DB2 Performance Expert for Multiplatforms for Business Intelligence

Torsten Steinbach

DB2 Performance Expert for Multiplatforms Development - Technical Lead

torsten@de.ibm.com

IBM Germany

0. About this document

This document focuses on the typical and most important performance issues that occur in Business Intelligence environments and how DB2 Performance Expert can be used to analyze and fix them. This document is not intended to be a complete DB2 BI tuning guide, nor is it intended to be a complete DB2 PE guide. For comprehensive information about DB2 PE, refer to the official DB2 PE library. Another helpful source of information is the IBM Redbook *DB2 Performance Expert for Multiplatforms*, Version 2 (SG24-6470). The author thanks the BI Center of Competency Team (Surendra Parlapalli, Rao Chinnam, Randy Holmes, Ron Sherman) for their participation and contribution in providing content as well as validation of the tool with the BI/Warehousing showcase projects.

The document contains two main parts. The first part introduces DB2 PE and the features that are most relevant to Business Intelligence systems. The second part contains a set of helpful and detailed scenarios that show you how to use DB2 PE to monitor and tune a Business Intelligence environment.

The document is based on DB2 Performance Expert for Multiplatforms Version 2.2 with Fix pack 1 and DB2 Version 8 with Fix pack 10. Some of the descriptions and screenshots that illustrate operating system-level monitoring are specific to AIX and Linux and are slightly different than those for Solaris.

1. Introduction

DB2 Performance Expert for Multiplatforms (DB2 PE) is a general purpose DB2 performance monitoring and analysis framework. It is flexible enough to be applied in all types of DB2 environments. In this document the focus is on how to leverage DB2 PE in DB2 Business Intelligence environments.

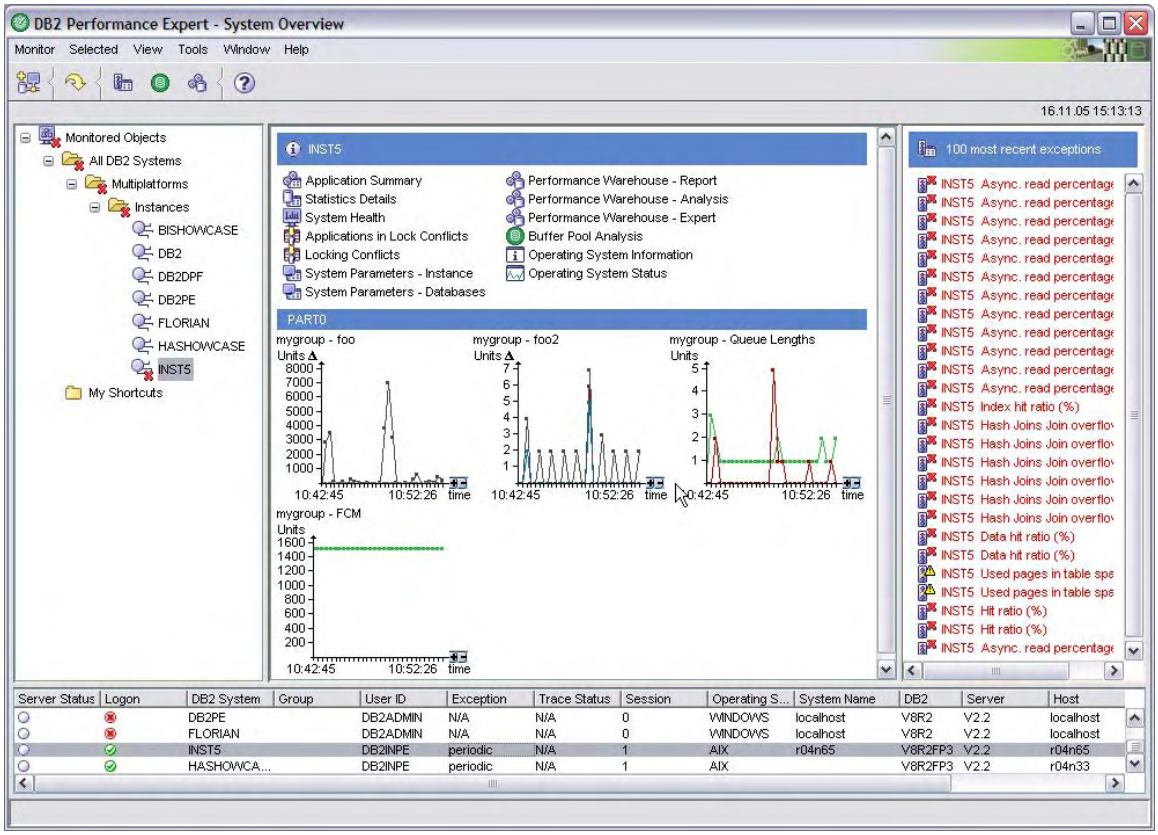
DB2 PE is based on a three-tier monitoring architecture. Tier 1 is the monitored DB2 system. Tier 2 is the DB2 PE server that remotely monitors tier 1. It also enables all the advanced features like performance history capturing, performance warehousing, and exception processing. Tier 3 is the DB2 PE client that connects to the DB2 PE server and is used to perform all monitoring tasks.

BI systems usually have typical data models and workloads, as well as typical performance problems. DB2 PE can help you to sort out these problems and find the best solutions.

2. DB2 PE features for Business Intelligence

This section describes the monitoring and analysis features of DB2 PE and explains why these features are valuable for Business Intelligence systems.

The entry point for all DB2 PE features is the DB2 PE System Overview window. You use this window to select and monitor different DB2 systems from the navigation tree which is located in the leftmost pane. The center pane contains links to the major monitoring and analysis features, as well as a number of diagrams that display key performance indicator data. The rightmost pane displays the performance exceptions that DB2 PE has detected on the currently selected system.



2.1 DPF monitoring and skew detection

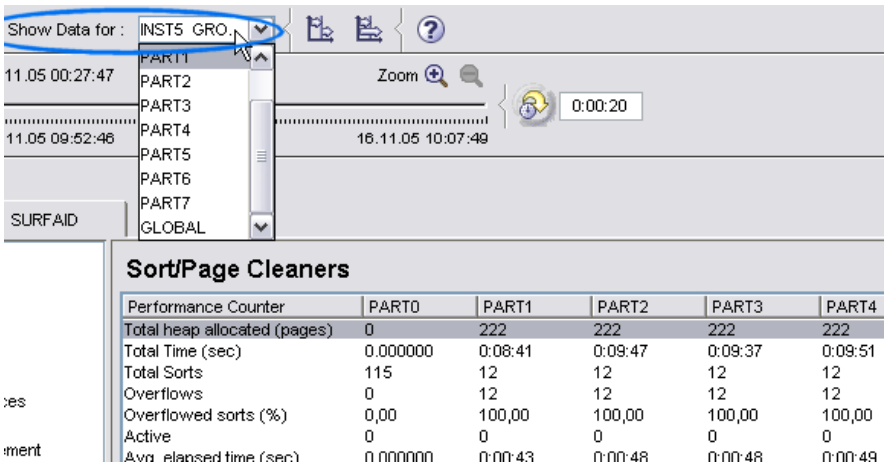
DB2 PE is fully enabled to monitor a multi-partitioned DB2 system. With a single mouse click your can retrieve monitoring information for all. You can conveniently get an overview over your entire distributed system.

This capability also is the basis for quick and easy detection of skews. A *skew* is an uneven distribution of behavioral or residual distribution of performance indicators over

the distributed system components. Skews are always potential reasons for performance problems.

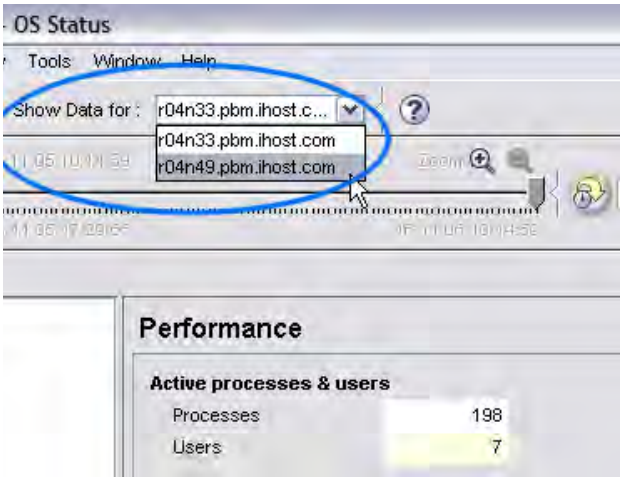
2.1.1 Partition-level skews

Most DB2 performance indicators can be retrieved for each partition. DB2 PE allows you to display an aggregated overview for these performance indicators (called a *global view*) or, you can display the current values for a selected partition (called a *partition view*). Finally, DB2 PE provides a special view mode called the *group view* (shown in the following picture), which provides a matrix view of individual values. This view allows you to look for skews in the different performance indicators.



2.1.2 System-level skews

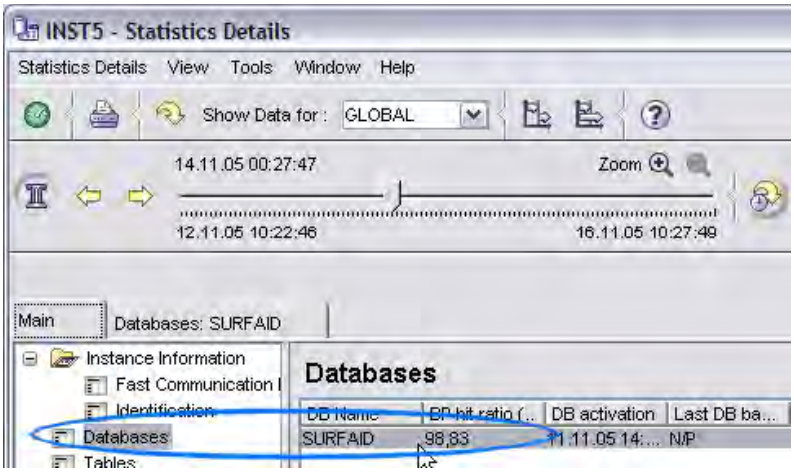
Because DB2 PE also provides operating system-level information it also offers you a special group view for that data where the group members are comprised of machines rather than partitions. This list is comprised of all distinct machines that are part of the distributed DB2 instance.



2.2 Engine monitoring

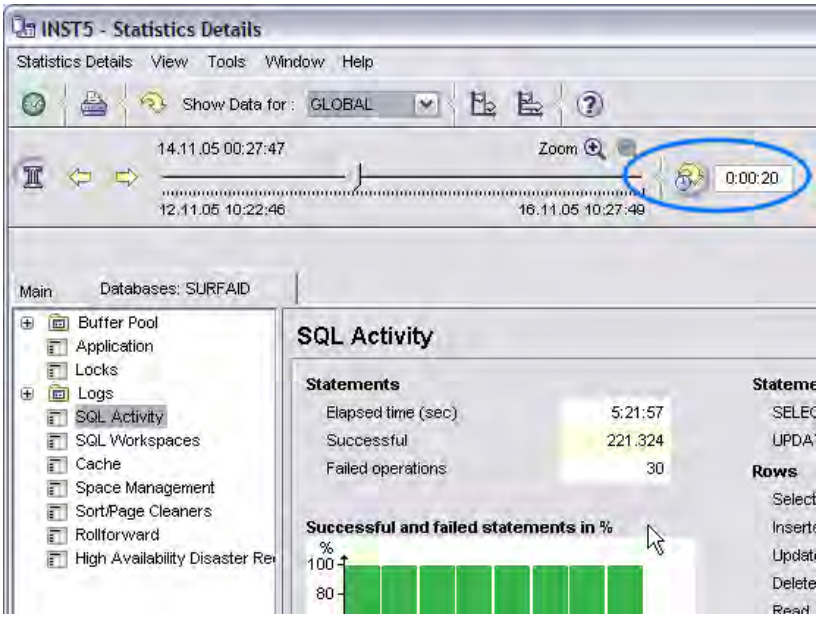
DB2 PE provides monitoring at the DB2 engine level (that is, performance metrics on the level of instances, databases, and subsequent entities such as tablespaces, bufferpools, and tables). You can access this data by double-clicking **Statistic Details** on the DB2 PE System Overview window.

The Statistic Details panels are divided into categories in a navigation tree that is located in the leftmost pane. You can find the metrics that are related to a subsequent level (for example, a database) by double-clicking one of the entries in the associated category.



Doing so opens another tabbed pane in the same window that contains details about the selected database.

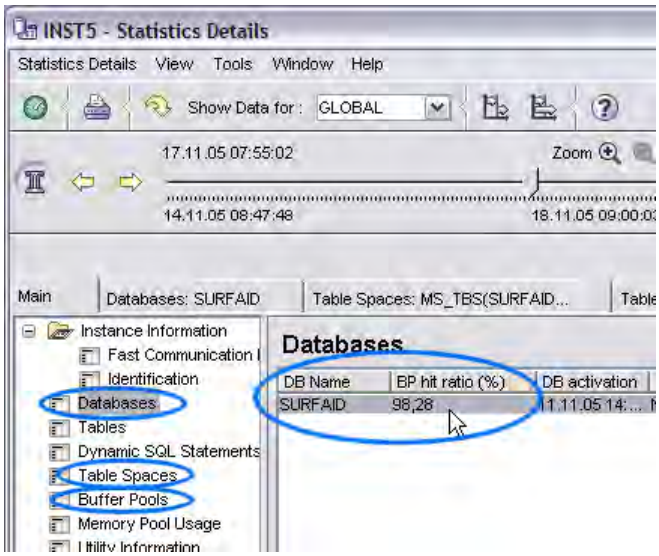
At the top of the panel is an auto-refresh option.



Some of the data can be seen as key performance indicators that you typically check first in a Business Intelligence environment. Some of the most important ones are:

BP hit ratio

Bufferpool hit ratio indicates the effectiveness of the bufferpool. Even though it is desirable to have the ratio at a rather high value (> 80%), this is typically not achievable for larger Business Intelligence systems. Increasing the bufferpool size might have a minimal effect on the hit ratio. Its number of pages might be so large, that the statistical chance of a hit is not improved by increasing the size.

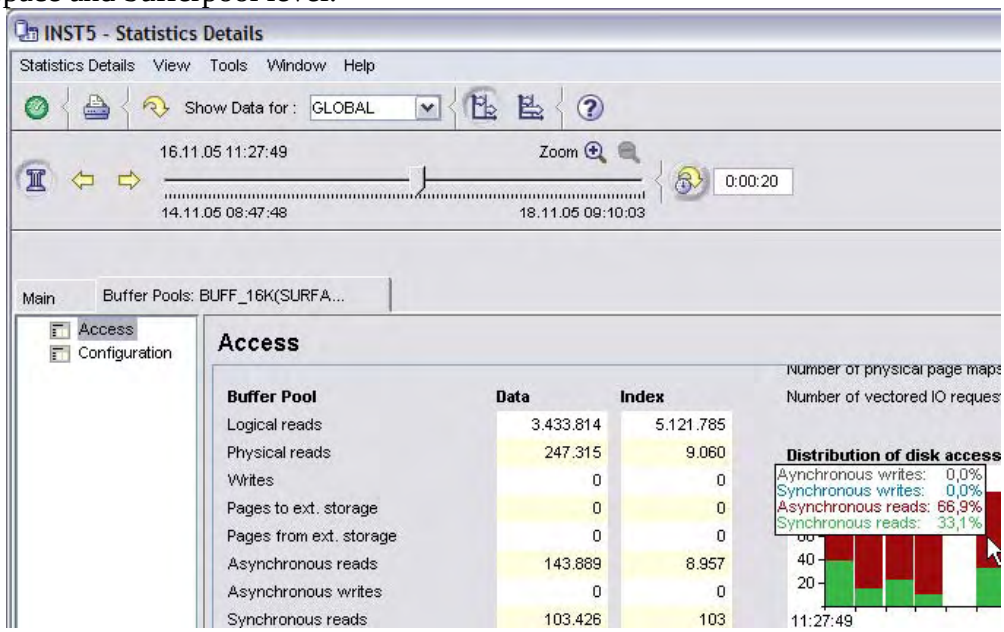


This metric can be retrieved on three different levels of the DB2 engine: database, tablespace, bufferpool.

Asynchronous read ratio

In Business Intelligence systems, a large amount of table data typically needs to be read. DB2 provides the feature of sequential prefetching, which allows pages to be fetched asynchronously ahead of access if the optimizer expects the next rows that will be requested to be on these consecutive pages. With the asynchronous read ratio metric (synchronous vs. asynchronous reads), you can see to what extent pages are read via this method. A high number is usually desired for Business Intelligence systems. As with the

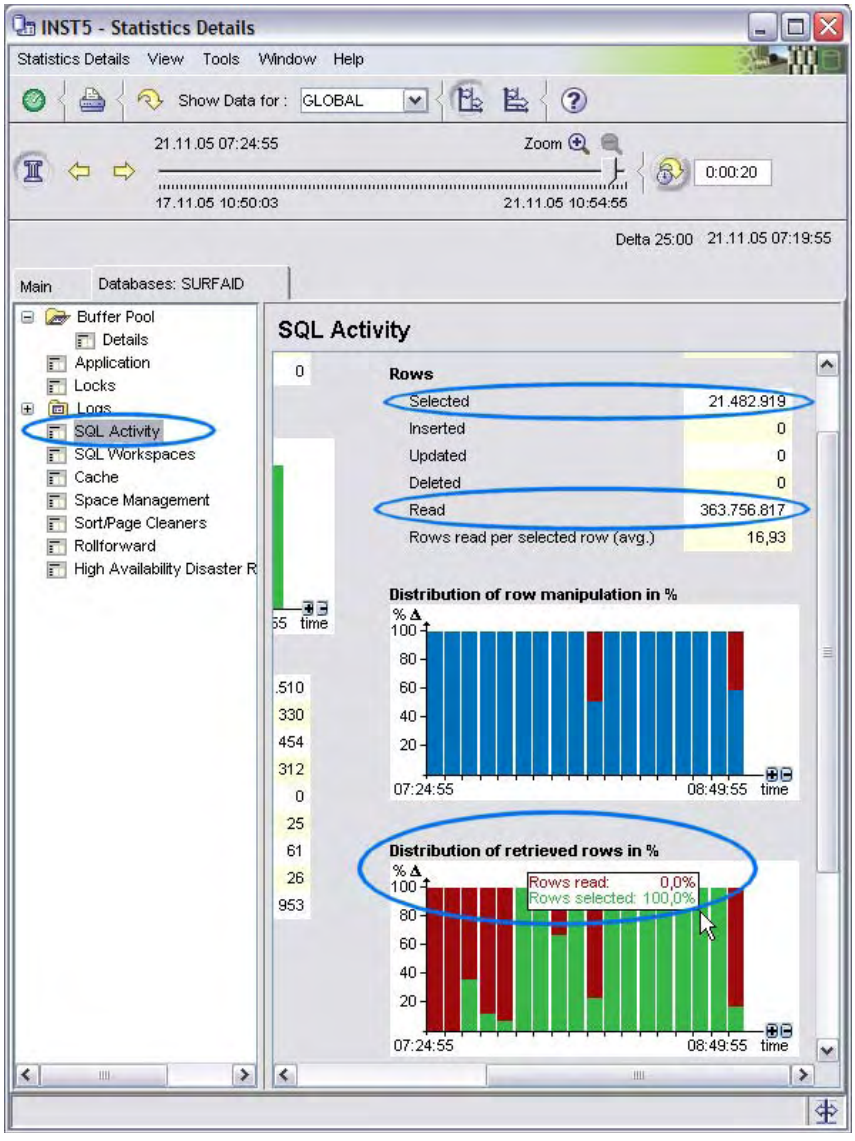
bufferpool hit ratio, the asynchronous read ratio metric is available on database, tablespace and bufferpool level.



Read efficiency

How efficiently queries are executed (with regard to how many rows must be read from disk compared to how many rows are actually returned – that is, how many rows are part of the result set) is a crucial metric. If this number is relatively high, DB2 reads much more data than the user actually requests.

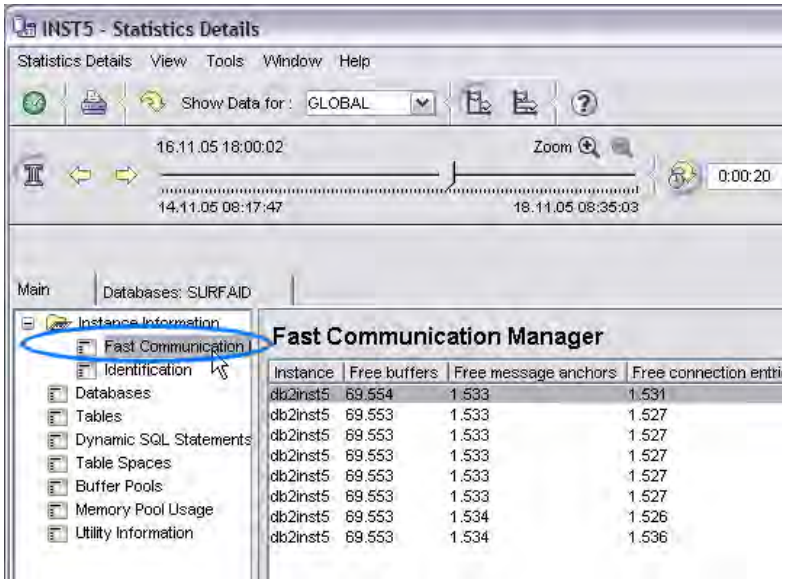
You can check this metric on database, tablespace, bufferpool, and table level.



FCM buffers

The Fast Communication Manager (FCM) is a key component of DB2 data partitioning feature (DPF). It is used for communication between the partitions. That's why DB2 PE provides a dedicated pane for its performance metrics.

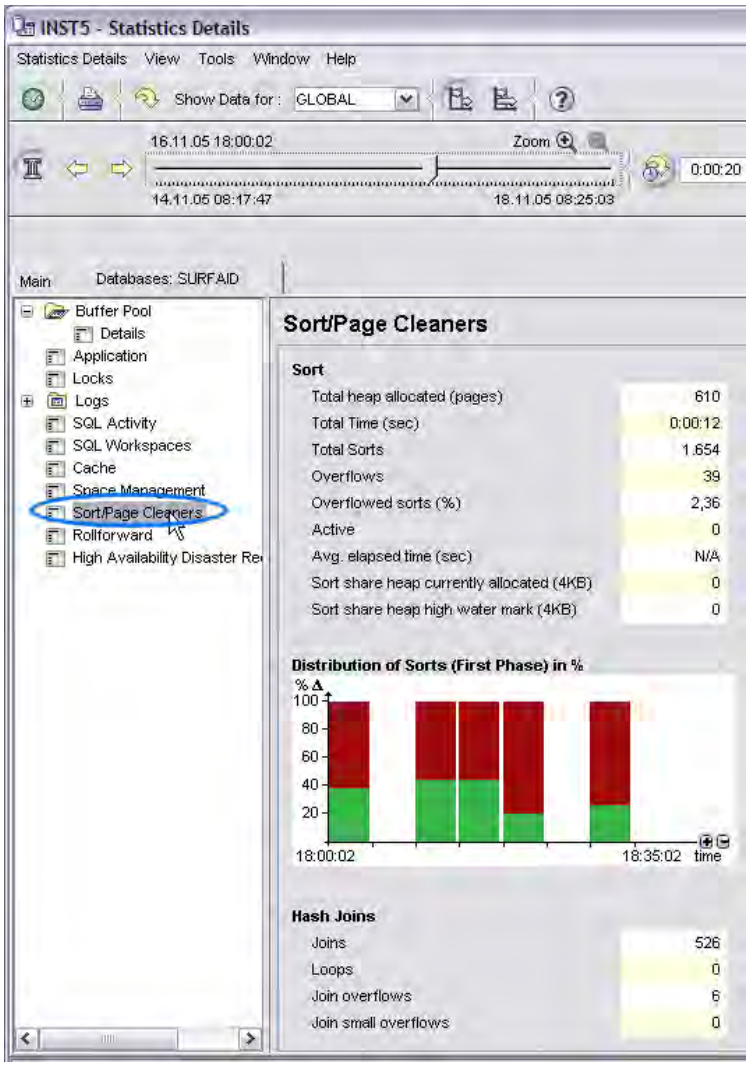
Section 3.10 describes FCM tuning in more detail.



Sort

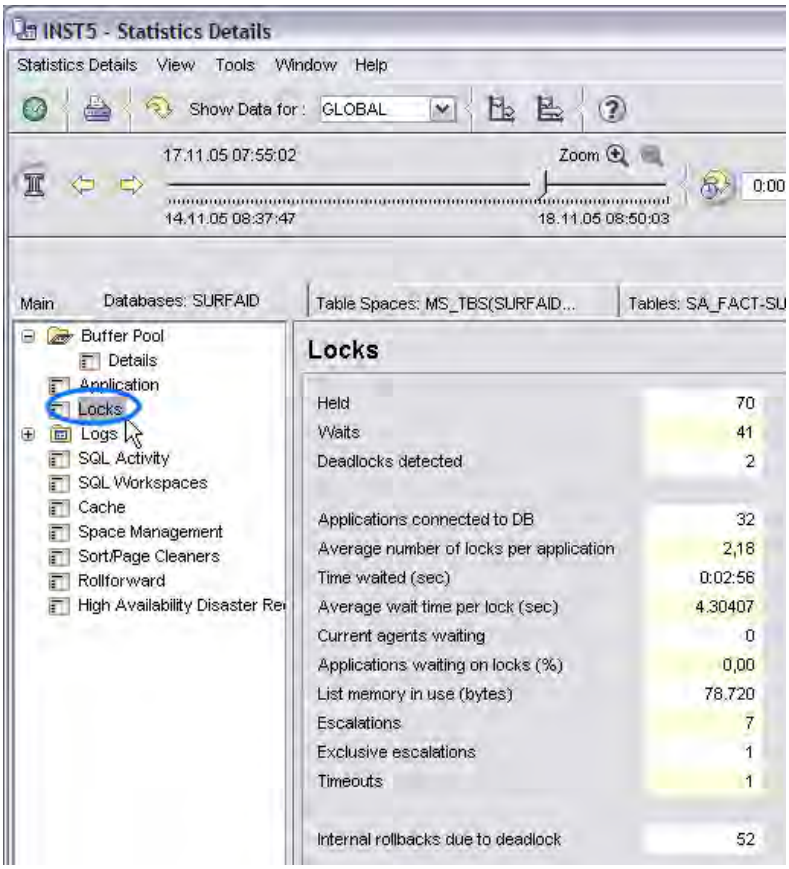
Sorting data is typically necessary to a large extent for execution of Business Intelligence SQL workloads. Tuning the engine accordingly is a key task for Business Intelligence administrators. For this reason, DB2 PE displays the key sort performance indicators for the DB2 engine on a dedicated pane in the Statistic Details window.

Section 3.2, describes how to address sorting issues.



Locking

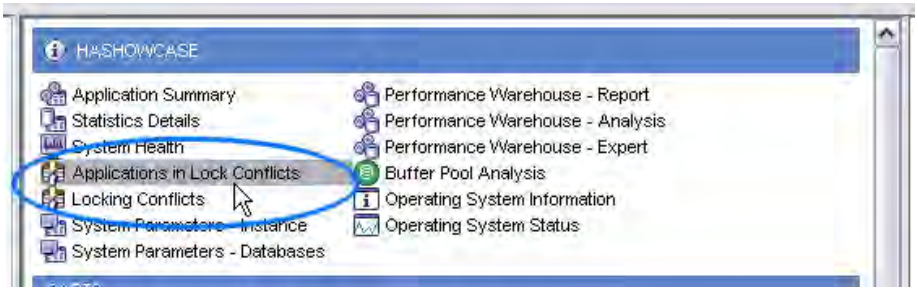
When Business Intelligence systems require concurrent incremental load of new data while the query workload continues, you must also check for locking issues. A dedicated Locks pane gives you the best overview of this.



In addition, DB2 PE proactively informs you about deadlocks, including the relevant details.

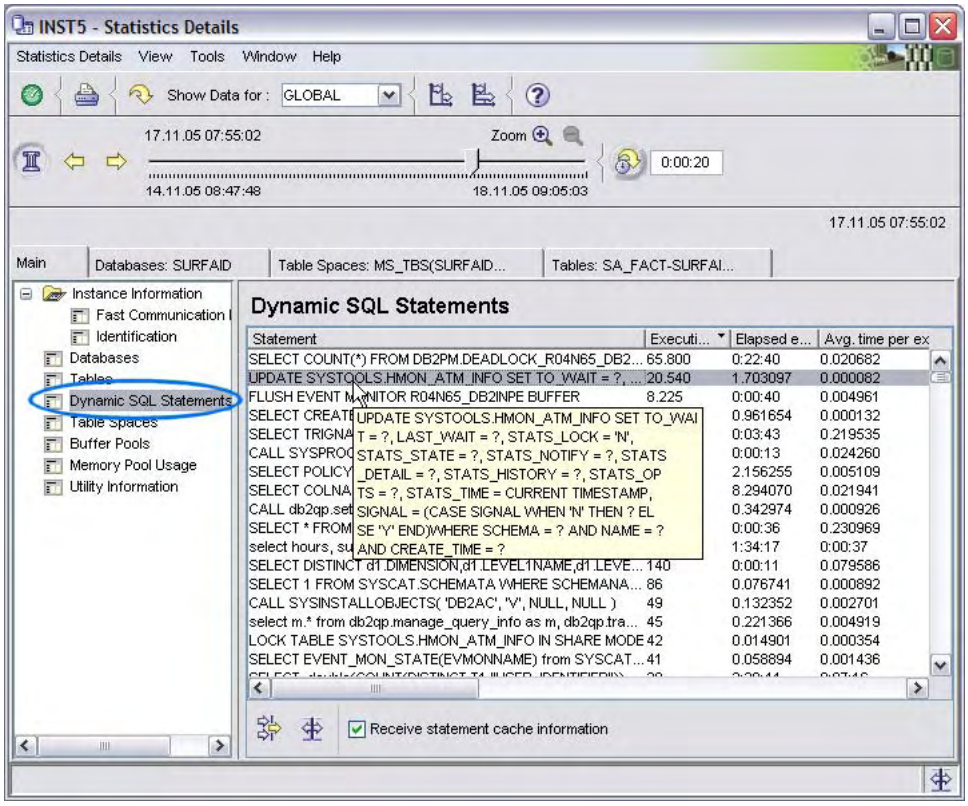
Furthermore, the System Overview window provides access to special dialogs for analyzing locking conflicts.

For more details about tuning load, see section 3.7.



Dynamic SQL statements

The main purpose of a Business Intelligence system is to return the answers for analytic queries. Therefore, it is crucial to keep track of the queries that the Business Intelligence system is running and of how they are performing. The best starting point is the Dynamic SQL Statements pane in the Statistics Details window. It shows you each statement that was recently executed along with key properties such as number of executions, compile time, and execution time.



2.3 Application monitoring and tracing

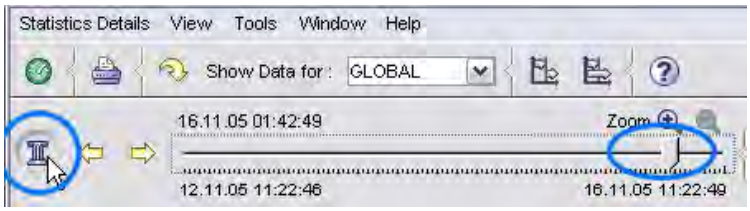
Applications are a crucial element to monitor in Business Intelligence environments because they typically execute long running queries. DB2 PE provides a special view for application specific metrics. Click **Application Summary** from the DB2 PE System Overview window.

The window that opens shows a list of all active applications and includes a couple of important performance metrics. You can see all details about a certain application by double-clicking it. The Application Details pane opens, which can be used in a manner that is similar to the Statistic Details pane to navigate to different types of application metrics. Most of the metrics that are described above for the DB2 engine level are

available on a per-application level here (bufferpool hit ratio, asynchronous read ratio, read efficiency, and sort metrics).

2.4 History analysis

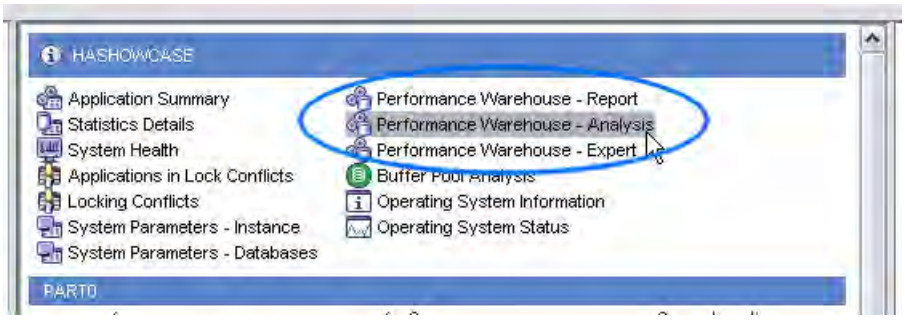
DB2 PE provides a convenient way to monitor the historical performance of a DB2 system. The panels that are opened from the DB2 PE System Overview window are opened in online monitoring mode by default. With a single mouse-click you can switch to history mode to display the exact same metrics. A history slider allows you to conveniently browse through historical data.



By default, this history data wraps after 50 hours. But you are free to configure any other number. Just be aware, the more hours that you keep, the larger the performance database on the DB2 PE server must be able to grow.

2.5 Performance Warehouse analysis

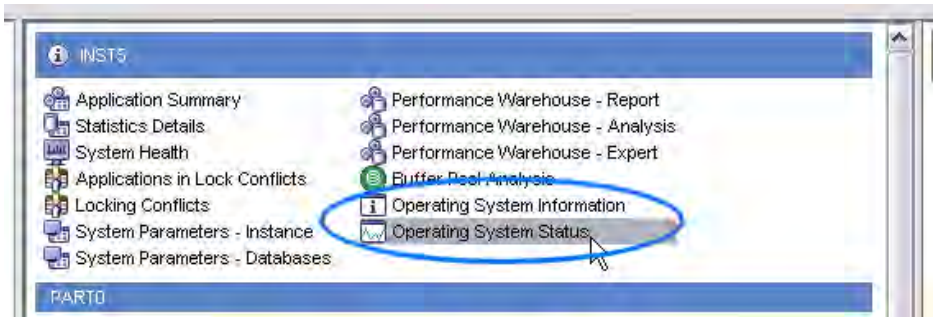
DB2 PE has special long-term storage for performance data called the Performance Warehouse (PWH). In addition to this data, a set of analyzing features is provided. A predefined set of HTML reports can be generated, predefined and custom queries can be executed, and predefined and custom rules of thumb can be evaluated to check for performance problems that were experienced by the Business Intelligence system over time.



The query capability is referenced extensively throughout the remainder of this document.

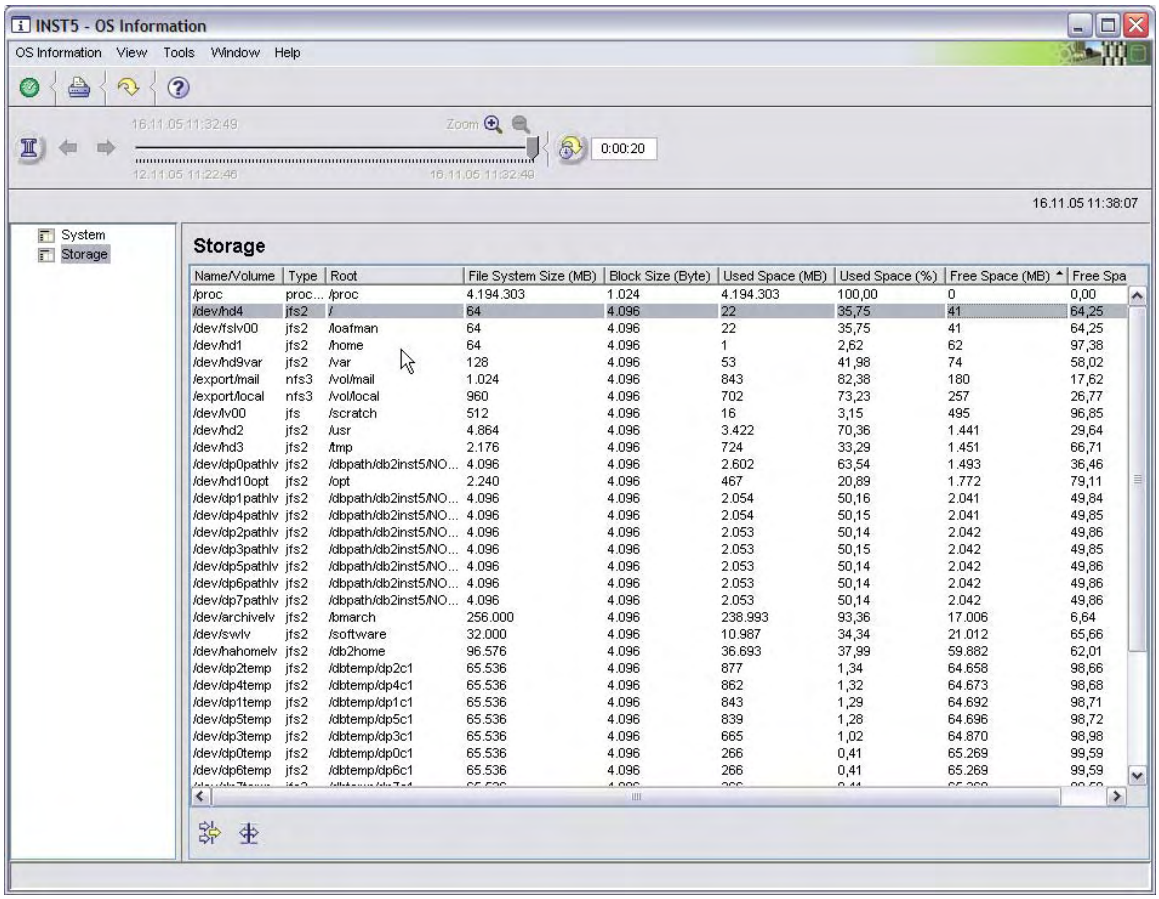
2.6 Operating System monitoring

In addition to a large variety of DB2 metrics, DB2 PE also keeps track and displays OS-level metrics. There are two links in the DB2 PE System Overview window to that data. **Operating System Information** links to the rather static data, such as configured memory, installed CPUs, and filesystem information. **Operating System Status** links to more volatile metrics about paging, processes, CPU consumption, and disk utilization. All OS-level data is enabled for history monitoring.



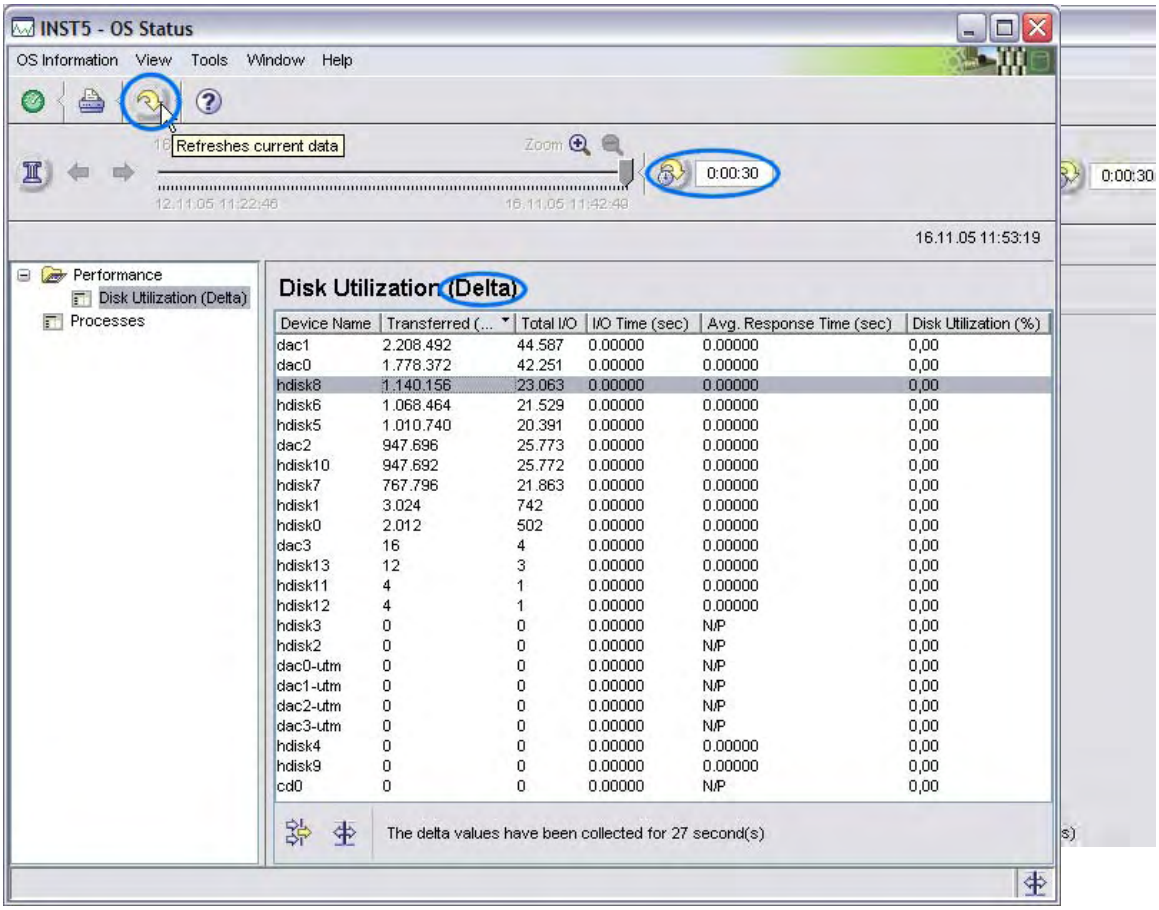
2.6.1 Filesystem usage

As your Business Intelligence system grows over time, you typically want to verify that the storage resources are still sufficient. You might also want to keep an eye on storage that is needed only temporarily. This is important because Business Intelligence systems tend to perform large sorts that might require some dedicated disk space. The OS Information window provides an overview of storage resources.



2.6.2 Disk I/O

Click the Disk Utilization (Delta) category of the OS Status window to display disk I/O metrics. This pane shows the transferred (that is, read and written) kilobytes and total number of I/O operations. Note that this data is based on delta intervals. You can either manually refresh the pane or see this data based on the time between the recent two refreshes, or you can use the auto-refresh option.



2.6.3 CPU utilization, queues, and paging

The OS Status window also contains information about CPU and paging in the Performance category.

The **Run Queue Length** field indicates how many processes are actually ready for execution, and the **Block Queue Length** field shows the number of processes that are waiting for I/O to finish.

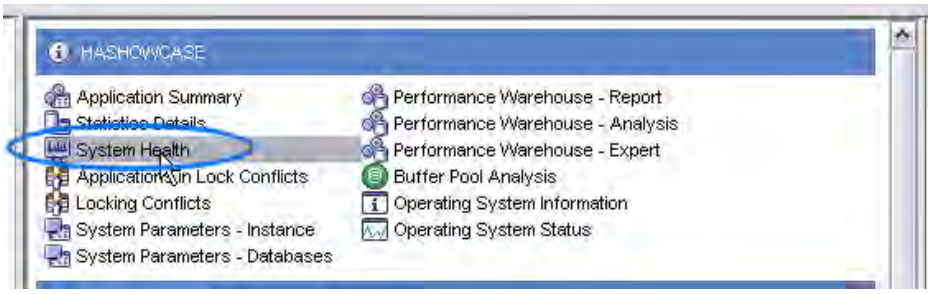
Paging is counted in the number of pages being aged in or paged out.
The CPU utilization is divided into user time, system time, I/O wait time, and idle time.

Both CPU utilization and paging data is based on delta values. You can either click the **Refresh** button to see the data based on the time between previous and current refresh, or you can use the auto-refresh feature.

The paging rate and block queue length data can also be visualized in the System Health pane, and they can be used for threshold definitions for exceptions.

2.7 Visualization

For all performance indicators that have been described so far, DB2 PE also provides a visual way of displaying them in the System Health window.



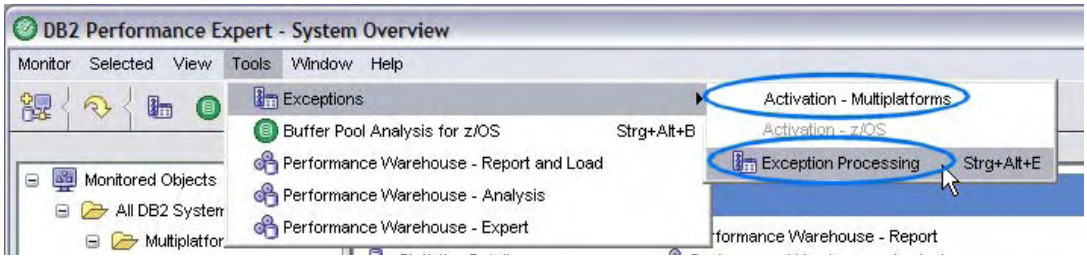
In this window you can set up and view a number of dashboards with predefined or custom defined diagrams (called *data views*). Like the other dialogs that have been described so far, the System Health window is fully enabled for history monitoring and for DPF monitoring (group views).

This allows you to set up dashboards with the Business Intelligence key performance indicators as described in section 3.11.

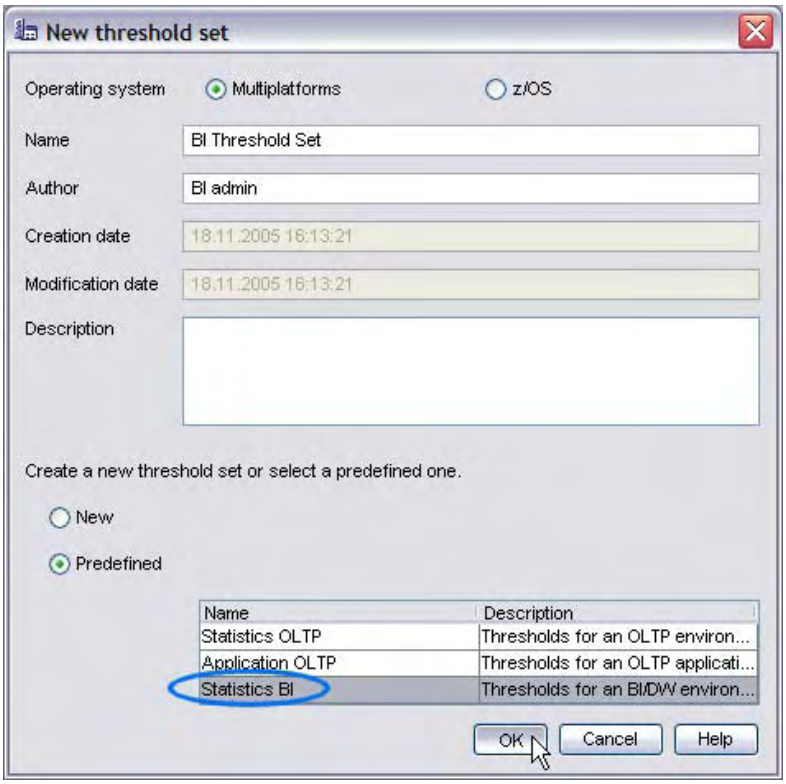
2.8 Threshold exceptions / predefined threshold set

A Business Intelligence system typically has certain threshold values for performance indicators that you should be aware of when they are exceeded. For this purpose, DB2 PE has the built-in feature of periodic exception processing, which basically means that it checks defined performance indicators against threshold definitions and raises an exception (visually, by email, or by executing custom code).

To enable this feature, create a set of threshold definitions and activate them so that they are considered in periodic exception processing.



When you create such a new threshold set, you can use the built-in Business Intelligence template so that you do not need to start from scratch.



3. Business Intelligence Performance Tuning Scenarios with DB2 PE

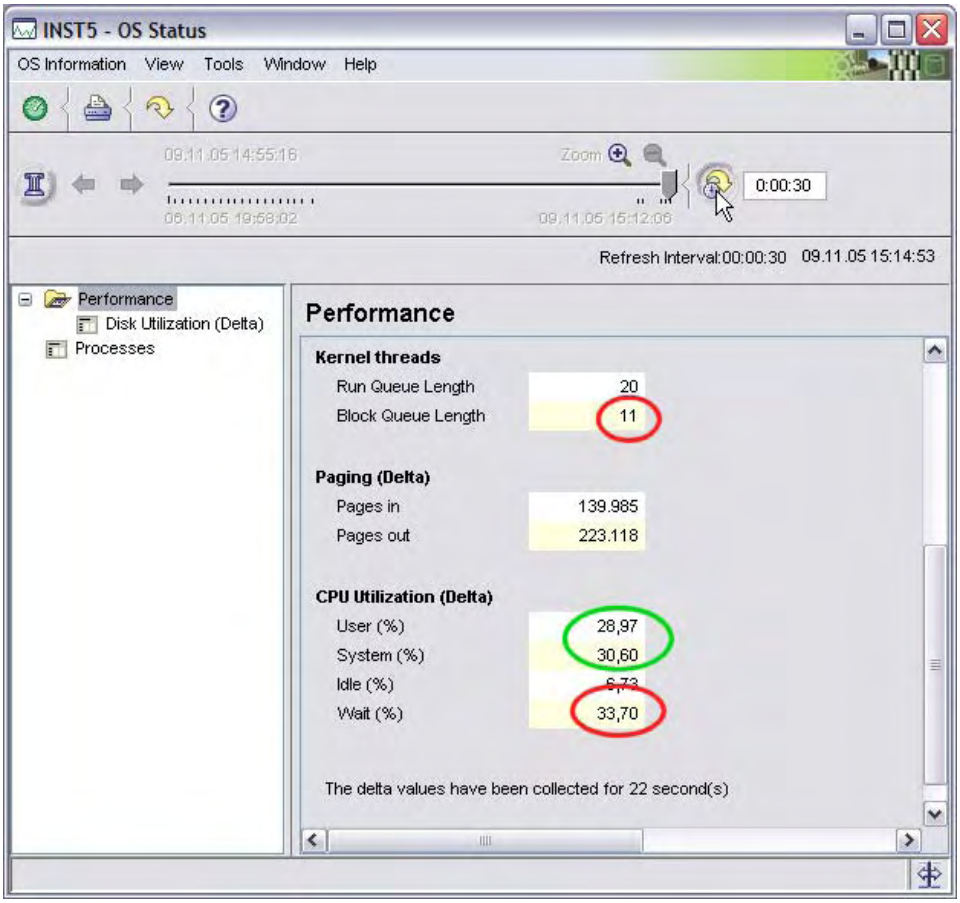
This section describes a set of typical Business Intelligence tuning scenarios and how DB2 PE facilitates them.

3.1 Check if system is CPU bound

The desired state is for the system to be CPU bound, that is, to not spend precious CPU cycles waiting on any resources, such as I/O.

3.1.1 General approach

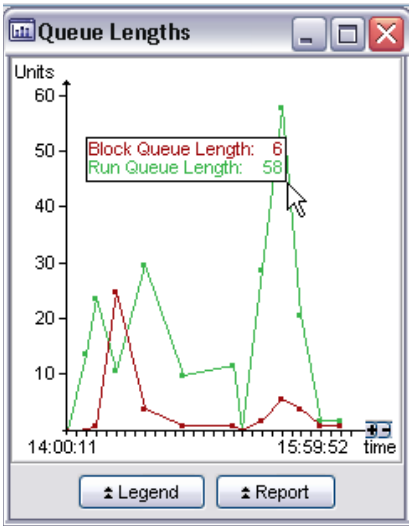
A straight-forward method to determine if the monitored system is CPU bound is to open the OS Status window and to display the Performance pane. Next, switch to auto-refresh mode so that you permanently are presented with the current state of the system. Then, check the **Block Queue Length** field. It should have a low value near 0 because it counts each process that currently has to wait for a resource such as disk or network.



Next, check the CPU utilization fields. The majority of the time should be consumed by user and system time. If wait time is a two-digit percentage over a longer time, the system is I/O-bound rather than CPU-bound and must be revised accordingly.

3.1.2 Visualization

The queue length metrics can also be visualized in the System Health and System Overview windows. Open the System Health window, right-click a data group, and select **New…**. Name the new data view "Queue Lengths" and select the category **Operating System Status, Performance** before clicking **Next**. Now select the counters "Run Queue Length" and "Block Queue Length" and click **Next**. Select **Dynamic scale** and click **Next**. Now select the chart type you like, for example, Line chart, to display a data view like the one shown here. Now click **Finish**.



If you want the data view that you just created to be displayed in System Overview window, right-click it

and select **Display in System Overview**.

3.1.3 Exceptions

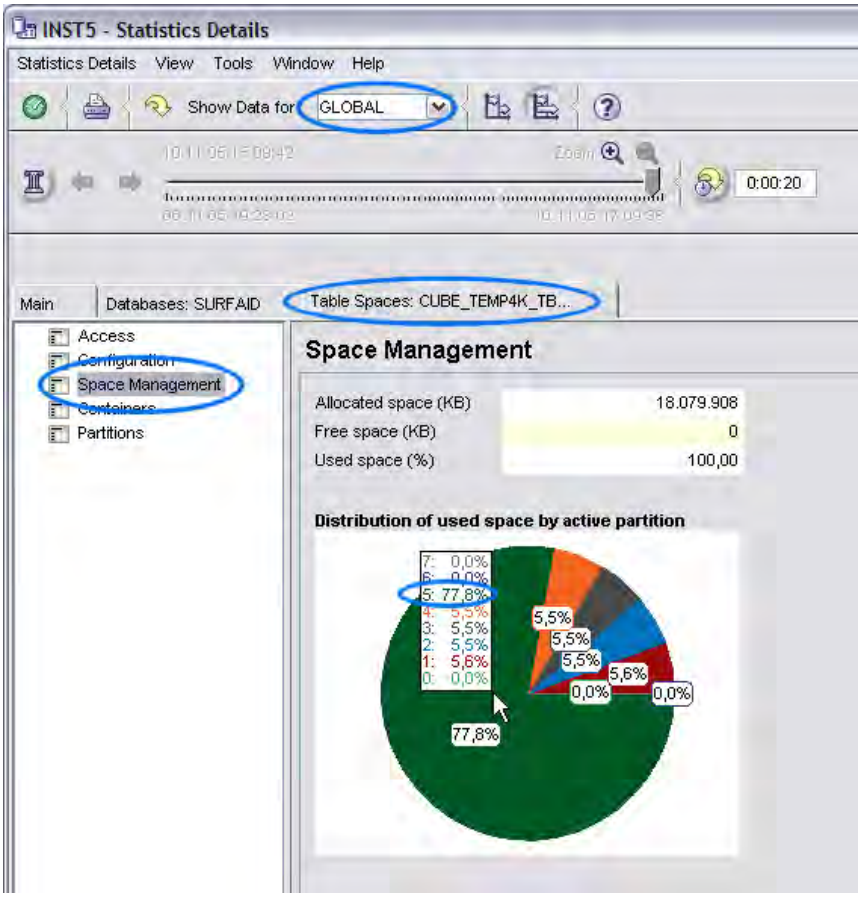
The Block Queue Length can also be automatically checked by DB2 PE against a threshold. To enable this feature: 1. Click **Tools->Exceptions->Exception Processing**. 2. Double-click a threshold set and click **Threshold Set->New Threshold**. 3. Select the Operating System Status exception category and the exception field **Block Queue Length**. Specify a warning and problem threshold depending on your system. Typically the block queue length should be below 10, so a warning threshold of 10 and a problem threshold of 15 might be a good start. 4. Click **OK**. 5. Activate this threshold set by clicking **Tools->Exceptions->Activation – Multiplatform**. In the dialog, select the threshold set and then press the start button for **Periodic Exception Processing**.

For other exception processing related features, such as e-mail notification, refer to the DB2 PE documentation.

3.1.4 Identify I/O bound partitions

Use the following method to identify I/O bound partitions. This method involves checking for temporary tablespace skew. 1. Open Statistical Details pane for “Table Spaces” and double-click the system temporary space that you want to check in that exercise. 2. Display the “Space Management” pane. Make sure that you are in Global view mode.

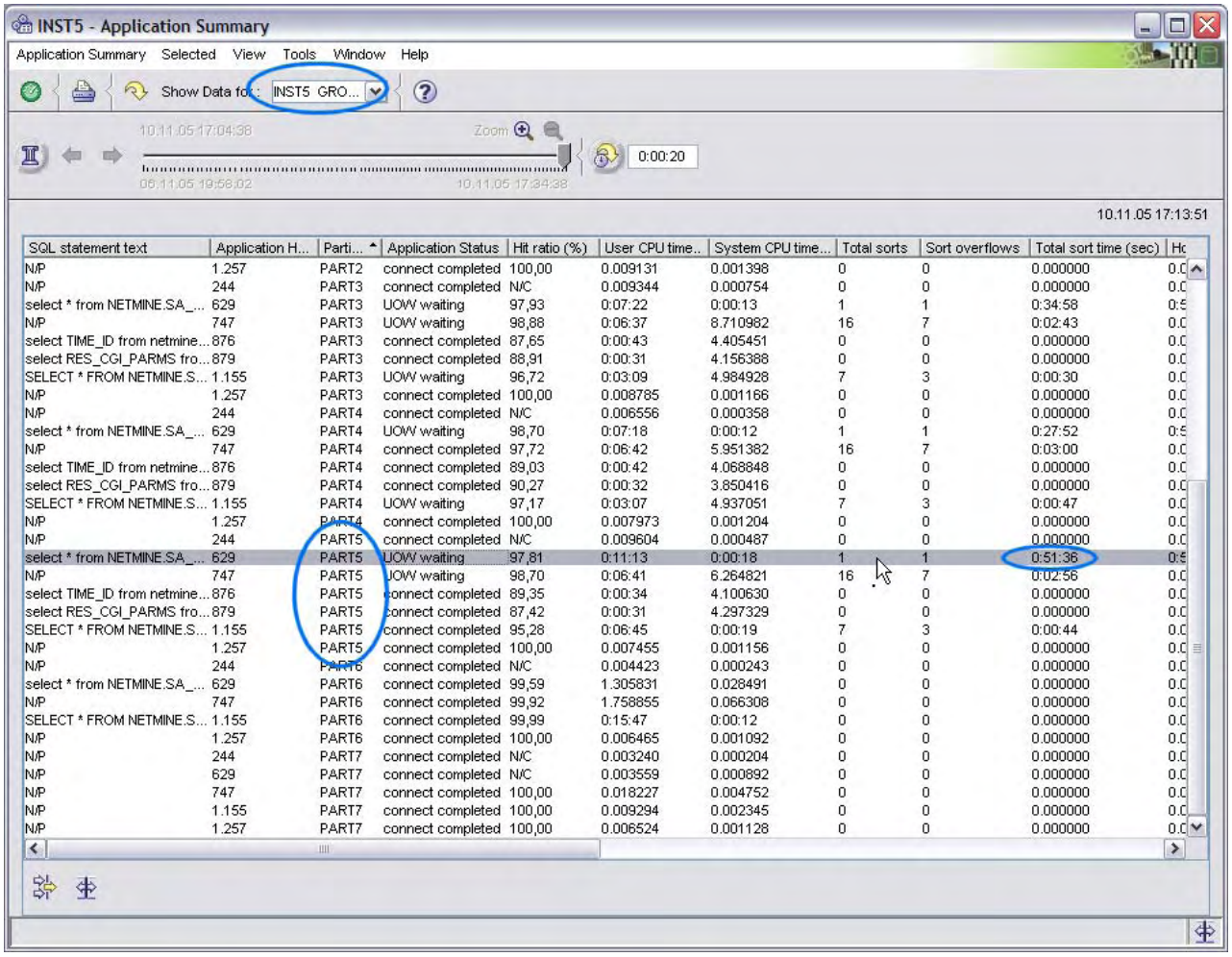
If there is repeating data skew on temporary tablespaces in the same partitions, this is an indication that these partitions are I/O bound or at least significantly more I/O bound than the other partitions.



Before concluding that a partition is I/O bound based on this method, you must eliminate other potential reasons for data skew on temporary tablespaces, which could be:

- Data skew on the base tables so that some partitions have to sort more than others
- Current workloads are using only certain or even a single partition because the base tables to not span the same partitions as the temporary tablespace does

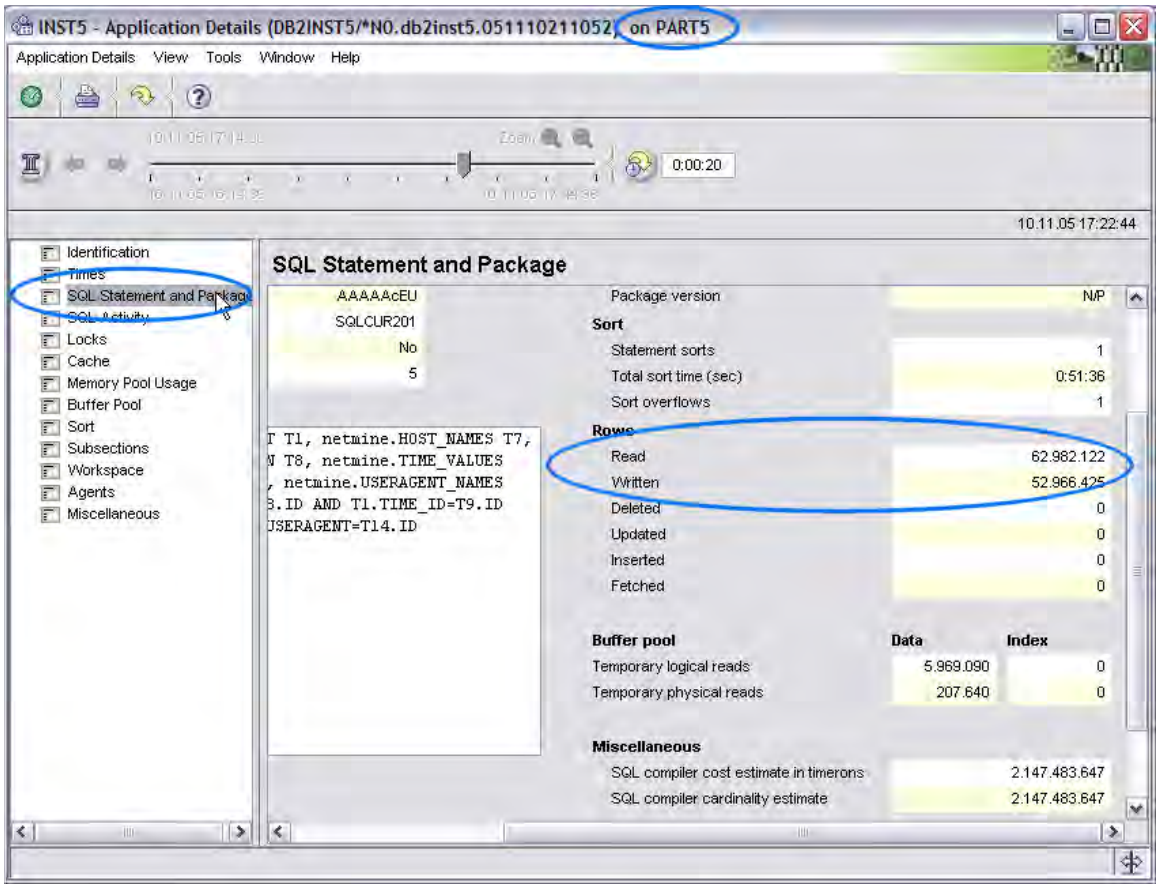
To eliminate the first of these reasons display the Application Summary window in group view. Sort the table on the **Partition** column and check all entries of the partition with skew detected above. Within this group, look for the entry with the highest sort time. This partition is a good candidate to check out if it is causing the skew in the temporary tablespace.



SQL statement text	Application H...	Parti...	Application Status	Hit ratio (%)	User CPU time..	System CPU time...	Total sorts	Sort overflows	Total sort time (sec)	Ho
N/P	1.257	PART2	connect completed	100,00	0.009131	0.001398	0	0	0.000000	0.0
N/P	244	PART3	connect completed	N/C	0.009344	0.000754	0	0	0.000000	0.0
select * from NETMINE.SA_...	629	PART3	UOW waiting	97,93	0:07:22	0:00:13	1	1	0:34:58	0:5
N/P	747	PART3	UOW waiting	98,88	0:06:37	8.710982	16	7	0:02:43	0.0
select TIME_ID from netmine...	876	PART3	connect completed	87,65	0:00:43	4.405451	0	0	0.000000	0.0
select RES_CGI_PARMS fro...	879	PART3	connect completed	88,91	0:00:31	4.156388	0	0	0.000000	0.0
SELECT * FROM NETMINE.S...	1.155	PART3	UOW waiting	96,72	0:03:09	4.984928	7	3	0:00:30	0.0
N/P	1.257	PART3	connect completed	100,00	0.008785	0.001166	0	0	0.000000	0.0
N/P	244	PART4	connect completed	N/C	0.006556	0.000358	0	0	0.000000	0.0
select * from NETMINE.SA_...	629	PART4	UOW waiting	98,70	0:07:18	0:00:12	1	1	0:27:52	0:5
N/P	747	PART4	connect completed	97,72	0:06:42	5.951382	16	7	0:03:00	0.0
select TIME_ID from netmine...	876	PART4	connect completed	89,03	0:00:42	4.068848	0	0	0.000000	0.0
select RES_CGI_PARMS fro...	879	PART4	connect completed	90,27	0:00:32	3.850416	0	0	0.000000	0.0
SELECT * FROM NETMINE.S...	1.155	PART4	UOW waiting	97,17	0:03:07	4.937051	7	3	0:00:47	0.0
N/P	1.257	PART4	connect completed	100,00	0.007973	0.001204	0	0	0.000000	0.0
N/P	244	PART5	connect completed	N/C	0.009604	0.000487	0	0	0.000000	0.0
select * from NETMINE.SA_...	629	PART5	UOW waiting	97,81	0:11:13	0:00:18	1	1	0:51:36	0:5
N/P	747	PART5	UOW waiting	98,70	0:06:41	6.264821	16	7	0:02:56	0.0
select TIME_ID from netmine...	876	PART5	connect completed	89,35	0:00:34	4.100630	0	0	0.000000	0.0
select RES_CGI_PARMS fro...	879	PART5	connect completed	87,42	0:00:31	4.297329	0	0	0.000000	0.0
SELECT * FROM NETMINE.S...	1.155	PART5	connect completed	95,28	0:06:45	0:00:19	7	3	0:00:44	0.0
N/P	1.257	PART5	connect completed	100,00	0.007455	0.001156	0	0	0.000000	0.0
N/P	244	PART6	connect completed	N/C	0.004423	0.000243	0	0	0.000000	0.0
select * from NETMINE.SA_...	629	PART6	connect completed	99,59	1.305831	0.028491	0	0	0.000000	0.0
N/P	747	PART6	connect completed	99,92	1.758855	0.066308	0	0	0.000000	0.0
SELECT * FROM NETMINE.S...	1.155	PART6	connect completed	99,99	0:15:47	0:00:12	0	0	0.000000	0.0
N/P	1.257	PART6	connect completed	100,00	0.006465	0.001092	0	0	0.000000	0.0
N/P	244	PART7	connect completed	N/C	0.003240	0.000204	0	0	0.000000	0.0
N/P	629	PART7	connect completed	N/C	0.003559	0.000892	0	0	0.000000	0.0
N/P	747	PART7	connect completed	100,00	0.018227	0.004752	0	0	0.000000	0.0
N/P	1.155	PART7	connect completed	100,00	0.009294	0.002345	0	0	0.000000	0.0
N/P	1.257	PART7	connect completed	100,00	0.006524	0.001128	0	0	0.000000	0.0

Open the details view by double-clicking the entry of interest and check the rows read and written counters. Now compare these to the same counters of the details panels of the other partitions of the same application handle.

If the numbers are approximately the same, then the data skew on the base tables is likely not the cause of the skew on the temporary tablespace.



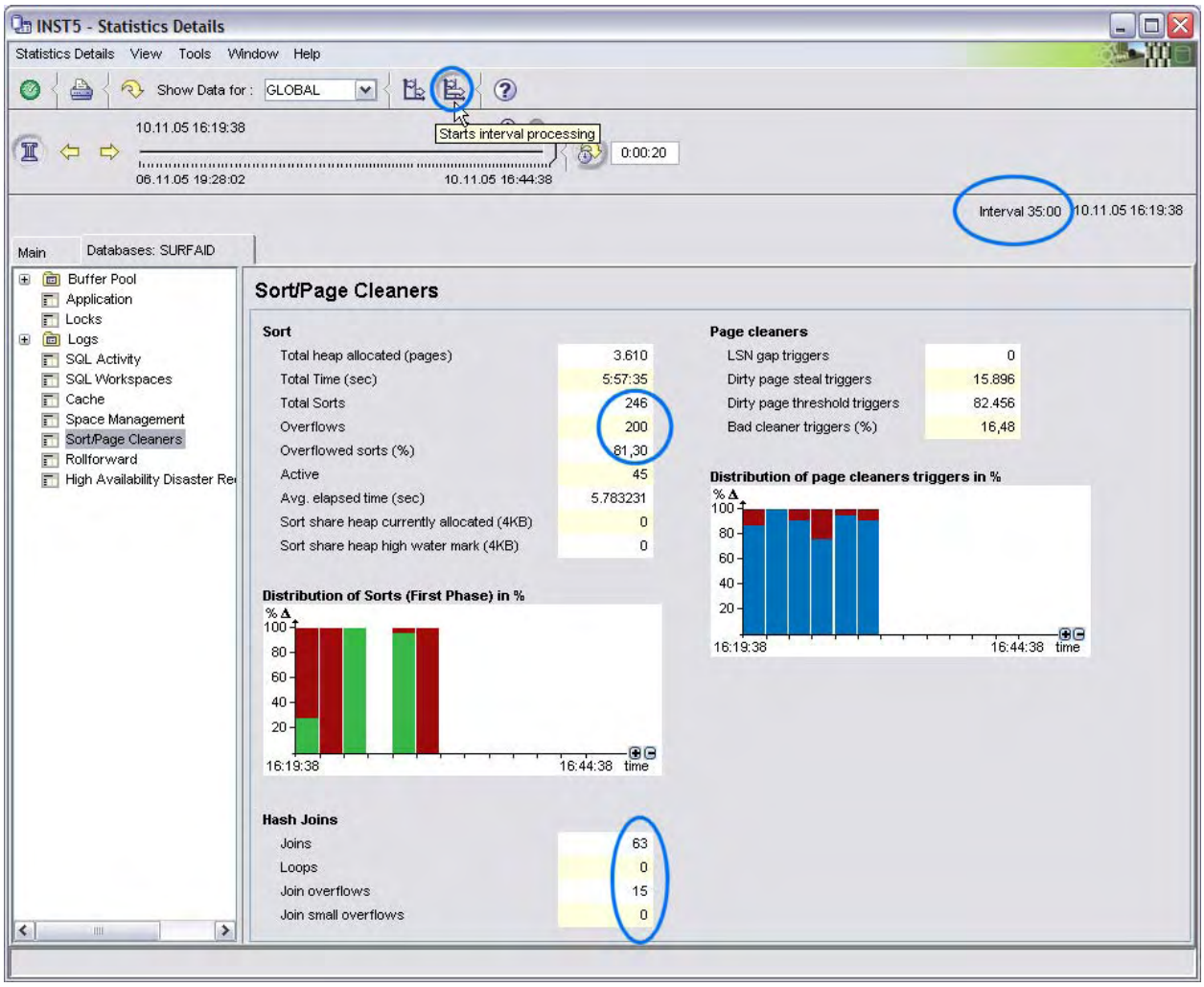
3.2 Sort and hash Join tuning

Sorting usually has a large impact in BI systems. For example, ORDER BY, GROUP BY, DISTICT, HAVING, or UNION clauses require the data to be sorted if there is no matching index on the according columns. So it is crucial to make sure that sorting is done in an optimal way.

A good starting point is the Sort/Page Cleaner pane in statistic details for a particular database. Check for sort overflows and overflowed sorts (%). These occur when a sort cannot be done in memory anymore.

Another reason for sorting to occur is hash joining. There are special indicators for these sort types. You should look at the ratio between the number of joins and join overflows. Check for hash join overflows.

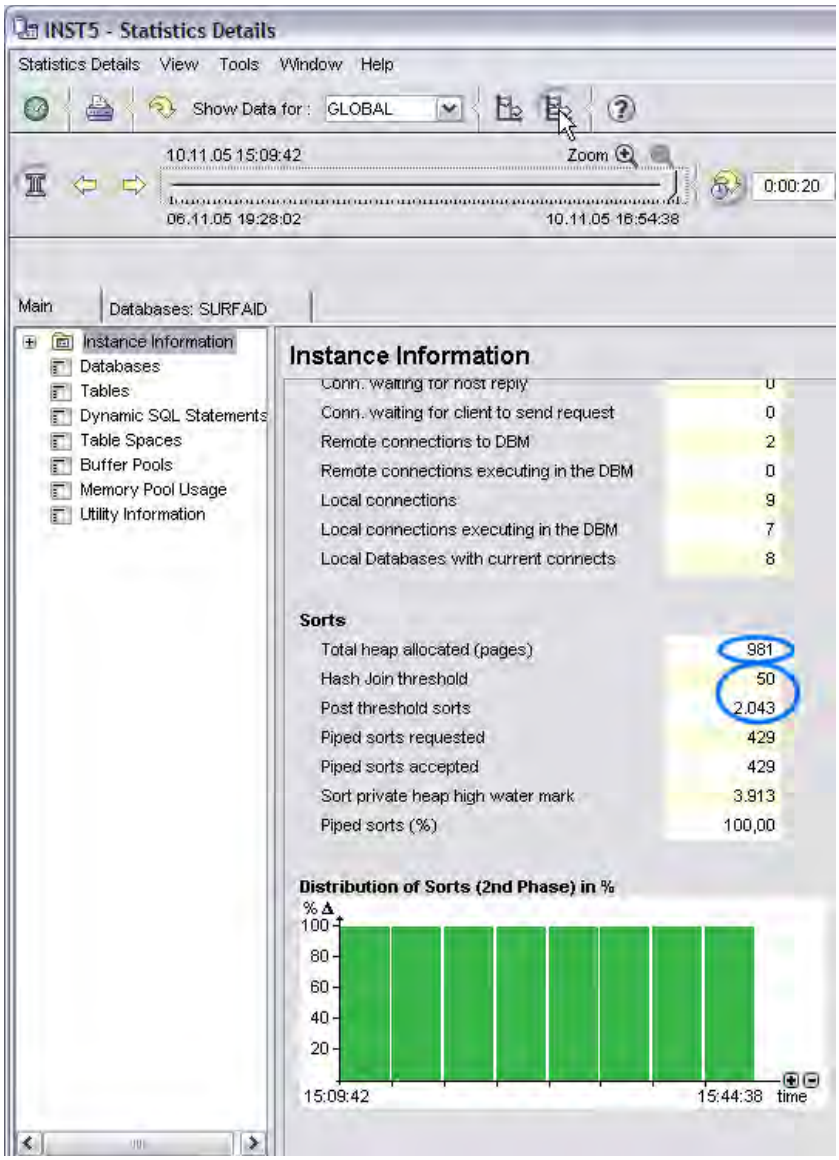
- Statistic Details window: the **Join small overflows** and **Join Overflows** fields
- System Overview Data View for Small Join Overflows vs. Join Overflows
- If Join small overflows are more than 10% of Join Overflows, increase the sort heap.
- If Join overflows are rather high, increase the sort threshold.



If you detect many overflows (of sorts or hash joins) you should determine how to lower them. Increasing the sort heap (database configuration) should help you if you see many join overflows in conjunction with many join small overflows. However, if your problem is a high number of sort overflows, you should check the overall number of post-threshold sorts, which is available in the Instance Information pane of the Statistic Details window. If the number of sort overflows significantly exceeds the number of post-threshold sorts, increasing the sort heap should help you.

Attention: When you compare these numbers take into account that post-threshold sorts counts for all sorts on all databases, while sort overflows are always counted on a per database basis.

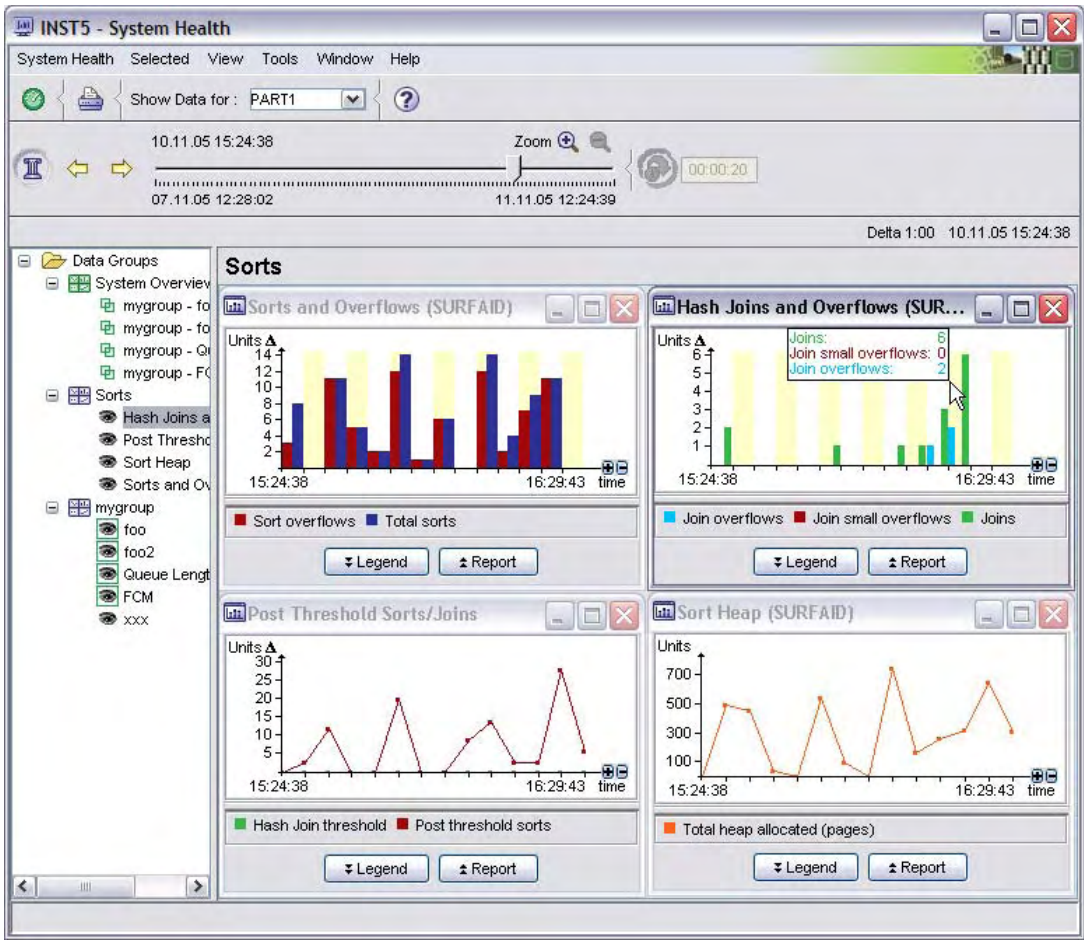
Before increasing the sort heap, you can use the Instance Information pane to determine how many pages are currently allocated.



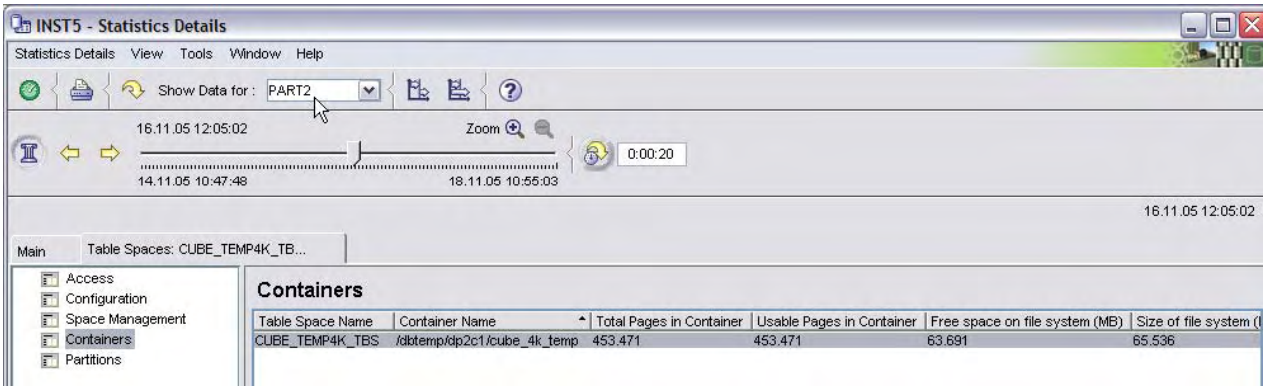
If a significant number of sorts (more than 5%) end as post-threshold sorts or as Hash join threshold (these are post-threshold hash joins), consider increasing the sort heap threshold in the database manager configuration.

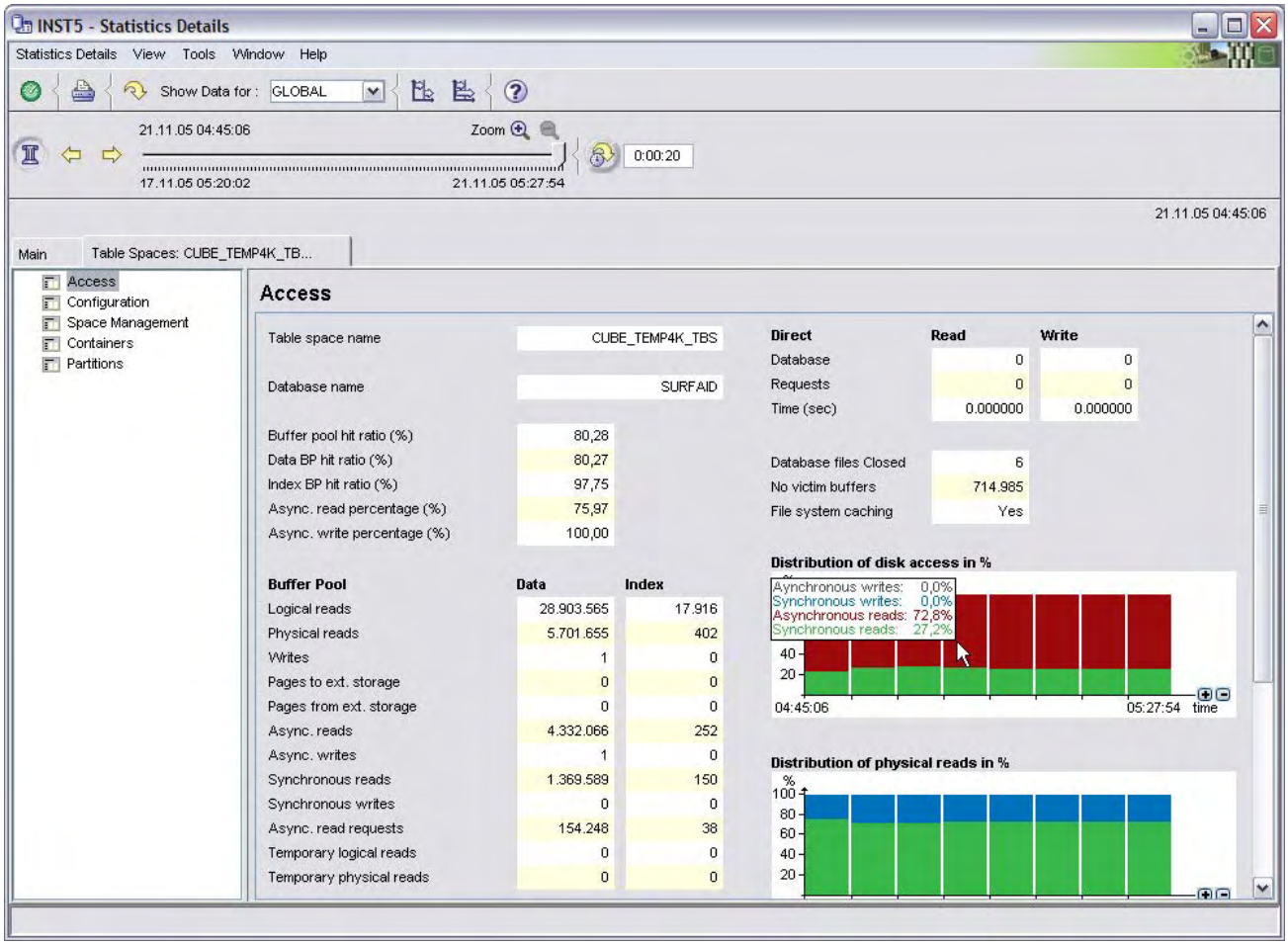
A more general way to resolve sort problems is to avoid sorting through the use of materialized query tables (MQTs). (You can create MQTs on typical joins, ORDER BYs, GROUP BYs, and so on your base tables.) For more information about MQTs, see section 3.9.

You can also set up data views for sort performance indicators as illustrated in the following figure to support a more intuitive detection of the aforementioned sort problems.



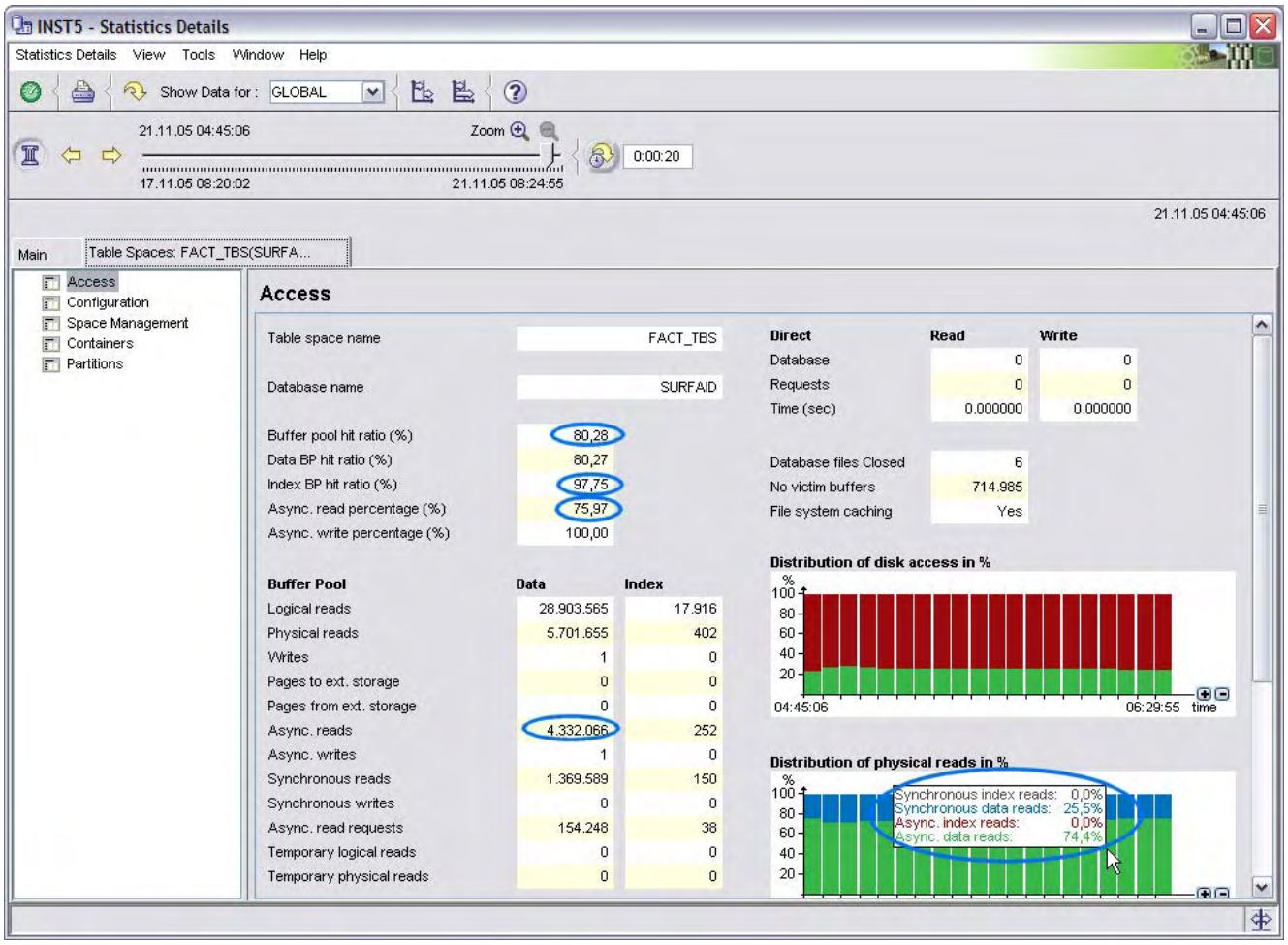
If you detect many sort or hash join overflows, you might first want to check how much data is actually spilling to disk for being sorted. You can do this by looking at the temporary table space consumption and usage in the Statistics Details window for the temporary table space in the Containers and Access panes.





3.3 Page I/O tuning

Business Intelligence workloads impose a large amount of I/O on tablespaces. For this reason, it is essential to have a close look at page I/O metrics. You can do this analysis at the database, bufferpool, and tablespace level, as well as at the application level. This scenario focuses on the tablespace level. 1. Open the Statistic Details window and display the Table Spaces pane. 2. Double-click a tablespace. You should start with your most critical tablespace (the one where the fact table or tables are located). The Access pane opens, which contains details about page I/O. 3. Evaluate the following metrics:



Bufferpool hit ratio

Check the bufferpool hit ratio. The higher it is the less page I/O has to be done to the tablespace containers. In large Business Intelligence systems it is typically not possible to achieve a high overall bufferpool hit ratio. Therefore, it is a good practice to at least tune the index page hit ratio. If your index data is located in a separate tablespace than the table data, examine the hit ratio for that tablespace and tune it by gradually increasing the bufferpool size, (which can be done dynamically) and by running a test workload against the system. While you are doing this, watch the bufferpool hit ratio for the index tablespace in the Access pane. Stop when increasing the bufferpool size does not increase the hit ratio anymore.

You can determine which bufferpool is caching data for this tablespace by looking at the Configuration pane in the **Bufferpool – Currently being used** field, which displays the ID of the responsible bufferpool.

Prefetching

Prefetching is a technique that is used by DB2 to decouple I/O operations to the tablespace containers from the page requests by the process that is executing a statement.

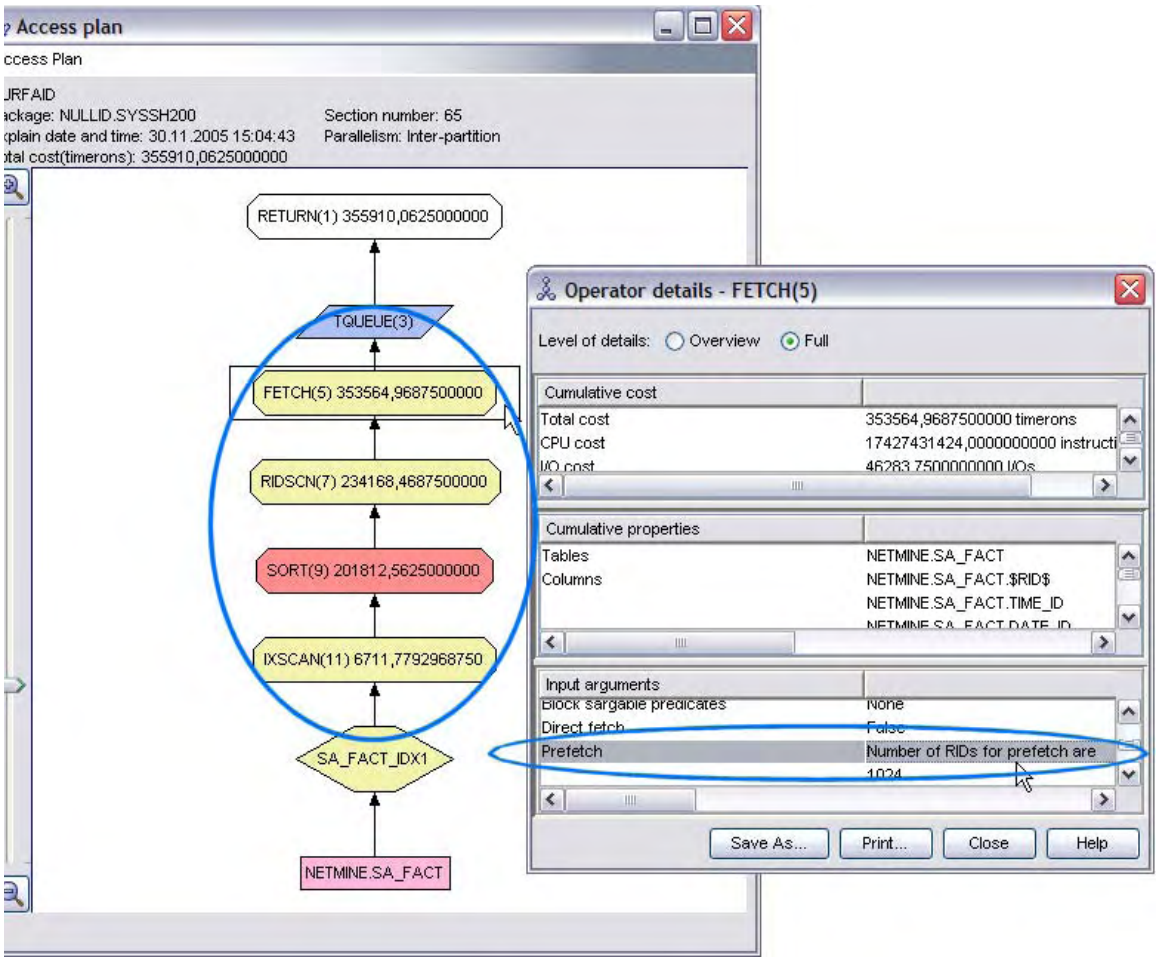
DB2 anticipates the pages that will be requested next and then asynchronously prefetches them to the bufferpool while the actual statement execution continues.

The amount of prefetching can be seen in the **Async. Reads** and **Async. Read percentage (%)** fields and in the **Distribution of physical reads in %** graph of the Access pane.

In addition, you can check single statements if they do prefetching if you look at the access plan of the statements (as described in section 3.5.4, “Understanding statement plan).

One type of prefetching is sequential prefetching, which is chosen when pages on disk are expected to be in the physically needed order (also referred to as *cluster ratio*). Sequential prefetching is an option for table scans. In the explain output, look for nodes of type TBSCAN and double-click them. In the **Input arguments** box, check the **Prefetch** argument to see if sequential prefetching is being performed for that table scan.

Another type of prefetching is list prefetching, which is chosen when pages on disk are not in the required order but the table is accessed through the use of an index. For list prefetching the RIDs that are retrieved from the index are sorted, and then the list of pages that they point to is prefetched. In the explain output look for chains of nodes of type IXSCAN, SORT, RIDSCN, and FETCH. Double-click the FETCH node. In the **Input arguments** box check the **Prefetch** argument to see if list prefetching is being performed for that table scan.



Sequential prefetching is the preferred way of performing table scans. You can support DB2 choosing and performing sequential prefetching by ensuring a good cluster ratio on the columns the table is sorted over right after a table scan. This is achieved by maintaining an index on the same columns in the same sort order and declaring it as the CLUSTERING INDEX. If you have created a clustering index on an existing table, you should run REORG and RUNSTATS on that table after you create the index.

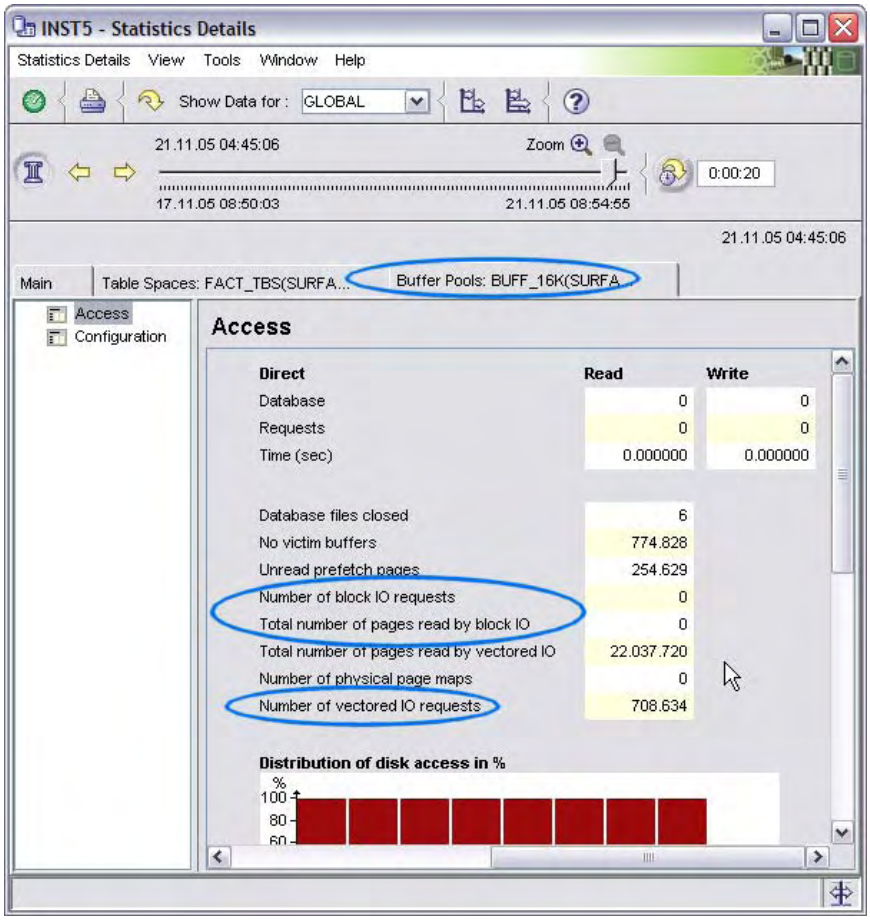
If you determine that DB2 does a lot of prefetching (by evaluating the metrics in the Access pane), you should determine if your system is I/O bound by looking at the I/O wait time indicator (as described in section 3.1.1). If this is the case, try to increase the PREFETCHSIZE parameter for the tablespaces where prefetching is done. But be aware that prefetching might not have been the cause of the I/O waits.

Increasing the PREFETCHSIZE too much leads to wasted I/O operations, which will be visible as a lower bufferpool hit ratio. So as you increase PREFETCHSIZE, also pay attention to the hit ratio. As soon as it goes down you might have increased PREFETCHSIZE too much. If you are monitoring on a bufferpool level you also have the counter available for “Unread prefetch pages” that provides direct information about prefetching effectiveness.

Block-based bufferpools

A further step to tune sequential prefetching is to also allow the pages in the bufferpool to be in the same order as they are on disk, which allows DB2 to use single I/O operations to read entire sequences of contiguous pages into the bufferpool. To enable this you must create the bufferpool with the option to reserve a fraction of the entire pool for block-based I/O.

You can monitor how effective this option is set up in the bufferpool details in Access pane of the Statistics Details window. If you have set up a block-based bufferpool and you see that the **Number of block based IO requests** field contains a very low number or almost the same as is in the **Number of vectored IO requests** field, you should consider changing the block size parameter of the bufferpool. The block size should be aligned with the extent size of the tablespaces. If this is not the case you might waste space in the block area of the bufferpool. If the value of the **Total number of pages read by block IO** field divided by the value of the **Number of block IO requests** field is much less than the defined block size, your block size might be too high with regard to the extent size of the tablespaces.



Read efficiency

This important I/O-related performance indicator has already been described to some extent in section 2.2. If you encounter many read rows but see only a few selected rows

you should revise your queries and determine if you are doing many table scans, which could be avoided by an appropriate index. Also, you might consider using MQTs for frequently joined tables. See section 3.5.4 for details about the execution plan.

3.4 Detecting skews

There are two basic approaches to detect skews with DB2 PE:

- You can set up a visualization dashboard in the System Health window for performance indicators that you want to know the skew on the partitions for.
- You can use the table displays of the group views in the data panels of DB2 PE.

3.4.1 Visualized skew detection

“The following figure is an example of a dashboard that has been set up in System Health to identify skews on sort behavior. See section 3.11 for details about setting up data views.

Another interesting way to visually recognize data skew is available in the statistics details pane as described in 3.4.1.2

3.4.1.1 Bufferpool hit ratio skew

The bufferpool hit ratio can be evaluated on different levels:

- Database
- Tablespace
- Bufferpool
- Application

For all of these levels, DB2 PE also facilities checking for skews in the bufferpool hit ratio.

Open the Statistic Details window, switch to group view, and select the single panes for Database, Table Spaces, and Buffer Pools to verify the hit ratio distribution over the partitions.

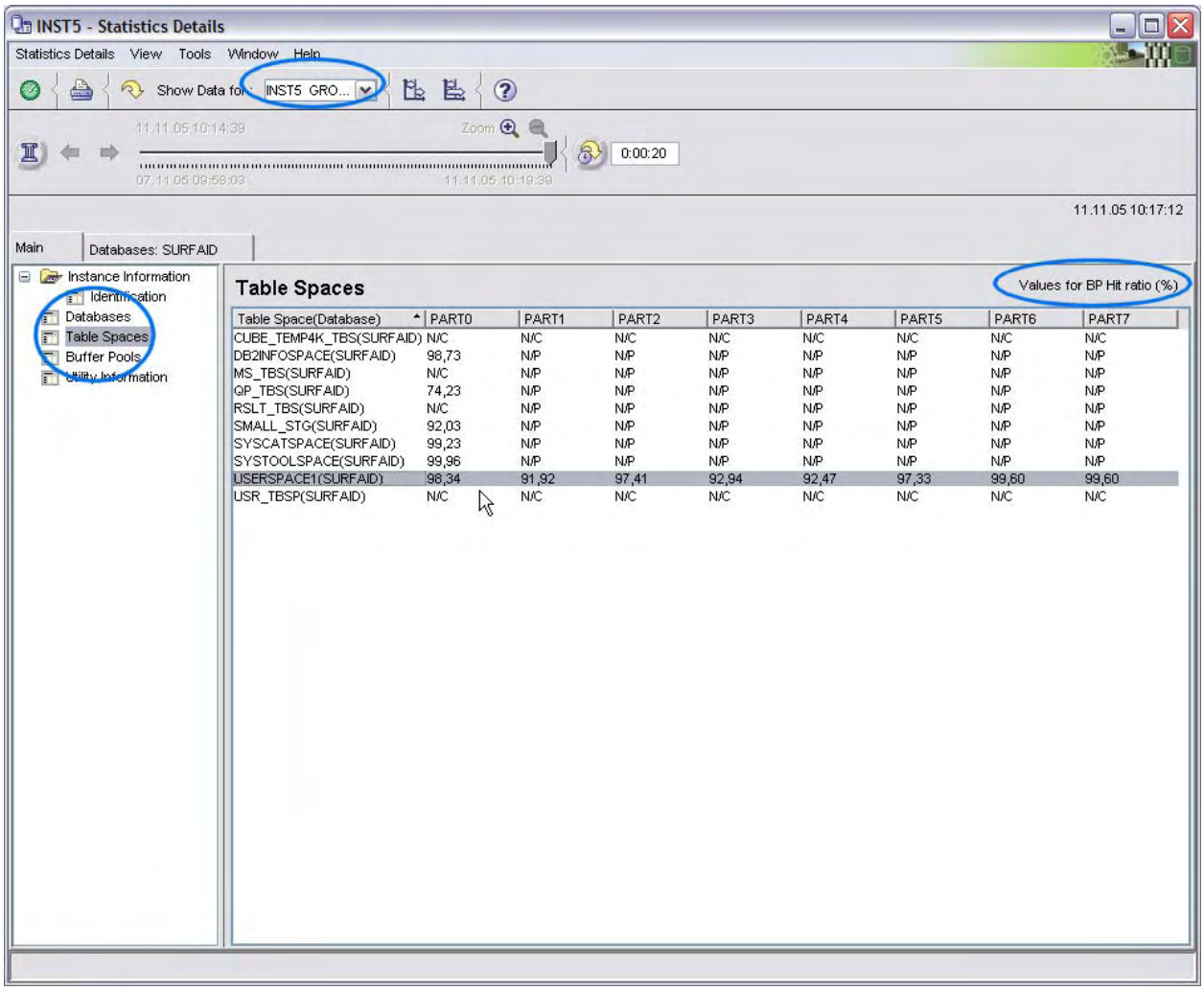
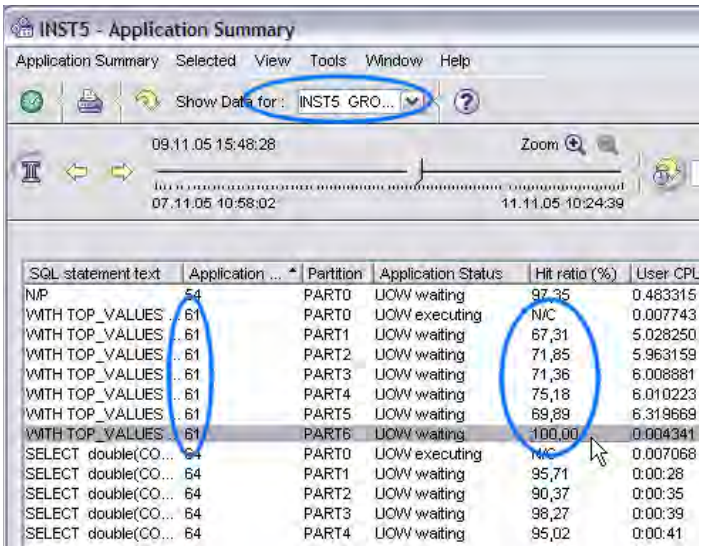


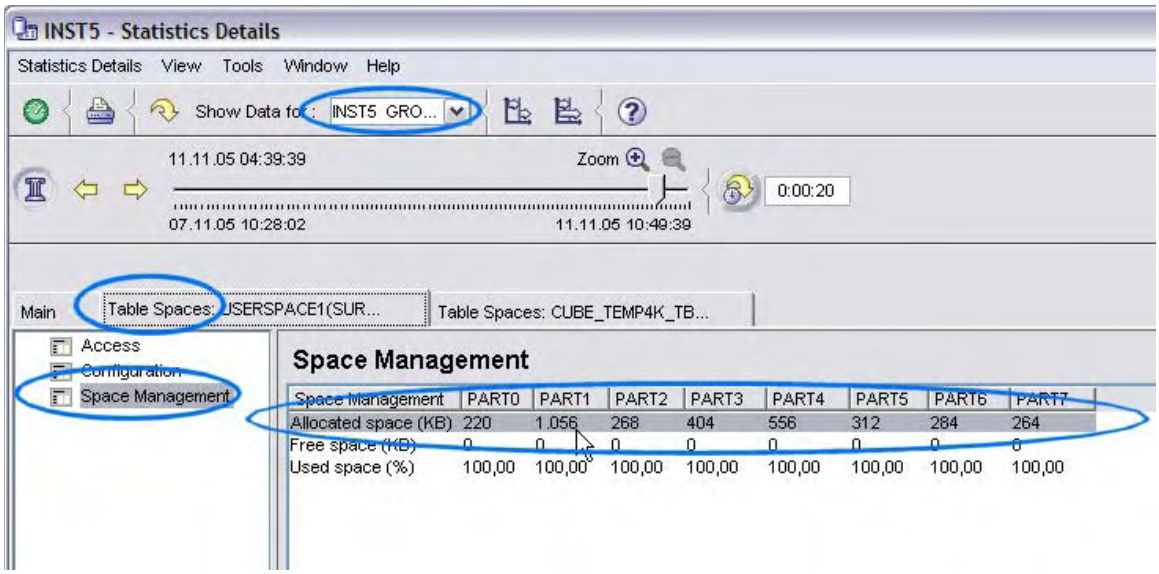
Table Space(Database)	PART0	PART1	PART2	PART3	PART4	PART5	PART6	PART7
CUBE_TEMP4K_TBS(SURFAID)	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C
DB2INFOSPACE(SURFAID)	98,73	N/P	N/P	N/P	N/P	N/P	N/P	N/P
MS_TBS(SURFAID)	N/C	N/P	N/P	N/P	N/P	N/P	N/P	N/P
QP_TBS(SURFAID)	74,23	N/P	N/P	N/P	N/P	N/P	N/P	N/P
RSLT_TBS(SURFAID)	N/C	N/P	N/P	N/P	N/P	N/P	N/P	N/P
SMALL_STG(SURFAID)	92,03	N/P	N/P	N/P	N/P	N/P	N/P	N/P
SYSCATSPACE(SURFAID)	99,23	N/P	N/P	N/P	N/P	N/P	N/P	N/P
SYSTOOLSPACE(SURFAID)	99,96	N/P	N/P	N/P	N/P	N/P	N/P	N/P
USERSPACE1(SURFAID)	98,34	91,92	97,41	92,94	92,47	97,33	99,60	99,60
USR_TBSP(SURFAID)	N/C	N/C	N/C	N/C	N/C	N/C	N/C	N/C

If you are interested in hit ratio values divided by index and data, you can find them by drilling down to the database or tablespace details respectively.
To check for the hit ratio distribution on application level, open the Application Summary window, switch to group view, and sort over the **Application Handle** column to see application details for each partition and grouped together per application. Then check the **Hit ratio (%)** column for uneven distribution of the hit ratio.



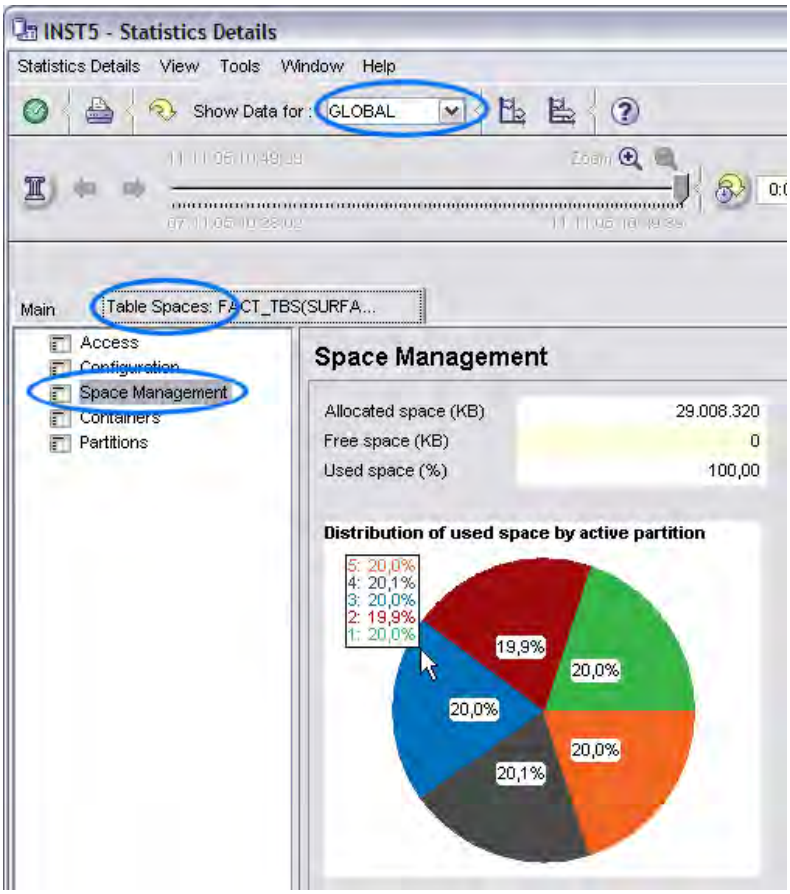
3.4.1.2 Data skew

DB2 PE provides means to check for data distribution at the database and tablespace level. Open the Statistics Details window, select the Table Spaces pane, and double-click the tablespace that you are interested in. Switch to group view and select the Space Management pane. A table is displayed that contains the allocated and free space of the selected tablespace on each single partition.



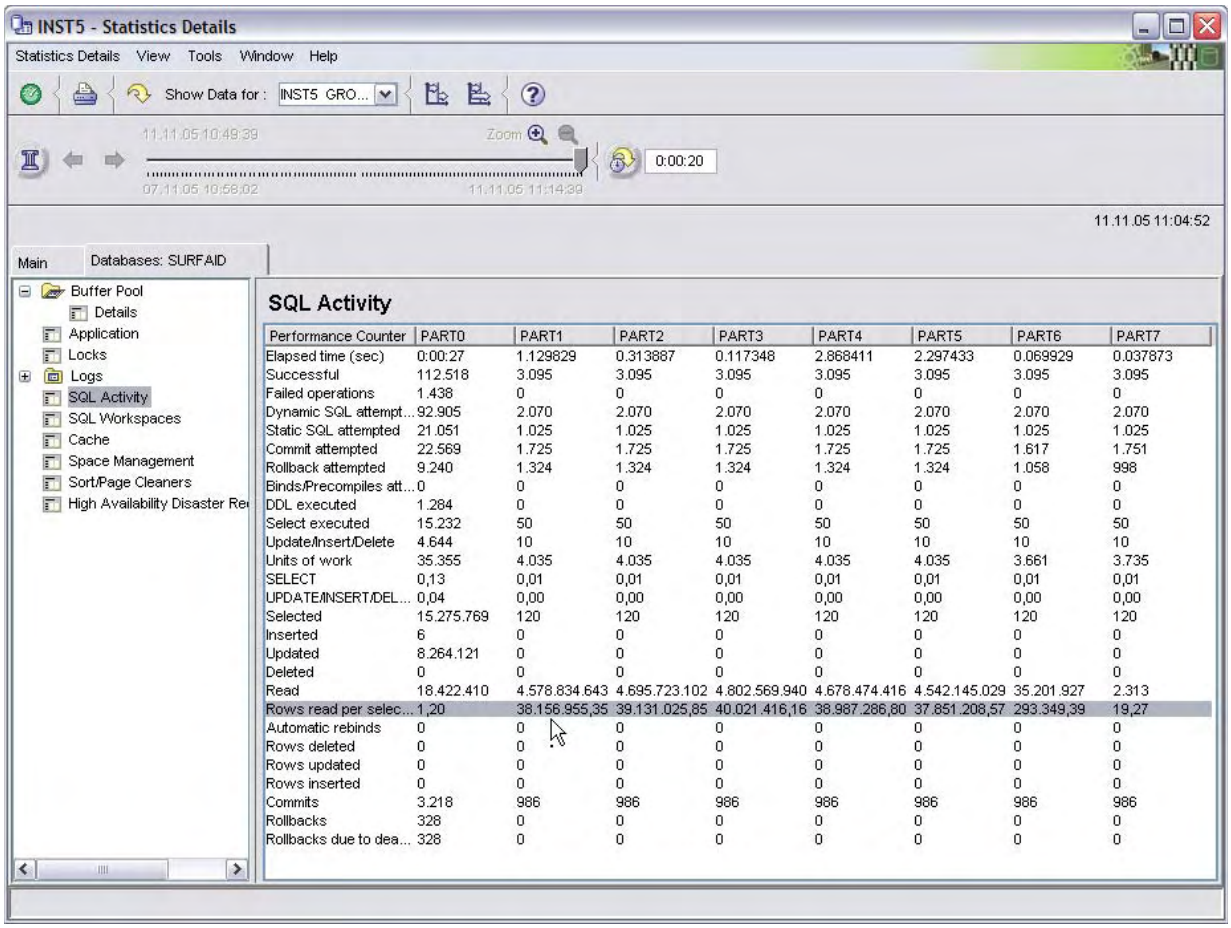
There is also a visual way to check for data skews on tablespace level. Open the Tablespaces – Space Management pane and switch to global view. A pie chart is displayed that visualizes the data distribution of the tablespace in terms of used pages.

Note that this pie chart is available only in online monitoring. Due to technical reasons, it is not available in history mode.



3.4.1.3 Other skews

The group view feature of DB2 PE provides ways to look for skews on many more performance indicators. As an example, the following figure shows how to check for skews on DB2 I/O data, such as rows read and rows selected.



Performance Counter	PART0	PART1	PART2	PART3	PART4	PART5	PART6	PART7
Elapsed time (sec)	0:00:27	1.129829	0.313887	0.117348	2.868411	2.297433	0.069929	0.037873
Successful	112.518	3.095	3.095	3.095	3.095	3.095	3.095	3.095
Failed operations	1.438	0	0	0	0	0	0	0
Dynamic SQL attempt...	92.905	2.070	2.070	2.070	2.070	2.070	2.070	2.070
Static SQL attempted	21.051	1.025	1.025	1.025	1.025	1.025	1.025	1.025
Commit attempted	22.569	1.725	1.725	1.725	1.725	1.725	1.617	1.751
Rollback attempted	9.240	1.324	1.324	1.324	1.324	1.324	1.058	998
Binds/Precompiles att...	0	0	0	0	0	0	0	0
DDL executed	1.284	0	0	0	0	0	0	0
Select executed	15.232	50	50	50	50	50	50	50
Update/Insert/Delete	4.644	10	10	10	10	10	10	10
Units of work	35.355	4.035	4.035	4.035	4.035	4.035	3.661	3.735
SELECT	0,13	0,01	0,01	0,01	0,01	0,01	0,01	0,01
UPDATE/INSERT/DEL...	0,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Selected	15.275.769	120	120	120	120	120	120	120
Inserted	6	0	0	0	0	0	0	0
Updated	8.264.121	0	0	0	0	0	0	0
Deleted	0	0	0	0	0	0	0	0
Read	18.422.410	4.578.834.643	4.695.723.102	4.802.569.940	4.678.474.416	4.542.145.029	35.201.927	2.313
Rows read per selec...	1,20	38.156.955,35	39.131.025,85	40.021.416,16	38.987.286,80	37.851.208,57	293.349,39	19,27
Automatic rebinds	0	0	0	0	0	0	0	0
Rows deleted	0	0	0	0	0	0	0	0
Rows updated	0	0	0	0	0	0	0	0
Rows inserted	0	0	0	0	0	0	0	0
Commits	3.218	986	986	986	986	986	986	986
Rollbacks	328	0	0	0	0	0	0	0
Rollbacks due to dea...	328	0	0	0	0	0	0	0

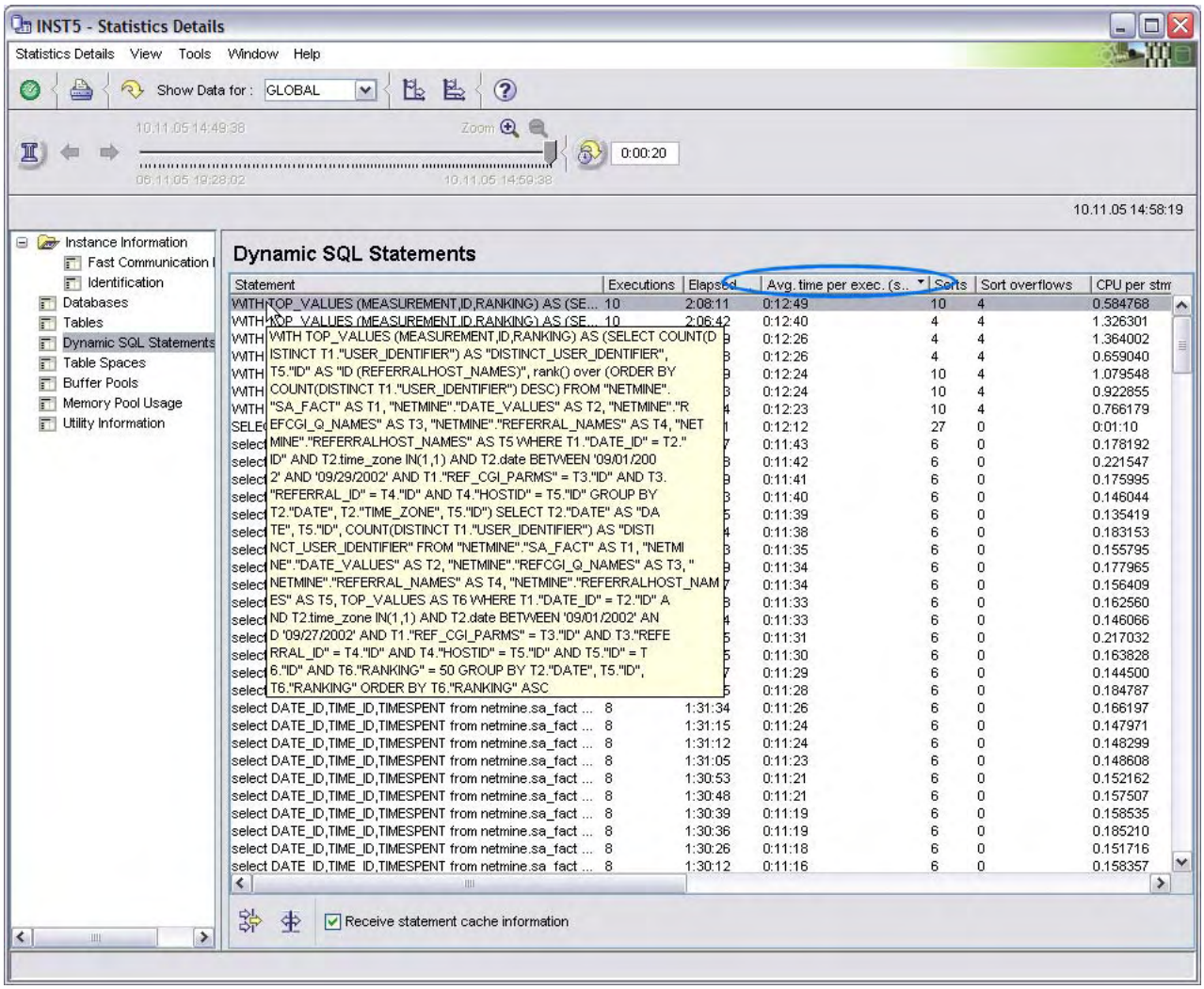
In this example figure, two partitions deviate from the average rows read per selected number. Partition 0 returns almost each row read because it is the coordinator partition in this particular scenario. Partition 6, however, has a multiple of the average rate and might need some attention.

You can look for skews on the other important performance indicators by opening the associated panes and switching to group view.

3.5 Understanding long-running queries

3.5.1 Identifying the top 10 statements

Whether you want to tune you Business Intelligence system or you want to find the source for poor performance in general, you need a starting point. A good way to start is to find the top 10 statements with regard to execution time. This can easily be achieved by opening the Statistics Details window and accessing the Dynamic SQL Statements pane. Click the **Receive statement cache information** button at the bottom of the pane, and DB2 PE will request and display this data. Now sort the **Avg. time per execution** column. You might also want to use other columns to find the top 10 statements, such as the **Executions** and **Elapsed time** columns.



You can also find the top 10 statements by examining the performance history. To do so, switch to history mode and scroll to the point in time that you want to perform this check for.

3.5.2 Linking top 10 statements to execution details

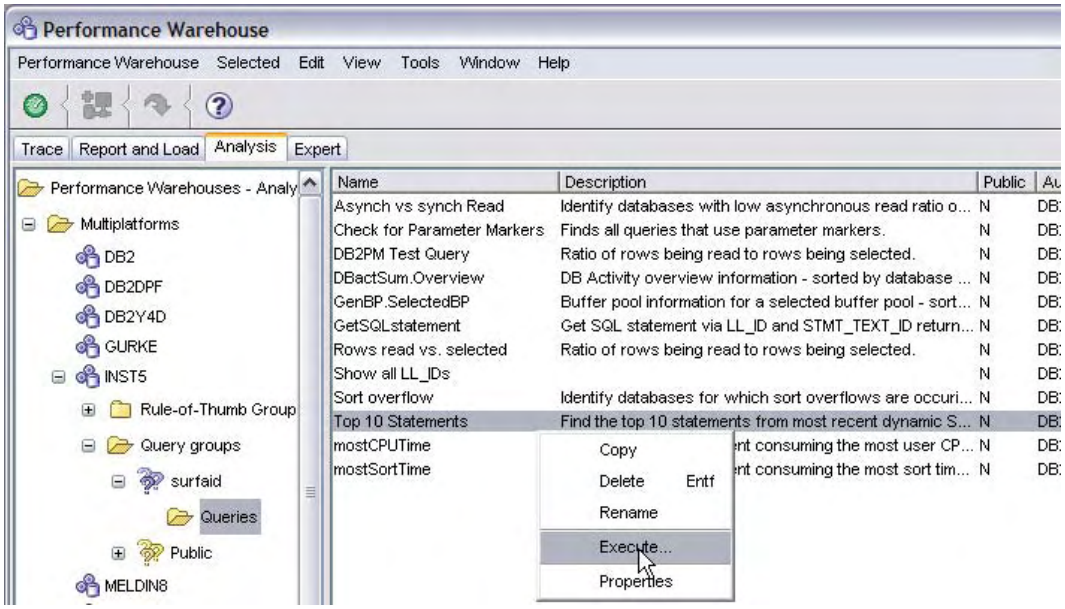
The dynamic SQL statement information shown in the previous section represents aggregated data over all executions of these statements. This means that it is not directly possible to drill down to the applications (as displayed in the application summary) that executed these statements.

But you can leverage the Query facility in Performance Warehouse to define and execute a query that shows you all entries when one of the top 10 statements has been captured in the application summary. To do so: 1. Open the Performance Warehouse – Analysis window, go to Query groups, and create a private group is one does not already exist. 2.

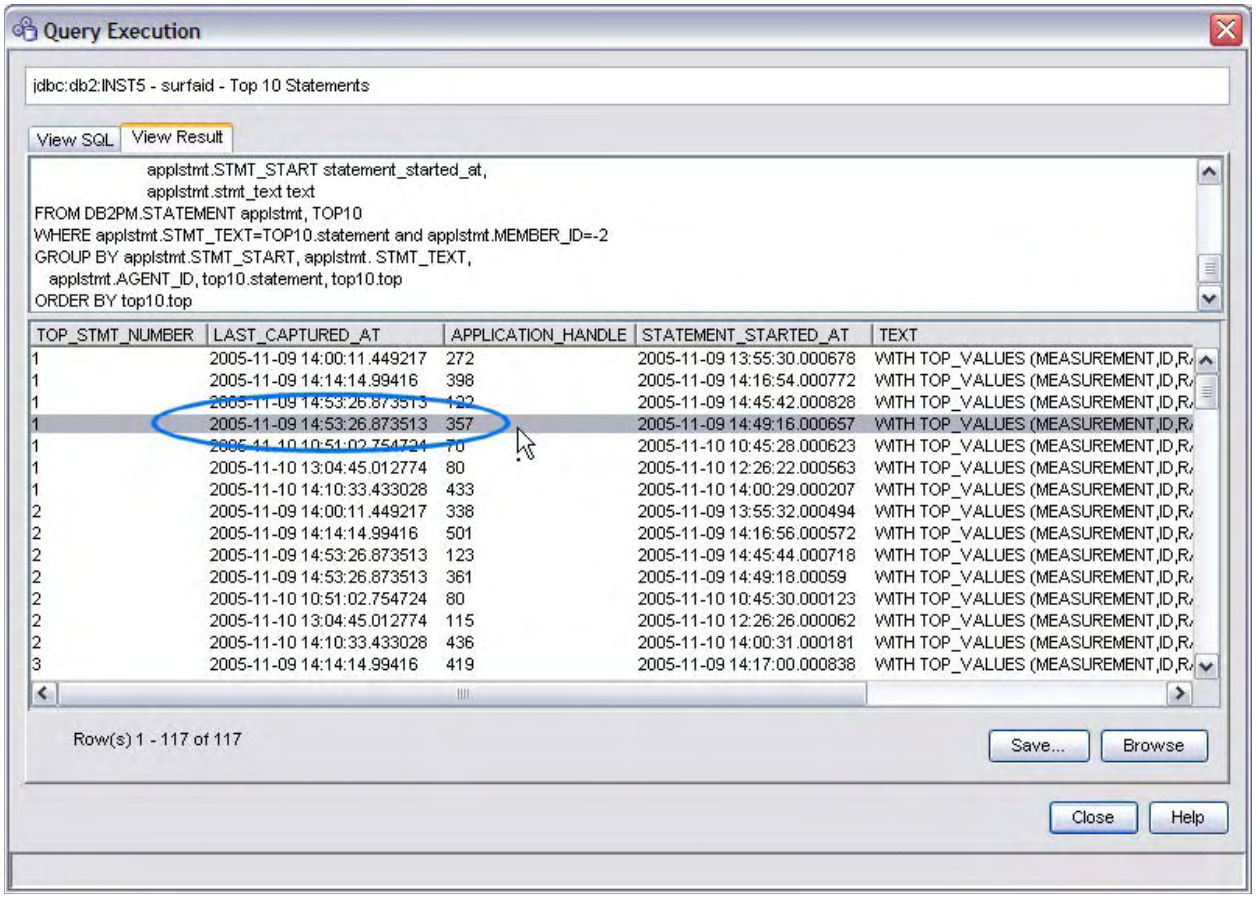
Right-click the group and select **Create**. Give the new query a name and description and paste the following query text in the appropriate entry field on the Definition pane.

```
WITH
LAST(last_ts) AS
      (SELECT MAX(HT_TIMESTAMP)
        FROM DB2PM.HISTORYTOC
        WHERE HT_DATA='DYNAMICSTATEMENTCACHE'),
TOP10(statement, ms_per_execution, top) AS
      (SELECT distinct sql.STMT_TEXT, sql.ATIMEP_EXECUTIONS,
              ROW_NUMBER( ) over
                  (ORDER BY ATIMEP_EXECUTIONS DESC) AS TOP
            FROM DB2PM.DYNSQL sql, LAST
            WHERE sql.INTERVAL_TO=LAST.last_ts
                  AND MEMBER_ID = -2
            ORDER BY ATIMEP_EXECUTIONS DESC
            FETCH FIRST 10 ROWS ONLY)
SELECT TOP10.top top_stmt_number,
       MAX(applstmt.INTERVAL_TO) last_captured_at,
       applstmt.AGENT_ID application_handle,
       applstmt.STMT_START statement_started_at,
       applstmt.STMT_TEXT text
      FROM DB2PM.STATEMENT applstmt, TOP10
      WHERE applstmt.STMT_TEXT = TOP10.statement
            AND MEMBER_ID = -2
      GROUP BY applstmt.STMT_START, applstmt. STMT_TEXT,
               applstmt.AGENT_ID, top10.statement, top10.top
      ORDER BY top10.top
```

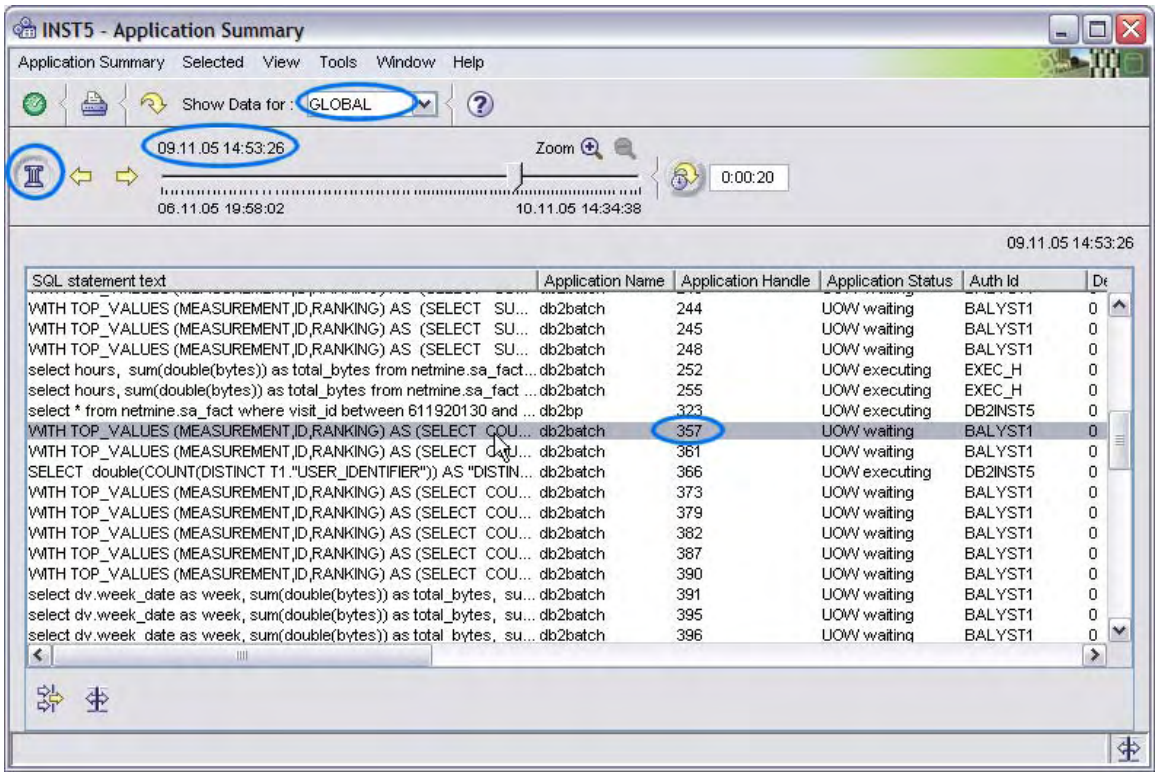
3. Click **OK** to save the query. 4. Right-click the query and select **Execute. I**n the menu that opens, click **Execute** again. Depending on how much data you have recorded in the history, this query might run for several minutes.



When the query finishes running, results such as those that are shown in the following figure are displayed. These results show one row per point in time when an application executed one of the top 10 statements.

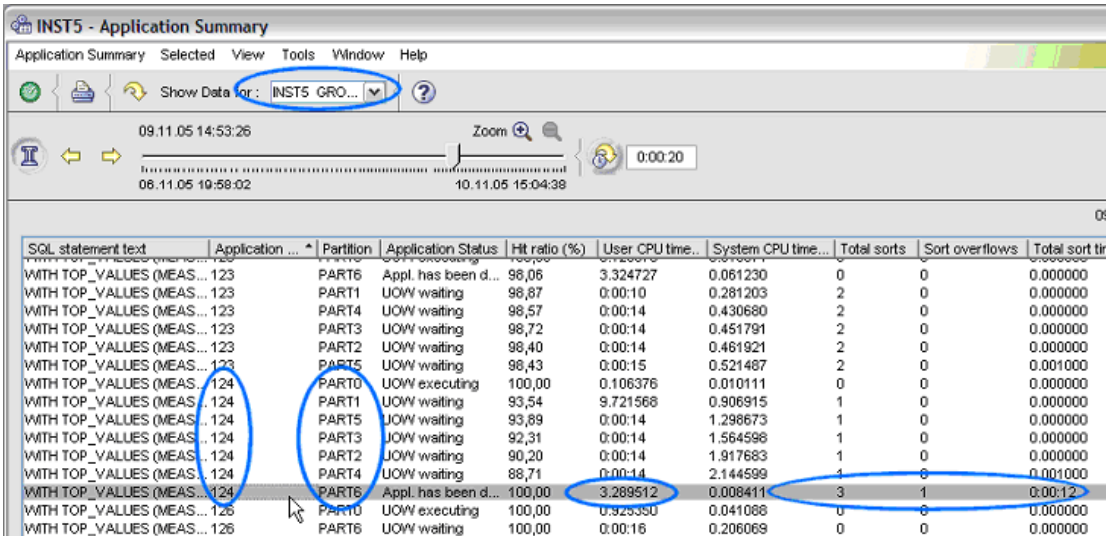


5. Select one of these entries, note the according point in time, and open the Application Summary window. Switch to history mode and scroll to the same point in time. Look for the same application handle as in the selected entry of the query results. Double-click the entry to display the details about the application and you are done.



3.5.3 Checking for skews per query execution

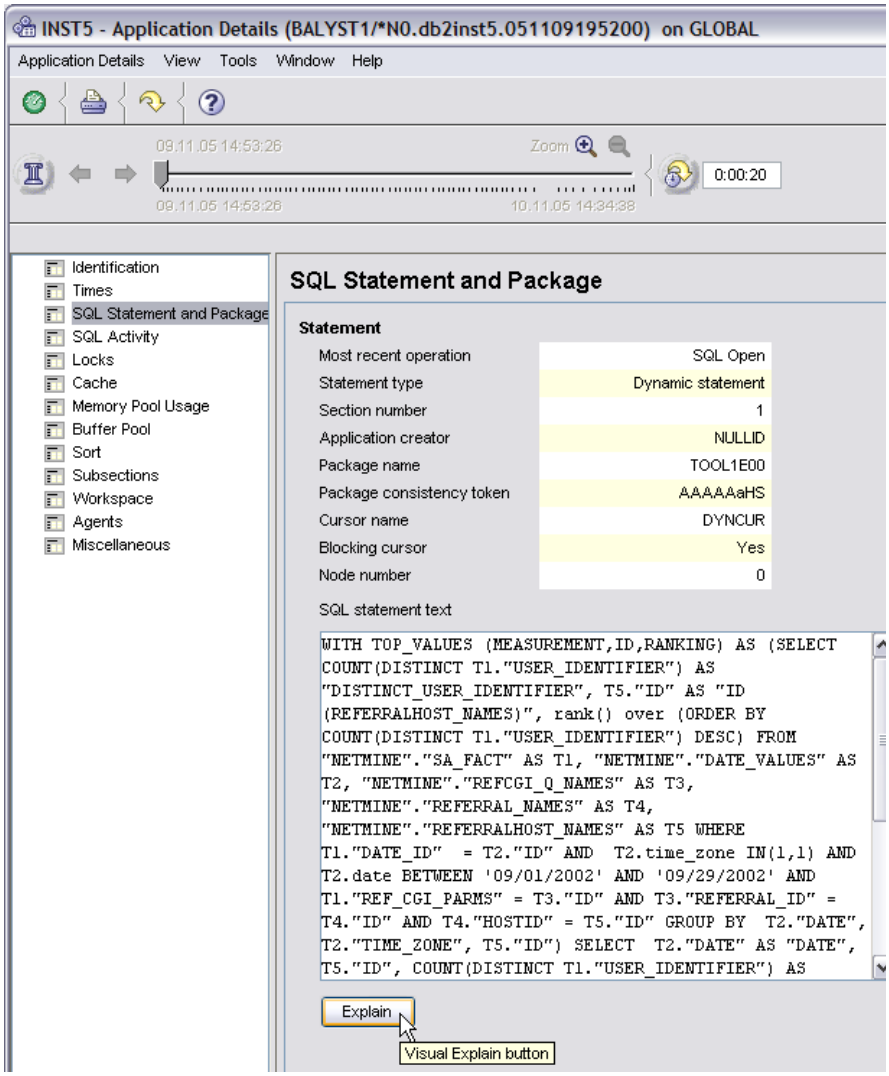
In the Application Summary window, you can check for different types of skews regarding query executions. 1. Switch to group view and sort over the **Application handle** column. 2. Scroll to the application that you are interested in. You will see one row per partition involved in the query execution for the current statement. 3. Check the columns of important performance indicators, such as user CPU time, system CPU time, sort overflows, and hit ratio for skews over the different partitions.



If you have a higher number of partitions you might want to use the filter function to limit the table content to just the application handle that you are interested in. You can then sort more freely over the columns with performance indicators that you want to check for skews.

3.5.4 Understanding the statement plan

After you have found an entry of interest in the Application Summary window you can drill-down into details by double-clicking the entry. In addition to other details you will see information about the **SQL statement and package** in the associated category on the leftmost pane. From here you can also explain the statement by clicking **Explain**.

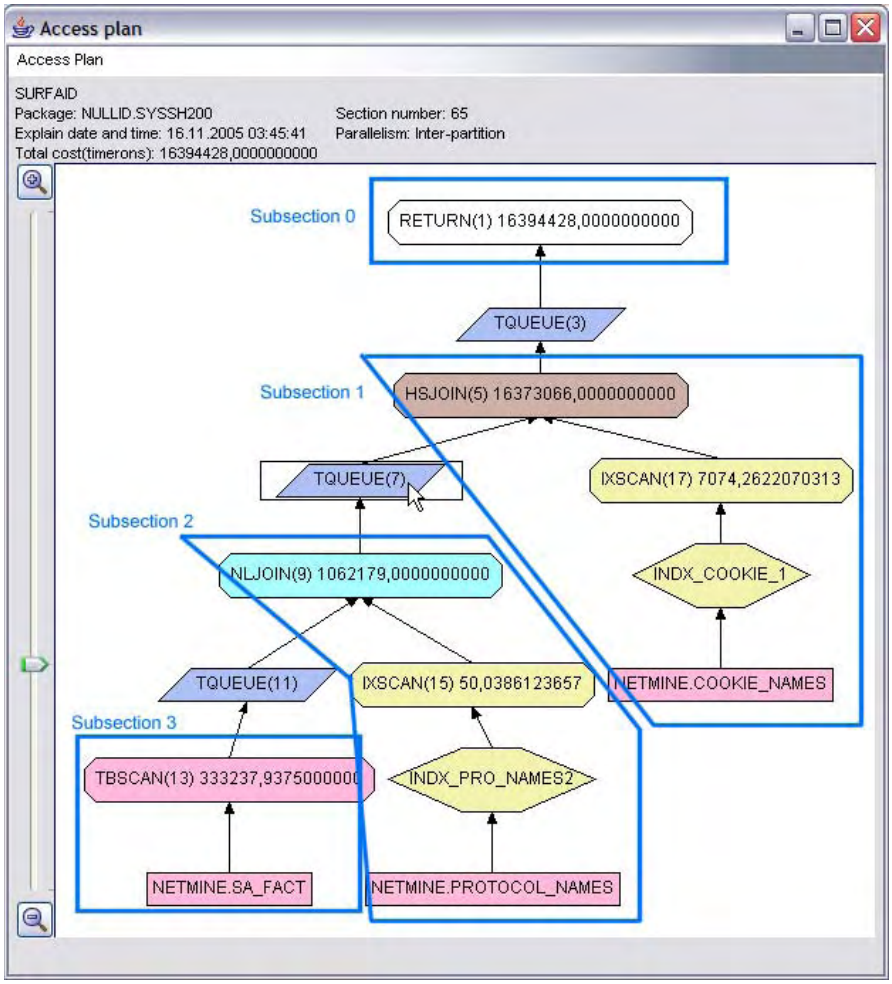


Clicking **Explain** opens the Visual Explain tool (as known from DB2 GUI tools), which you can use to explore the plan details. The numeric values in each box represent the

estimated effort for that operation in an imaginary unit called *timerons*. You can identify the hot spots of the plan by looking for the big increases of those numbers that are higher in the tree. Boxes that are higher in the hierarchy include the aggregated timerons on their direct children boxes (that is, if the number of a parent box is just slightly more than the sum of the numbers in all children boxes, the operation that is represented by that parent box is very cheap).

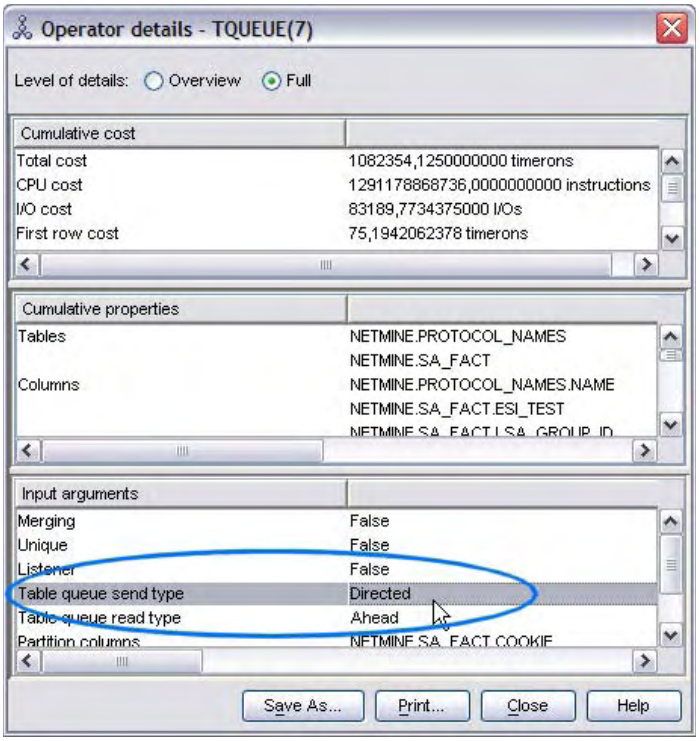
For general information about interpreting explain output please refer to the DB2 documentation.

Note that the same explain capability of DB2 PE is also available directly from the Dynamic SQL Statements pane of the Statistics Details window after you have opened the details pane for a selected statement.



In DPF environments a statement plan is broken down into subsections. Each subsection is executed in a dedicated agent; potentially the same subsection is executed many times in parallel by multiple partitions. The results of one subsection are the input data to the following subsection. The data between subsections is exchanged through the use of table queues. A table queue can be imagined as a never-materialized temporary table within a

communication buffer between partitions. Visual Explain allows you to detect the different subsections. To do so, look for table queues. Each part of the plan tree between table queues is a single subsection. The previous figure shows a statement with four subsections outlined in blue boxes.



An interesting component of the plan is the table queue (TQUEUE), of which there are different types. To determine the type, double-click on a table queue box and look at the **Table queue send type**.

You should look for table queues send types of Directed because this type might impose a potential performance issue. They are usually inevitable if two larger tables need to be joined over columns where at least one of them is not part of a partition key. As a consequence, one or even both of the tables must first be scanned and dynamically hashed on the join column. Then the hashed data is sent via table queues to the according partition where the actual join is performed. This operation results in a lot of data being transferred over the wire if the tables are large.

Requirement: Even if both tables have the join column in the partition key, a directed table queue is required if the two tables have different partition maps (that is, if they are stored in tablespaces of different node groups).

If you detect directed table queues, you might want to reconsider the partition keys or the distribution of tables to node groups, or you might want to revise the queries. The optimum goal to strive for is collocated joins between the tables. You can recognize them by using Visual Explain to look for joins between tables that do not involve a table queue. A common approach to working with different partition keys of tables that you need to join is to create and maintain an MQT on one of the tables with a different partition key than the other joined table. The optimizer will then decide to rewrite queries that these tables to a join that accesses the MQT instead of the base table, which results in a collocated join.

If you detect broadcast table queues for small dimension tables, consider creating replicated MQTs on them (CREATE TABLE … AS … REPLICATED), which means

that they physically exist redundantly on each partition and which allows to optimizer to choose collocated joins when these tables are joined (for example, with the fact table).

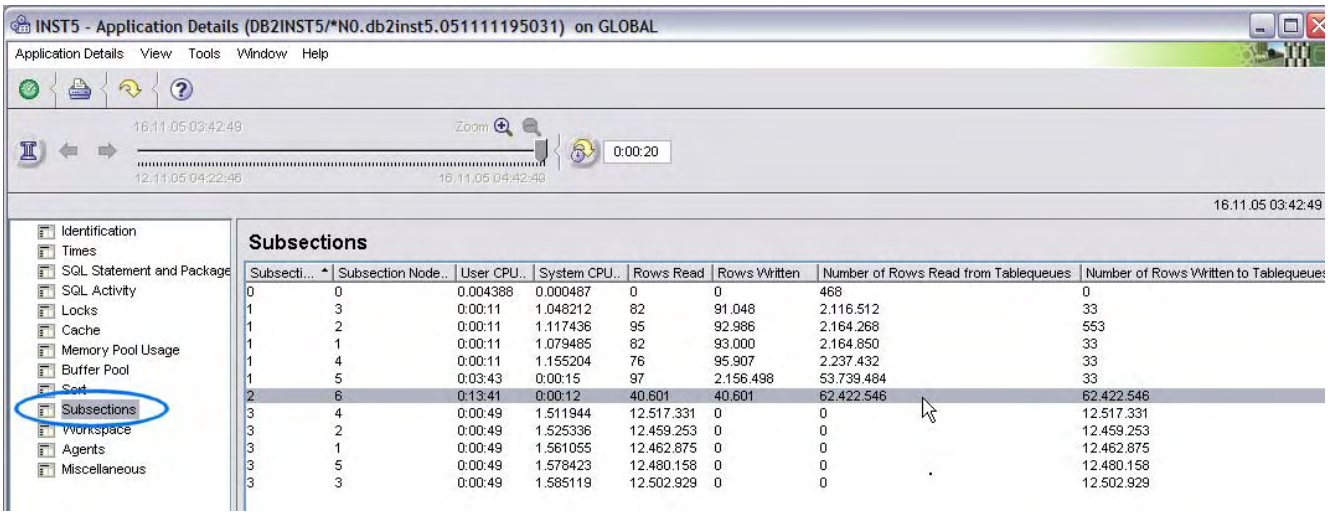
3.5.4.1 Monitoring subsections

In addition to the approach to that is described in section 3.5.3, you can also perform a detailed analysis of the execution per subsection by using the Subsections pane of the Application Details window. The table displays one entry per subsection per partition.

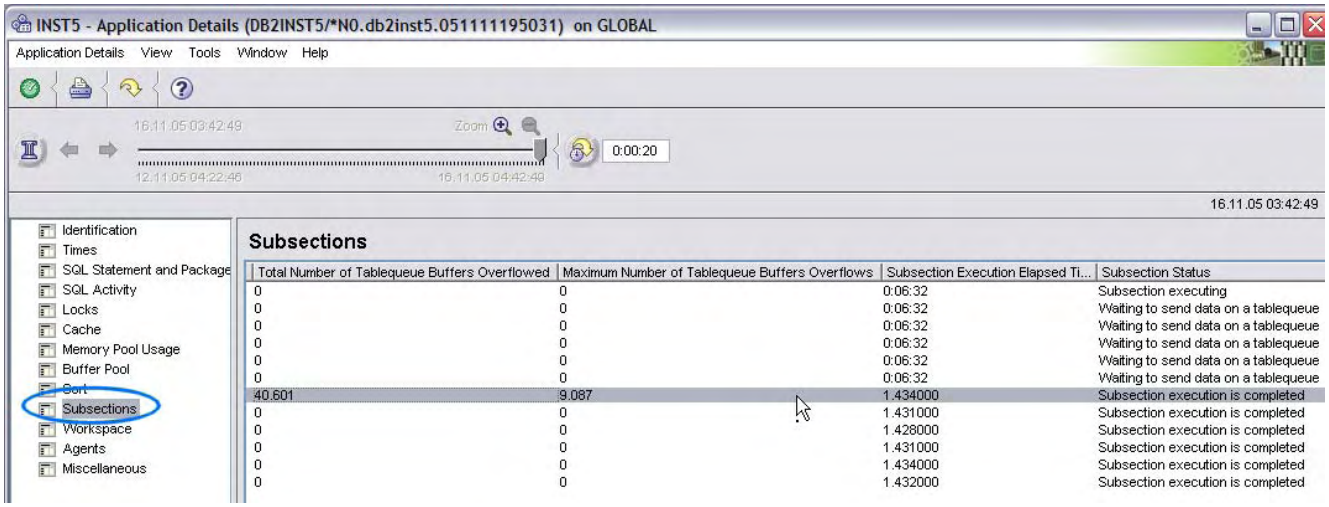
In the following example (which represents the same query that is presented in the Visual Explain example in section 3.5.4) you can see that subsection 3 is executed in parallel by partitions 1 through 5. Within these five entries you can look for skews in the areas of CPU, rows read/written, and table queue length. The latter indicates how much data is read and written from/to a table queue (as shown in the **Number of Rows Read from Tablequeues** and **Number of Rows Written to Tablequeues** columns). If there are significant deviations for the same subsection (for example, partition 5 of subsection 1 in the picture below) a partition might be over-utilized. The reason could be data skew.

Generally, high numbers of table queue lengths are an indicator for directed table queues that might need your attention (as described in section 3.5.4).

The same properties can also be used to understand more about the selectivity of the subsections. For example, subsection 2 (nested loop join with table PROTOCOL_NAMES) has a selectivity of 100 % because it reads the same number of rows from its input table queue as it writes to its output table queue. On the other side, subsection 1 (hash join with COOKIE_NAMES) has a very low selectivity so far (which might change because the query is still executing and subsection 1 is still waiting to send data via the table queue to subsection 0).



Subsecti...	Subsection Node..	User CPU..	System CPU..	Rows Read	Rows Written	Number of Rows Read from Tablequeues	Number of Rows Written to Tablequeues
0	0	0.004388	0.000487	0	0	468	0
1	3	0:00:11	1.048212	82	91.048	2.116.512	33
1	2	0:00:11	1.117436	95	92.986	2.164.268	553
1	1	0:00:11	1.079485	82	93.000	2.164.850	33
1	4	0:00:11	1.155204	76	95.907	2.237.432	33
1	5	0:03:43	0:00:15	97	2.156.498	53.739.484	33
2	6	0:13:41	0:00:12	40.601	40.601	62.422.546	62.422.546
3	4	0:00:49	1.511944	12.517.331	0	0	12.517.331
3	2	0:00:49	1.525336	12.459.253	0	0	12.459.253
3	1	0:00:49	1.561055	12.462.875	0	0	12.462.875
3	5	0:00:49	1.578423	12.480.158	0	0	12.480.158
3	3	0:00:49	1.585119	12.502.929	0	0	12.502.929



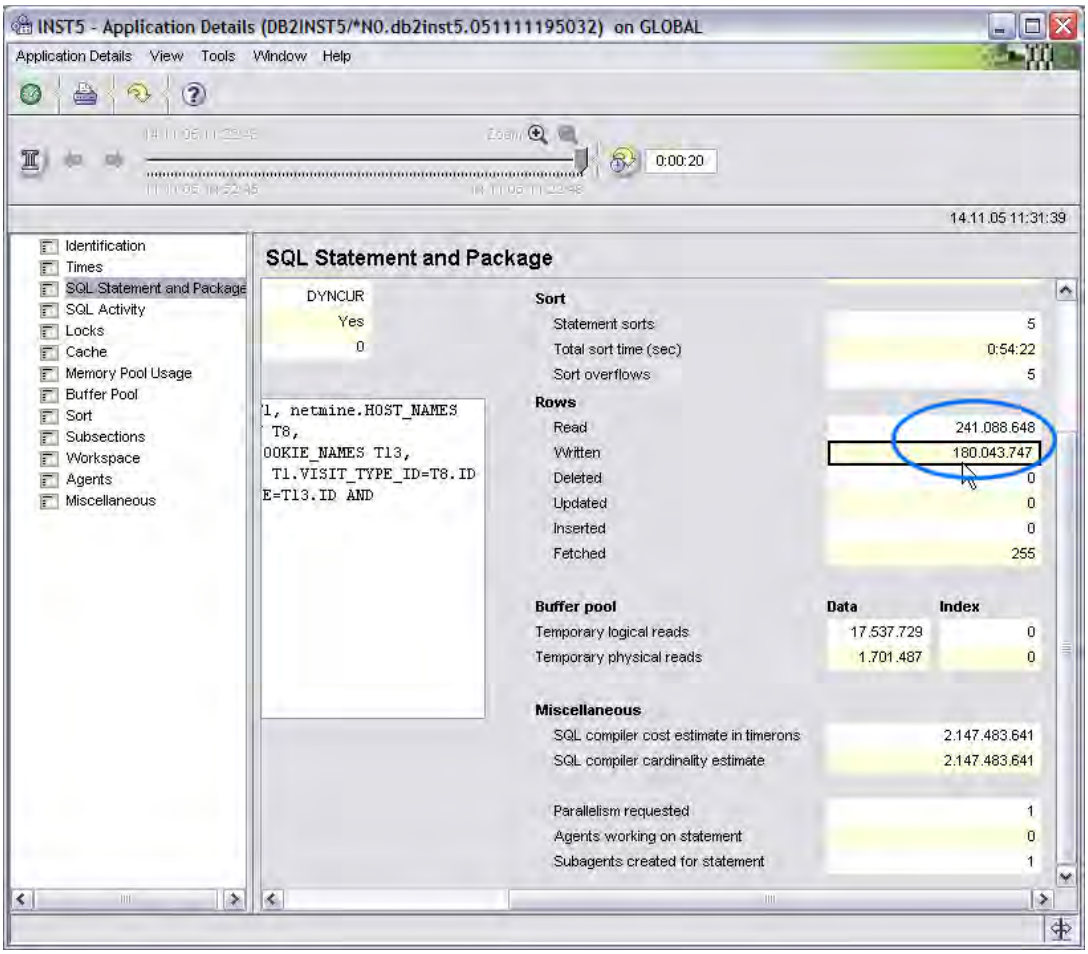
By scrolling horizontally you can check for other properties.
The **Subsection Status** column indicates the current state of query execution. In the previous figure you can see that subsections 3 and 2 are already done. Subsection 1 is actually done as well but still needs to send some data to subsection 0, which is the coordinator subsection and which is currently executing (that is, receiving data).

The **Total Number of Tablequeue Buffers Overflowed** and **Maximum Number of Tablequeue Buffers Overflows** columns indicate if data had to be written to temporary tablespace. This can occur if there are multiple partitions reading for the same table queue and some of them are not fast enough to read the data. The sending subsection is then spilling the data for the slower receivers to disk so that it can continue to send the data to the other receivers.

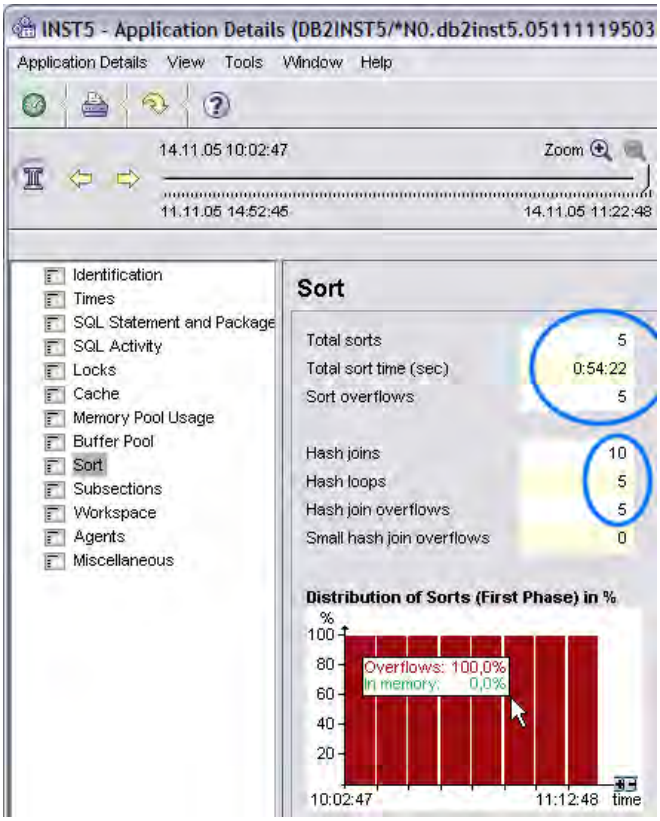
3.5.5 Check for SORT issues

A general indicator that large SORTs are happening and might be the cause for a slow query execution is a high rows written rate. Even though the statement is just a SELECT, rows are written to disk if a SORT or HASH JOIN cannot be done in memory. The following figure shows a rather high number of rows being written to disk, which is indicated in the application details.

Note: You can also check for **Rows Written** in dynamic SQL table in the Statistics Details window. Here you can likely sort over this column to see the statements that are writing to disk the most.



To verify your suspicion of a sort problem you should determine if the application that is executing this statement has many sort or hash join overflows. You can find this information in the Sort category of the Application Details window.

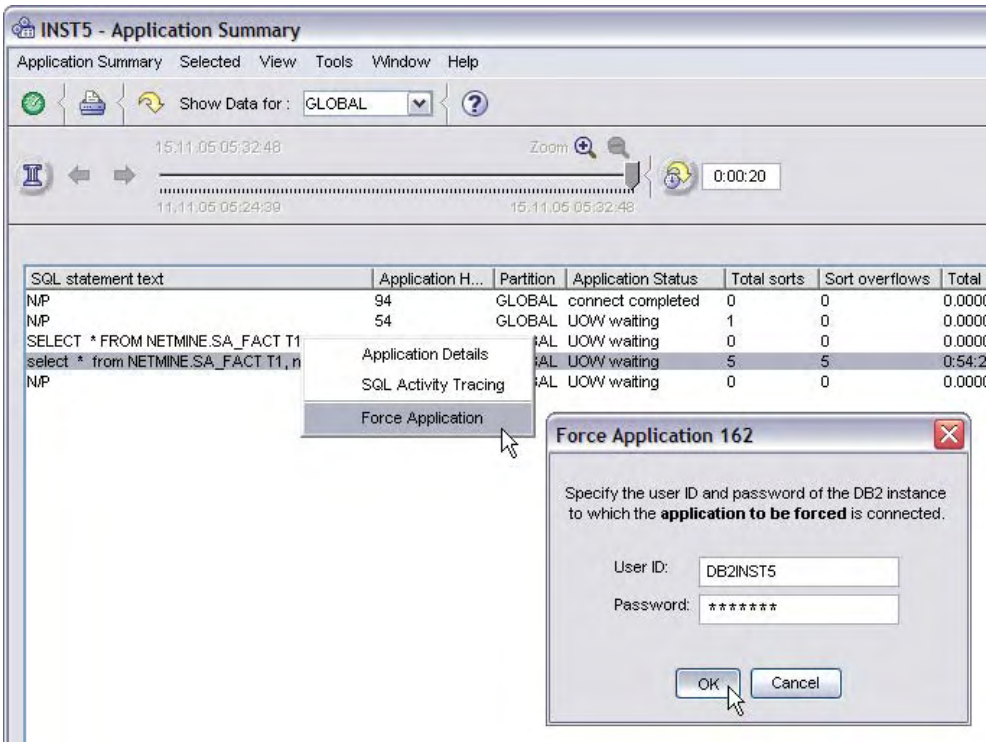


If you are interested in sort indicators (for example, if you want to find the top statements with regard to sort issues), you can also use the Dynamic SQL Statements window pane of the Statistics Details window and sort over **Sort overflows** or **Rows written** columns. See section 3.5.1 for more information.

3.5.6 Canceling long-running queries

If you identify a long-running query that is having a severe negative impact on the entire performance of your system, and you do not have access to or influence on the person or application that has issued this query, you can use DB2 PE to cancel this query. The associated application will be rolled back and closed immediately.

To do so, open the Application Summary window in global view, right-click the application that issued the query, and click **Force Application**. Then either enter the user ID (and password) that started that application or a user ID with at least SYSMAINT authority on the monitored instance.

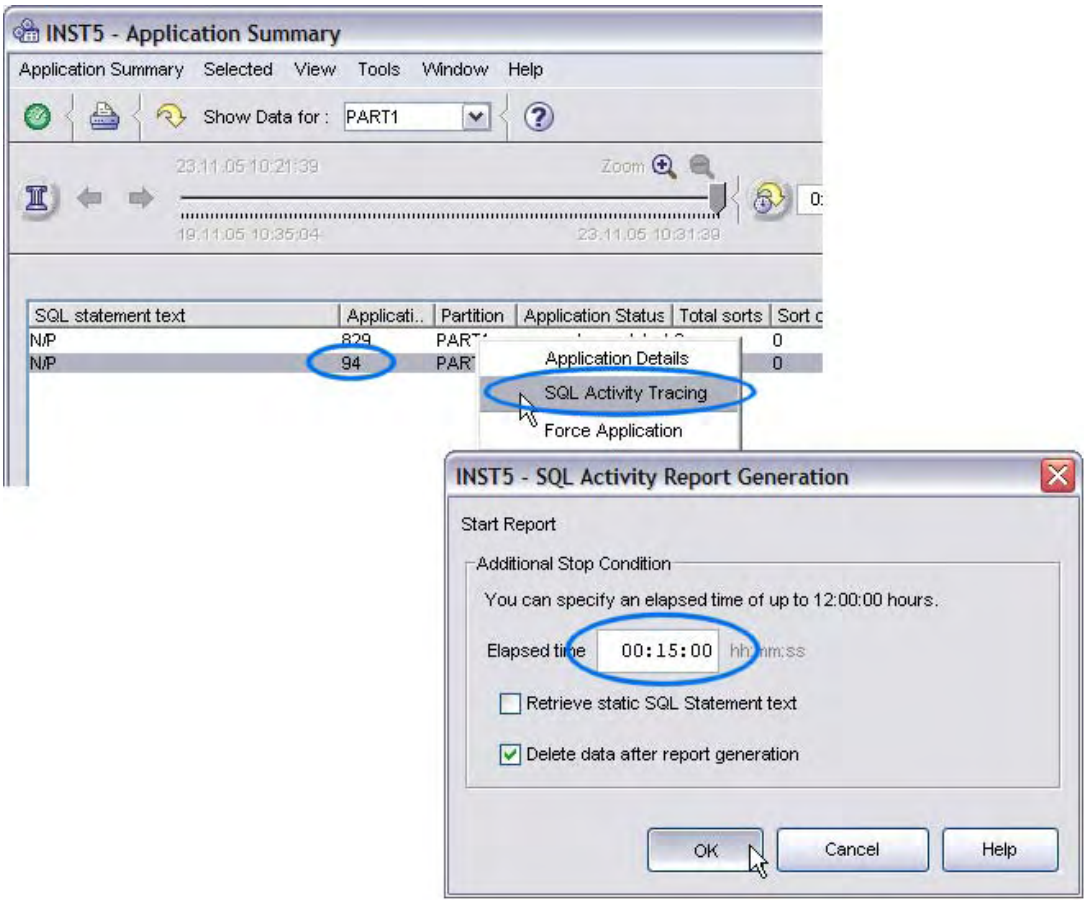


3.6 SQL tracing

Sometimes you need to perform a complete SQL trace of an application or, in some cases, entire Business Intelligence systems in order to better understand a performance problem. DB2 PE provides means to run ad-hoc SQL traces for a selected application, as well as to schedule and perform complete traces for an entire Business Intelligence system. DB2 PE offers a set of features to help you analyze the data that is collected in these SQL traces. These features include the ability to generate SQL Activity HTML reports and a set of predefined analyzing queries.

The simplest way to do ad-hoc tracing of a current application is to open the Application Summary window in partition or group view, right-click the application that you want to trace, and select **SQL Activity Tracing**. In the **SQL Activity Report Generation** dialog that opens specify how long the trace should run, if it should also capture static SQL (in case your application uses it), and if the data should be deleted from the PE database after the report has been created.

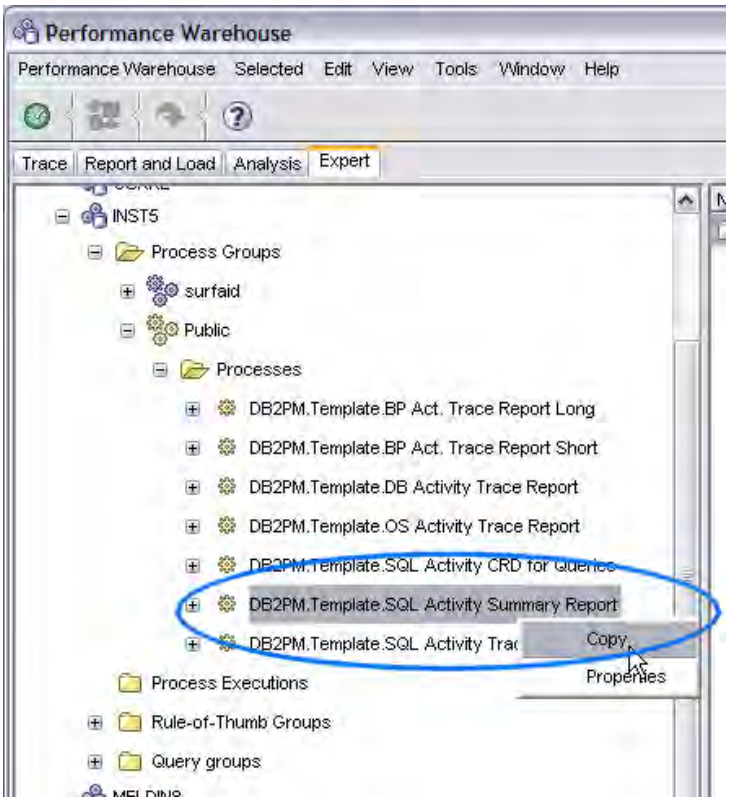
Now let the application perform the action that you want to trace and click **OK** in this dialog to start the trace.



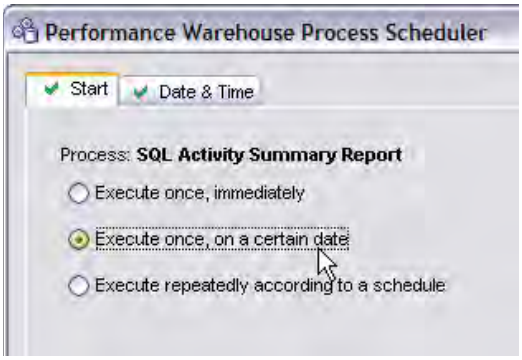
After the specified tracing time is reached and after some internal processing of the data, the results are displayed in an HTML report that outlines all of the SQL statements of the selected application and their execution metrics during the trace time.

If you do not have a specific application that you want to trace but rather you want to trace the entire SQL workload, you can do this by scheduling a process in the DB2 PE Performance Warehouse dialog. The DB2 PE Performance Warehouse dialog offers additional options for the data that you collect. For example, in addition to generating reports, you can also just collect the SQL activity trace and then run some analytic queries or rules of thumb against it.

To generate an SQL activity report in Performance Warehouse: 1. Double-click **Performance Warehouse - Expert** in the System Overview window, navigate to your instance, and expand the Process Groups folder. 2. If you have only the predefined Public group, create a custom group by selecting **Create** from the context menu and providing a name for the group. 3. Right-click **Public->Processes->DB2PM.Template SQL Activity Summary Report** and select **Copy** from the menu that opens. 4. Select the previously created new process group and click **OK**.



5. Navigate to the previously created process group, right-click **Processes->SQL Activity Summary Report** and select **Execute…** from the menu that opens. In the dialog that opens, select whether you want to start the trace immediately (similar to the option that was described above for a single application), to run it at a scheduled time, or to set up a schedule if you want to have the trace collected periodically.

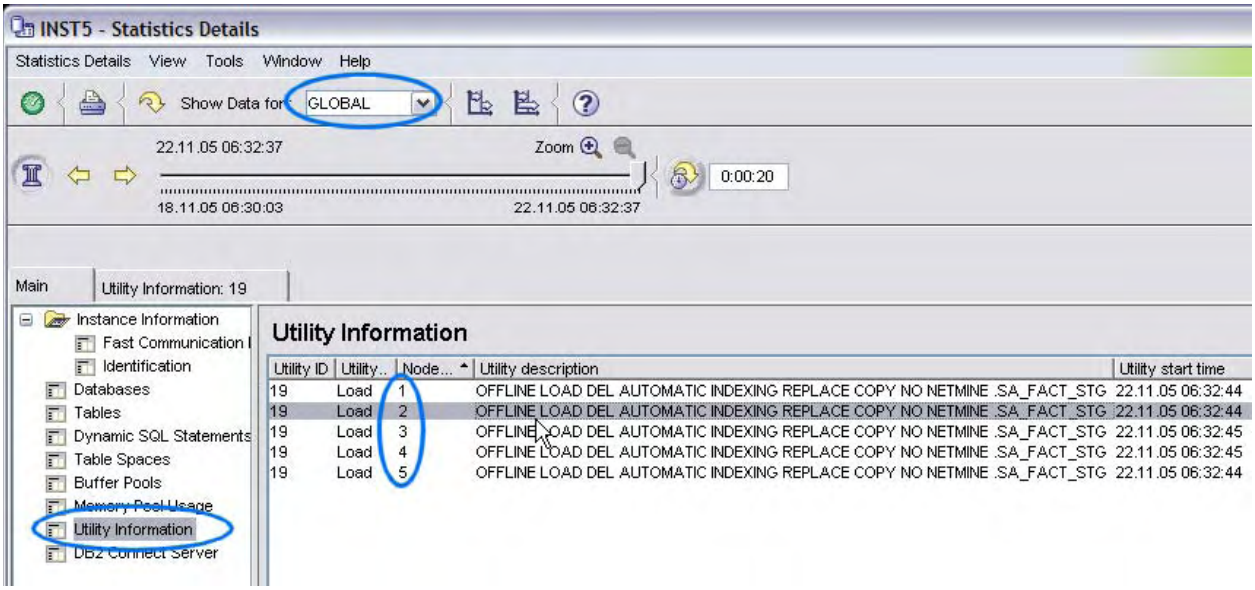


3.7 Monitoring and tuning load

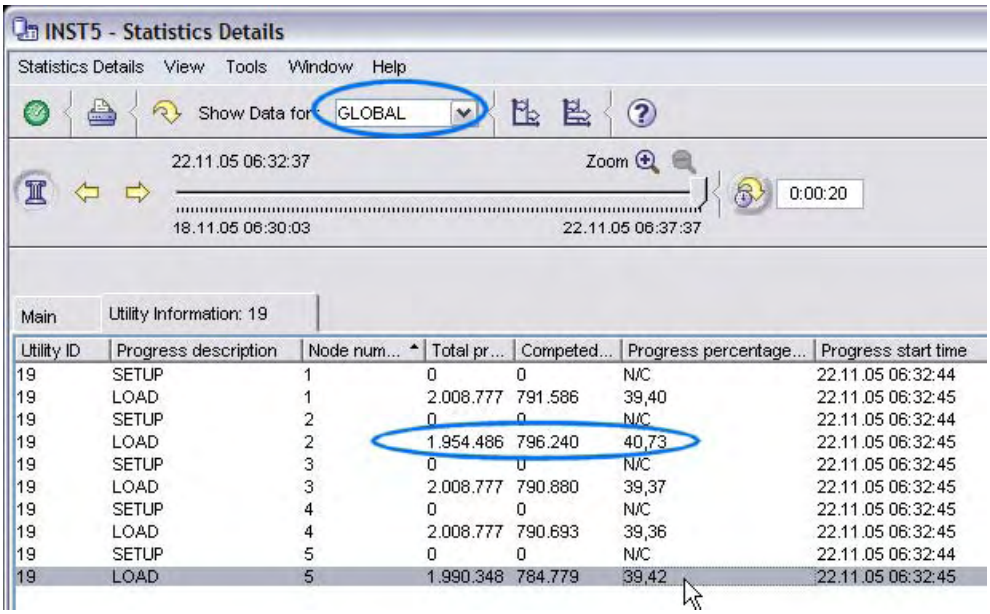
Not only do Business Intelligence systems need to handle analyzing the workload, they also need to accommodate the periodical incremental loading of new data into the tables. The load actually consists of three steps. First there is the actual load, import, or plain insert of new data to the tables. Then the MQTs need to be refreshed. Finally the statistics need to be updated. The data might first be loaded to staging tables, followed by an

INSERT…SELECT statement that moves the data from the staging table to the actually queried table.

DB2 PE provides dedicated means to monitor load processes. To do so: 1. From the Statistic Details window, select the Utility Information pane and display it in global view. If the load is currently running, you will see one entry per partition that the load affects, as shown in the following figure. If there were multiple load processes running at the same time, you can distinguish them by examining the **Utility ID** column.

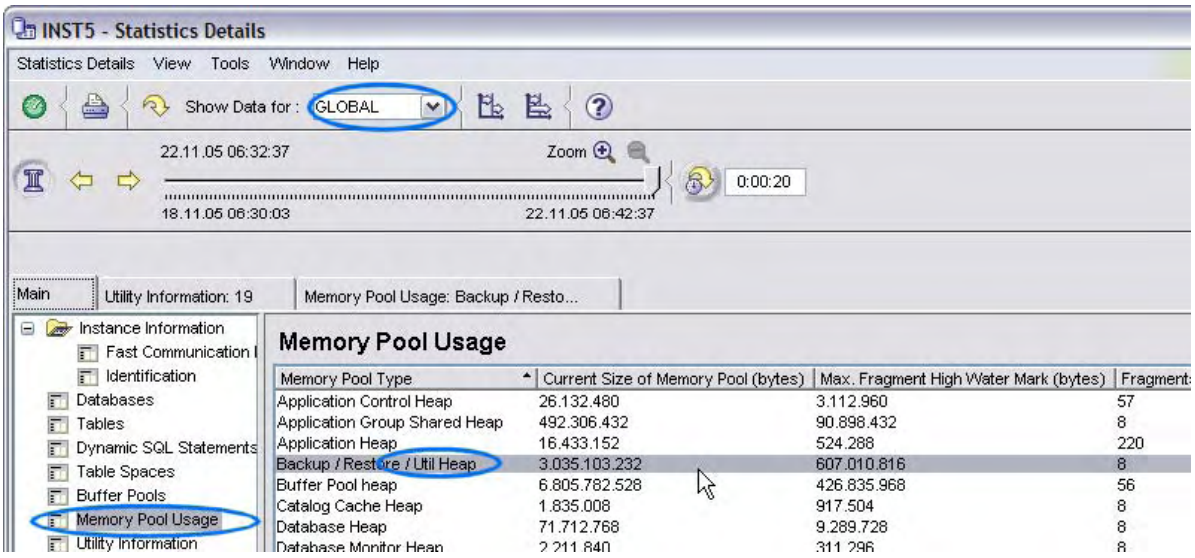


2. Double-click any of the entries to display details about the current status of the selected load process on each partition. As shown in the following figure, you can track the progress of the load steps on the single partitions by looking at the **Progress percentage complete** column for the different nodes. Use the manual or auto-refresh option to track the progress over time.

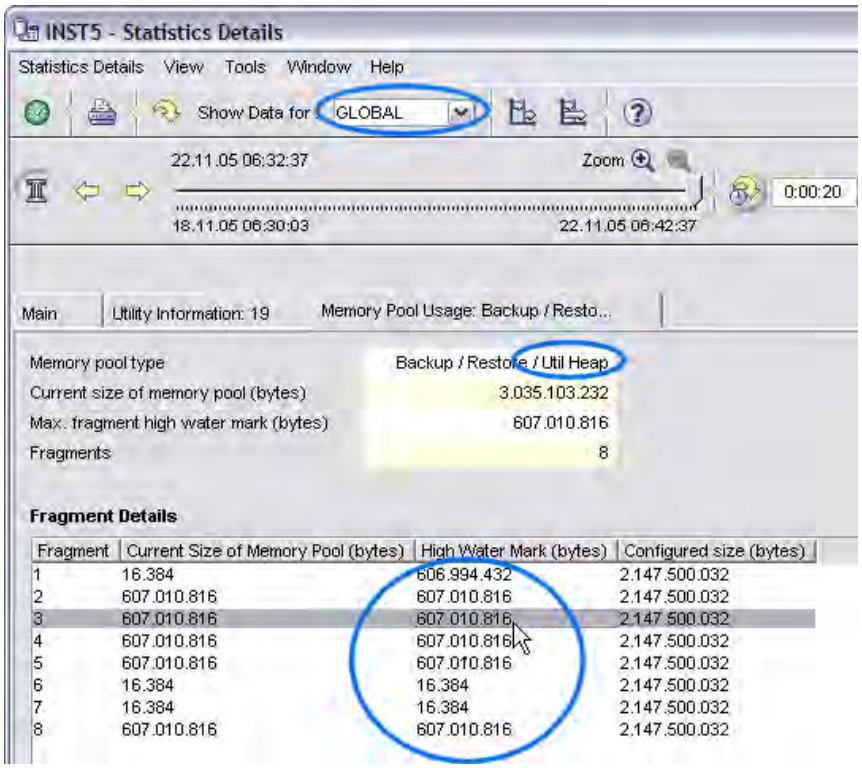


3.7.1 Utility heap

The load process uses the utility heap as memory buffer. When you load data into MDC tables this buffer is heavily employed to build the MDC blocks. You can monitor the utility heap in the Memory Pool Usage pane of the Statistic Details window. Look for the heap type Backup / Restore / Util Heap and double-click it.



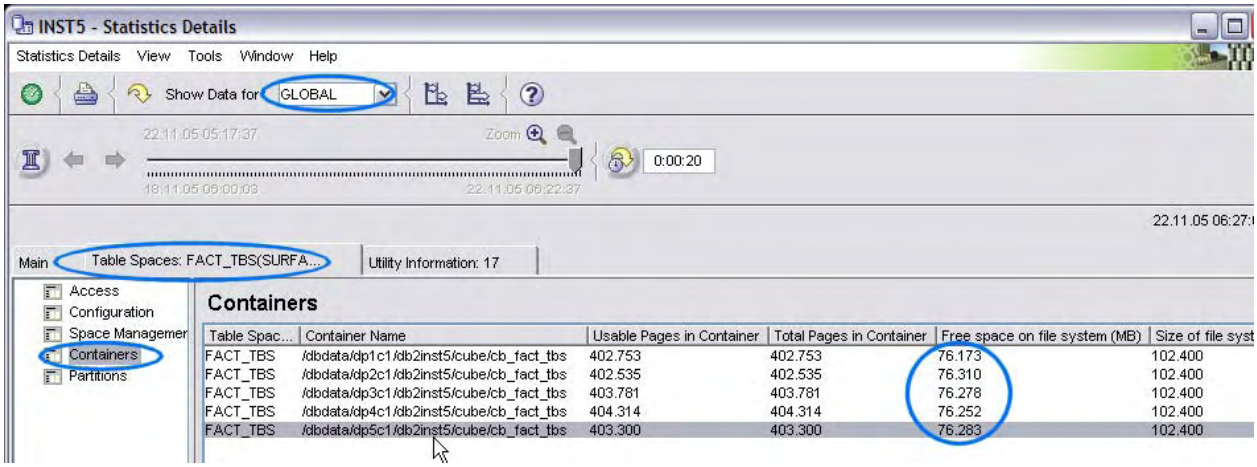
The Statistics Details window that opens contains a **High Water Mark (bytes)** column that displays the heap on each partition. Unfortunately, due to the way DB2 exposes these metrics, DB2 PE cannot display the actual node number here (the fragment number is not the node number). But if you want to see the high water mark for a dedicated partition, you can switch to partition view for that partition.



Check the high watermark against the values in the **Configured size (bytes)** column. If it is near or equal to that size you should consider increasing the utility heap size.

3.7.2 Monitoring tablespaces

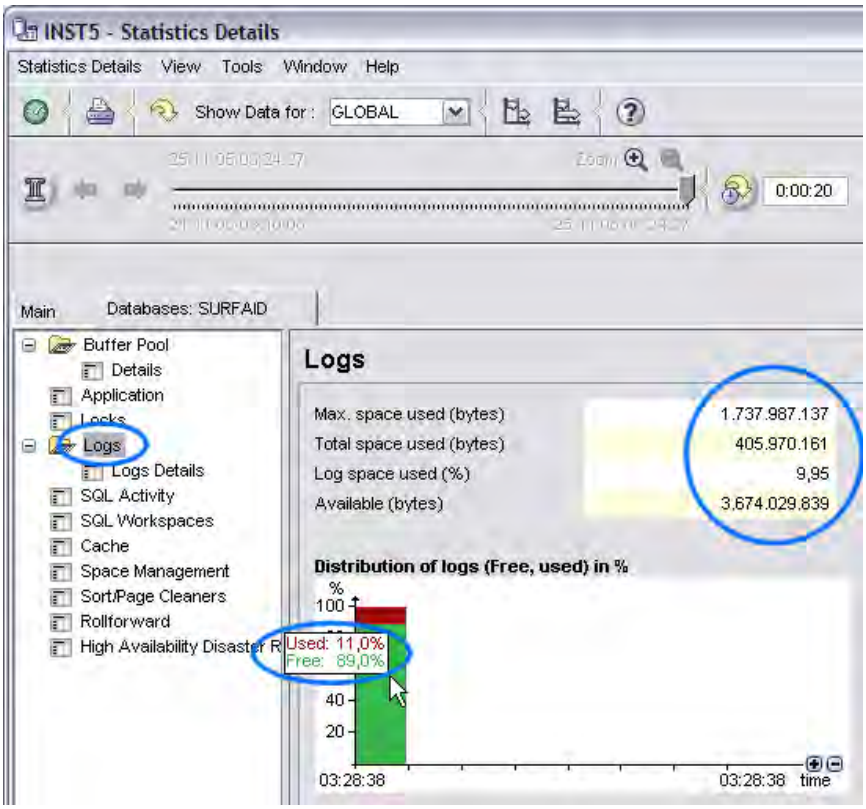
Before you start the load or as the load is running you should make sure that the tablespaces are not nearing their maximum capacity. You can track the size of the tablespaces in the Statistics Details window. To do so, select the Table Spaces category and double-click the tablespace. Then select the Containers category. Check the **Free space on file system (MB)** column to see how much space is left for the containers to grow.



3.7.3 Log space

If you are loading to staging tables first, you will do the INSERT...SELECT from the staging tables to the actually queried tables after the load is complete. This is a transactional process, which leads to transaction logging. There are two potential issues to be aware of:

The first is that your available transaction log space might run full. This can happen if you do the incremental INSERT in one or a few big transactions (for example, you have one INSERT…SELECT for the entire data of the staging table). You can check your transaction log space usage for your database in the Logs pane of the Statistics Details window.



The problem of log space running full can be resolved either by increasing it or by having more granular transactions (for example, by splitting up the INSERT…SELECT from the staging table to many statements with according range predicates).

The second potential problem is that that logging can cause a bottleneck due to I/O. You should watch the logging page activity (in the same pane as the log space usage) and the disk I/O of your disks where the transaction log files are located. See section 2.6.2 for more information.

Logging I/O problems can be countered by distributing the filesystem for transaction log files over more physical disks. You can also try to increase the size of the log buffer (LOGBUFSZ in the database configuration).

3.7.4 Concurrent load and analytics

Oftentimes, the incremental load must run concurrently with the ongoing analytic workload. In these cases you must also check for concurrency issues in which the queries are blocked by a load or vice versa. It is especially true for concurrent load and analytics that the previously mentioned staging tables are used for loading the data to the actual tables and then doing INSERT…SELECT operations into the actual tables.

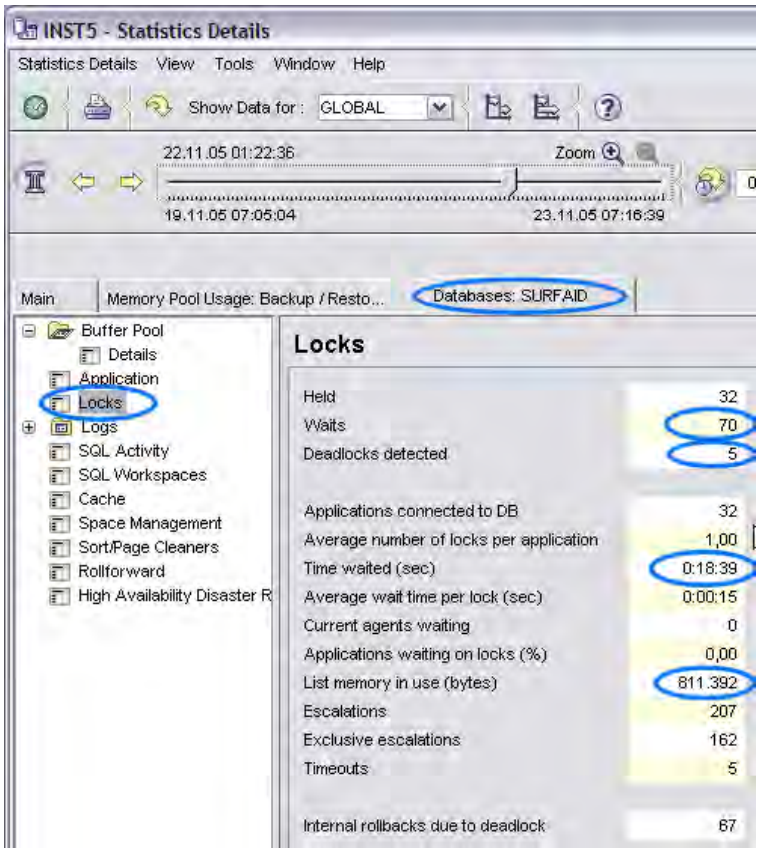
There are different strategies to avoid concurrency issues. For example, you can issue the queries with isolation level Uncommitted Read (UR). This strategy causes the queries to read rows that have been inserted but not yet committed, as if the commit already

happened. Another strategy is to set the DB2 registry variable DB2_SKIPINSERTED=YES, which causes the query to ignore the not-yet-committed rows. In both strategies, the queries are not blocked by the concurrent process that is inserting new data into the table.

But before thinking about this you should determine if you have concurrency problems and, if so, what the cause is.

3.7.5 Detecting concurrency issues

One issue potential issue is that some applications must wait for others for a longer time. You can check this in the Locks pane of the Statistics Details window. Look at the values in the **Waits** and the **Time waited (sec)** fields and see if the wait time is high with regards to a selected delta interval.

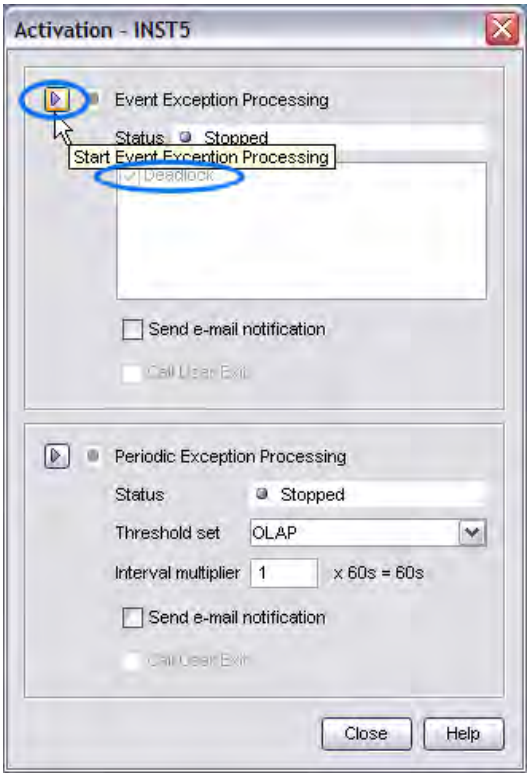


Another potential issue is that you are experiencing deadlocks (as determined by looking at the **Deadlocks detected** field).

A potential reason for both problems could be that you are experiencing lock escalations too (as shown in the **Escalations** field). If you are, you should tune your lock list size. Check the **List memory in use (bytes)** field. If it is near the configured lock list size, you should increase your lock list size (LOCKLIST). To determine the current lock list size, look at the **Maximum storage for lock list (4 KB)** field in the System Parameters – Database window, which is an be opened from the System Overview window. If not, you should increase the **Max. percent of lock list before escalation** (MAXLOCKS) parameter to a higher value because it limits how much of the lock list can be consumed by a single application in percent.

Another way to mitigate problems with lock list size is to avoid locking altogether by using Uncommitted Read (UR) isolation level.

It is not always possible to modify the lock list to solve concurrency issues. When it is not possible, you must analyze these issues in more detail. If you want to find out which applications are conflicting with each other, you can use the Applications in Lock Conflicts pane from the System Overview window. If you would rather know which tables that the lock waits are happening on, you can use the Locking Conflicts pane.



If you want to analyze deadlocks you should start the event exception processing for deadlocks in the Activation dialog (From the System Overview window, click **Tools-> Exceptions->Activation – Multiplatform**). As soon as a deadlock is detected, DB2 PE will raise an exception and display it in the System Overview window, where you can double-click it to drill down to the deadlock details.

3.8 Parameter marker check

BI systems typically execute analysis statements that take longer per definition. For that purpose the optimization of statements by DB2 is a crucial performance factor. Therefore, the DB2 optimizer should get as much information as possible for the statement compilation, including the values of all predicates. It is normally better to not use parameter markers in the statements but rather to specify the values as literals directly. This leads to DB2 statement compilation with literally each execution. But the time that is required by this approach is normally less than the time that would be spent executing of a less optimal plan that the optimizer has created due the missing knowledge of parameter marker values.

The following query can be used to check all statements that are executed by the system for the use of parameter marker values. Refer to section 3.5.2 for information about creating new queries. The result list shows one entry per such statement along with its average execution time. Use the PWH query facility to store and execute this statement.

```
SELECT sql.STMT_TEXT statement_using_par_markers,
           AVG(sql.ATIMEP_EXECUTIONS) avg_exec_time,
           MIN (INTERVAL_TO) first_captured_at,
           MAX(INTERVAL_TO) last_captured_at
      FROM DB2PM.DYNSQL sql
           WHERE MEMBER_ID = -2 AND LOCATE(STMT_TEXT, '?') <> 0
           GROUP BY sql.STMT_TEXT
```

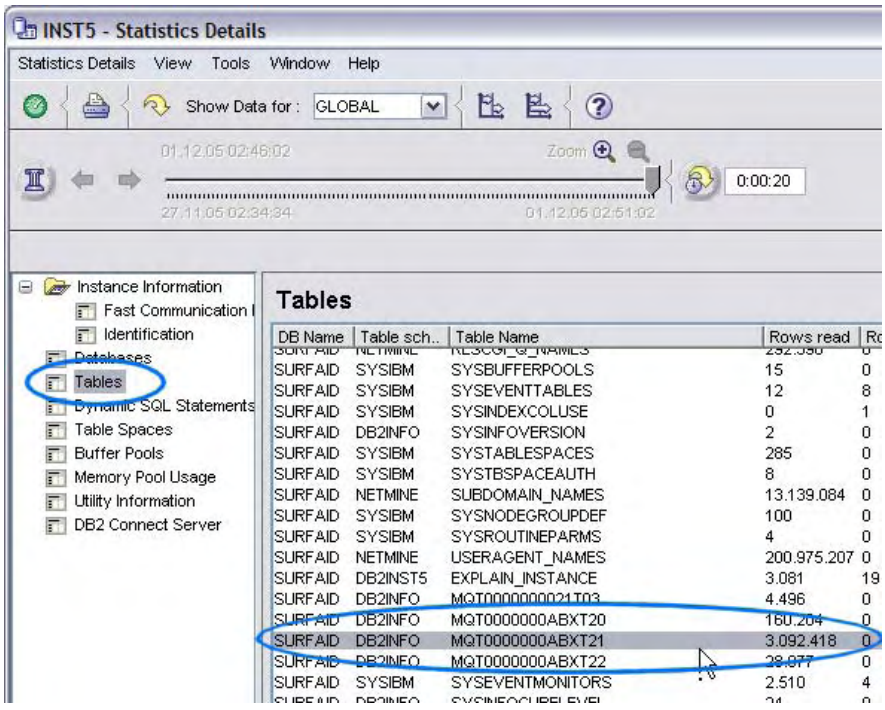
```
ORDER BY AVG(sql.ATIMEP_EXECUTIONS)
```

3.9 Verify MQT effectiveness

Materialized query tables (MQTs) are an elementary building block for Business Intelligence performance. However, it is not desirable to maintain MQTs that provide zero or even rather low performance benefit to the actual Business Intelligence workload that is executed. You can use DB2 PE to verify if and to what extend a MQT is used.

Use the explain capability (see section 3.5.4) to determine if certain queries are using MQTs or not. You can recognize that in the explain output by looking for the base tables that the query text actually refers to. If there is an eligible MQT for the query, the optimizer rewrites it to run against the MQT. In the explain output you will see MQT(s) that are being accessed instead of some or all base tables.

If you want to see to what extent MQTs are used in total, open the Statistics Details window and navigate to the Tables category. Select the check box at the bottom to make DB2 PE retrieve and display table access statistics. Then, locate the MQTs and compare their I/O statistics (in the **Rows read** column) with those of the base tables to understand to what extent MQTs are used over base tables.



3.10 FCM tuning

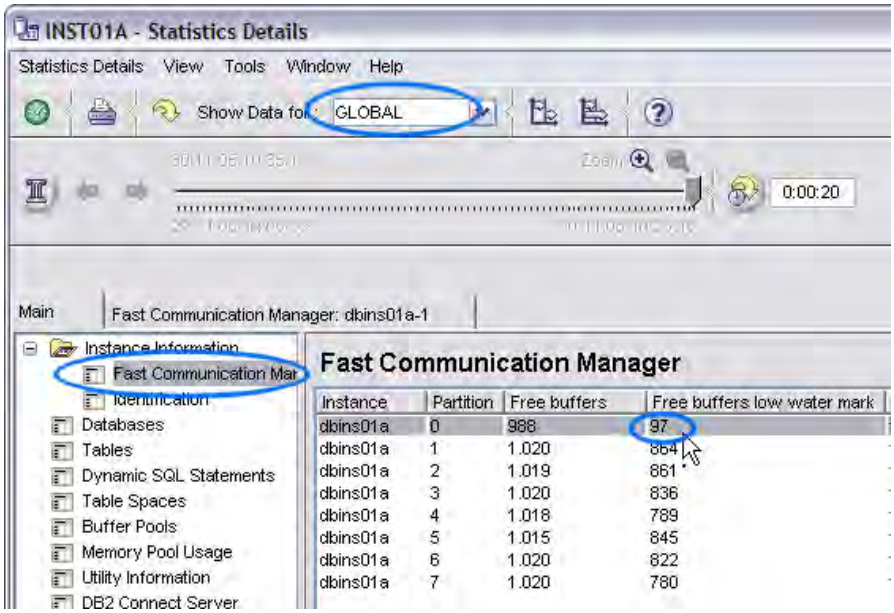
Between partitions on different physical machines, the underlying transport infrastructure for the table queues (see section 3.5.4) is the fast communication manager (FCM). The

main configuration parameter for it is the number of FCM buffers (num_fcm_buffers) on the database manager level. You can determine if this parameter is set appropriately by opening the Statistics Details window, navigating to **Instance Information -> Fast Communication Manager,** and switching to global view.

The key property to monitor is the **Free buffers low water mark**. It should not be too low (for example, less than 5% of num_fcm_buffers) or even 0 because this means that the number of FCM buffers are not sufficient to satisfy the actual inter-partition communication. You should increase the num_fcm_buffers parameter, or alternatively try to achieve more collocated joins (see section 3.5.4) to avoid some need for shipping data between partitions.

Alternately, a rather high number for the low water mark (for example, more than 50% of num_fcm_buffers) is also a cause for concern. There are several reasons for a high number for the low water mark, including:

- Too many FCM buffers are configured (your workload just doesn't need that much).
- All partitions are on the same machine (no FCM communication is needed).
- No workload is running.
- Network capacity is not sufficient to handle FCM communication.



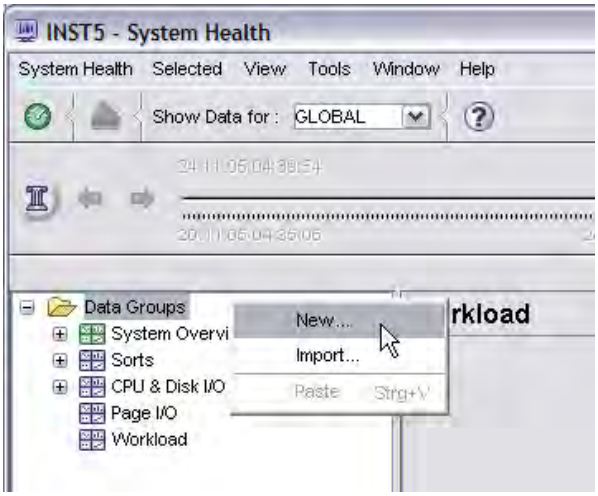
If you see skew on the low water mark on certain partitions, check these partitions for issues such as I/O bound execution.

3.11 Dashboard monitoring of key Business Intelligence performance indicators

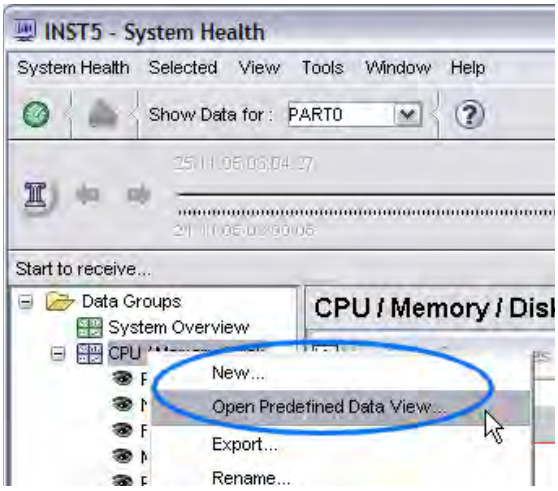
At several places throughout this document, you have probably noticed the capability of DB2 PE to visualize performance data in different types of diagrams based on the System Health window (see section 2.7). This window allows you to set up data visualization in a very flexible way, including the custom setup of dashboards for different Business Intelligence monitoring topics.

To set up your Business Intelligence performance dashboards: 1. Open the System Health window from the System Overview window. 2. Create new data groups (one for each dashboard) by right-clicking **Data Groups** and selecting **New…**. A suggested set of dashboards is:

- Sorts
- CPU / Memory / Disk
- Page I/O
- Workload
- FCM
- Incremental Load
- Storage



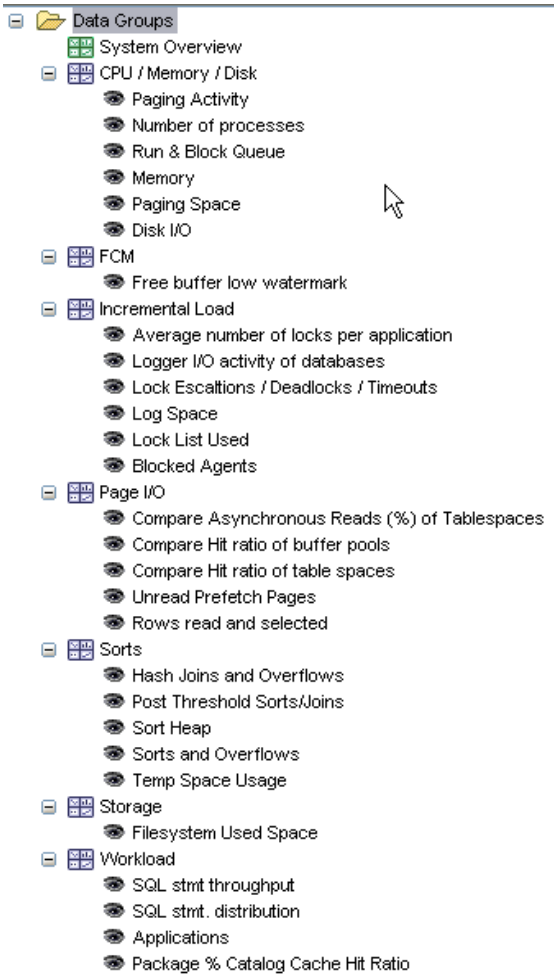
3. Now you can define the single thresholds in the data groups. DB2 PE provides a set of predefined data views, as well as the possibility to freely define custom ones (by clicking **New…**). For the custom data views you can choose between different data categories and then select one or multiple counters of that category. There is a rich set of options for displaying the diagram. For details refer to the DB2 PE documentation.

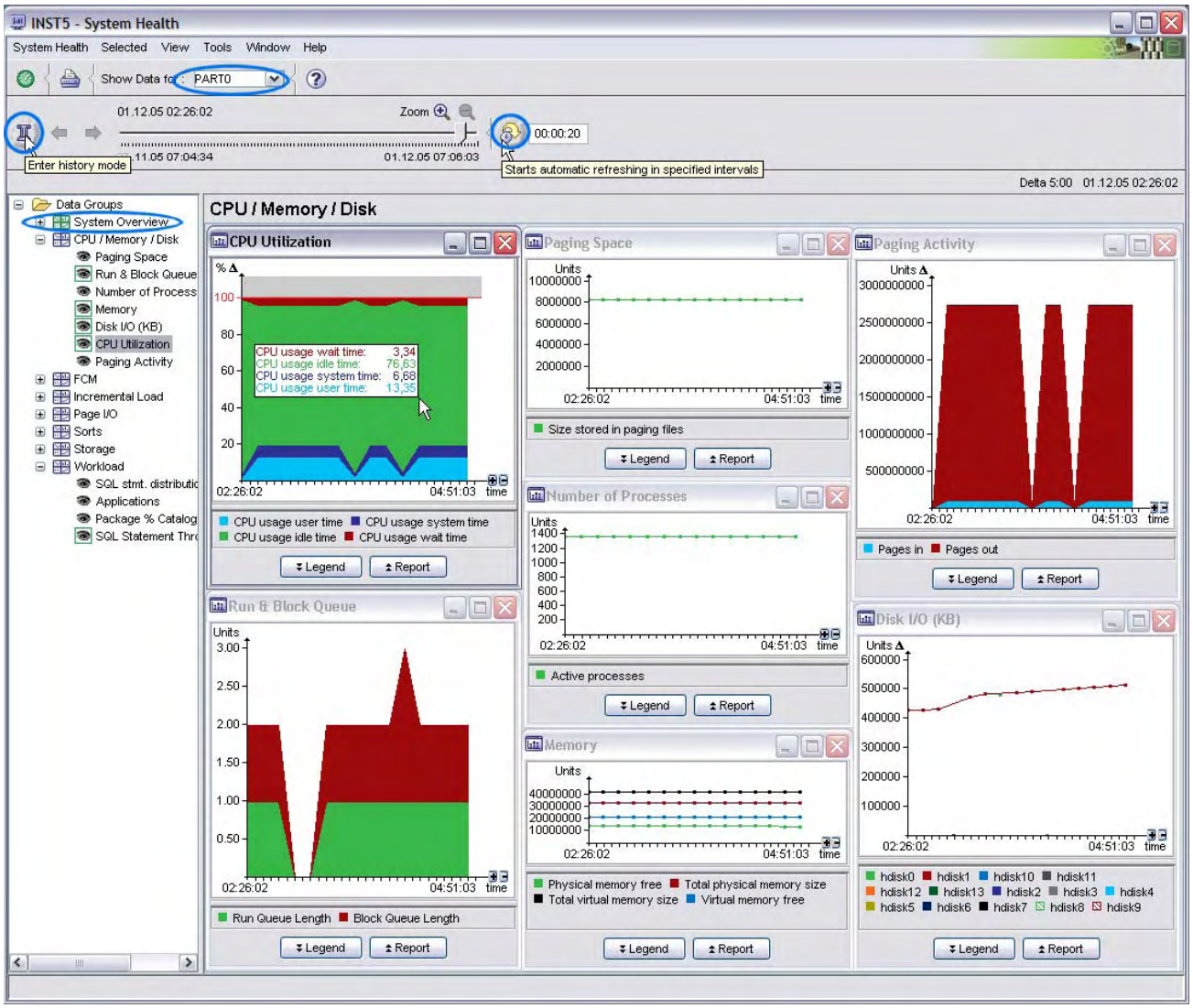


The System Health window supports exporting data view definitions per group to an XML file, which allows you to distribute your own dashboard definitions to other DB2 PE clients and to import them there. This document is also accompanied with a Business Intelligence-specific set of data view definitions that you can use to set up your Business Intelligence dashboards with minimal effort.

The figure on the right shows the data view definitions that are found in these supplementary XML files. Refer to "Appendix – Importing Business Intelligence data " for information about deploying these Business Intelligence data views to your DB2 PE system.

By using the System Health window, you have a very powerful means of intuitive access to your Business Intelligence performance because it supports online monitoring via the auto-refresh option as well as history browsing, as is the case with any other DB2 PE monitoring panel.





As demonstrated in section 3.4.1, you can also apply the partition group view to the System Health window, which allows you to see a visualized display of your monitoring data for each single partition side by side.

The System Health window is also the dialog that you use to set up the most important data views for being displayed in the System Overview window for your system. Just check **Display in System Overview** in the context menu of a data view. The icon visually indicates that it is now also available in the System Overview window. The **System Overview** node holds all of the data views that have been marked in this way.

4. Conclusion

Business Intelligence performance is heavily related to DB2 performance, both of which require tooling support to understand and resolve performance issues. DB2 Performance Expert for Multiplatforms provides a very rich set of DB2 performance monitoring capabilities that you can use to perform many Business Intelligence performance tuning tasks. An especially convenient feature is DB2 PE's DPF-monitoring capability, which makes DB2 PE an extremely useful component of Business Intelligence environments. DB2 PE's advanced monitoring techniques, such as history monitoring, exception processing, performance warehousing, and performance visualization, provide a great deal of flexibility and value.

Appendix – Importing Business Intelligence data views

DB2 PE provides a set of Business Intelligence data view definitions that you can use to help you to set up your Business Intelligence dashboard quickly and easily. These definitions are provided to you as a set of XML files in the installation directory of your client in **`samples\SystemHealth directory`**. To deploy them to your DB2 PE installation:

1. Open the System Health window, right-click **Data Groups**, and select **Import…**.
2. Navigate to the directory where you unpacked the zip file, select an XML file, and then click **Open**.
3. Depending on the XML file that you selected, additional dialogs might be displayed. Enter the appropriate information in these dialogs:

   For **`BI_DataViews_CPU_Memory_Disk.xml`**: Specify the disks that you want to monitor I/O for.

   For **`BI_DataViews_FCM.xml`**: Specify the partitions that you want to monitor FCM buffers for.

   For **`BI_DataViews_Sorts.xml`**, **`BI_DataViews_IncrementalLoad.xml`**, **`BI_DataViews_PageIO.xml`** and **`BI_DataViews_Workload.xml`**: If you have configured multiple databases of your monitored DB2 instance for monitoring in DB2 PE, select the database that you want to monitor sorting, data loading, page I/O, and general workload for.

   For **`BI_DataViews_Sorts.xml`**: Specify the disks where your temporary tablespaces are placed.

   For **`BI_DataViews_Storage.xml`**: Specify the file systems for which you want to monitor the usage.