT VALUES (7, 22.0)
INSERT INTO T_STATISTIQUES_STT VALUES (8, 22.0)
INSERT INTO T_STATISTIQUES_STT VALUES (9, 22.0)
INSERT INTO T_STATISTIQUES_STT VALUES (10, 22.0)
INSERT INTO T_STATISTIQUES_STT VALUES (11, 22.0)
```

A nouveau la médiane est facile à déterminer de visu :

```
STT_ID      STT_VALEUR
----------- ----------------------
1           22
7           22
8           22
9           22
10          22

11          22                    <-- la médiane est là

3           22,5
5           23
6           23,5
4           24
2           27,5
```

Appliquons la requête de la solution 2 a ce nouveau panel de données :

```
MEDIANE
----------------------
NULL
```

Patatras… Aucune valeur ne sort ! Que s'est-il passé ?

5 - Troisième tentative

Constatons tout d'abord que l'ensemble inférieur est formé d'une seule valeur. Dès lors les opérations ensemblistes ne sont pas capables de faire la distinction entre les différentes valeurs. La limite entre les deux ensembles se positionne donc en dehors (ici dessous) de l'équilibre des lignes, et rien n'est retourné par la requête principale.

Ensuite, nous avons vu que suivant que le nombre de ligne était pair ou impair, la formulation de la requête est différente.
Une solution pour contourner le problème des lignes paires et impaires est de dédoubler le nombre des lignes. Cela ne change rien à la médiane de multiplier par deux les occurrences si ces nouvelles occurrences sont des lignes dupliquées, mais cela permet d'avoir l'assurance que l'on travaille sur un nombre de lignes paires.

Faisons cela à l'aide d'une vue :

```
CREATE VIEW V_STATISTIQUES_STT
AS
   SELECT *
   FROM   T_STATISTIQUES_STT
   UNION ALL
   SELECT *
   FROM   T_STATISTIQUES_STT
```

Dès lors on peut exprimer notre précédente requête par :

```
SELECT STT_VALEUR
FROM   V_STATISTIQUES_STT STT
WHERE  (SELECT COUNT(*)
        FROM   T_STATISTIQUES_STT) <=
               (SELECT COUNT(*)
                FROM   V_STATISTIQUES_STT AS SOU
                WHERE  SOU.STT_VALEUR >= STT.STT_VALEUR)
   AND (SELECT COUNT(*)
        FROM   T_STATISTIQUES_STT) <=
              (SELECT COUNT(*)
               FROM   V_STATISTIQUES_STT AS SOU
               WHERE  SOU.STT_VALEUR <= STT.STT_VALEUR)
```

Appliqué à notre panel, cette solution donne :

```
STT_VALEUR
----------------------
22
22
22
22
22
22
```

```
22
22
22
22
22
22
```

Auquel il suffit de demander un calcul de moyenne pour trouver la solution. Quelques petits malins m'aurait dit : « mais non, il suffit de faire un distinct et le tour est joué »... je vous laisse deviner pourquoi cela risque de ne pas marcher !

```
SELECT AVG(STT_VALEUR) AS MEDIANE
FROM   (SELECT STT_VALEUR
        FROM   V_STATISTIQUES_STT STT
        WHERE  (SELECT COUNT(*)
                FROM   T_STATISTIQUES_STT)
                       <= (SELECT COUNT(*)
                           FROM   V_STATISTIQUES_STT AS SOU
                           WHERE  SOU.STT_VALEUR <= STT.STT_VALEUR)
           AND (SELECT COUNT(*)
                FROM   T_STATISTIQUES_STT)
                       <= (SELECT COUNT(*)
                           FROM   V_STATISTIQUES_STT AS SUR
                           WHERE  SUR.STT_VALEUR >= STT.STT_VALEUR) ) AS T
```

Solution 3

Qui donne :

```
MEDIANE
----------------------
22
```

Cette requête est-elle parfaite ?
Ajoutons donc une dernière ligne :

```
INSERT INTO T_STATISTIQUES_STT VALUES (12, 22.5)
```

Que donne la requête des différentes solutions ?

Requête solution 1 :

```
STT_ID      STT_VALEUR
----------- ----------------------
```

Requête solution 2 :

```
MEDIANE
----------------------
NULL
```

Requête solution 3 :

```
MEDIANE
----------------------
22,125
```

Et la solution 3 n'est même pas bonne… En effet, vérifions à l'œil :

```
STT_ID      STT_VALEUR
----------- --------------------
1           22
7           22
8           22
9           22
10          22

11          22                  <-- médiane 1 \
                                                Moyenne : 22,25
12          22,5                <-- médiane 2 /

3           22,5
5           23
6           23,5
4           24
2           27,5
```

Simplement parce que le dédoublement des valeurs peut conduire à des occurrences quadruples ou plus encore des valeurs. Mais par définition la médiane dans ce cas étant la moyenne des deux valeurs, il suffit d'ajouter un DISTINCT dans le calcul de la moyenne !

6 - Quatrième tentative (la bonne !)

Finalement notre requête pour exprimer la médiane est en définitive :

```
SELECT AVG(DISTINCT STT_VALEUR) AS MEDIANE
FROM   (SELECT STT_VALEUR
        FROM   V_STATISTIQUES_STT STT
        WHERE  (SELECT COUNT(*)
                FROM   T_STATISTIQUES_STT) <=
                       (SELECT COUNT(*)
                        FROM   V_STATISTIQUES_STT AS SOU
                        WHERE  SOU.STT_VALEUR <= STT.STT_VALEUR)
           AND (SELECT COUNT(*)
                FROM   T_STATISTIQUES_STT) <=
                       (SELECT COUNT(*)
                        FROM   V_STATISTIQUES_STT AS SUR
                        WHERE  SUR.STT_VALEUR >= STT.STT_VALEUR) ) AS T
```

Solution 4

Qui peut s'exprimer sans la vue :

```
SELECT AVG(DISTINCT STT_VALEUR) AS MEDIANE
FROM   (SELECT STT_VALEUR
        FROM   (SELECT *
                FROM   T_STATISTIQUES_STT
                UNION  ALL
                SELECT *
                FROM   T_STATISTIQUES_STT) STT
        WHERE  (SELECT COUNT(*)
                FROM   T_STATISTIQUES_STT)
                    <= (SELECT COUNT(*)
```

```
                        FROM   (SELECT *
                                FROM   T_STATISTIQUES_STT
                                UNION  ALL
                                SELECT *
                                FROM   T_STATISTIQUES_STT) AS SOU
                        WHERE  SOU.STT_VALEUR <= STT.STT_VALEUR)
          AND (SELECT COUNT(*)
               FROM   T_STATISTIQUES_STT)
                   <= (SELECT COUNT(*)
                       FROM   (SELECT *
                               FROM   T_STATISTIQUES_STT
                               UNION  ALL
                               SELECT *
                               FROM   T_STATISTIQUES_STT) AS SUR
                       WHERE  SUR.STT_VALEUR >= STT.STT_VALEUR) ) AS T
```

Ou via les expressions de table, par :

```
WITH
T_DOUBLE AS
(SELECT *
 FROM   T_STATISTIQUES_STT
 UNION  ALL
 SELECT *
 FROM   T_STATISTIQUES_STT),
T_COUNT AS
(SELECT COUNT(*) AS N
 FROM   T_STATISTIQUES_STT)
SELECT AVG(DISTINCT STT_VALEUR) AS MEDIANE
FROM   (SELECT STT_VALEUR
        FROM   (SELECT *
                FROM   T_DOUBLE) STT
        WHERE  (SELECT *
                FROM   T_COUNT)
                       <= (SELECT COUNT(*)
                           FROM   (SELECT *
                                   FROM   T_DOUBLE) AS SOU
                           WHERE  SOU.STT_VALEUR <= STT.STT_VALEUR)
           AND (SELECT *
                FROM   T_COUNT)
                       <= (SELECT COUNT(*)
                           FROM   (SELECT *
                                   FROM   T_DOUBLE) AS SUR
                           WHERE  SUR.STT_VALEUR >= STT.STT_VALEUR) ) AS T
```

Nous appellerons cette solution **la solution de Date**, car c'est Chris Date qui en a donné la première fois l'expression.

7 - Autres formulations

Chris Date et Joe Celko se sont bien battus par papiers interposés pour trouver les failles de ces différentes problématiques de l'expression de la médiane. Ils en ont tirés différentes expressions, dont celle que je viens de vous donner est la première version corrigés de la médiane de Date.

Joe Celko propose de calculer le nombre de lignes à retourner par les deux sous ensembles à l'aide de la demie valuation du nombre total des lignes. La, requête est intéressante à décomposer :

1. les valeurs au dessus sont exprimées par :

```
SELECT ST1.STT_ID, ST1.STT_VALEUR
FROM   T_STATISTIQUES_STT AS ST1
       INNER JOIN T_STATISTIQUES_STT AS ST2
             ON ST1.STT_VALEUR <= ST2.STT_VALEUR
GROUP  BY ST1.STT_ID, ST1.STT_VALEUR
HAVING COUNT(*) <= (SELECT CEILING(COUNT(*) / 2.0)
                    FROM T_STATISTIQUES_STT)
```

```
STT_ID      STT_VALEUR
----------- ----------------
3           22.5
12          22.5
5           23.0
6           23.5
4           24.0
2           27.5
```

2. les valeurs en dessous sont exprimées par :

```
SELECT ST1.STT_ID, ST1.STT_VALEUR
FROM   T_STATISTIQUES_STT AS ST1
       INNER JOIN T_STATISTIQUES_STT AS ST2
             ON ST1.STT_VALEUR >= ST2.STT_VALEUR
GROUP  BY ST1.STT_ID, ST1.STT_VALEUR
HAVING COUNT(*) <= (SELECT CEILING(COUNT(*) / 2.0)
                    FROM T_STATISTIQUES_STT)
```

```
STT_ID      STT_VALEUR
----------- -----------------
1           22.0
7           22.0
8           22.0
9           22.0
10          22.0
11          22.0
```

3. Il ne suffit plus que de prendre le minimum des valeurs au dessus et le maximum des valeurs en dessous pour obtenir notre solution en une ou deux lignes :

```
SELECT MIN(STT_VALEUR) AS STT_VALEUR
/* valeurs au dessus */
FROM   (SELECT ST1.STT_ID, ST1.STT_VALEUR
        FROM   T_STATISTIQUES_STT AS ST1
               INNER JOIN T_STATISTIQUES_STT AS ST2
                     ON ST1.STT_VALEUR <= ST2.STT_VALEUR
        GROUP  BY ST1.STT_ID, ST1.STT_VALEUR
        HAVING COUNT(*) <= (SELECT CEILING(COUNT(*) / 2.0)
                            FROM T_STATISTIQUES_STT) ) SUR
UNION  ALL
```

```
SELECT MAX(STT_VALEUR) AS STT_VALEUR
/*  valeurs en dessous */
FROM   (SELECT ST1.STT_ID, ST1.STT_VALEUR
        FROM   T_STATISTIQUES_STT AS ST1
               INNER JOIN T_STATISTIQUES_STT AS ST2
                     ON ST1.STT_VALEUR >= ST2.STT_VALEUR
        GROUP  BY ST1.STT_ID, ST1.STT_VALEUR
        HAVING COUNT(*) <= (SELECT CEILING(COUNT(*) / 2.0)
                            FROM T_STATISTIQUES_STT) ) SOU
```

Qui donne :

```
STT_VALEUR
----------------------
22.5
22.0
```

4. Nous savons désormais quoi faire, et la requête globale s'exprime de la sorte :

```
SELECT AVG(STT_VALEUR) AS MEDIANE
FROM (SELECT MIN(STT_VALEUR) AS STT_VALEUR
      -- valeurs au dessus
      FROM   (SELECT ST1.STT_ID, ST1.STT_VALEUR
              FROM   T_STATISTIQUES_STT AS ST1
                     INNER JOIN T_STATISTIQUES_STT AS ST2
                           ON ST1.STT_VALEUR <= ST2.STT_VALEUR
              GROUP  BY ST1.STT_ID, ST1.STT_VALEUR
              HAVING COUNT(*) <= (SELECT CEILING(COUNT(*) / 2.0)
                                  FROM T_STATISTIQUES_STT) ) SUR
      UNION
      SELECT MAX(STT_VALEUR) AS STT_VALEUR
      -- valeurs en dessous
      FROM   (SELECT ST1.STT_ID, ST1.STT_VALEUR
              FROM   T_STATISTIQUES_STT AS ST1
                     INNER JOIN T_STATISTIQUES_STT AS ST2
                           ON ST1.STT_VALEUR >= ST2.STT_VALEUR
              GROUP  BY ST1.STT_ID, ST1.STT_VALEUR
              HAVING COUNT(*) <= (SELECT CEILING(COUNT(*) / 2.0)
                                  FROM T_STATISTIQUES_STT) ) SOU ) T
```

Nous appellerons donc cette nouvelle solution **la solution de Date & Celko**.

Celko nous a donné une variante qui est la suivante :

Formulation avec des vues :

```
CREATE VIEW V_SUR
AS
SELECT T1.STT_ID, T1.STT_VALEUR
FROM   T_STATISTIQUES_STT AS T1
       INNER JOIN T_STATISTIQUES_STT AS T2
             ON T1.STT_VALEUR <= T2.STT_VALEUR
GROUP  BY T1.STT_ID, T1.STT_VALEUR
HAVING COUNT(*) <= (SELECT CEILING(COUNT(*) / 2.0)
                    FROM   T_STATISTIQUES_STT);
```

```
CREATE VIEW V_SOUS
AS
SELECT T1.STT_ID, T1.STT_VALEUR
FROM   T_STATISTIQUES_STT AS T1
       INNER JOIN T_STATISTIQUES_STT AS T2
             ON T1.STT_VALEUR >= T2.STT_VALEUR
GROUP  BY T1.STT_ID, T1.STT_VALEUR
HAVING COUNT(*) <= (SELECT CEILING(COUNT(*) / 2.0)
                    FROM   T_STATISTIQUES_STT);

CREATE VIEW V_MEDIAN
AS
SELECT DISTINCT STT_VALEUR
FROM   T_STATISTIQUES_STT
WHERE  STT_VALEUR = (SELECT MAX(STT_VALEUR)
                     FROM   V_SOUS)
UNION ALL
SELECT DISTINCT STT_VALEUR
FROM   T_STATISTIQUES_STT
WHERE  STT_VALEUR = (SELECT MIN(STT_VALEUR)
                     FROM   V_SUR);

SELECT AVG(STT_VALEUR) AS MEDIANE
FROM   V_MEDIAN
```

La même avec des tables dérivées (sous requeêtes dans la clause FROM)

```
SELECT AVG(STT_VALEUR) AS MEDIANE
FROM   (SELECT DISTINCT STT_VALEUR
        FROM   T_STATISTIQUES_STT
        WHERE  STT_VALEUR =
           (SELECT MAX(STT_VALEUR)
            FROM   (SELECT T1.STT_ID, T1.STT_VALEUR
                    FROM   T_STATISTIQUES_STT AS T1
                           INNER JOIN T_STATISTIQUES_STT AS T2
                                 ON T1.STT_VALEUR >= T2.STT_VALEUR
                    GROUP  BY T1.STT_ID, T1.STT_VALEUR
                    HAVING COUNT(*) <=
                       (SELECT CEILING(COUNT(*) / 2.0)
                        FROM   T_STATISTIQUES_STT ) ) AS TX )
        UNION ALL
        SELECT DISTINCT STT_VALEUR
        FROM   T_STATISTIQUES_STT
        WHERE  STT_VALEUR =
           (SELECT MIN(STT_VALEUR)
            FROM   (SELECT T1.STT_ID, T1.STT_VALEUR
                    FROM   T_STATISTIQUES_STT AS T1
                           INNER JOIN T_STATISTIQUES_STT AS T2
                                 ON T1.STT_VALEUR <= T2.STT_VALEUR
                    GROUP  BY T1.STT_ID, T1.STT_VALEUR
                    HAVING COUNT(*) <=
                       (SELECT CEILING(COUNT(*) / 2.0)
                        FROM   T_STATISTIQUES_STT ) ) AS TY ) ) AS TZ
```

Nous appellerons cette variante **Celko 1**.

Dans son intéressant ouvrage « SQL Cookbook » Anthony Molinaro nous donne une autre expression que j'ai traduite pour notre exemple :

```
SELECT AVG(STT_VALEUR)
FROM (SELECT E.STT_VALEUR
      FROM   T_STATISTIQUES_STT E
             CROSS JOIN T_STATISTIQUES_STT D
      GROUP BY E.STT_VALEUR
      HAVING SUM(CASE
                   WHEN E.STT_VALEUR = D.STT_VALEUR THEN 1
                   ELSE 0
                 END) >= ABS(SUM(SIGN(E.STT_VALEUR - D.STT_VALEUR)))
     ) AS T
```

Elle est en fait inspirée des travaux de Abromovich, Alexandrova et Birger (1993-1994) et remaniée par Celko.
Nous l'appellerons la solution **Molinaro**.

D'autres formulations ne sont pas aptes à traiter notre cas. Parmi celles-ci notons la 2eme formule de Date et celle de Murchison…

Date propose une simplification de la formule classique :

```
SELECT AVG(DISTINCT STT_VALEUR) AS MEDIANE
FROM   T_STATISTIQUES_STT
WHERE  STT_VALEUR IN
          (SELECT MIN(STT_VALEUR)
           FROM   T_STATISTIQUES_STT
           WHERE  STT_VALEUR IN
              (SELECT T2.STT_VALEUR
               FROM   T_STATISTIQUES_STT AS T1
                      INNER JOIN T_STATISTIQUES_STT AS T2
                            ON T2.STT_VALEUR <= T1.STT_VALEUR
               GROUP  BY T2.STT_VALEUR
               HAVING COUNT(*) <= (SELECT CEILING(COUNT(*) / 2.0)
                                   FROM   T_STATISTIQUES_STT))
           UNION
           SELECT MAX(STT_VALEUR)
           FROM   T_STATISTIQUES_STT
           WHERE  STT_VALEUR IN
              (SELECT T2.STT_VALEUR
               FROM   T_STATISTIQUES_STT AS T1
                      INNER JOIN T_STATISTIQUES_STT AS T2
                            ON T2.STT_VALEUR >= T1.STT_VALEUR
               GROUP  BY T2.STT_VALEUR
               HAVING COUNT(*) <= (SELECT CEILING(COUNT(*) / 2.0)
                                   FROM   T_STATISTIQUES_STT)))
```

Cette requête donne 23 comme résultat au lieu de 22,25. Il faut donc l'éviter car elle ne marche qua dans certains cas grande dispersion, pas de NULL).

La requête de Murchison repose sur une bonne idée : concaténer la clef de la table (à supposer que l'on en ait une) avec la colonne contenant la valeur médiane à calculer. Ainsi toutes les lignes seront distinctes.

```
SELECT AVG(STT_VALEUR)
FROM   T_STATISTIQUES_STT AS T1
WHERE  EXISTS(SELECT COUNT(*)
              FROM   T_STATISTIQUES_STT AS T2
              WHERE  CAST(STT_VALEUR AS CHAR(32))
```

```
                    + CAST(T2.STT_ID AS CHAR(32)) >=
                    CAST(STT_VALEUR AS CHAR(32))
                    + CAST(T1.STT_ID AS CHAR(32))
             HAVING COUNT(*) = (SELECT FLOOR(COUNT(*) / 2.0 )
                                FROM   T_STATISTIQUES_STT)
                OR  COUNT(*) = (SELECT CEILING(COUNT(*) / 2.0)
                                FROM   T_STATISTIQUES_STT))
-- notez que vous pouvez remplacer la clause HAVING
-- par la formulation suivante :
/*
             HAVING COUNT(*) = (SELECT (COUNT(*) + 1 ) / 2.0
                                FROM   T_STATISTIQUES_STT)
                OR  COUNT(*) = (SELECT (COUNT(*) / 2.0) + 1
                                FROM   T_STATISTIQUES_STT))
*/
```

Malheureusement elle ne donne pas non plus le bon résultat dans notre cas et sort une médiane à 24 ou 22,5 suivant la façon dont est écrite la clause HAVING !

Celko à proposé une méthode intéressante réalisée avec des tables de travail. La première étape consiste à créer une table de travail ayant toutes les occurrences de valeur et le nombre de fois ou elles apparaissent. Dans notre cas cela donne :

```
SELECT STT_VALEUR, COUNT(*) AS NOMBRE
FROM   T_STATISTIQUES_STT
GROUP  BY STT_VALEUR

STT_VALEUR           NOMBRE
-------------------- -----------
22                   6
22,5                 2
23                   1
23,5                 1
24                   1
27,5                 1
```

La seconde étape consiste à créer une seconde table de travail calqué sur la première avec 2 colonnes supplémentaires de recoupement permettant de savoir dans quelle partie de la série on se trouve.
Notez que pour mieux voir la progression de la chose, j'ai préféré utiliser une CTE au lieu d'une vue…

```
WITH VALEURS
AS
(
SELECT STT_VALEUR, COUNT(*) AS NOMBRE
FROM   T_STATISTIQUES_STT
GROUP  BY STT_VALEUR
)
SELECT T2.STT_VALEUR, T2.NOMBRE,
       SUM(T1.NOMBRE) - T2.NOMBRE AS PRE,
       SUM(T1.NOMBRE) AS CUMUL
FROM   VALEURS AS T1
       INNER JOIN VALEURS AS T2
             ON T1.STT_VALEUR <= T2.STT_VALEUR
GROUP  BY T2.STT_VALEUR , T2.NOMBRE
```

```
STT_VALEUR           NOMBRE      PRE         CUMUL
-------------------- ----------- ----------- -----------
22                   6           0           6
22,5                 2           6           8
23                   1           8           9
23,5                 1           9           10
24                   1           10          11
27,5                 1           11          12
```

Dès lors la solution est simple : avec le nombre de ligne *n* divisé par deux, soit la médiane est située entre le précompte et le cumul, soit la médiane est située sur le précompte d'une ligne et le cumul d'une autre. Ce qui se formule ainsi :

```
WITH VALEURS
AS
(
SELECT STT_VALEUR, COUNT(*) AS NOMBRE
FROM   T_STATISTIQUES_STT
GROUP  BY STT_VALEUR
),
COMPTE
AS
(
SELECT T2.STT_VALEUR, T2.NOMBRE,
       SUM(T1.NOMBRE) - T2.NOMBRE AS PRE,
       SUM(T1.NOMBRE) AS POST
FROM   VALEURS AS T1
       INNER JOIN VALEURS AS T2
             ON T1.STT_VALEUR <= T2.STT_VALEUR
GROUP  BY T2.STT_VALEUR , T2.NOMBRE
)
SELECT AVG(STT_VALEUR) AS MEDIANE
FROM   COMPTE C
WHERE  (    C.POST > (SELECT MAX(POST) / 2.0 FROM COMPTE)
        AND C.PRE  < (SELECT MAX(POST) / 2.0 FROM COMPTE))
   OR       C.POST = (SELECT MAX(POST) / 2.0 FROM COMPTE)
   OR       C.PRE  = (SELECT MAX(POST) / 2.0 FROM COMPTE)
```

Et donne un parfait bon résultat !

Nous appellerons cette requête **Celko 2**.

8 - La médiane en SQL:1999

La présence des fonctions de fenêtrage permet une écriture plus rapide du calcul de la médiane en SQL.
Rappelons que les fonctions de fenêtrage sont des fonctions qui s'appliquent aux lignes du résultat afin des les enrichir, par exemple par des colonnes d'énumération ou de rangement (*ranking*). Ces fonctions sont par exemple RANK() ou ROW_NUMBER() et possèdent une sous clause de nom OVER permettant de définir le sens du rangement (ORDER BY) et la partition, c'est-à-dire l'éventuel niveau de rupture pour relacer le rangement.

La syntaxe générale en est la suivante :

```
<partition_function)()
   OVER ( ORDER BY <liste_colonne_ordre>
        [ PARTITION BY <liste_colonne> ] )

   <liste_colonne_ordre> ::
      col1 [ ASC | DESC ] [ , col2 [ ASC | DESC ] [ ... ] ]

   <liste_colonne > ::
      col1 [ , col2 [ ... ] ]

<partition_function> ::
{ RANK | DENSE_RANK | PERCENT_RANK | ROW_NUMBER | CUM_DIST }
```

Il faut savoir aussi que les fonctions de fenêtrage s'appliquent aussi aux fonctions de calcul d'agrégat statistique. Avec une particularité, pour le COUNT(*) la clause ORDER BY n'existe pas…

Une idée est donc de trier toutes nos lignes sous la forme d'une série avec un ROW_NUMBER et de supprimer les lignes des parties hautes et basses qui ne nous intéressent pas.

Voyons ce que donne une première requête :

```
SELECT *, ROW_NUMBER() OVER(ORDER BY STT_VALEUR) AS RANG
FROM   dbo.T_STATISTIQUES_STT
```

```
STT_ID      STT_VALEUR           RANG
----------- -------------------- --------
1           22                   1
7           22                   2
8           22                   3
9           22                   4
10          22                   5
11          22                   6
12          22,5                 7
3           22,5                 8
5           23                   9
6           23,5                 10
4           24                   11
2           27,5                 12
```

Pour trouver la ou les bonnes lignes il est souhaitable d'avoir en sus le nombre de ligne. C'est là que nous allons utiliser le COUNT(*) en « fonction de fenêtrage » :

```
SELECT *, ROW_NUMBER() OVER(ORDER BY STT_VALEUR) AS RANG,
          COUNT(*) OVER() AS NOMBRE
FROM   dbo.T_STATISTIQUES_STT
```

```
STT_ID      STT_VALEUR           RANG                 NOMBRE
----------- -------------------- -------------------- -----------
1           22                   1                    12
7           22                   2                    12
```

```
8           22                  3                   12
9           22                  4                   12
10          22                  5                   12
11          22                  6                   12
12          22,5                7                   12
3           22,5                8                   12
5           23                  9                   12
6           23,5                10                  12
4           24                  11                  12
2           27,5                12                  12
```

Il va maintenant falloir prendre une seule ligne si NOMBRE est impair ou 2 si nombre est pair. Cela doit se faire de la sorte :

```
SELECT *
FROM   (SELECT *, ROW_NUMBER() OVER(ORDER BY STT_VALEUR) AS RANG,
                  COUNT(*) OVER() AS NOMBRE
        FROM   T_STATISTIQUES_STT) AS T
WHERE  (    MOD(NOMBRE, 2) = 0
        AND RANG IN (NOMBRE / 2, (NOMBRE / 2) + 1 ))
  OR   (    MOD(NOMBRE, 2) = 1
        AND RANG = NOMBRE)
```

***NOTA** : cette requête est en pur SQL normatif. Pour SQL Server remplacer la fonction MOD (modulo) par l'opérateur %.*

En effet une fonction de fenêtrage ne peut apparaître en filtrage ni dans une clause WHERE ni dans une clause HAVING de la requête qui la porte car, opérant sur les lignes du résultat, ses valeurs ne sont connues qu'après calcul de toutes les lignes. Il convient donc d'utiliser systématiquement une sous requête de type « table dérivée » pour se faire (sous requête dans la clause FROM.

L'application de cette requête nous donne :

```
STT_ID      STT_VALEUR           RANG                NOMBRE
----------- -------------------- ------------------- -----------
11          22                   6                   12
12          22,5                 7                   12
```

Dont nous savons désormais que nous devons faire la moyenne !

```
SELECT AVG(STT_VALEUR) AS MEDIANE
FROM   (SELECT *
        FROM   (SELECT *, ROW_NUMBER() OVER(ORDER BY STT_VALEUR) AS RANG,
                          COUNT(*) OVER() AS NOMBRE
                FROM   T_STATISTIQUES_STT) AS T
        WHERE  (    MOD(NOMBRE, 2) = 0
                AND RANG IN (NOMBRE / 2, (NOMBRE / 2) + 1 ))
          OR   (    MOD(NOMBRE, 2) = 1
                AND RANG = NOMBRE / 2) ) AS T
```

***NOTA** : cette requête est en pur SQL normatif. Pour SQL Server remplacer la fonction MOD (modulo) par l'opérateur %.*

Nous appellerons cette nouvelle solution **la solution SQL:1999**.

9 - La médiane avec la clause TOP (n) de SQL Server 2005

Unea autre façon de calculer une médiane est d'utiliser une clause que j'exècre car parfaitement anti relationnelle, comme la clause TOP de SQL Server (mais on trouve les mêmes horreurs chez MySQL par exemple avec la clause LIMIT…)

Voici une expression de la chose. Elle consiste à calculer le nombre de ligne haute et basse de sélectionner la max et le min de chacun des ensemble, de les réunir et bien entendu d'en faire la moyenne :

```
SELECT AVG(VAL)
FROM   (SELECT MAX(STT_VALEUR) AS VAL
        FROM  (SELECT TOP(SELECT  CASE
                                     WHEN N % 1 = 0 THEN (N /2) + 1
                                     ELSE N / 2
                                  END AS TOPN
                           FROM (SELECT COUNT(*) AS N
                                 FROM   dbo.T_STATISTIQUES_STT) AS T)
                                                                  STT_VALEUR
               FROM   dbo.T_STATISTIQUES_STT
               ORDER  BY STT_VALEUR) AS T1

        UNION

        SELECT MIN(STT_VALEUR) AS VAL
        FROM  (SELECT TOP(SELECT  CASE
                                     WHEN N % 1 = 0 THEN (N /2) + 1
                                     ELSE N / 2
                                  END AS TOPN
                           FROM (SELECT COUNT(*) AS N
                                 FROM   dbo.T_STATISTIQUES_STT) AS T)
                                                                  STT_VALEUR
               FROM   dbo.T_STATISTIQUES_STT
               ORDER  BY STT_VALEUR DESC) AS T2
        ) AS T
```

Nous appellerons cette nouvelle solution **la solution Top** !.

10 – La médiane itérative…

Tant qu'à faire, pourquoi ne pas implémenter une médiane avec un curseur ?
Simple : il suffit de compter le nombre de lignes, puis d'aller au bon endroit dans un curseur ordonné, récupérer la ou les bonnes valeurs et ensuite de faire une moyenne.

Voici une telle procédure écrite en Transact SQL le langage procédural propre à SQL Server et Sybase :

```
CREATE PROCEDURE P_MEDIANE
AS
BEGIN
   CREATE TABLE #T_MEDIANE (V FLOAT);
   DECLARE @MILIEU INTEGER, @PAIR BIT, @VAL FLOAT;
   SELECT @MILIEU = COUNT(*) FROM T_STATISTIQUES_STT;
   IF @MILIEU % 1 = 0 SET @PAIR = 1 ELSE SET @PAIR = 0;
   SET @MILIEU = @MILIEU / 2;
   DECLARE C SCROLL CURSOR
   FOR
      SELECT STT_VALEUR
      FROM   T_STATISTIQUES_STT
      ORDER  BY STT_VALEUR
   FOR READ ONLY;
   OPEN C;
   FETCH ABSOLUTE @MILIEU FROM C INTO @VAL;
   INSERT INTO #T_MEDIANE VALUES (@VAL);
   IF @PAIR = 1
   BEGIN
      FETCH NEXT FROM C INTO @VAL;
      INSERT INTO #T_MEDIANE VALUES (@VAL);
   END;
   CLOSE C;
   DEALLOCATE C;
   SELECT AVG(V) AS MEDIANE FROM #T_MEDIANE;
END;
```

Nous appellerons cette nouvelle solution **la solution cursor**…

8 – Quelle est la meilleure ?

Bien entendu on peut se demander avec toutes ses différentes versions quelles sont celles qui sont les plus efficaces.

Pour réaliser notre test, j'ai farcis la table de quelques 10 000 lignes avec le script SQL suivant (MS SQL Server) :

```
-- insertion de 9989 lignes supplémentaires
DECLARE @ID INT;
SET @ID = 12;
WHILE @ID <> 10000
BEGIN
    SET @ID = @ID + 1;
    INSERT INTO dbo.T_STATISTIQUES_STT (STT_ID, STT_VALEUR)
     VALUES (@ID, RAND() * 25.0);
END;
```

Nous testerons une première fois les requêtes sans aucun index, puis avec un index créé sur la colonne STT_VALEUR.

```
CREATE INDEX X_VAL ON dbo.T_STATISTIQUES_STT (STT_VALEUR);
```

Voici les résultats de notre étude :

Requête	Date	Date & Celko	Celko 1	Molinaro	Celko 2	SQL :1999	Top	Cursor
Sans index	703 707	390 714	390 772	360 443	∞	20 393	116	20 239
Meilleur indexation	523 959	291 424	291 428	360 481	∞	20 390	81	20 234

Mesure des IO (entrées / sorties) pour les différentes requêtes avec et sans index.

And the winner is... La requête avec la clause TOP ! Notez que l'indexation n'a que peu d'effet sur le volume des données traitées sauf pour la requête vedette !

Notons l'extrême mauvais score de la formule Celko2. En effet nous n'avons pas pus arrivé au bout du calcul après une heure sur un serveur biprocesseur cadencé à 3,4 Ghz doté de 8 Go de RAM. En utilisant un panel de données plus fin, nous avons obtenu, pour cette formule, les éléments suivants :

Nombre de lignes	IO	time (s)
250	440 108	17
300	946 236	30
350	1 287 686	49
400	1 683 043	72
450	2 129 668	103
500	2 628 793	140

Ceci nous laisse supposer que le nombre des IO avec cette formule devrait être très supérieur à 50 000 000... Autrement dit, cette formule est inutilisable...