Étude d'Indexation
Avec SQL Server 2000/2005

La présente étude permet de comprendre l'intérêt de l'indexation, le gain qu'elle apporte et les techniques à mettre en oeuvre.
Pour plus d'intérêt, nous verrons une démonstration spectaculaire au cours de laquelle nous passerons d'un coût de requête de 96 000 avec un modèle non indexé à un coût de 2... soit un gain de 48 000.
Bien entendu il s'agit d'un cas d'école, mais formateur en diable !

Par Frédéric Brouard - MVP SQL Server
Expert SQL et SGBDR, Auteur de :

- SQL, Développement, Campus Press 2001
- SQL, collection Synthex, Pearson Education 2005, co écrit avec Christian Soutou
- http://sqlpro.developpez.com (site de ressources sur le langage SQL et les SGBDR)
- Enseignant aux Arts & Métiers et à l'ISEN Toulon

0 - Le problème :

Avec la table dbo.T_EMPLOYEE_EMP dont la structure est la suivante :

```
CREATE TABLE dbo.T_EMPLOYEE_EMP
(    EMP_ID             int IDENTITY NOT NULL PRIMARY KEY,
     EMP_NOM            char(32) NOT NULL,
     EMP_PRENOM         varchar(25),
     EMP_TITRE          char(8),
     EMP_ADRESSE1       varchar(38),
     EMP_ADRESSE2       varchar(38),
     EMP_ADRESSE3       varchar(38),
     EMP_CP             char(8),
     EMP_VILLE          varchar(32),
     EMP_TEL            char(20),
     EMP_GSM            char(20),
     EMP_DATE_ENTREE    datetime,
     EMP_INDICE         float,
     EMP_SALAIRE        int,
     EMP_MATRICULE      uniqueidentifier DEFAULT NEWID(),
     EMP_SERVICE        varchar(16),
     EMP_SEXE           char(5))
```

Constituée d'une unique structure de stockage (table en cluster indexé sur Primary Key), comportant 1 253 500 lignes.

La requête : *comptez le nombre d'hommes et de femmes du service RH.*

1 - Reproduire le problème dans votre base

Pour reproduire le problème dans votre SGBDR SQL Server, restaurez la base DB_TEST_INDEX à l'aide du script SQL suivant, en remplaçant les ??? par les chemins de fichiers adéquats (le premier étant le chemin du fichier de sauvegarde de la base, les suivant le répertoire de destination des fichiers de la base).

```
RESTORE DATABASE DB_TEST_INDEX
FROM DISK = '???\BACKUP_DATABASE_AM_DB_TEST_INDEX.BDB'
WITH REPLACE,
```

```
     MOVE 'DB_TEST_INDEX'     TO '???\fichier_de_donnees_de_la_base_DB_TEST_INDEX.DATA',
     MOVE 'DB_TEST_INDEX_log' TO '???\journal_de_transaction_de_la_base_DB_TEST_INDEX.JT'
GO
```

Ceci ne termine pas pour autant le travail car il nous reste à charger plus en avant la base de nouveaux employés...
Pour cela une simple requête suffit. Attention cependant, cette requête peut prendre beaucoup de temps parce qu'elle manipule plus d'un million de lignes... mais il chargera suffisament votre table pour représenter un volume de données significatif !

```
INSERT INTO dbo.T_EMPLOYEE_EMP
SELECT E1.EMP_NOM, E2.EMP_PRENOM, E2.EMP_TITRE,
       E1.EMP_ADRESSE1, E1.EMP_ADRESSE2, E1.EMP_ADRESSE3,
       E2.EMP_CP, E2.EMP_VILLE, E2.EMP_GSM, E1.EMP_TEL,
       E2.EMP_DATE_ENTREE, E1.EMP_INDICE, E2.EMP_SALAIRE,
       NEWID(), E1.EMP_SERVICE, E2.EMP_SEXE
FROM   dbo.T_EMPLOYEE_EMP E1
       CROSS JOIN dbo.T_EMPLOYEE_EMP E2
WHERE  E1.EMP_ID < E2.EMP_ID
  AND  E2.EMP_ID % 10 = 3
```

Pour vérifier que vous avez suffisament de lignes en lançant la commande suivante :

```
sp_spaceused 'dbo.T_EMPLOYEE_EMP'
```

Vous devriez obtenir les données suivantes :

```
name              rows        reserved      data          index_size    unused
----------------- ----------- ------------- ------------- ------------- ---------
T_EMPLOYEE_EMP    1253500     258344 KB     257472 KB     840 KB        32 KB
```

...ou quelque chose de similaire !

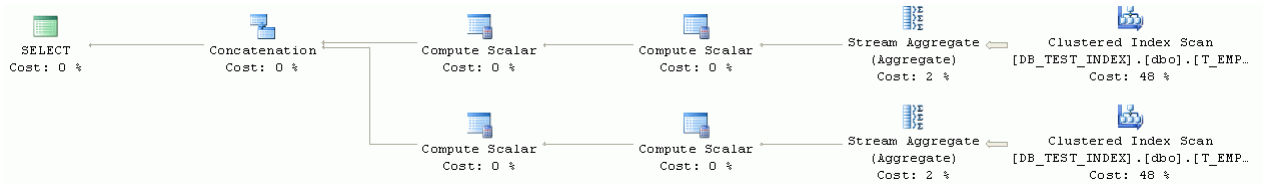
En fait cette table d'employés compte donc plus de 1 250 000 salariés et occupe 251 Mo de données plus 0,8 Mo pour les pages de navigation de l'index cluster.

1 – Situation actuelle

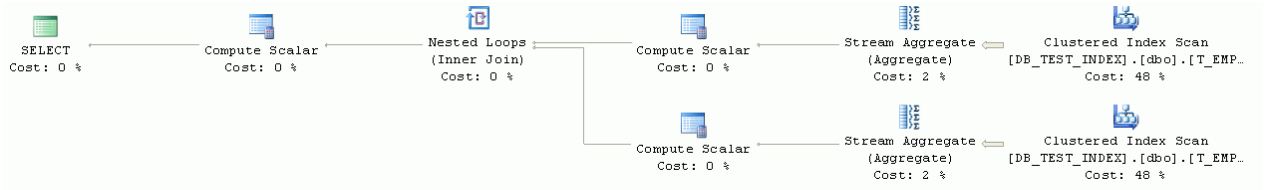
Voici diverses expression de requête pour cette demande et les plans de requête associés.

```
-- 1
SELECT COUNT(*), EMP_SEXE
FROM   T_EMPLOYEE_EMP
WHERE  EMP_SERVICE = 'RH'
GROUP  BY EMP_SEXE
```

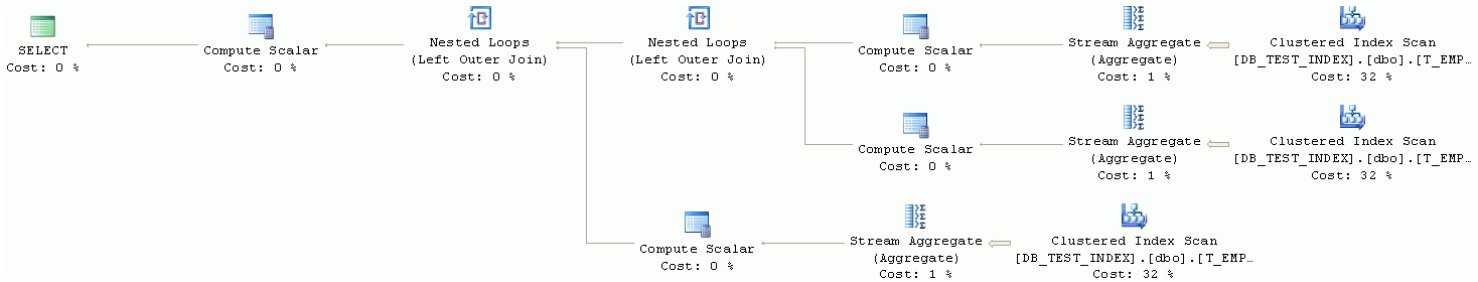

```
-- 2
SELECT COUNT(*) AS NOMBRE, 'Homme' AS SEXE
FROM   T_EMPLOYEE_EMP
WHERE  EMP_SERVICE = 'RH'
  AND  EMP_SEXE = 'Homme'
UNION
SELECT COUNT(*) AS NOMBRE, 'Femme' AS SEXE
FROM   T_EMPLOYEE_EMP
WHERE  EMP_SERVICE = 'RH'
  AND  EMP_SEXE = 'Femme'
GO
```



```sql
-- 3
SELECT COUNT(*) AS HOMME,
       (SELECT COUNT(*)
        FROM   T_EMPLOYEE_EMP
        WHERE  EMP_SERVICE = 'RH'
          AND  EMP_SEXE = 'Femme') AS FEMME
FROM   T_EMPLOYEE_EMP
WHERE  EMP_SERVICE = 'RH'
  AND  EMP_SEXE = 'Homme'
GO
```



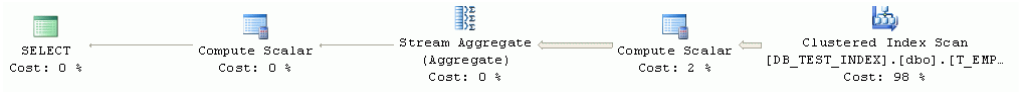

```sql
-- 4
SELECT COUNT(*) - (SELECT COUNT(*)
                   FROM   T_EMPLOYEE_EMP E2
                   WHERE  E2.EMP_SERVICE = E1.EMP_SERVICE
                     AND  EMP_SEXE = 'Femme') AS HOMME,
       (SELECT COUNT(*)
        FROM   T_EMPLOYEE_EMP E3
        WHERE  E3.EMP_SERVICE = E1.EMP_SERVICE
          AND  EMP_SEXE = 'Femme') AS FEMME
FROM   T_EMPLOYEE_EMP E1
WHERE  EMP_SERVICE = 'RH'
GROUP  BY EMP_SERVICE
GO
```



```
-- 5 (valable uniquement sur SQL Server 2005)
WITH T_EMP
AS
(SELECT COUNT(*) AS N
 FROM   T_EMPLOYEE_EMP
 WHERE  EMP_SERVICE = 'RH'
   AND  EMP_SEXE = 'Femme')
SELECT COUNT(*) - (SELECT N FROM T_EMP) AS HOMME,
       (SELECT N FROM T_EMP) AS FEMME
FROM   T_EMPLOYEE_EMP E1
WHERE  EMP_SERVICE = 'RH'
GROUP  BY EMP_SERVICE
GO
```

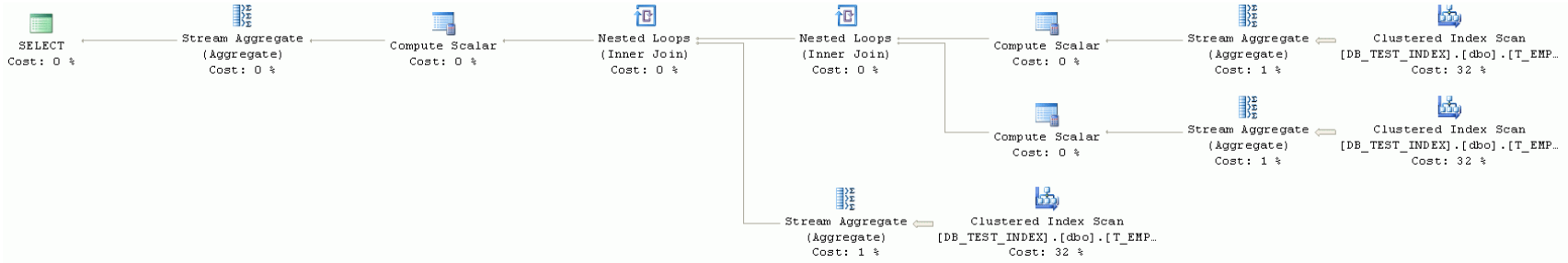
Même plan que requête 4

```
-- 6
SELECT SUM(CASE EMP_SEXE
             WHEN 'Homme' THEN 1
             WHEN 'Femme' THEN 0
           END) AS NOMBRE_HOMME,
       SUM(CASE EMP_SEXE
             WHEN 'Homme' THEN 0
             WHEN 'Femme' THEN 1
           END) AS NOMBRE_FEMME
FROM  dbo.T_EMPLOYEE_EMP
WHERE EMP_SERVICE= 'RH'
GO
```



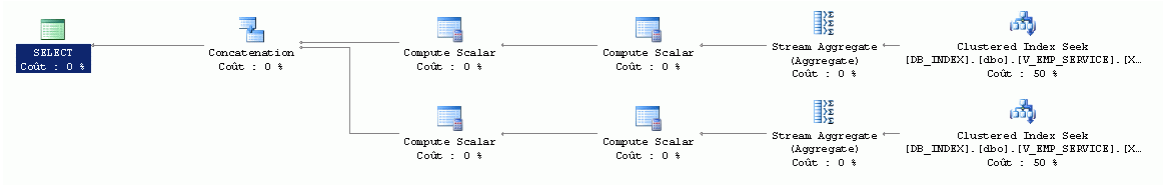
```sql
-- 7
SELECT COUNT(EMP_SEXE) AS NOMBRE,
       CASE EMP_SEXE
        WHEN 'Femme' THEN 'Femme'
        WHEN 'Homme' THEN 'Homme'
        ELSE 'Unknown'
       END AS SEXE
FROM   dbo.T_EMPLOYEE_EMP
WHERE  EMP_SERVICE= 'RH'
GROUP  BY EMP_SEXE
GO
```


```sql
-- 8
SELECT DISTINCT EMP_SERVICE,
       (SELECT COUNT(*)
        FROM   dbo.T_EMPLOYEE_EMP
        WHERE  EMP_SEXE = 'Homme'
        AND    EMP_SERVICE= 'RH') AS HOMME,
       (SELECT COUNT(*)
        FROM   dbo.T_EMPLOYEE_EMP
        WHERE  EMP_SEXE = 'Femme'
        AND    EMP_SERVICE= 'RH') AS FEMME
FROM  dbo.T_EMPLOYEE_EMP
WHERE EMP_SERVICE = 'RH'
GO
```




```sql
-- 9 (valable uniquement sur SQL Server 2005)
WITH
CTE_RH AS
(SELECT EMP_SEXE
```

```
 FROM   dbo.T_EMPLOYEE_EMP
 WHERE  EMP_SERVICE = 'RH' ),
CTE_FEMME AS
(SELECT *
 FROM   CTE_RH
 WHERE  EMP_SEXE = 'Femme'),
CTE_HOMME AS
(SELECT *
 FROM   CTE_RH
 WHERE  EMP_SEXE = 'Homme')
SELECT COUNT(*) AS N, 'FEMME'
FROM   CTE_FEMME
UNION
SELECT COUNT(*) AS N, 'HOMME'
FROM   CTE_HOMME
```



Exécutez les requêtes dans une fenêtre de requête du Query Analyzer (SQL 2000) ou de SL Server management Studio (SQL Server 2005) en ayant préalablement lancé la commande :

```
SET STATISTICS IO ON
```

Qui permet d'afficher le nombre des pages lues (I/O = entrées sorties). Ce paramètre est le plus significatif en matière d'affinage de requêtes. Il est trivial de comprendre que plus le volume de données à manipuler est minime, plus rapide sera la requête !
Si vous avez des lectures physiques de disque, c'est que toutes les données de votre table ne sont pas dans le cache. Relancez plusieurs fois les requêtes, cela aidera à faire persister les données en mémoire.

En terme de lectures logiques de pages (SET STATISTICS IO ON), les résultats sont les suivants :

- Les requêtes 1, 6 et 7 consomment 32 296 pages
- Les requêtes 2, 3 et 9 consomment 64 592 pages (soit 2 fois plus)
- Les requêtes 4, 5 et 8 consomment 96 888 pages (soit 3 fois plus)

2 – Ajout d'un index mono colonne sur EMP_SEXE

Il semble à priori intéressant de placer un index sur le sexe des employés puisque l'on peut constater que la plupart des requêtes utilisent cette colonne comme critère dans la clause WHERE.

```
CREATE INDEX X_SEX ON T_EMPLOYEE_EMP (EMP_SEXE)
```

Ceci n'a aucun effet sur aucune requête. Les résultats restent inchangés. Pourquoi ? L'index n'est tout simplement pas sélectif. Avec deux valeurs seulement dans le panel de données il n'y a pas vraiment d'intérêt à l'utiliser !
L'index est retiré.

```
DROP INDEX T_EMPLOYEE_EMP.X_SEX
```

3 – Ajout d'un index mono colonne sur EMP_SERVICE

Tentons un index sur le service cette colonne est bien plus discriminante. En principe le moteur devrait en tenir compte.

```
CREATE INDEX X_SRV ON T_EMPLOYEE_EMP (EMP_SERVICE)
```

En terme de lectures logiques de pages (SET STATISTICS IO ON), les requêtes 4, 5 et 8 passent de 96 888 pages à 64 651, soit une économie de 33%. Le moteur a donc tenu compte de cet index et l'on peut le constater en étudiant les différents plans de requête qui en résulte. Cependant cet index oblige à retourner dans la table afin de trouver l'information complémentaire sur le sexe des employés...

L'index est retiré.

```
DROP INDEX T_EMPLOYEE_EMP.X_SRV
```

4 – Ajout d'un index bi colonne sur EMP_SEXE / EMP_SERVICE (index couvrant)

A priori un index bicolonne semble plus intéressant car il sera couvrant, c'est à dire que la seule lecture de l'index suffit à répondre à la requête...

```
CREATE INDEX X_SEXSRV ON T_EMPLOYEE_EMP (EMP_SEXE, EMP_SERVICE)
```

Les résultats changent du tout au tout.

- Les requêtes 2, 3 et 9 passent de 64 592 pages à 80, soit une économie de 99,9 %
- Les requêtes 4 et 5 passent de 96 888 pages à 4 544, soit une économie de 95 %
- Les requêtes 1, 6 et 7 passent de 32 296 pages à 4 486, soit une économie de 86 %
- La requête 8 passe de 32 296 à 4 566, soit une économie de 85 %

Les plans de requêtes montrent que la lecture des données est passée dans la plupart des cas d'un balayage (SCAN) à une recherche dans l'index (SEEK). Ceci explique les performances nettement améliorées.
Cependant notez qu'à ce stade, les requêtes les plus performantes (2 et 3) ne sont pas d'une écriture évidente.

Pour les meilleures performances nous avons divisé par près de 400 fois le nombre des lectures. Nous avons là un bon index !

L'index est retiré.

```

```

5 – Ajout d'un index bi colonne sur EMP_SERVICE / EMP_SEXE (index couvrant)

Renversons l'ordre de l'index bicolonne en plaçant en tête la colonne EMP_SERVICE plus discriminante.

```
CREATE INDEX X_SRVSEX ON T_EMPLOYEE_EMP (EMP_SERVICE, EMP_SEXE)
```

Les résultats s'améliorent encore.

- Les requêtes 2, 3 et 9 passent de 64 592 pages à 81, soit une économie de 99,87 %
- Les requêtes 4 et 5 passent de 96 888 pages à 137, soit une économie de 99,86 %
- La requête 8 passe de 32 296 à 158, soit une économie de 99,51 %
- Les requêtes 1, 6 et 7 passent de 32 296 pages à 77, soit une économie de 99,76 %

Les plans de requêtes montrent encore une fois que la lecture des données est passée dans la plupart des cas d'un balayage (SCAN) à une recherche dans l'index (SEEK). Ceci explique les performances nettement améliorées.
Notez qu'à ce stade, parmi les requêtes les plus performantes figure la 1 (écriture la plus classique) mais aussi la 6 et la 7 avec le comptage imbriqué dans la structure CASE.
Pour les meilleures performances nous avons divisé par près de 420 fois le nombre des lectures. Nous avons là un bon index !

L'index est retiré.

```
DROP INDEX T_EMPLOYEE_EMP.X_SRVSEX
```

6 – Ajout d'un index mono colonne sur EMP_SERVICE incluant EMP_SEXE (index couvrant - SQL Server 2005)

SQL Server 2005 permet de créer des index ayant des colonnes surnuméraires non indexées, ce qui permet de couvrir artificiellement certaines requêtes. Cela est une bonne idée. Essayons de la mettre en oeuvre en indexant le service et en ajoutant le sexe.

```
CREATE INDEX X_SRV_SEX ON T_EMPLOYEE_EMP (EMP_SERVICE) INCLUDE (EMP_SEXE)
```

Les résultats ne s'améliorent pas.

Les requêtes 2, 3 et 9 passent de 81 pages à 154.
Les requêtes 4 et 5 passent de 137 pages à 231.

La requête 8 passe de 158 à 231.

Les requêtes 1, 6 et 7 sont stables à 77 pages lues.

On peut cependant supposer que :

- l'index est moins volumineux que celui précédemment posé
- la mise à jour de cet index sera moins consommatrice de ressources

Pour comparaisons, rajouter l'index précédent à l'aide de la commande suivante :

```
CREATE INDEX X_SEXSRV ON T_EMPLOYEE_EMP (EMP_SEXE, EMP_SERVICE)
```

La taille des index peut être vue indifféremment à l'aide d'une de ces deux requêtes :

```
SELECT name, used
FROM   sys.sysindexes
WHERE  name LIKE 'X?_%' ESCAPE '?';
```

```
DECLARE @DBID INT, @OBID INT
SELECT @DBID = DB_ID(), @OBID = OBJECT_ID('dbo.T_EMPLOYEE_EMP')
SELECT DISTINCT i.name, used
FROM   sys.sysindexes i
       CROSS APPLY sys.dm_db_index_physical_stats(@DBID, @OBID, NULL, NULL, 'DETAILED') As ips
WHERE  name LIKE 'X?_%' ESCAPE '?'
  AND  i.indid = ips.index_id;
```

```
name                 used
-------------------- -----------
X_SRVSEX             4486
X_SRV_SEX            4483
```

Elles montrent que la différence est peu sensible : 3 pages sur 4486 soit moins de 0,1% de différence...

7 – Création d'une vue indexée

Supprimez préalablement tous les index posés.

Notre dernière étape consiste à créer une vue que l'on indexera, c'est à dire en fait une table calculée en permanence par rapport à la définition de la vue. SQL Server appelle cela "vue indexée" et Oracle, "vue matérialisée"...

Création de la vue :

```
CREATE VIEW V_EMP_SSC
WITH SCHEMABINDING
AS
SELECT COUNT_BIG(*) AS NOMBRE,
       EMP_SEXE AS SEXE,
       EMP_SERVICE AS SERVICE
FROM   dbo.T_EMPLOYEE_EMP
GROUP  BY EMP_SERVICE, EMP_SEXE;
```

Création de l'index sur la vue

```
CREATE UNIQUE CLUSTERED INDEX X_SSC ON V_EMP_SSC (SERVICE, SEXE);
```

Essayez maintenant le code suivant :

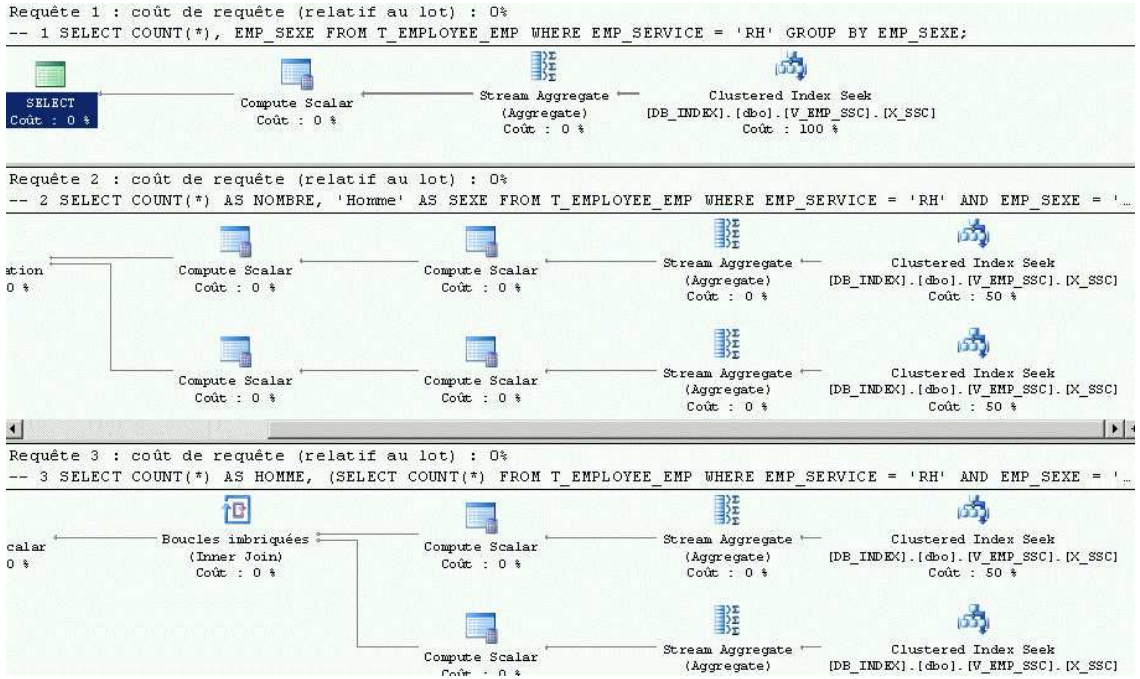
```
SET STATISTICS IO ON;
GO

SELECT NOMBRE, SEXE
FROM   V_EMP_SSC
WHERE  SERVICE = 'RH';
GO
```

Vous devriez obtenir :

```
Table 'V_EMP_SSC'... logical reads 2...
```

La vue indexée à divisé par 16148 la meilleure requête initiale et à encore permis d'économiser 97 % de pages à lire sur la meilleure requête pourvue d'un index couvrant...
Si vous êtes en édition Enterprise ou en édition Developper relancez toutes les requêtes en ayant pris soin d'effacer tous les index à l'exclusion de la vue indexée. La plupart des requêtes vont bénéficier de la vue indexée car le moteur est capable de substituer les appels à la table par la vue équivalente.

Regardez les plans de requête, seules les requêtes 6 et 7 ne bénéficient pas de cette amélioration.



Le gain est spectaculaire. Par rapport à l'étape 5 (en moyenne la meilleure) :

- La requête 2 passe de 81 à 2 soit une économie de 97,5%
- Les requêtes 2, 3 et 9 passent de 81 à 4, soit une économie de 95 %
- Les requêtes 4 et 5 passent 137 à 6, soit une économie de 96 %
- La requête 8 passe 158 à 6, soit une économie de 96 %

- Les requêtes 6 et 7 ne bénéficient pas de cette amélioration et reviennent à 32 296 au lieu de 77.

8 – PREMIÈRE CONCLUSION :

Nous pouvons dire que par rapport à la lecture initiale de la table nous avons divisé par 16 148 le traitement et donc le temps de réponse... Pas mal... non ?

De l'importance d'une bonne indexation !

9 - ENCORE PLUS

Étudiez maintenant le fait que les données EMP_SEXE et EMP_SERVICE font l'objet de deux tables à part schématisées comme suit :

```
CREATE TABLE T_SEX
(SEX_ID       SMALLINT NOT NULL PRIMARY KEY IDENTITY ,
 SEX_LIBELLE  CHAR(5) NOT NULL);
GO

CREATE TABLE T_SERVICE_SRV
(SRV_ID       SMALLINT NOT NULL PRIMARY KEY IDENTITY,
 SRV_LIBELLE  VARCHAR(36));
GO
```

Et que la table des employées en tire profit de la sorte :

```
CREATE TABLE dbo.T_EMPLOYEE2_EMP
(    EMP_ID             int IDENTITY NOT NULL PRIMARY KEY,
     EMP_NOM            char(32) NOT NULL,
     EMP_PRENOM         varchar(25),
     EMP_TITRE          char(8),
     EMP_ADRESSE1       varchar(38),
     EMP_ADRESSE2       varchar(38),
```

```
      EMP_ADRESSE3      varchar(38),
      EMP_CP            char(8),
      EMP_VILLE         varchar(32),
      EMP_TEL           char(20),
      EMP_GSM           char(20),
      EMP_DATE_ENTREE   datetime,
      EMP_INDICE        float,
      EMP_SALAIRE       int,
      EMP_MATRICULE     uniqueidentifier DEFAULT NEWID(),
      SRV_ID            smallint FOREIGN KEY REFERENCES T_SERVICE_SRV (SRV_ID),
      SEX_ID            smallint FOREIGN KEY REFERENCES T_SEX (SEX_ID));
GO
```

Voici le script de basculement de l'ancienne version de la table à la nouvelle modélisation multitabulaire :

```
INSERT INTO dbo.T_SEX
SELECT DISTINCT EMP_SEXE
FROM   dbo.T_EMPLOYEE2_EMP;
GO

INSERT INTO dbo.T_SERVICE_SRV
SELECT DISTINCT EMP_SERVICE
FROM   dbo.T_EMPLOYEE2_EMP;
GO

SET IDENTITY_INSERT dbo.T_EMPLOYEE2_EMP ON;
INSERT INTO dbo.T_EMPLOYEE2_EMP (EMP_ID, EMP_NOM, EMP_PRENOM, EMP_TITRE, EMP_ADRESSE1,
                                 EMP_ADRESSE2, EMP_ADRESSE3,
                                 EMP_CP, EMP_VILLE, EMP_TEL, EMP_GSM, EMP_DATE_ENTREE, EMP_INDICE,
                                 EMP_SALAIRE, EMP_MATRICULE, SRV_ID, SEX_ID)
SELECT EMP_ID, EMP_NOM, EMP_PRENOM, EMP_TITRE, EMP_ADRESSE1, EMP_ADRESSE2, EMP_ADRESSE3,
       EMP_CP, EMP_VILLE, EMP_TEL, EMP_GSM, EMP_DATE_ENTREE, EMP_INDICE,
       EMP_SALAIRE, EMP_MATRICULE, SRV_ID, SEX_ID
FROM   dbo.T_EMPLOYEE2_EMP E
       LEFT OUTER JOIN dbo.T_SEX S
            ON E.EMP_SEXE = S.SEX_LIBELLE
       LEFT OUTER JOIN dbo.T_SERVICE_SRV R
            ON E.EMP_SERVICE = R.SRV_LIBELLE;
SET IDENTITY_INSERT dbo.T_EMPLOYEE2_EMP OFF;
```

```
GO
```

Voyons quels identifiants ont été attribués aux deux sexes :

```
SELECT * FROM T_SEX;

SEX_ID SEX_LIBELLE
------ -----------
1      Femme
2      Homme
```

Et quel identifiant est attribué au service RH ?

```
SELECT * FROM T_SERVICE_SRV WHERE SRV_LIBELLE = 'RH';

SRV_ID SRV_LIBELLE
------ ------------------------------------
7      RH
```

Posons maintenant ce que nous savons être l'un des meilleurs index :

```
CREATE INDEX X_SRVSEX ON dbo.T_EMPLOYEE2_EMP (SRV_ID, SEX_ID);
```

Voyons maintenant comment nos requêtes légèrement modifiées en tirent profit :

```
SET STATISTICS IO ON

-- 1
SELECT COUNT(*), CASE SEX_ID
                    WHEN 1 THEN 'FEMME'
                    WHEN 2 THEN 'HOMME'
                 END
FROM   T_EMPLOYEE2_EMP
WHERE  SRV_ID = 7
GROUP  BY SEX_ID;
GO
```

```
-- 2
SELECT COUNT(*) AS NOMBRE, 'Homme' AS SEXE
FROM   T_EMPLOYEE2_EMP
WHERE  SRV_ID = 7
  AND  SEX_ID = 2
UNION
SELECT COUNT(*) AS NOMBRE, 'Femme' AS SEXE
FROM   T_EMPLOYEE2_EMP
WHERE  SRV_ID = 7
  AND  SEX_ID = 1;
GO

-- 3
SELECT COUNT(*) AS HOMME,
       (SELECT COUNT(*)
        FROM   T_EMPLOYEE2_EMP
        WHERE  SRV_ID = 7
          AND  SEX_ID = 1) AS FEMME
FROM   T_EMPLOYEE2_EMP
WHERE  SRV_ID = 7
  AND  SEX_ID = 2;
GO

-- 4
SELECT COUNT(*) - (SELECT COUNT(*)
                   FROM   T_EMPLOYEE2_EMP E2
                   WHERE  E2.SRV_ID = E1.SRV_ID
                     AND  SEX_ID = 1) AS HOMME,
       (SELECT COUNT(*)
        FROM   T_EMPLOYEE2_EMP E3
        WHERE  E3.SRV_ID = E1.SRV_ID
          AND  SEX_ID = 1) AS FEMME
FROM   T_EMPLOYEE2_EMP E1
WHERE  SRV_ID = 7
GROUP  BY SRV_ID;
GO

-- 5
```

```
WITH T_EMP
AS
(SELECT COUNT(*) AS N
 FROM   T_EMPLOYEE2_EMP
 WHERE  SRV_ID = 7
   AND  SEX_ID = 1)
SELECT COUNT(*) - (SELECT N FROM T_EMP) AS HOMME,
       (SELECT N FROM T_EMP) AS FEMME
FROM   T_EMPLOYEE2_EMP E1
WHERE  SRV_ID = 7
GROUP  BY SRV_ID;
GO

-- 6
SELECT SUM(CASE SEX_ID
             WHEN 2 THEN 1
             WHEN 1 THEN 0
           END) AS NOMBRE_HOMME,
       SUM(CASE SEX_ID
             WHEN 2 THEN 0
             WHEN 1 THEN 1
           END) AS NOMBRE_FEMME
FROM  dbo.T_EMPLOYEE2_EMP
WHERE SRV_ID = 7;
GO

-- 7
SELECT COUNT(SEX_ID) AS NOMBRE,
       CASE SEX_ID
       WHEN 1 THEN 'Femme'
       WHEN 2 THEN 'Homme'
       ELSE 'Unknown'
       END AS SEXE
FROM   dbo.T_EMPLOYEE2_EMP
WHERE  SRV_ID = 7
GROUP  BY SEX_ID;
GO

-- 8
```

```
SELECT DISTINCT
       (SELECT COUNT(*)
        FROM   dbo.T_EMPLOYEE2_EMP
        WHERE  SEX_ID = 2
        AND    SRV_ID = 7) AS HOMME,
       (SELECT COUNT(*)
        FROM   dbo.T_EMPLOYEE2_EMP
        WHERE  SEX_ID = 1
        AND    SRV_ID = 7) AS FEMME
FROM  dbo.T_EMPLOYEE2_EMP
WHERE SRV_ID = 7;
GO

-- 9
WITH
CTE_RH AS
(SELECT SEX_ID
 FROM   dbo.T_EMPLOYEE2_EMP
 WHERE  SRV_ID = 7 ),
CTE_FEMME AS
(SELECT *
 FROM   CTE_RH
 WHERE  SEX_ID = 1),
CTE_HOMME AS
(SELECT *
 FROM   CTE_RH
 WHERE  SEX_ID = 2)
SELECT COUNT(*) AS N, 'FEMME'
FROM   CTE_FEMME
UNION
SELECT COUNT(*) AS N, 'HOMME'
FROM   CTE_HOMME
```

Le résultat permet de voir que l'on gagne encore par rapport aux versions précédentes :

- Les requêtes 1, 6, 7 passent à 52 pages
- Les requêtes 2, 3, 9 passent à 56 pages
- Les requêtes 4 et 5 passent à 94 pages
- La requête 8 passe à 60 pages.

Le meilleur gain par rapport à l'étape 5 ou 6 est de 77 / 52 soit : 32%, c'est considérable.

Bien entendu il nous faudrait être plus explicite et interroger ces tables non pas des ID, mais par des libellés comme c'était le cas précédemment

Ainsi l'exécution de :

```
SELECT COUNT(*), SEX_LIBELLE
FROM   T_EMPLOYEE2_EMP E
       INNER JOIN dbo.T_SEX S
             ON E.SEX_ID = S.SEX_ID
       INNER JOIN dbo.T_SERVICE_SRV R
             ON E.SRV_ID = R.SRV_ID
WHERE  SRV_LIBELLE = 'RH'
GROUP  BY SEX_LIBELLE;
```

Donne :

```
Table 'T_SEX'.           Nombre d'analyses 0, lectures logiques  4...
Table 'Worktable'.       Nombre d'analyses 0, lectures logiques  0...
Table 'T_EMPLOYEE2_EMP'. Nombre d'analyses 1, lectures logiques 52...
Table 'T_SERVICE_SRV'.   Nombre d'analyses 1, lectures logiques  2...
```

Soit 58 pages lues, c'est-à-dire un gain de 77 / 58 = 25%
C'est toujours quelque chose d'appréciable.

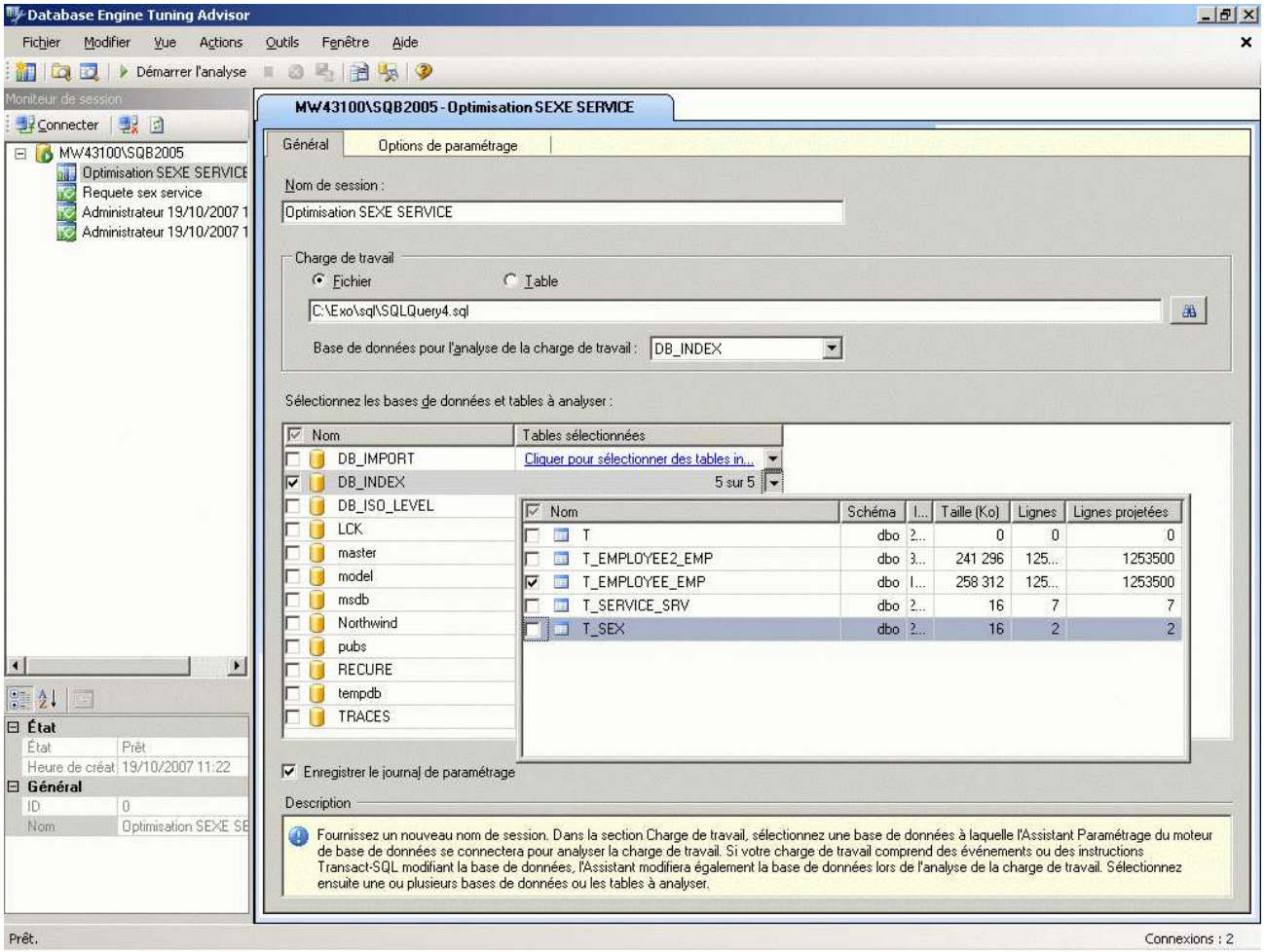
10 – SECONDE CONCLUSION

Une bonne indexation ne suffit pas. Un modèle fortement normalisé fait gagner une place appréciable au stockage des données. Dès lors les requêtes sont moins gourmandes en pages à lire et les performances à tous les niveaux (lecture comme mise à jour) augmentent.

En bref : indexation + modèle normalisé = meilleures performances possibles !

11 - TOUJOURS PLUS (SQL Server 2005 - exclusif)

SQL Server 2005 est doté d'un outil d'assistance au tuning. Ce nouvel outil « Database Tuning Advisor (DTA) » en français Assistant de paramétrage du moteur de bases de données, est accessible depuis le menu Outil de SSMS (SQL Server Management Studio).
Il permet d'auditer un lot de requête ou une table de trace et de livrer des préconisations. Voyons si les automates sont aussi performants que les humains, et confions lui nos 9 requêtes à analyser.



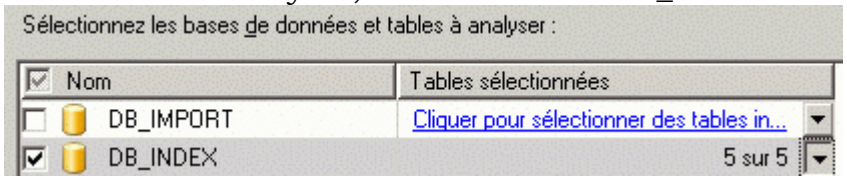
Pour cela :

1) donnez un nom à la session de travail (ici « Optimisation SEXE SERVICE »)

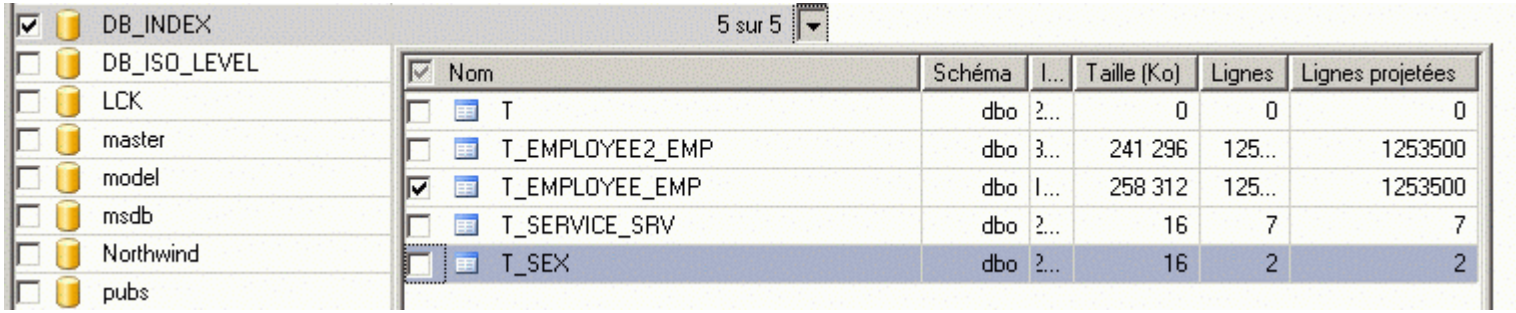
2) Cochez « Charge de Travail / Fichier » et appelez le fichier SQL contenant les 9 requêtes que vous aurez eu soin d'y mettre (ici SQLQuery4.sql)

3) Sélectionnez dans la liste déroulante la base de données pour l'analyse de la charge de travail comme étant celle qui contient notre table d'employés (ici DB_INDEX)

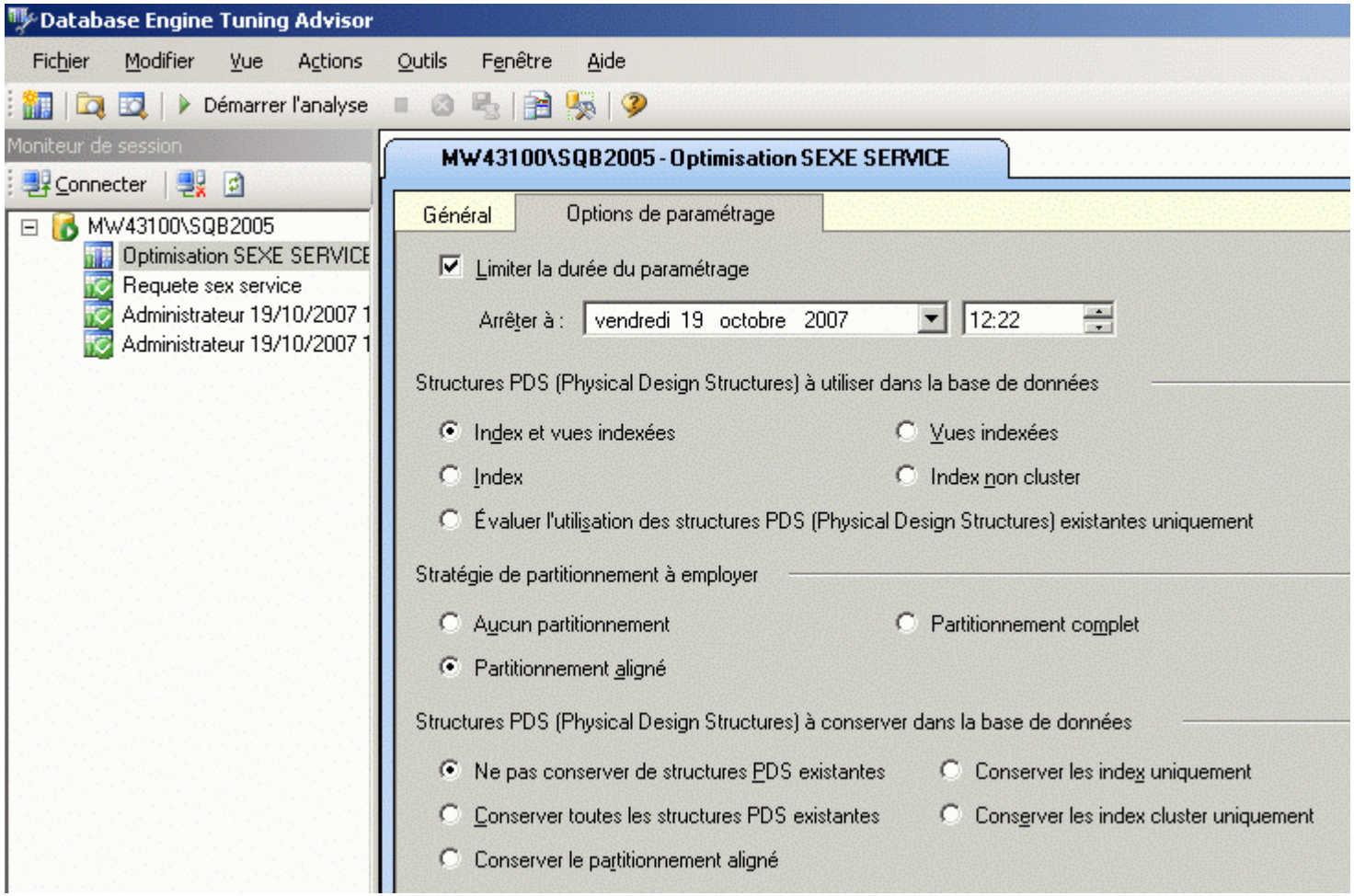
4) Dans la liste des bases de données et tables à analyser, cochez la base DB_INDEX



5) Déroulez la liste dans la colonne Table sélectionnées à la même ligne et décochez toutes les tables sauf T_EMPLOYEE_EMP



En ce faisant vous avez donné la liste des requêtes SQL à analyser, la base et les objets (tables) concernés. L'analyse est donc paramétrée au niveau de son contenu, voyons maintenant ce que nous pouvons demander à cet outil.
Pour ce faire, allez dans l'onglet « Options de paramétrage »

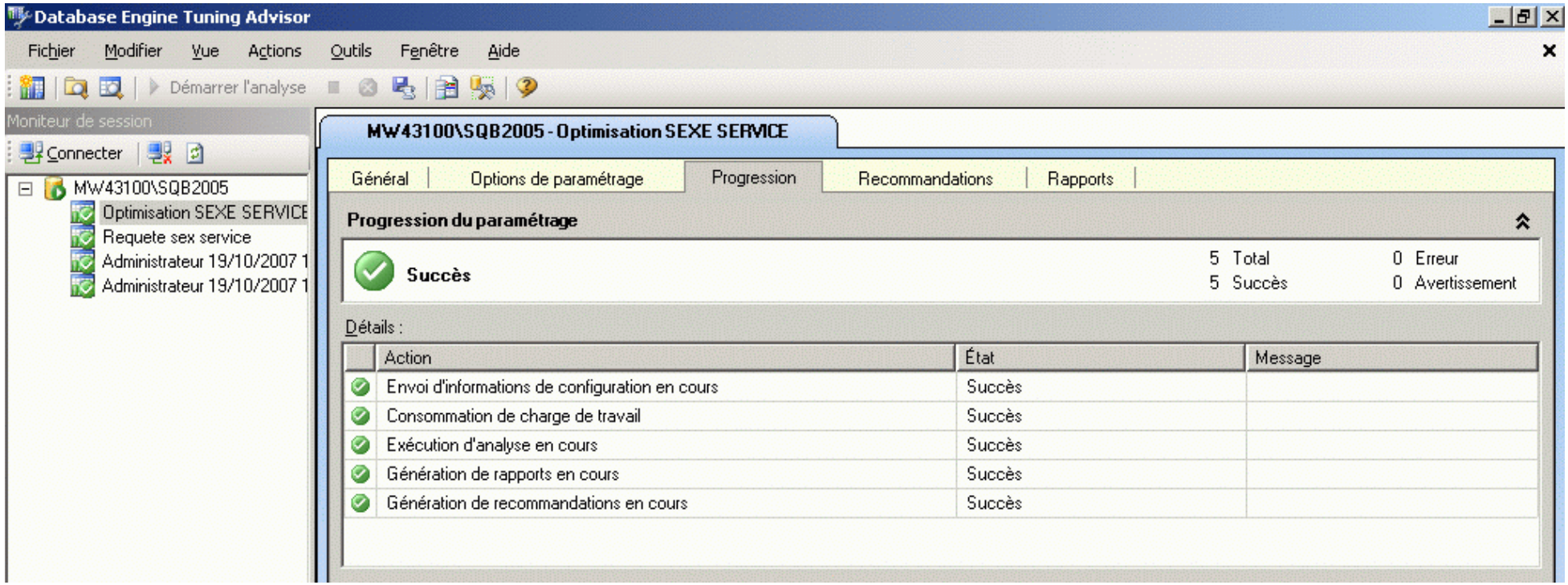


Cochez les cases :

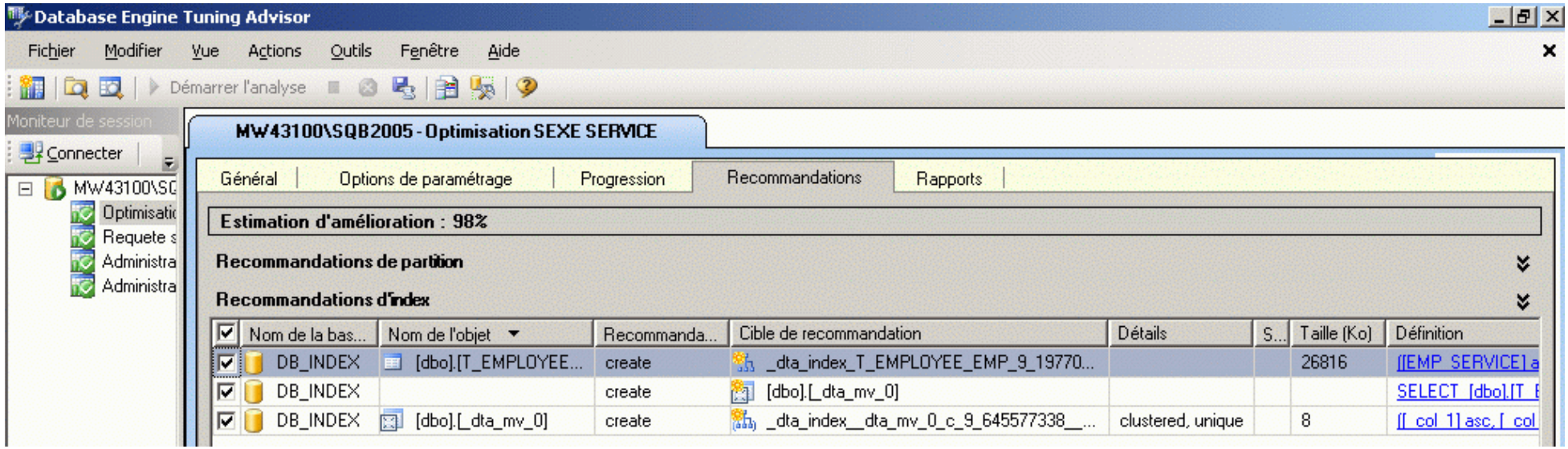
1) Structures PDS (Physical Design Structures) existantes uniquement : Index et vues indexées
2) Stratégie de partitionnement à employer : Partitionnement aligné
3) Structures PDS (Physical Design Structures) à conserver dans la base de données : ne pas conserver de structures PDS existantes

Avec ce paramétrage vous indiquer à l'assistant qu'il peut se baser sur des tables, des index et des vues indexées pour faire son travail, qu'il peut même créer des partitions voire casser les tables pour en proposer de nouvelles (refactoring de base de donnéezs).

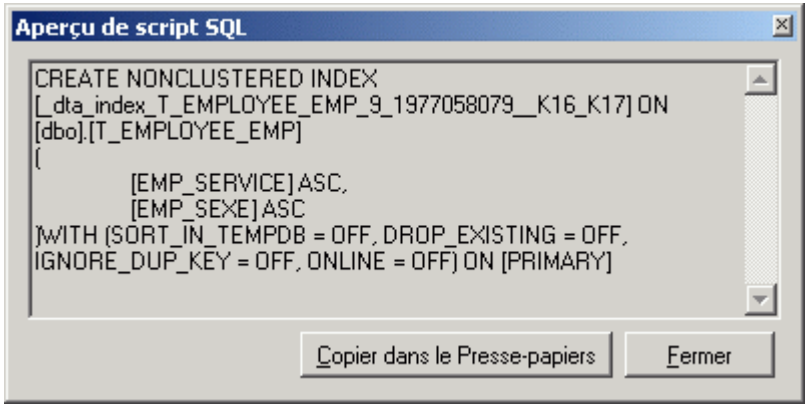
Votre assistant est suffisament paramétré pour commencer à travailler. Lancez son exécution en cliquant dans le menu « démarrez l'analyse » (triangle vert)



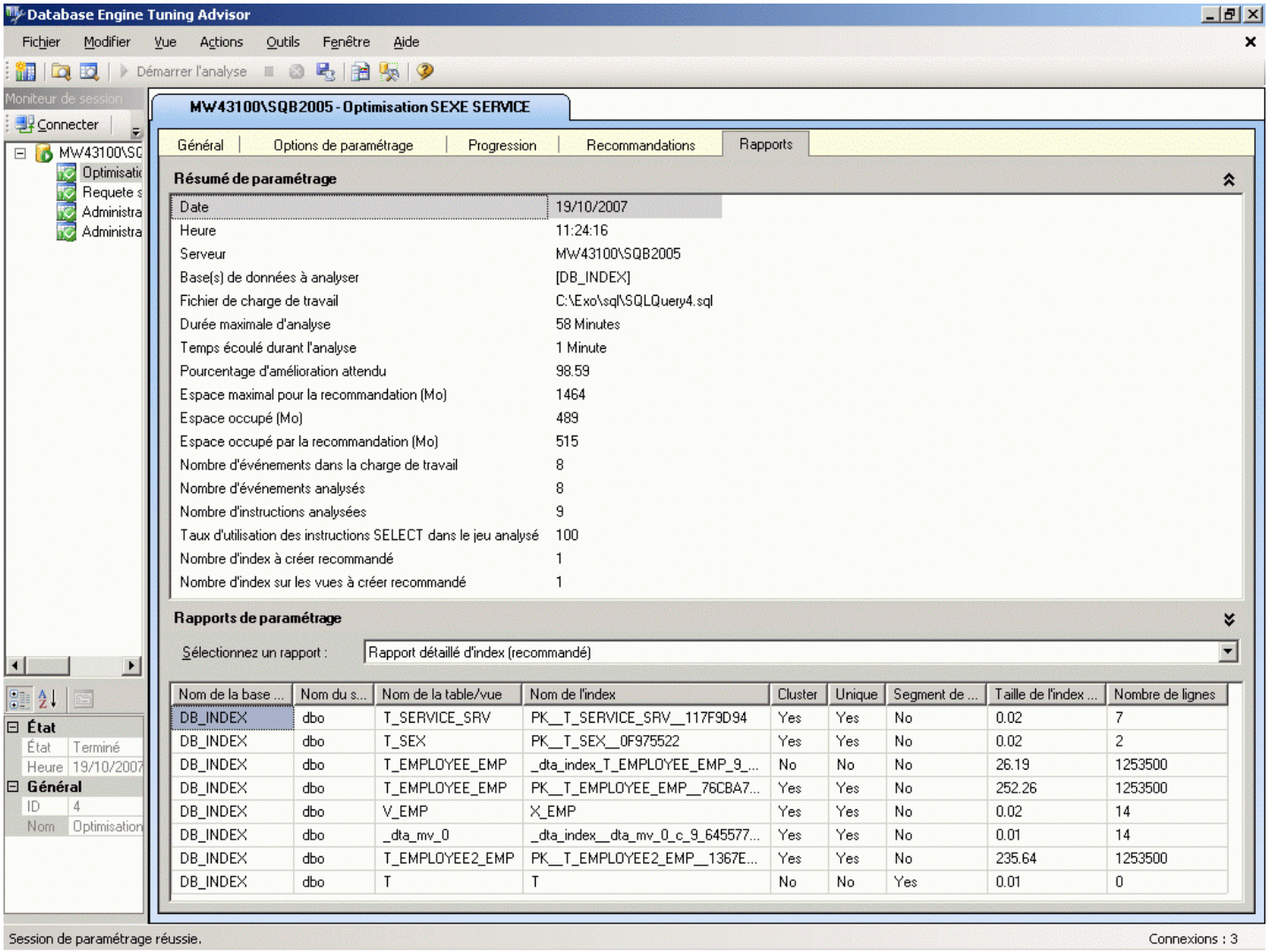
Une fois l'écran de progression passé, vous obtenez les conseils de DTA :



Ici DTA assure que le gain sera de 98% en appliquant les recommandations. Chacun des conseils peut être vu sous l'angle d'un script SQL. Il suffit pour cela de cliquer sur le lien pour voir apparaître le détail de chaque script SQL :



En allant dans l'onglet rapport, vous pouvez obtenir une présentation imprimable des recommandations de DTA avec différentes mesures de gain.



Bien entendu l'ensemble des commandes SQL peut être visualisé dans un seul script. Pour cela cliquez sur l'icône « script SQL » de la barre de menu…

Voici le script SQL que DTA nous a fournit :

```
use [DB_INDEX]
go
SET QUOTED_IDENTIFIER ON
go
SET ARITHABORT ON
go
SET CONCAT_NULL_YIELDS_NULL ON
go
SET ANSI_NULLS ON
go
SET ANSI_PADDING ON
go
SET ANSI_WARNINGS ON
go
SET NUMERIC_ROUNDABORT OFF
go
CREATE VIEW [dbo].[_dta_mv_0] WITH SCHEMABINDING
 AS
SELECT  [dbo].[T_EMPLOYEE_EMP].[EMP_SERVICE] as _col_1,
        [dbo].[T_EMPLOYEE_EMP].[EMP_SEXE] as _col_2,
        count_big([DB_INDEX].[dbo].[T_EMPLOYEE_EMP].[EMP_SEXE]) as _col_3,
        count_big(*) as _col_4
FROM    [dbo].[T_EMPLOYEE_EMP]
GROUP   BY  [dbo].[T_EMPLOYEE_EMP].[EMP_SERVICE],  [dbo].[T_EMPLOYEE_EMP].[EMP_SEXE]
go
SET ARITHABORT ON
go
SET CONCAT_NULL_YIELDS_NULL ON
go
SET QUOTED_IDENTIFIER ON
go
SET ANSI_NULLS ON
go
SET ANSI_PADDING ON
go
SET ANSI_WARNINGS ON
```

```
go
SET NUMERIC_ROUNDABORT OFF
go
CREATE UNIQUE CLUSTERED INDEX [_dta_index__dta_mv_0_c_9_645577338__K1_K2]
ON [dbo].[_dta_mv_0]
(    [_col_1] ASC,
     [_col_2] ASC
)
WITH (SORT_IN_TEMPDB  = OFF,
       DROP_EXISTING  = OFF,
       IGNORE_DUP_KEY = OFF,
       ONLINE         = OFF) ON [PRIMARY]
go

CREATE NONCLUSTERED INDEX [_dta_index_T_EMPLOYEE_EMP_9_1977058079__K16_K17]
ON [dbo].[T_EMPLOYEE_EMP]
(    [EMP_SERVICE] ASC,
     [EMP_SEXE] ASC
)
WITH (SORT_IN_TEMPDB  = OFF,
       DROP_EXISTING  = OFF,
       IGNORE_DUP_KEY = OFF,
       ONLINE         = OFF) ON [PRIMARY]
go
```

Critique du travail de DTA

Ce script comporte la création d'une vue indexable et deux index, l'un sur la vue l'autre sur la table…

La vue comporte une colonne de plus que celle que nous avions décidé de faire. Mais cette colonne s'avère inutile car redondante. Elle compte les occurrences de EMP_SEXE en plus des occurrences de lignes, le résultat étant le même.

La pose des deux index est aussi redondante pour 7 des 9 requêtes. L'index de table, n'est nécessaire que pour les deux requêtes « CASE ».

Le score de DTA est donc très mitigé : la vue indexée est plus volumineuse et un index inutile est créé. Mais il est vrai que DTA agit en bon logicien alors que nous pauvres humains, travaillons en sémanticiens. Bref, ce n'est pas encore demain que l'humain sera remplacé par la machine !