

Performance du SQL dynamique avec DB2 pour z/OS Version 8

Thomas Baumann, La Mobilière (Suisse)

thomas.baumann@mobi.ch



**Réunion du Guide DB2A
Jeudi 3 avril 2008
Infotel, Bagnolet (93)
France**

La Mobilière
Assurances & prévoyance



This presentation describes a methodical approach to database tuning, which has been developed, tested and used for emergency performance tuning after an application's rollout, but has also been shown to be quite helpful for routine database monitoring. We will discuss answers to the following three questions derived from commands, metrics and tools available in each shop:

- a) Are the application's critical queries being served in the most effective manner?
- b) Is DB2 making optimal use of resources so as to reach the desired performance levels?
- c) Are there enough primary resources available for DB2's consumption and are they configured adequately, given the current workload?

Objectifs



l'auditeur

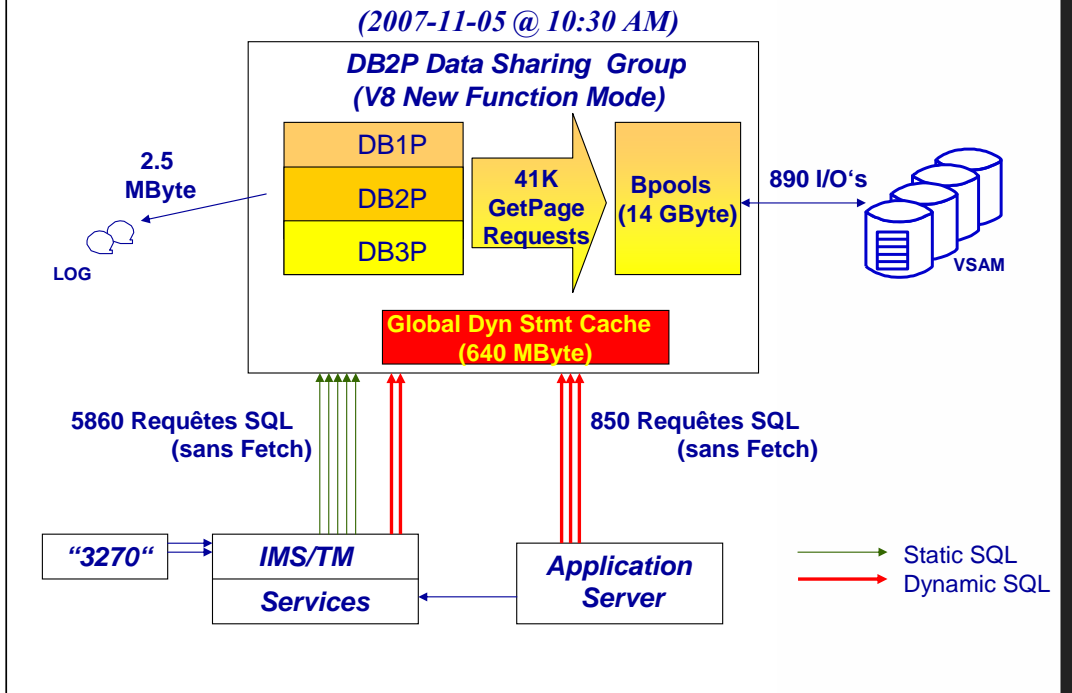
- **Sait aborder le tuning avec une approche méthodologique**
- **Apprend à connaître les spécificités du tuning pour le SQL dynamique**
- **Découvre les secrets du cache pour le SQL dynamique**
- **Obtient un bref aperçu des outils disponibles en matière de diagnostic de la performance du SQL dynamique**

First, we will discuss how to identify those ‘queries from hell’ which could potentially monopolize your whole system. Then – with a finer granularity – we will discuss metrics and techniques to further optimize both your dynamic queries as your system as a whole.

Eventually, we will discuss the question of having enough system resources or if there is a need to increase your system’s overall size.

We will conclude with an overview of some of the tools available for dynamic query tuning.

Une seconde dans la vie du DB2 dans la Mobilière



This slide shows the workload at Swiss Mobiliar's OLTP environment as measured 07/18/2006.

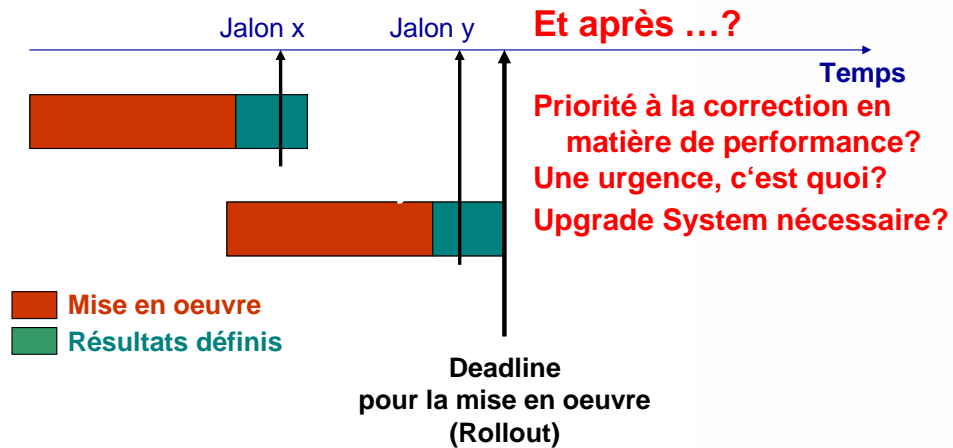
Disclaimer



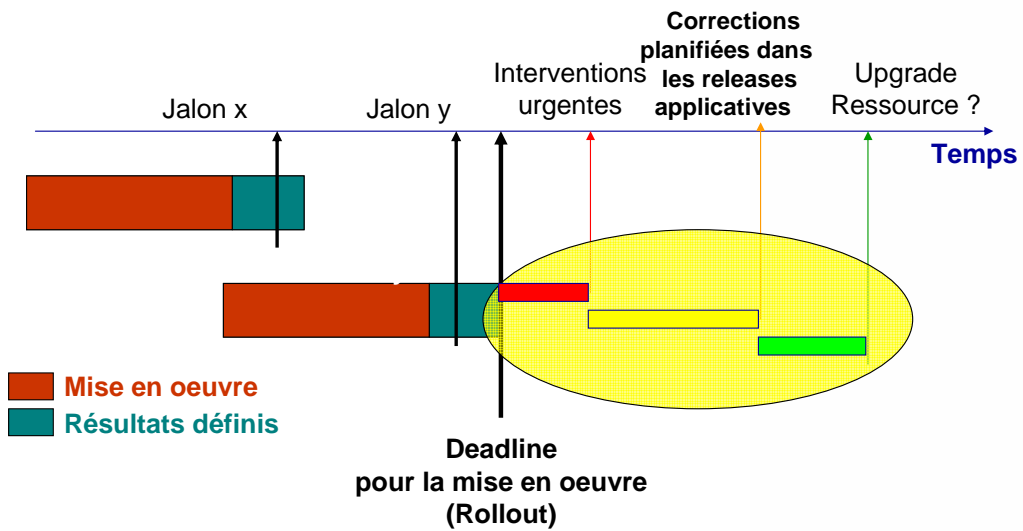
- The information contained in this presentation has not been submitted to any formal Swiss Mobiliar or other review and is distributed on an 'as is basis' without any warranty either express or implied. The use of this information is a user responsibility.
- The procedures, results and measurements presented in this paper were run in either the test and development environment or in the production environment at Swiss Mobiliar in Berne, Switzerland. They have not been tested under all circumstances. There is no guarantee that the same or similar results will be obtained elsewhere. Users attempting to adapt these procedures and data to their own environments do so at their own risk. All procedures presented have been designed for educational purposes only.



MVM (Modèle de procédé de la Mobilière):



After the application's rollout, the structured project methodology ends. During project consolidation, it is often hard to get sufficient resources to fix performance problems, because priority is often given to correct functional deficiencies only.



The methodology presented in this paper starts here: It defines milestones with performance metrics that need to be met. Milestones with a scope ranging from a single SQL statement to the overall performance of the application and the database subsystem. If the criterions of the first step were missed, an emergency corrective action is automatically scheduled.

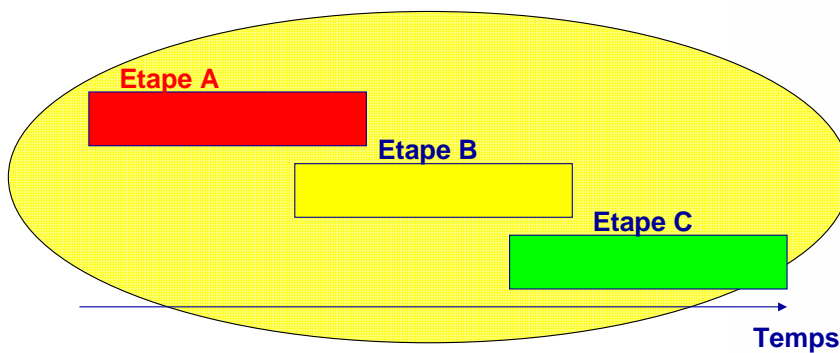
In the remainder of this presentation, we will discuss this methodology, and we will present you a tool which incorporates this methodical approach.



Méthodologie pour SQL Troubleshooting Tuning

➤ Etape A: Y a-t-il des requêtes SQL extrêmement critiques?

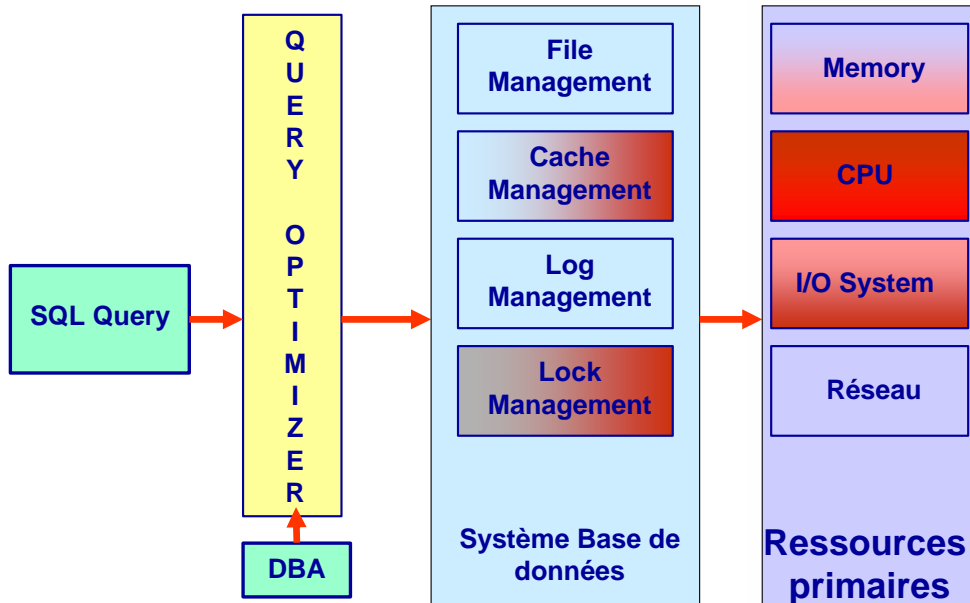
- Etape B: La charge de travail des gestionnaires de base de données est-elle soutenue de façon adéquate?
- Etape C: Les ressources système sont-elles suffisantes (CPU, Memory)? De la redondance pourrait-elle aider (MQT)?



This BMW troubleshooting (or emergency database performance) tuning methodology is divided in three steps: In each of these steps, there is one single question which is analyzed in full detail, and the problems detected during this step must be resolved before entering into the next step.

Step 1 detects extremely inefficient queries (*,queries from hell'*). Once they are corrected, the next step is initiated: Different thresholds answer the question '*are my queries served the optimal way?*'. This might concern both SQL queries and database runtime system features and parameters. Only in the third step, after all the problems detected in the first two steps are fully resolved, the question '*Do my queries get sufficient resources ?*' will be asked.

Etape A: Y a-t-il des requêtes SQL extrêmement critiques?



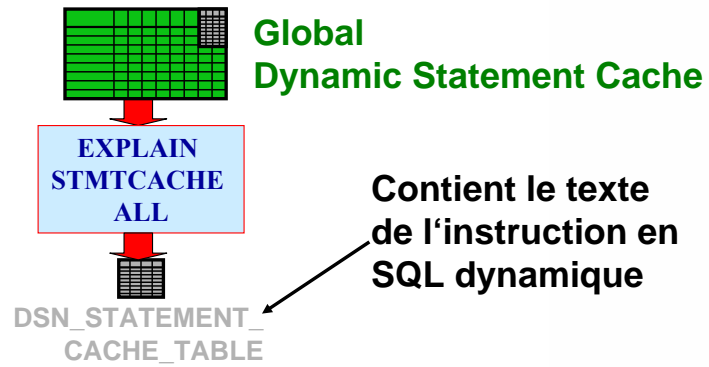
In a system with single very inefficient SQL statements, these might be the typical symptoms: A very high CPU usage rate, an I/O system close at the limits, bufferpools which do not reach their performance goals, and some locking problems.

All these 'trouble areas' are marked in red colour – the more red the more urgent they want your attention.

And all caused by a few inefficient SQL statements. That is why we start investigating this system from left to right, from SQL queries to basic resources and not vice versa.



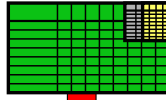
EXPLAIN STMTCACHE ALL



Analysis of the global dynamic statement cache has become easy in DB2 for z/OS V8: Simply define the `dsn_statement_cache_table`, and populate it by executing `EXPLAIN STMTCACHE ALL`.



EXPLAIN STMTCACHE ALL
avec **-STA TRACE(MON) CLASS(1) IFCID(318)**



Global
Dynamic Statement
Cache

“The collection of statistics for statements in the dynamic statement cache can increase the processing cost for those statements. When IFCID 0318 is inactive, DB2 tracks the statements in the dynamic statement cache, but does not accumulate the statistics as those statements are used. When you are not actively monitoring the cache, you should turn off the trace for IFCID 0318.”

performance

The collection of statistics for statements in the dynamic statement cache can increase the processing cost for those statements. When IFCID 0318 is inactive, DB2 tracks the statements in the dynamic statement cache, but does not accumulate the statistics as those statements are used. When you are not actively monitoring the cache, you should turn off the trace for IFCID 0318.

BMW Etape A1: Trouver les „Requêtes démoniaques“



Identifier les requêtes les plus gourmandes en CPU:

CPU par Exécution de requête > 0.5 sec

& \sum_h CPU par requête > 5 min

```
SELECT STMT_TEXT
FROM DSN_STATEMENT_CACHE_TABLE
WHERE STAT_CPU > 300
AND STAT_CPU / STAT_EXEC > 0.5
```

This very first criterion identifies the most inefficient dynamic SQL statements the ones which should be attacked at first – the *queries from hell*.

The screenshot displays a performance analysis tool interface. At the top left, the word 'Ag' is visible. Below it is a table with the following data:

STAT_EXEC
1
4
3
23
6
5
2
1
4
10

To the right of the grid is another table with a single column containing repeated entries: '.G, C99996_G, C971...'. Below the grid, the text 'Compte les points noirs...' is displayed. The interface also includes a paperclip icon and a search bar at the top right.

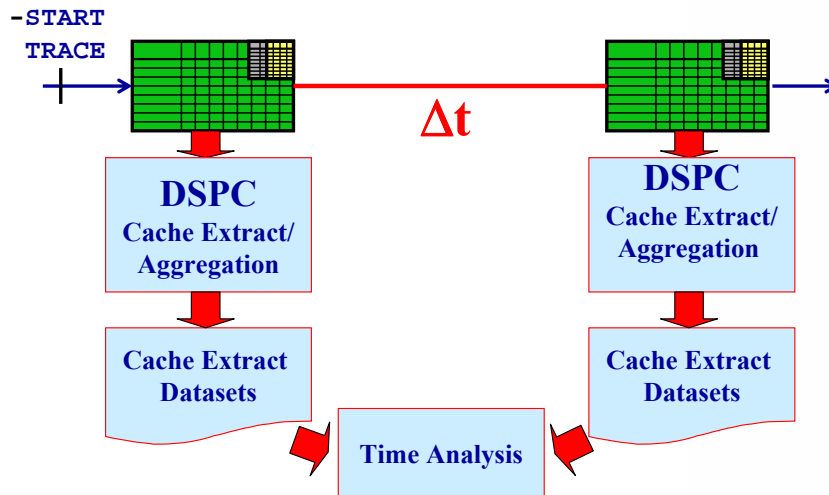
However, EXPLAIN STMTCACHE ALL produces information on an extremely granular level only (for each single cache entry). If you are not using parameter markers, or if you use many similar statements (for example, the only difference between two statements is the number of IN-list entries), you get many rows which in fact refer to the same kind of SQL statement.

Any kind of statement aggregation is essential to performance tuning. This presentation discusses two approaches to statement aggregation:

Statement Text aggregation (building statement groups according to identical statement query text while ignoring constants)

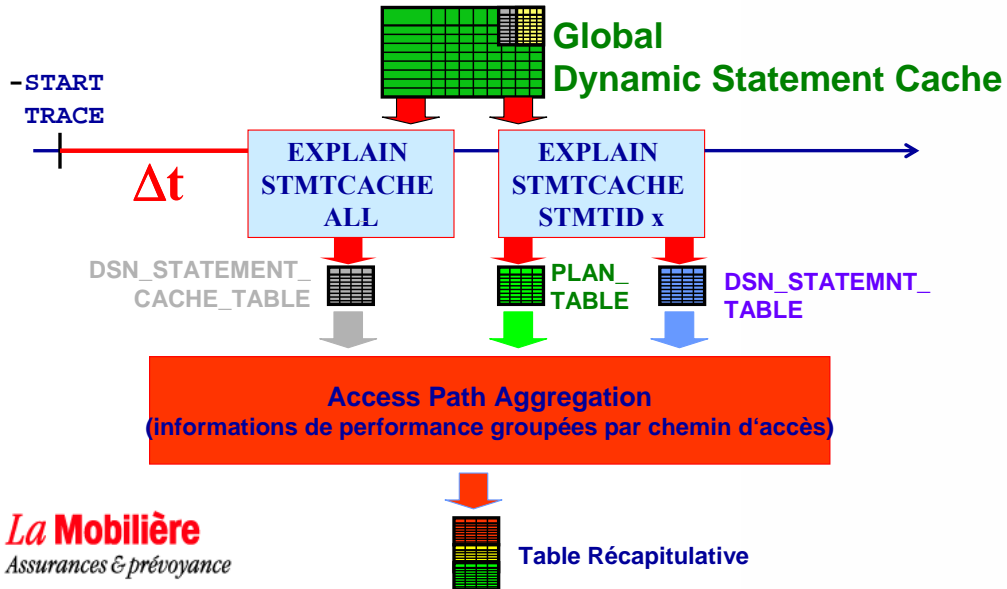
Access Path aggregation (building statement groups according to all queries with identical access paths).

DSPC: Agrégation basée sur le texte



Analysis of the global dynamic statement cache has become easy in DB2 for z/OS V8: Simply define the `dsn_statement_cache_table`, and populate it by executing `EXPLAIN STMTCACHE ALL`.

M3: Agrégation par chemins d'accès



Analysis of the global dynamic statement cache has become easy in DB2 for z/OS V8: Simply define the `dsn_statement_cache_table`, and populate it by executing `EXPLAIN STMTCACHE ALL`.

**BMW Etape A1:
Trouver les „Requêtes démoniaques“**



Critère de mesure A1: Utilisation de CPU extrêmement importante

Identifier les requêtes les plus gourmandes en CPU avec

CPU par exécution de requête > 0.5 sec

& \sum_h CPU par groupe d'instructions *) > 5 min

*) Un *groupe d'instructions* se compose de toutes les instructions SQL regroupées selon un critère de groupement (par exemple „texte d'instruction“ ou „chemin d'accès utilisé“).

This very first criterion identifies the most inefficient dynamic SQL statements the ones which should be attacked at first – the *queries from hell*.



Critère de mesure A2: Utilisation inefficace du „Dynamic Statement Cache“

Proportion de “cached Statements” non référencés

> 80% de tous les “cached Statements”

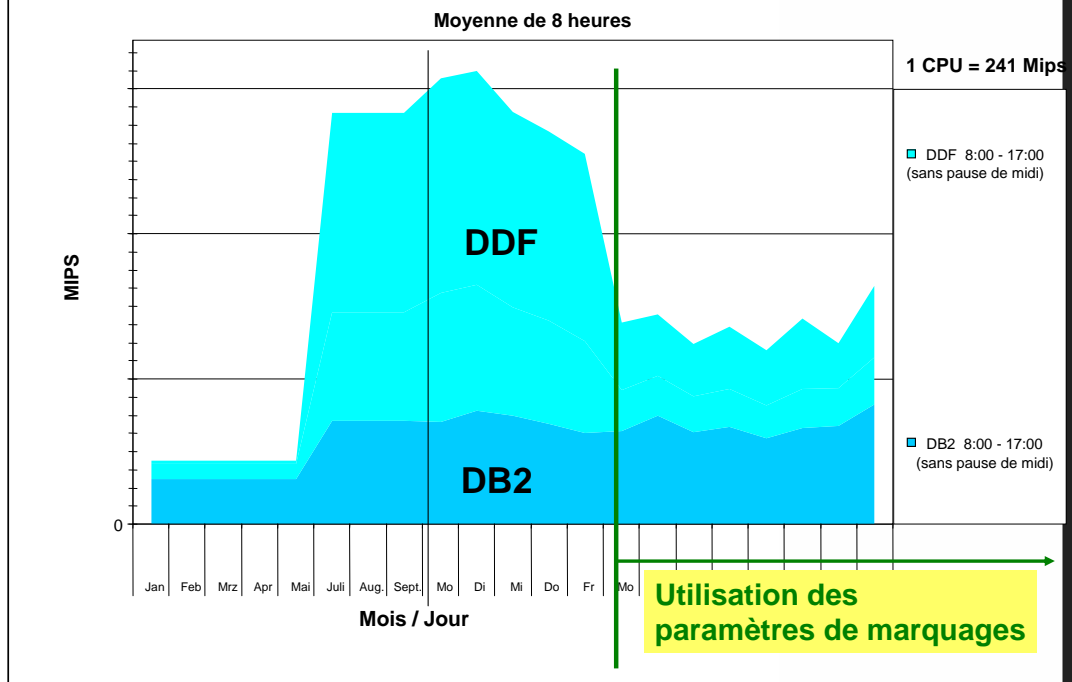
One of the largest problems in dynamic SQL tuning is the usage of parameter markers: If you don't use them, every single instance of a dynamic statement is considered to be a separate statement, and needs a complete bind process. And will be placed in the dynamic statement cache even if it will never be rereferenced.

On the other hand, if you use parameter markers for all kind of values, the optimizer can not benefit from frequency values statistics.

In general, you should use parameter markers for all numbers and names which are really variable, such as client numbers, customer names, addresses, etc.

The second step in our tuning approach counts the number of unreferenced statements within the cache and compares this number with the total number of cached statements:

BMW Etape A2: Paramètres de marquage manquants



One of the largest problems in dynamic SQL tuning is the usage of parameter markers: If you don't use them, every single instance of a dynamic statement is considered to be a separate statement, and needs a complete bind process. And will be placed in the dynamic statement cache even if it will never be rereferenced.

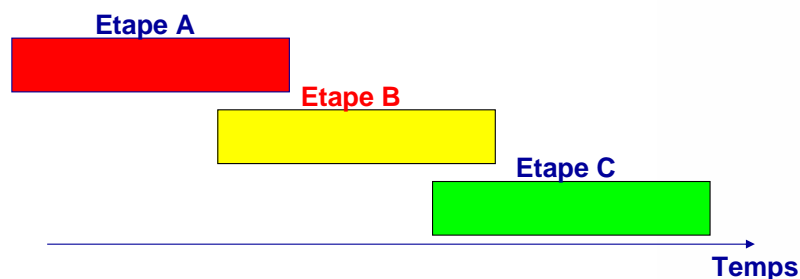
On the other hand, if you use parameter markers for all kind of values, the optimizer can not benefit from frequency values statistics.

In general, you should use parameter markers for all numbers and names which are really variable, such as client numbers, customer names, addresses, etc.

The second step in our tuning approach counts the number of unreferenced statements within the cache and compares this number with the total number of cached statements:

Méthodologie pour SQL Troubleshooting Tuning

- Etape A: Y a-t-il des requêtes SQL extrêmement critiques?
- **Etape B: La charge de travail des gestionnaires de base de données est-elle soutenue de façon adéquate?**
- Etape C: Les ressources système sont-elles suffisantes (CPU,Memory)?
De la redondance pourrait-elle aider (MQT)?

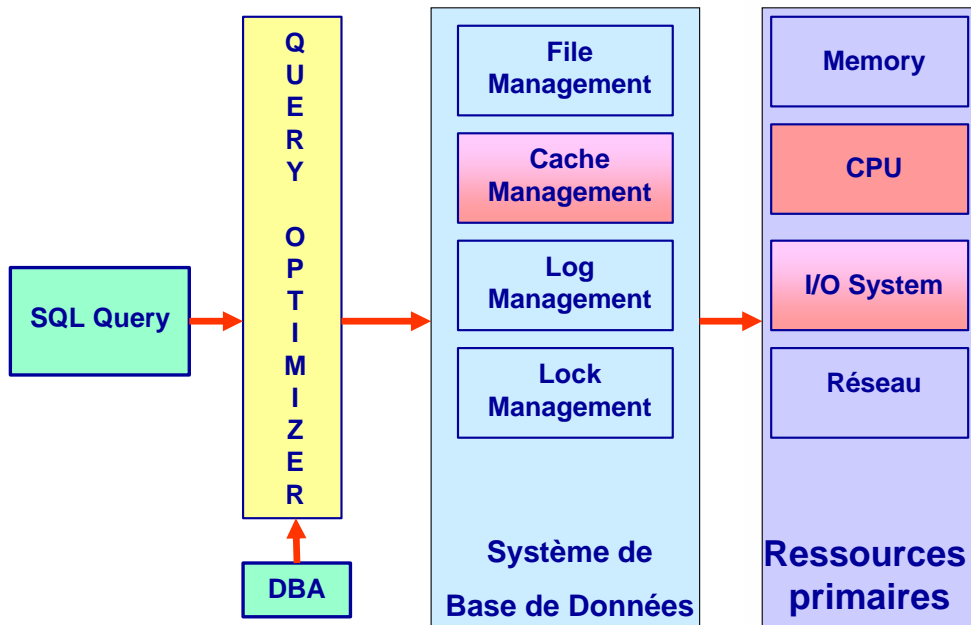


Now the first step has been done. The most critical queries are corrected. If they would still be in the system, there is a large probability that they would critically impact the results of the analysis of this second step – and leading to the wrong performance tuning actions.

If this methodology is used for routine monitoring, the first step should not identify any candidates.

Optimization opportunities detected during this second step do not require immediate (emergency) actions, but should be corrected until the next application release or system maintenance window. .

Etape B: La charge de travail des gestionnaires de base de données est-elle soutenue de façon efficiente?



After tuning the most urgent SQL problems, there are still some areas covered in red which require your attention. So we don't want to lose too much time and start to resolve these problems immediately.



$\frac{\text{Nombre de lignes analysées}}{\text{Nombre de lignes "fetched"}} > 10$

$\& \sum_h \text{CPU par groupe d'instructions}^*) > 30 \text{ sec}$

*) Un *groupe d'instructions* se compose de toutes les instructions SQL regroupées selon un critère de groupement (par exemple „texte d'instruction“ ou „chemin d'accès utilisé“).

This criterion lists SQL statements which need a certain (too large) amount of rows to be analyzed before finding a row which corresponds to the where predicates. This might have different reasons:

- no indexes available which cover the query's predicates

- stage 2 predicates

- old or insufficient catalog statistics

etc.

BMW Etape B1: Prédicats inefficients



```
SQL Statement Performance Information
COMMAND ==>                                SCROLL ==> CSR
ID: DB3P0424 10301045                      Extract from : 2006.04.24 10:30:53
                                                to          : 2006.04.24 10:45:15
Aggr. ID : 1.857.158                        in Cache since : 2006.04.22 09:03:18
SQL Type : SELECT                           # Stmt execs : 13
-----
                                         Total          Average
CPU Time .....: 0:00:20.587893          0:00:01.583684
Elapsed Time .....: 0:00:23.529180          0:00:01.809936
# Synchronous Buffer reads .....: 16
# Getpage Operations .....: 727.677          55.975
# Rows examined for Statement ...: 1.211.327          93.179
# Rows returned for Statement ..: 39              3
# Sorts performed for Statement .: 0
# IDX Scans performed for Statement : 1.709          131
# TS Scans performed for Statement : 0              0
Wait Time Synchronous I/O .....: 0:00:00.356062          0:00:00.027389
```



This criterion lists SQL statements which need a certain (too large) amount of rows to be analyzed before finding a row which corresponds to the where predicates. This might have different reasons:

- no indexes available which cover the query's predicates
- stage 2 predicates
- old or insufficient catalog statistics

etc.



Critère de mesure B2: Eviter (Sort-) Workfiles

Identifier et optimiser le SQL avec

**(Accès à la table DSNWFQB ou
Nombre de tris > 0)**

& \sum_h CPU par groupe d'instructions*) > 30 sec

*) Un *groupe d'instructions* se compose de toutes les instructions SQL regroupées selon un critère de groupement (par exemple „texte d'instruction“ ou „chemin d'accès utilisé“).

The DSNWFQB table is a table which you or your DBA colleagues probably never have defined. It represents the materialization of a query block during query execution (a *workfile* temporarily created by DB2), and has a great potential of causing performance problems.



Critère de mesure B3: forte dépense de verrouillage

Identifier et optimiser les requêtes SQL avec

*Wait Time Lock par Instruction SQL > 0.01 sec et
Wait Time Lock > 5% du Elapsed Time*

& \sum_h Elapsed Time par groupe d'instructions) > 30 s*

*) Un *groupe d'instructions* se compose de toutes les instructions SQL regroupées selon un critère de groupement (par exemple „texte d'instruction“ ou „chemin d'accès utilisé“).

Instead of looking at queries which cause deadlocks and timeouts, the queries reported here show a high amount of time spent in waiting for locks, but without necessarily reaching the timeout limits.



Critère de mesure B4: Problèmes du Query Optimizer

Identifier, analyser et optimiser les requêtes SQL avec

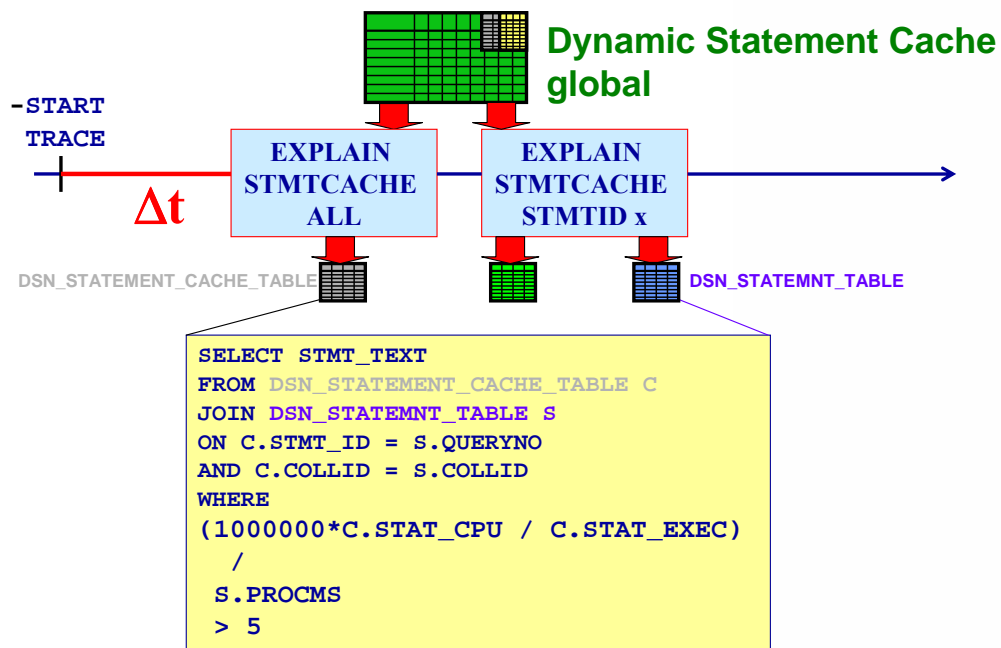
$$\frac{\text{CPU mesuré (dsn_statement_cache_table)}}{\text{CPU estimé (dsn_statemnt_table)}} > 5$$
$$\& \sum_h \text{CPU par groupe d'instructions}^*) > 20 \text{ sec}$$

*) Un *groupe d'instructions* se compose de toutes les instructions SQL regroupées selon un critère de groupement (par exemple „texte d'instruction“ ou „chemin d'accès utilisé“).

If the optimizer chooses a wrong access path, the final cpu measurements are very often far away from what it initially estimated.

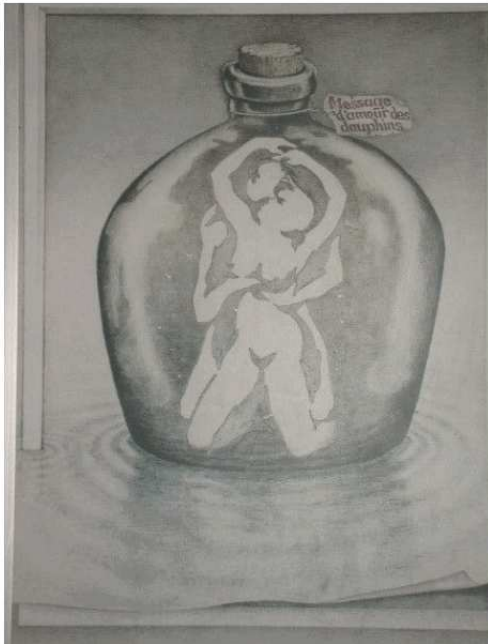
The ratio of measured compared to estimated CPU is a very good starting point to identify the queries which have inefficient access paths, often due to missing or wrong statistical information at the time the query was initially boud or prepared.

Comparaison de l'estimation de performance avec le résultat mesuré



This is how to get the information to calculate the *measured vs. estimated cpu ratio*. The EXPLAIN STMTCACHE STMTID x statement does not recalculate the access path, but does insert the originally calculated access path information into the plan_table, and the original cpu estimate into the dsn_statement_table's PROCMS column.

Regarde d'abord cette image en te concentrant. Que vois-tu ?
Lis l'explication en dessous pour comprendre ce que tu as réellement vu !
Intéressant non !



Des études ont montré que les enfants ne reconnaissent pas cette image „intime“, car leur mémoire ne connaît pas encore de telle situation.

Ce que les enfants voient, ce sont 9 dauphins.

Les vois-tu aussi ?

This typical response time tuning scenario starts by analyzing the PM07 (elapsed time) report. Special attention is given to statement groups with a high value of sync I/O operations per query (SYNCIO_PER_EXEC) and a low number of processed rows (ROW_PROCESSED_PER_EXEC).

Let us look at the second row from the bottom: For an average query execution, 33 rows are fetched into the application, which causes 28 getpage requests and 10 synchronous I/O operations per query execution. There were 246 executions of this access path (NO_OF_EXECUTIONS), and there exist 6 different statements within this statement group.



Critère de mesure B5: grandeur du „Cache Dynamic SQL Statement“

Cache residency time minimal < 10 min
→ Cache sous-dimensionné

Formule approximative:

$$\frac{\# \text{ Stmts en Cache} - \frac{\# \text{ Stmts en Cache} * \text{ hit ratio}}{\text{Log} (\# \text{ Stmts en Cache})}}{\sum \text{ nouvelles references par sec.}} < 600$$

You might wonder how we found an upper limit of 10 minutes to be a good value to determine the dynamic statement cache's size. In fact we also do.

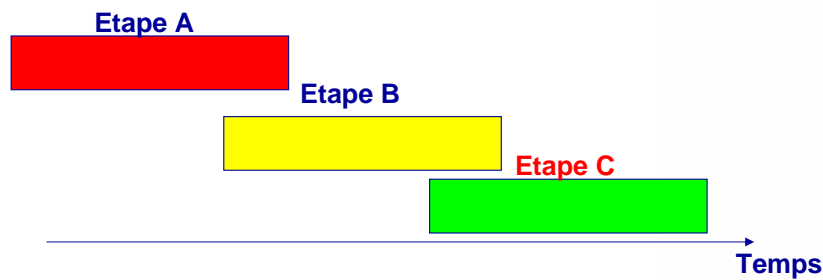
What we tested and measured was the impact of a larger dynamic statement cache (with a minimum cache residency time of up to 60 minutes). The benefit for the overall system was not much better than with a cache size according to the 10 minutes limit. However, with a smaller dynamic statement cache, the overall numbers and specifically the values measured for prepare-statements, were higher.

So we decided to have 10 minutes as our performance goal.



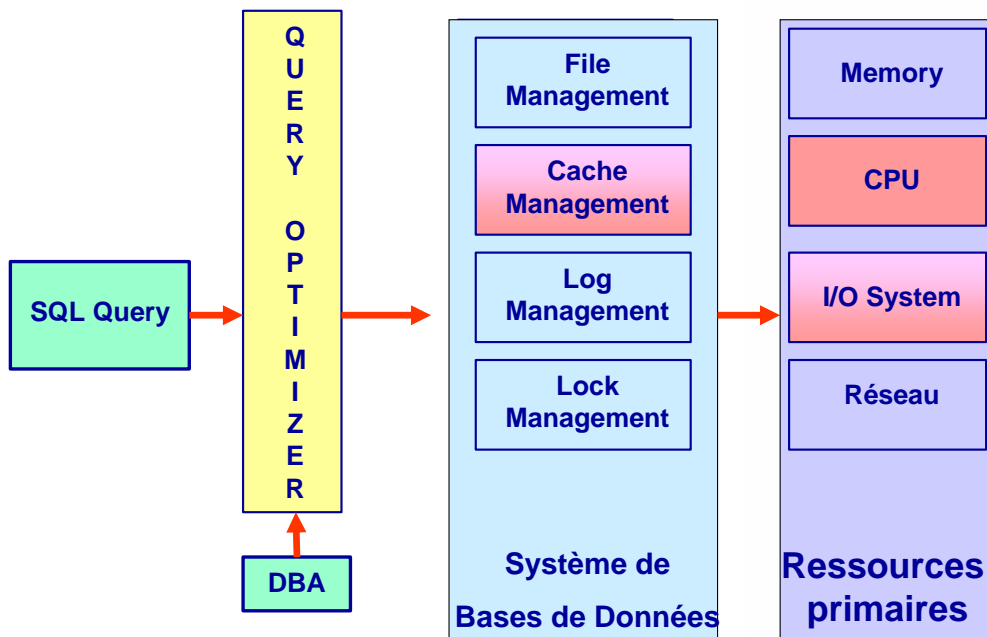
Méthodologie pour SQL Troubleshooting Tuning

- Etape A: Y a-t-il des requêtes SQL extrêmement critiques?
- Etape B: La charge de travail des gestionnaires de base de données est-elle soutenue de façon adéquate ?
- **Etape C: Les ressources système sont-elles suffisantes (CPU,Memory)?
De la redondance pourrait-elle aider (MQT)?**



Now the second step of the methodology has been finished. Again – all the problems identified must be resolved before the third step can start..

Etape C: les ressources système sont-elles suffisantes?



Now it is time to concentrate on primary resources. Or – alternatively – you can start to investigate the most I/O consuming queries, detecting unused indexes, selecting candidates for materialized query tables, etc.



Goulots d'étranglement identifiés :

C1) Contrôle Memory :

Bufferpool re-read en l'espace de 2min > 0, Paging ?

C2) Contrôle I/O :

Moyenne Sync I/O Suspension > 8msec

C3) Contrôle Réseau:

End-to-end response time/temps de traitement SQL > 2

C4) Contrôle CPU :

'Wait for CPU' / temps de traitement SQL > 0.5

Some simple resource checks.

Dynamic SQL outils de diagnostic des performances



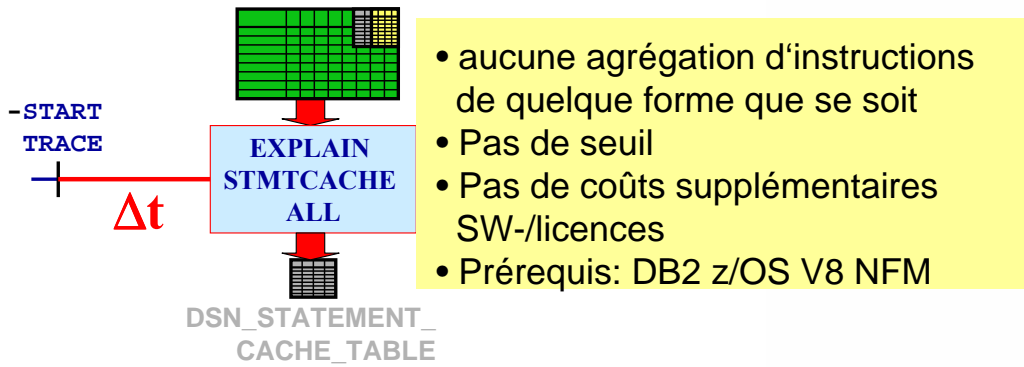
- EXPLAIN STMTCACHE ALL (native SQL Statement)
- M3
Condition préalable: DB2 z/OS V8 NFM
- DSPC Dynamic Statement Performance Control
Condition préalable: DB2 z/OS V6ff
- Quelques „Statement Level Query Monitors“
(ne sont pas basés sur Dynamic Statement Cache)
 - Apptune for DB2 (BMC)
 - CA Detector for DB2 (CA)
 - DB2 Query Monitor (IBM)



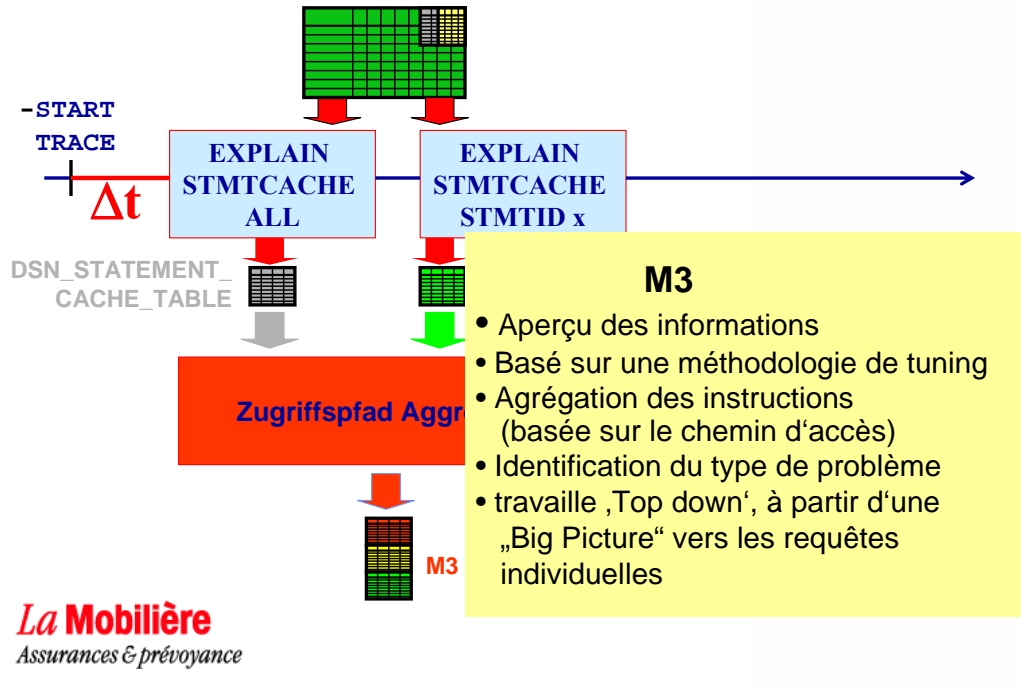
These are some of the tools available. We will shortly compare them, and will finish with some hints when to use what.



EXPLAIN STMTCACHE ALL



Analysis of the global dynamic statement cache has become easy in DB2 for z/OS V8: Simply define the `dsn_statement_cache_table`, and populate it by executing `EXPLAIN STMTCACHE ALL`.



The idea which was driving M3 development was the implementation of the tuning methodology described in this presentation.

M3



Advanced Query Tuning

M3 V2.2

Refresh

Dynamic SQL Performance Diagnosis for DB2P

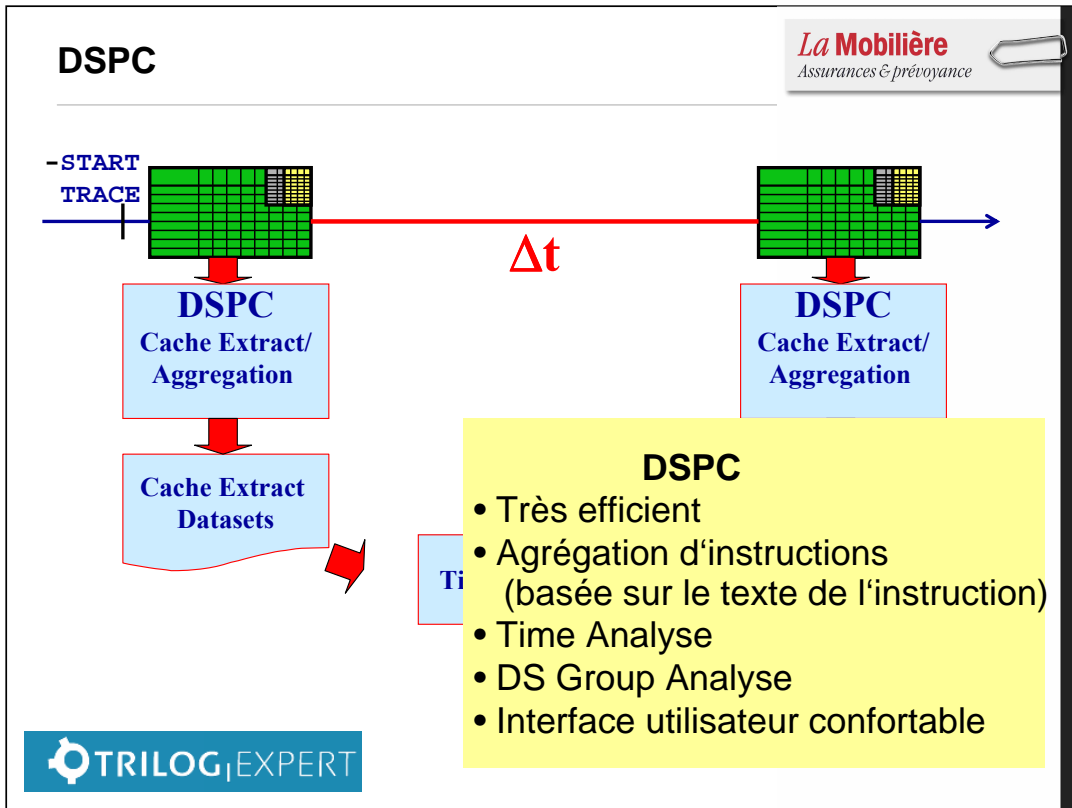
Trace start 2007/9/24 10:11 am
Cache snapshot 2007/9/24 10:27 am

Emergency Tuning		Overall	% Change	Critical Applications	% Change
ETD1	No of Stmt Groups with high CPU / Statement Ratio	0	0%	0	0%
ETD2	% of Unreferenced Cached Statements	56%	-5%	66%	0%



M3
Summary
Table

PMO3	Optimization Potential	62%	3%	57%	2%
PMO4	Unused Indexes	20	11%	n/a	n/a
PMO5	Total CPU Time per second	0.08500	4%	0.07147	-7%
PMO6	Total Sync I/O Time per second	1.59921	11%	1.38093	8%
PMO7	Total Elapsed Time per second	1.84253	11%	1.59572	7%
PMO8	CPU Seconds / Processed Row	0.00004	-50%	0.00007	-13%
PMO9	Elapsed Seconds / Processed Row	0.00096	-45%	0.00166	3%
DynSQL Performance Index (100=31.08.2007) DSPi		206.19	92.59%	107.03	118.32
					7.52%



The idea which was driving M3 development was the implementation of the tuning methodology described in this presentation.



```

DSPC 1.5.0 - SQL Statement Performance Information
COMMAND ==> SCROLL ==> CSR
ID: SYS3      DB2P      Extract from   : 2007.09.24 13:19:19
                  to     : 2007.09.24 13:21:40
Aggr. ID : 113.977   in Cache since : 2007.09.24 13:19:45
SQL Type : SELECT   # Stmt execs  : 8
-----
                                Total      Average
                                More:
CPU Time .....: 0:00:03.133727  0:00:00.391715  +
Elapsed Time .....: 0:00:03.989092  0:00:00.498636
# Synchronous Buffer reads .....: 0 0
# Getpage Operations .....: 9.695 1.212
# Rows examined for Statement .....: 490.821 61.353
# Rows returned for Statement .....: 2.399 300
# Sorts performed for Statement .....: 8 1
# IDX Scans performed for Statement : 0 0
# TS Scans performed for Statement .: 16 2
    
```





DSPC 1.5.0 - SQL Statement Text Information Row 1 to 19 of 43
COMMAND ==> SCROLL ==> CSR

ID : SYS3 DB2P Extract from : 2007.09.24 13:19:19
to : 2007.09.24 13:21:40
SQL Stmt ID: 113.977 in Cache since: 2007.09.24 13:19:45
Executed : 8 Total CPU : 00:00:03.133727
Client : U901550 Transaction : abtntsrv1.exe
Package : SYSSH200

```
-----  
SELECT *  
FROM DB2PVIEW.VEOPENPAR  
WHERE EDITINGSTATUS  
IN (?, ?)  
AND ASSIGNMENTSTATUS IS NULL  
AND LASTCHANGEDTHROUGH  
IN (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,  
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,  
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,  
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,  
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)  
UNION  
SELECT *  
FROM DB2PVIEW.VEOPENPAR  
WHERE EDITINGSTATUS  
IN (?, ?)  
AND ASSIGNMENTSTATUS IS NULL
```



Quel outil pour quelle situation ?



- **Exemple 1: Emergency Tuning (DSPC)**
- **Exemple 2: Response Time Tuning (DSPC ou M3)**
- **Exemple 3: A la recherche des index non utilisés (M3)**

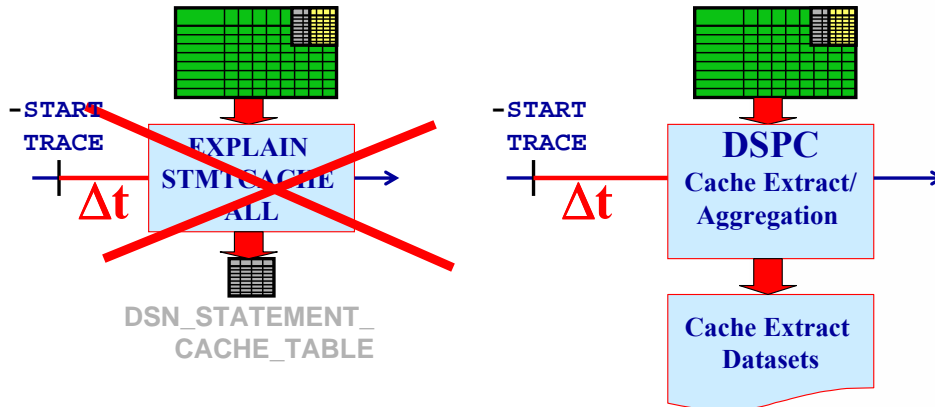
These are some of the tools available. We will shortly compare them, and will finish with some hints when to use what.

Exemple 1: Emergency Tuning



```
ICH70001I U104958 LAST ACCESS AT 03:24:07 ON TUESDAY, FEBRUARY 12,2007  
IKJ56455I U104958 LOGON IN PROGRESS AT 07:52:44 ON FEBRUARY 13, 2007  
***
```

Emergency Tuning: Oubliez EXPLAIN STMTCACHE ALL



„Même avec une priorité maximale donnée au Job et un Statement Cache de grandeur moyenne, cette instruction peut durer „une éternité“ lorsque le système est surchargé“

„Avec une priorité maximale donnée au Job, le „Cache Extract“ se termine en quelques secondes. Agrégations et Jobs suivants sont possibles sur des LPAR différents.“

Exemple - Emergency Tuning



```

DSPC 1.5.0 - Single Cache Extracts
COMMAND ==>
Row 1 from 8
SCROLL ==> CSR

Line Commands: S Single SQL Statements      A Aggregated SQL Statements
                C Client X TRAN P Package    T Table Aggregation Informations
                TA Time Analysis              PI Performance Informations

LC System  DB2      Extract  from    HIT%    #SQL    Total    Total
 *         *         Date     Time    %       #       Elapsed  CPU
-----
s_ SYS3     DB2P      2007.10.29 08:07:00 97.64    9494   00:22:18.34 00:01:16.92
_  SYS3     DB2P      2007.09.24 13:21:40 94.09    17659  00:02:07.30 00:00:12.28
_  SYS3     DB2P      2007.09.24 13:19:19 94.08    17655  00:00:18.03 00:00:01.30
_  SYS3     DB2P      2007.04.16 13:33:07 94.42    20636  00:11:19.86 00:00:38.25
_  SYS3     DB2P      2007.04.16 13:25:58 94.42    20417  00:00:08.84 00:00:00.42
_  SYS3     DB2P      2007.03.27 15:28:56 94.56    20462  00:17:25.55 00:01:55.90
_  SYS3     DB2P      2007.03.27 15:18:52 94.56    20528  00:00:55.40 00:00:11.76
_  SYS3     DB2P      2007.03.27 07:51:27 94.50    27812  00:00:00.00 00:00:00.00
***** Bottom of data *****
    
```



Exemple - Emergency Tuning



```
DSPC 1.5.0 - SQL Information Display                               Row 1 from 5
COMMAND ==>                                                       SCROLL ==> CSR
ID: SYS3      DB2P      Extract from      : 2007.10.29 08:07:00
                  to      : 2007.10.29 08:07:00

Line Commands: PI Performance Information   T SQL Statement Text
                CI Client Information       TE / TV / TB Edit/View/Browse Text
                AC Add challenges           EC / VC / BC Edit/View/Browse Chall.
                CC Change challenges

LC      Total CPU   Total Elapsed  Exec#  Client  Package Table   SQL Type Cha
-----
PI 00:00:00.23744 00:00:02.324969  72 U901550 SYSSH20 TE0_TPA0 UPDATE
_  00:00:00.12879 00:00:01.769088  249 U901550 SYSSH20 TE0TRAC INSERT
  00:00:00.03823 00:00:00.091792  332 U901550 SYSSH20 TE0_TPA5 SELECT
  00:00:00.01074 00:00:00.081027  15 U901550 SYSSH20 TE0_TVER UPDATE
  00:00:00.00729 00:00:00.049892  43 U901550 SYSSH20 TE0_TPA0 SELECT
***** Bottom of data *****
```



Exemple - Emergency Tuning



```
DSPC 1.5.0 - SQL Statement Performance Information
COMMAND ==> _                                SCROLL ==> CSR

ID: SYS3      DB2P      Extract from      : 2007.10.29 08:07:00
                               to                : 2007.10.29 08:07:00
Aggr. ID : 26      in Cache since   : 2007.10.28 03:36:27
SQL Type : UPDATE   # Stmt execs    : 72
-----

                               Total          Average
CPU Time .....: 0:00:00.237444    0:00:00.003297
Elapsed Time .....: 0:00:02.324969    0:00:00.032291
# Synchronous Buffer reads .....: 231          3
# Getpage Operations .....: 399          6
# Rows examined for Statement .....: 0            0
# Rows returned for Statement .....: 73           1
# Sorts performed for Statement .....: 0            0
# IDX Scans performed for Statement .....: 73           1
# TS Scans performed for Statement ..: 0            0

Wait Time Synchronous I/O .....: 0:00:01.662263    0:00:00.023086
Wait Time LOCK and LATCH request ...: 0:00:00.000020    0:00:00.000000
Wait Time Sync. Exec. Unit Switch ...: 0:00:00.000000    0:00:00.000000
Wait Time Global Locks .....: 0:00:00.000000    0:00:00.000000
Wait Time Read by another Thread ...: 0:00:00.000000    0:00:00.000000
Wait Time Write by another Thread ...: 0:00:00.000000    0:00:00.000000

Isolation Bind Option on init.prep. : UR (UNCOMMITTED READ)
Currentdata Bind Option .....: NO
```

INTEC|EXPERT

Exemple - Emergency Tuning



```
DSPC 1.5.0 - SQL Statement Text Information          Row 1 to 4 of 4
COMMAND ==> _                                     SCROLL ==> CSR
ID          : SYS3          DB2P          Extract from : 2007.10.29 08:07:00
                                                to       : 2007.10.29 08:07:00
SQL Stmt ID: 26          in Cache since: 2007.10.28 03:36:27
Executed   : 72          Total CPU     : 00:00:00.237444
Client     : U901550     Transaction  : abtntsrv4.exe
Package    : SYSSH200
-----
update    DB2PVIEW.VE008001
set       C99993 = ?, C99994 = ?
where    (C95826 = ?)
and      (C95836 = ?)
***** Bottom of data *****
```

Exemple 2: Response Time Tuning

Emergency Tuning		Overall	% Change	Crit. Applications	% Change
ET01	No of Stmt Groups with high CPU / Statement Ratio	0	0%	0	0%
ET02	% of Unreferenced Cached Statements	58%	-2%	69%	3%
Performance Control: SQL Statement		Overall	% Change	Crit. Appl.	% Change
PC01	No of Stmt Groups with Inefficient Predicates	0	0%	0	0%
PC02	No of Stmt Groups with Resource Intensive Sorts	0	-100%	0	0%
PC03	No of Stmt Groups with Intensive Locking	0	0%	0	0%
PC04	No of Stmt Groups with Optimizer Challenges	0	0%	0	0%
Performance Control: Memory Management		Overall	% Change	Crit. Appl.	% Change
PC05	Dyn Statement Cache Min Residency Time (min)	46	-13%		
PC06	Local Buffer Pools Min Residency Time (min)	21	50%		
PC07	No of Failed Writes to Group Buffer Pool	0	0%		
PC08	No of Sync I/Os caused by Cross-Invalidation (per min)	0	0%		
Performance Control: Data Management		Overall	% Change	Crit. Appl.	% Change
PC09	Getpage requests / second	2188	-14%	2128	19%
Performance Management		Overall	% Change	Crit. Appl.	% Change
PM01	Tspce/Index Runstats Efficiency	n/a	n/a	n/a	n/a
PM02	Tspce/Index Reorg Efficiency	n/a	n/a	n/a	n/a
PM03	Optimization Potential	65%	-13%	61%	-3%
PM04	Unused Indexes	29	4%	n/a	n/a
PM05	Total CPU Time per second	0.07964	-19%	0.07357	-3%
PM06	Total Sync I/O Time per second	1.65479	-40%	1.45778	-20%
PM07	Total Elapsed Time per second	2.00487	-35%	1.78193	-14%
PM08	CPU Seconds / Processed Row	0.00007	-22%	0.00007	-13%
PM09	Elapsed Seconds / Processed Row	0.00188	-40%	0.00173	-23%
DynSQL Performance Index (100=31.08.2007) DSPI		113.43	38.85%	116.63	19.25%

This typical response time tuning scenario starts by analyzing the PM07 (elapsed time) report. Special attention is given to statement groups with a high value of sync I/O operations per query (SYNCIO_PER_EXEC) and a low number of processed rows (ROW_PROCESSED_PER_EXEC).

Let us look at the second row from the bottom: For an average query execution, 33 rows are fetched into the application, which causes 28 getpage requests and 10 synchronous I/O operations per query execution. There were 246 executions of this access path (NO_OF_EXECUTIONS), and there exist 6 different statements within this statement group.

Exemple 2: M3 Elapsed Time Query Detail Report

PROGRAM_NAME	CURSQLID	TOTAL_ELAPSED	NO_OF_ENTRIES	NO_OF_EXECUTIONS	SYNCIO_PER_EXEC	GETPAGE_PER_EXEC	STAT_PROW
M71460	U102316	230.888734	767	3684			
SYSSH200	U901550	193.989246	1	19004			
SYSSH200	U901550	174.589542	79	13560			
SYSSH200	SAT00L	138.752031	39	73			
SYSSH200	U901550	112.265125	1	4939			
SYSSH200	U901550	61.853462	4	475			
SYSSH200	U901550	53.271022	2	1082			
SYSSH200	U901550	47.638874	1	1271	6	53	1
SYSSH200	U901550	44.604221	1	142	0	9	19005
SYSSH200	U901550	40.605472	1	19210			
SYSSH200	U901550	39.430029	6	246	1	9	1
SYSSH200	U901550	39.041934	1	1433	1	1379	7
SYSSH200	U901550	37.520746	7	426	1	17	4940
SYSSH200	U901550	37.206385	1472	3371	9	22	358
SYSSH200	U901550	35.639041	6	625	5	66	146241
SYSSH200	U901550	34.927996	1	4736	2	2	1272
SYSSH200	U901550	32.343580	6	246	28	1316	12761
SYSSH200	U901550	31.953123	1	6316	0	1	19211
					13	37	1078
					2	2	1434
					7	24	456
					0	5	1
					4	16	138
					0	2	4736
					10	28	33
					0	1	6316
				

Vue de détail du Rapport PM07
(groupe d'instructions,
trié selon elapsed time)
(6 entrées en cache, 246
exécution)

This typical response time tuning scenario starts by analyzing the PM07 (elapsed time) report. Special attention is given to statement groups with a high value of sync I/O operations per query (SYNCIO_PER_EXEC) and a low number of processed rows (ROW_PROCESSED_PER_EXEC).

Let us look at the second row from the bottom: For an average query execution, 33 rows are fetched into the application, which causes 28 getpage requests and 10 synchronous I/O operations per query execution. There were 246 executions of this access path (NO_OF_EXECUTIONS), and there exist 6 different statements within this statement group.

Exemple 2: M3 Elapsed Time Query Detail Report (ff)



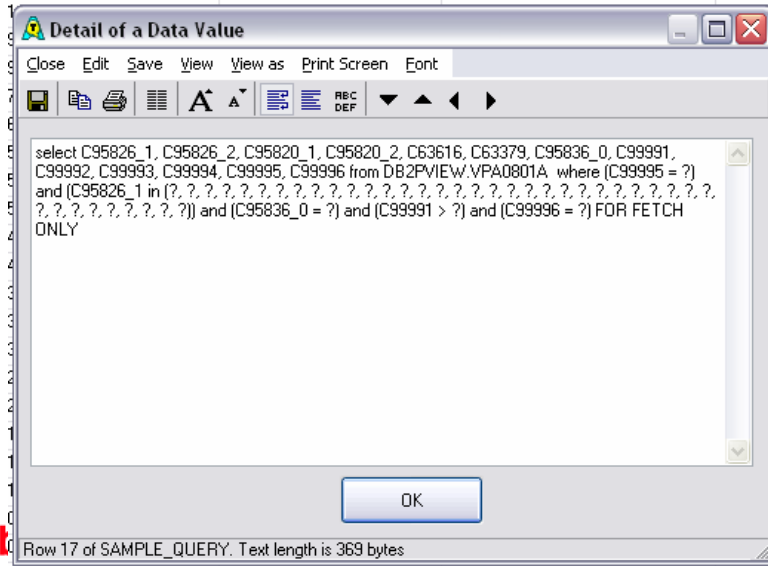
SAMPLE_QUERY	APIID
SELECT C97251_G, C97150_G, C97227_G, C97163_G, C97364_...	ODB2PVIEWMV714601RONS30N000DB2PROD×GESGBA6I3Y1D...
Insert into DB2PVIEW.VPA09412 (C95836_0, C95826, C95820, C9...	ODB2PRODTPA09410N
Insert into DB2PVIEW.VPA97922 (C95836, C95826, C95820, C958...	ODB2PRODTPA97920N
SELECT DB2PVIEW.VSAMELD1.C44301 FROM DB2PVIEW.VSA...	ODB2PROD×SAMELD2I1NL30N
Insert into DB2PVIEW.VPA0800A (C63306, C95809, C95836, C633...	ODB2PRODTPA08000N
select C95826, C99993 from DB2PVIEW.VEI08001 where (C99995...	ODB2PROD×EI08006N2N
select C95836_0, C95826, C95820, C95836_1, C95864, C99991, C...	ODB2PROD×PA09411I1N
Insert into DB2PVIEW.VVERTXT1 (C95836_1, C63391, C95820, C9...	ODB2PRODTVERTXT0N
delete from DB2PVIEW.VEQ97921 where (C99996 in [?, ?, ?, ?, ?..	ODB2PROD×EQ97924I2Y
Insert into DB2PVIEW.VEQ97921 (C95836, C95826, C95820, C958...	ODB2PRODTQ_TPA97920N
select C95836, C95826, C95820, C95815, C95891, C95832, C9584...	ODB2PROD×BA97923N3Y
Insert into DB2PVIEW.VVERDAT1 (C95836_0, C95826, C95820, C...	ODB2PRODTVERDAT0N
select C95826, C99993 from DB2PVIEW.VPA0800A where (C9999...	ODB2PROD×PA08007N2Y
update DB2PVIEW.VEQ08001 set C99993 = '2007-06-11-10.41.53...	ODB2PROD×EQ08002I2Y
select C97227, C99993 from DB2PVIEW.VEISORP1 where (C9999...	ODB2PROD×EISORP1N1N
Insert into DB2PVIEW.VEQTRAC1 (C40000, C40001, C43050, 1, C...	ODB2PRODTQTRAC0N
select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C633...	ODB2PROD×PA08013N3Y
Insert into DB2PVIEW.VEORTXT1 (C95836_1, C63391, C95820, C...	ODB2PRODTQ_TVERTXT0N

The access path used by this query is an index-only access (by index DB2PROD.XPA08013) with three columns as matching index columns. At first look, this seems to be okay. Could we make this access path any faster?

First of all, we analyze all 6 statement instances of this statement group: are they all looking similar?

Exmple 2: M3 Elapsed Time Query Detail Report (ff)

Contenu de l'une des instructions du groupe d'instructions sélectionné:



The screenshot shows a window titled "Detail of a Data Value" with a menu bar (Close, Edit, Save, View, View as, Print Screen, Font) and a toolbar. The main area contains a SQL query:

```
select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C63379, C95836_0, C99991,
C99992, C99993, C99994, C99995, C99996 from DB2PVIEW.VPA0801A where (C99995 = ?)
and (C95826_1 in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?)) and (C95836_0 = ?) and (C99991 > ?) and (C99996 = ?) FOR FETCH
ONLY
```

An "OK" button is at the bottom center. The status bar at the bottom left reads "Row 17 of SAMPLE_QUERY. Text length is 369 bytes".

A complete query text of one of the statements of this statement group.

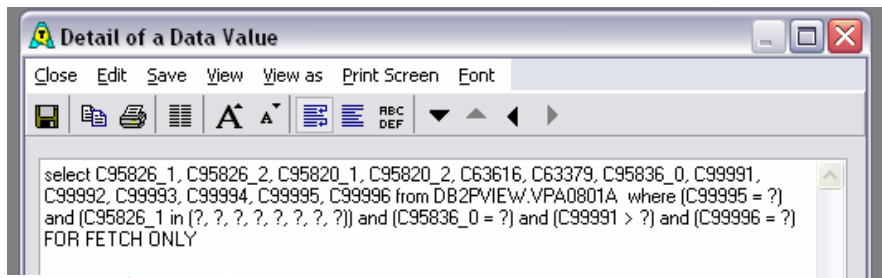
Exemple 2: M3 Instances Report



Présente toutes les instructions avec le même chemin d'accès:

STAT_EXEC	STAT_CPU	STAT_ELAP	STMT_TEXT
217	0.140316978155398	24.9742927700865	select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C633..
3	0.017192983072178	1.94238877389945	select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C633..
12	0.0268336618355676	3.45861363504447	select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C633..
12	0.0216637859157487	0.0297839539162084	select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C633..
1	0.00663515646375862	1.20040631387748	select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C633..
1	0.00446205307163444	0.738095417677187	select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C633..

La première instruction est exécutée le plus souvent:



Exemple 2: M3 Instances Report



Par souci d'exhaustivité, voici les autres instructions avec le même chemin d'accès:

```
select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C63379, C95836_0, C99991,
C99992, C99993, C99994, C99995, C99996 from DB2PVIEW.VPA0801A where (C99995 = ?)
and (C95826_1 in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?)) and (C95836_0 = ?) and (C99991 > ?) and (C99996 = ?) FOR FETCH
ONLY
```

```
select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C63379, C95836_0, C99991,
C99992, C99993, C99994, C99995, C99996 from DB2PVIEW.VPA0801A where (C99995 = ?)
and (C95826_1 in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?)) and (C95836_0 = ?) and (C99991 > ?) and (C99996 = ?) FOR FETCH
ONLY
```

```
select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C63379, C95836_0, C99991,
C99992, C99993, C99994, C99995, C99996 from DB2PVIEW.VPA0801A where ((C95826_1 in (?,
?, ?, ?, ?, ?, ?)) ) and ((C99995 = ?) ) and (C99996 = ?) FOR FETCH ONLY
```

```
select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C63379, C95836_0, C99991,
C99992, C99993, C99994, C99995, C99996 from DB2PVIEW.VPA0801A where (C99995 = ?)
and (C95826_1 in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?)) and (C95836_0 = ?) and (C99991 > ?)
and (C99996 = ?) FOR FETCH ONLY
```

```
select C95826_1, C95826_2, C95820_1, C95820_2, C63616, C63379, C95836_0, C99991,
C99992, C99993, C99994, C99995, C99996 from DB2PVIEW.VPA0801A where (C99995 = ?)
and (C95826_1 in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?)) and (C95836_0 =
?) and (C99991 > ?) and (C99996 = ?) FOR FETCH ONLY
```

Exemple 2: M3 Instances Report



Voici à quoi ressemble l'Index XPA08013:

DB2PROD.XPA08013

```
(C95826_1 ASC  
,C99995 ASC  
,C99996 ASC  
,C95826_2 ASC  
,C95820_1 ASC  
,C95820_2 ASC  
,C63616 ASC  
,C63379 ASC  
,C95836_0 ASC  
,C99991 ASC  
,C99992 ASC  
,C99993 ASC  
,C99994 ASC)
```

C95826_1, C99995, et C99996 sont utilisées en tant que colonnes de *matching index*.

Cela peut-il encore être amélioré ? – oui!
La séquence des index peut être changée comme suit:

C95836_0 en tant que quatrième colonne
C99991 en tant que cinquième colonne

5 parmi les 6 instances de requête profiteraient de cette modification; aucun impact sur la 6ème instance de requête

Exemple 2: M3 Index Report



D'autres requêtes (SQL dynamique) utilisent-elles cet index?
Avec plus de 3 colonnes utilisées en tant que *matching Index*?

APID
0DB2PROD\PA0801311Y
0DB2PROD\PA0801312Y
0DB2PROD\PA0801313Y
0DB2PROD\PA08013N2Y
0DB2PROD\PA08013N3Y

Le "M3 Index Report" montre tous les chemins d'accès qui contiennent l'index concerné. Le nombre de colonnes utilisées en tant que *matching Index* varie entre 1 et 3.

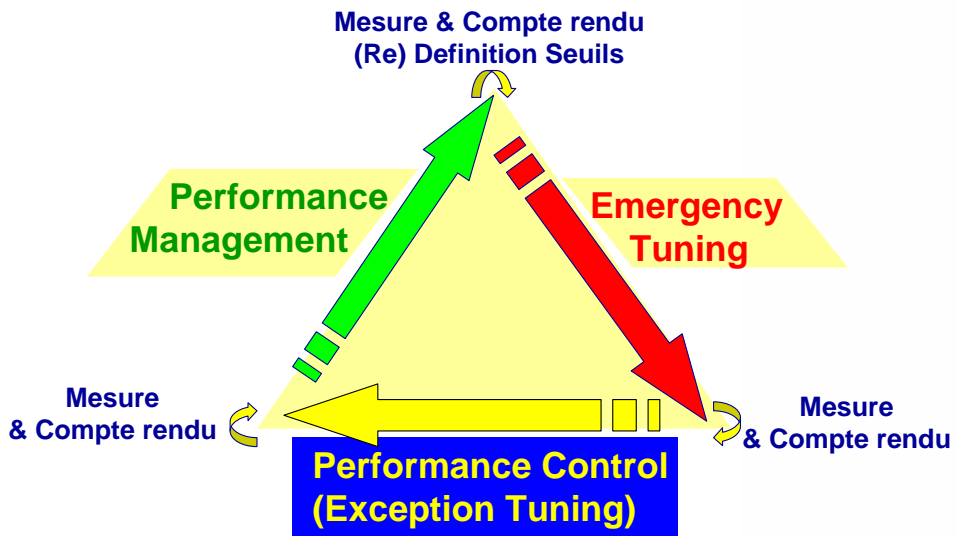
Exemple 2: M3 Index Report



D'autres requêtes (SQL statique) utilisent-elles cet index?
Avec plus de 3 colonnes utilisées en tant que *matching Index*?

```
SELECT PROGNAME, QUERYNO
FROM DB2PVIEW.PLAN_TABLE, sysibm.syspackdep
where progname=dname
      and accesscreator=bqualifier
      and accessname    =bname
      and bqualifier='DB2PROD'
      and bname        ='XPA08013'
AND MATCHCOLS        > 3
```

Condition préalable: les Packages ont été liés avec EXPLAIN(YES)





- Outstanding Performance
- Excellent Performance
- Acceptable Performance
- Bearable Performance
- Inacceptable Performance
- Unavailable Application

