Using AS/400 Database Monitor and Visual Explain
To Identify and Tune SQL Queries

Understand the basics of how to identify and tune the performance of SQL

Learn about the new graphical Operations Navigator Visual Explain tool

See how Database Monitor can assist in SQL performance

Redpaper

ibm.com/redbooks
Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, “Special notices” on page 39.

First Edition (January 2001)

This document was created or updated on January 9, 2001.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2000, 2001. All rights reserved.
Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
## Contents

**Preface** ................................................................. v  
The team that wrote this redpaper ......................................... v  
Comments welcome .......................................................... v  

**What is the Database Monitor for AS/400 tool?** ............................. 1  
  Collecting DB Monitor data ................................................. 1  
  Importing an SQL Performance Monitor .................................. 3  
  Special consideration on collecting DB Monitor data ................. 3  
  DB Monitor record types .................................................. 3  
  Global DB Monitor data fields ........................................... 4  
  How the query data is organized in DB Monitor ....................... 6  
  Linking query instances in Monitor data ................................ 7  

Reported DB Monitor record types ........................................ 8  
  1000 Record - SQL statement summary .................................. 9  
  3000 Record - Arrival sequence (table scan) .......................... 11  
  3001 Record - Using existing index .................................... 12  
  3002 Record - Index create ............................................ 13  
  3003 Record - Query sort ................................................ 15  
  3004 Record - Temporary file .......................................... 16  
  3006 Record - Access plan rebuild ..................................... 16  
  3007 Record - Summary optimization ................................... 17  
  3010 Record - Host variable and ODP implementation ............... 17  
  3021 Record - Bitmap created .......................................... 18  
  3022 Record - Bitmap merge ............................................ 18  
  3023 Record - Temporal hash table created ........................... 18  
  3028 Record - Grouping .................................................. 19  

Analyzing SQL statements involving JOINs ................................ 19  
  3014 Record - Generic query information ............................. 20  

Other DB Monitor records .................................................. 21  

**Operations Navigator Visual Explain** .................................. 23  
  Using Visual Explain .................................................... 23  
  Using Visual Explain interactively ..................................... 23  
  Using Visual Explain over DB Monitor collected data ................ 24  
  Visual Explain results ................................................... 25  

**Database performance analysis** ........................................ 27  
Database performance problems that are not easily avoidable .......... 27  
Problems that are generally avoidable ................................... 28  
Creating indexes ............................................................. 29  
For faster analysis ........................................................... 29  
  Altering insert with subselect data in a Monitor file ............... 30  

DB Monitor query examples 1 through 13 .................................. 30  
  Example 1: Identify the specific job(s) to analyze .................... 30  
  Example 2: Which type of SQL operations account for the most run time .......... 30  
  Example 3: Which SQL operations account for the most run time (list by text). ...... 31  
  Example 4: Which queries account for the most run time (list by QQUCNT) .......... 31  
  Example 5: Which queries account for the most run time ............... 31  
  Example 6: How do I determine how an individual query was run ............. 31  
  Example 7: Which queries are significantly affected by full opens ............. 33  
  Example 8: List all queries (sorted by text) that are doing full opens .......... 33
Preface

The purpose of this redpaper is to help you to understand the basics of identifying and tuning the performance of Structured Query Language (SQL) statements using DB2 Universal Database for AS/400 (DB2 UDB for AS/400). DB2 UDB for AS/400 provides a comprehensive set of tools for helping technical analysts tune SQL queries.

Database Monitor for AS/400 (DB Monitor) is part of the set of tools OS/400® provides for assisting in SQL performance analysis since Version 3 Release 6. It is used to analyze database performance problems after SQL requests have been run.

Operations Navigator Visual Explain is a new tool that shows the internal steps that DB2 UDB for AS/400 performs in response to an SQL query. Because of its graphical nature, it gives a new perspective on SQL query performance analysis.

This redpaper presents a series of tips and techniques based on DB Monitor and Visual Explain that may be used to demonstrate how to get the most out of both DB2 UDB for AS/400 and query optimizer when using SQL. You should be aware that usage of DB Monitor is technically demanding. IBM offers Learning Services courses on database tools and analysis to help you.

The team that wrote this redpaper

This redpaper was originally written by a team of specialists from the IBM Rochester Lab:

Greg Leibfried
Richard Odell
Rick Peterson
Dale Weber

This redpaper was recently updated by the International Technical Support Organization Rochester Center:

Hernando Bedoya
ITSO Rochester

Daniel Lema
IBM Colombia

Comments welcome

Your comments are important to us!

We want our redpapers to be as helpful as possible. Please send us your comments about this redpaper or other Redbooks in one of the following ways:

- Use the online evaluation form found at ibm.com/redbooks
- Send your comments in an Internet note to redbook@us.ibm.com
Using AS/400 Database Monitor and Visual Explain To Identify and Tune SQL Queries
What is the Database Monitor for AS/400 tool?

The Database Monitor for AS/400 tool (DB Monitor) is an AS/400-based tool used to gather performance related statistics for SQL queries run on AS/400. Data collected by the DB Monitor is stored in an AS/400 database file where it can be queried to help identify and tune performance problem areas. Results from the DB Monitor can be useful for batch jobs and online transactions. Also, the results can be used to look at SQL queries from a global system level for a specific job or a specific query. DB Monitor data is most useful if the user has a basic knowledge of AS/400 query optimization techniques.

Collecting DB Monitor data

Use Operations Navigator Database functions to start, pause, analyze, and stop DB Monitor. Alternatively you can use the STRDBMON command or STRPFRMON with STRDBMON(*YES) to start the DB Monitor. The DB Monitor will collect information on previously started jobs or new jobs started after the monitor collection has begun. Because of the volume of data collected, try to gather data for a specific job only. This makes analysis easier and keeps the DB Monitor file smaller. If this is not possible, collect data on all of the jobs, and use queries to select the specific jobs of interest.

It should also be noted that when the DB Monitor is gathering data, a significant amount of CPU utilization (20 to 30 percent) and disk usage may be temporarily required.

Starting DB Monitor

Using Operations Navigator database functions, you have two options for DB Monitor: summary and detailed. The summary provides basic information over table scan, data sorts, host variable use, optimizer time-outs, access paths considered, indexes used, indexes created, subselect processing, and temporary file use. With the detailed DB Monitor, you have detailed information about the query and its access path, which we describe in this redpaper.

Starting a summary SQL Performance Monitor starts a memory resident database monitor which is a lighter workload for the system. A detailed SQL Performance Monitor causes a heavier workload, equivalent to that of the STRDBMON... TYPE(*DETAILED). There is no equivalent to STRDBMON... TYPE(*SUMMARY) in Operations Navigator.

To start DB Monitor with Operations Navigator, right-click SQL Performance Monitors and, depending upon your needs, select New->Summary or New->Detailed (see Figure 1 on page 2). This takes you to the New SQL Performance Monitor dialog box. On the General tab, a name and library is requested for storing the collected data. In the Monitored Jobs tab you can select the jobs that you are interested in monitoring. Selecting all jobs may cause a significant overhead on the system, particularly if it is a detailed data collection.
Using AS/400 Database Monitor and Visual Explain To Identify and Tune SQL Queries

Pausing, Continuing, Ending, and Deleting a DB Monitor
Right-click a specific SQL Performance Monitor on Operations Navigator to obtain a menu like the one shown in Figure 2. You can pause or end an SQL Performance Monitor. When you end an SQL Performance Monitor, you cannot start it again. You can continue with an SQL Performance Monitor that was previously paused.

SQL Performance Monitor data table
Selecting Properties from the menu in Figure 2 displays a properties window with three tabs. Under Saved Data tab, you can find the library and file name for the DB Monitor captured data. You will need this to create your own queries over DB Monitor tables.
Importing an SQL Performance Monitor

You can import data collected from STRDBMON into SQL Performance Monitor using Operations Navigator. You can also import data collected on other systems. To do this, right-click SQL Performance Monitors icon (in Operations Navigator), select the Import option. Then the Import SQL Performance Monitor Files dialog window appears. Fill in the required fields: Monitor Name, File and Library where the monitor resides.

You can take a summary DB Monitor from the STRDBMON interactive command and import it into an Operations Navigator Detailed SQL Performance Monitor. It will import all of the necessary information, even for Visual Explain, except for row access information (useful only for analyzing non-SQL data access).

Special consideration on collecting DB Monitor data

Optimization records/data will not appear for queries that are already in reusable ODP mode when the monitor is started. To ensure the capture of this data for a batch job, start the DB Monitor before the job starts and collect it over the entire length of the job (or as much as needed). For online transactions, start the monitor before connecting to the QZDASOINIT server job to ensure optimization data is collected. The STRDBMON JOB(*ALL) command is needed in both of these cases.

If optimization data was not collected for a given query, run the query using Operations Navigator Run SQL Script or Start SQL (STRSSQL) command or other tools and collect debug messages or DB Monitor data. Obtain the query text from the DB Monitor data or from the step mode trace. If the DB Monitor is ended while the query is in progress, optimization data is collected but other data for that query (SQL text, etc.) is not.

DB Monitor record types

DB Monitor collects different data and stores records in a single table in the order of occurrence. Each record contained in the DB Monitor table contains a record type field. The DB Monitor uses the QQRID field to describe the type of information gathered in the particular record. The DB Monitor record types most often used for performance analysis are listed here:

Record types most often used (QQRID value)

<table>
<thead>
<tr>
<th>QQRID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>SQL summary record</td>
</tr>
<tr>
<td>3000</td>
<td>Arrival sequence</td>
</tr>
<tr>
<td>3001</td>
<td>Using existing index</td>
</tr>
<tr>
<td>3002</td>
<td>Index created</td>
</tr>
<tr>
<td>3003</td>
<td>Query sort</td>
</tr>
<tr>
<td>3004</td>
<td>Temporary results file</td>
</tr>
<tr>
<td>3006</td>
<td>Access plan rebuilt</td>
</tr>
<tr>
<td>3007</td>
<td>Summary optimization data (also known as index optimization data)</td>
</tr>
<tr>
<td>3010</td>
<td>Host variable and ODP implementation</td>
</tr>
<tr>
<td>3021</td>
<td>Bitmap created</td>
</tr>
<tr>
<td>3022</td>
<td>Bitmap merge</td>
</tr>
<tr>
<td>3023</td>
<td>Temporal hash table created</td>
</tr>
<tr>
<td>3026</td>
<td>Union merge</td>
</tr>
<tr>
<td>3027</td>
<td>Subquery merge</td>
</tr>
</tbody>
</table>
Using AS/400 Database Monitor and Visual Explain To Identify and Tune SQL Queries

DB Monitor record types: Other record types

- 3005 Table locked
- 3008 Subquery processing
- 3014 Generic query information
- 3018 STRDBMON and ENDDBMON data
- 3019 Records retrieved detail (only with *DETAIL)
- 3025 DISTINCT processing
- 3030 Query step processing

Record types 3000 to 3008, 3014, and 3021 to 3029 occur during a full open and can be referred to as optimization records. Optimization records are much like debug messages. These records are necessary to determine the access plan for any given query in the DB Monitor data.

Global DB Monitor data fields

The data fields that are common to all record types are:

- **QQJOB** Job name
- **QQUSER** Job user name
- **QQJNUM** Job number: The job number is very useful when multiple jobs are collected in one DB Monitor file.
- **QQJFLD** Join field: Contains information that uniquely identifies a job and includes job name, job user name, and job number.
- **QQTIME** Time that the record was created: The time record can be useful when trying to find out what queries were running in a given time period.
- **QQUCNT** Unique number given for each query within a job. QQUCNT links together all DB monitor records associated with all instances of a unique query within a job.

  The QQUCNT value assigned at full open time stays constant for all subsequent instances of that query. Non-ODP SQL operations (prepare, describe, commit) have QQUCNT = 0 and thus cannot be linked to a query. But the QQ1000 field in the prepare or describe 1000 record will contain the prepared SQL text.

  QQUCNT is not set for optimization records.

- **QQI5** Refresh counter: This record specifies the instance number for a unique query. It is used in conjunction with the QQUCNT value to look at a specific instance of a query and is only valid on 3010 and 1000 SQL summary records.

  Non-ODP 1000 records (commit, prepare, etc.) have QQI5 = 0.

- **QQRID** Record Identifier: Identifies the type of record.
Figure 3 shows an example of the fields in the DB Monitor table.

The combination of QQJFLD and QQUCNT provides a powerful tool to join different optimization records for the same SQL statement. We exploit this extensively in this redpaper.

Other fields common to many records are those that identify tables or indexes used:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QQTLN</td>
<td>Library of the table queried.</td>
</tr>
<tr>
<td>QQTFN</td>
<td>Name of the table queried.</td>
</tr>
<tr>
<td>QQPTLN</td>
<td>Base library.</td>
</tr>
<tr>
<td>QQPTFN</td>
<td>Base table.</td>
</tr>
<tr>
<td>QQILNM</td>
<td>Library of the index used.</td>
</tr>
<tr>
<td>QQIFNM</td>
<td>Name of the index used: *N when it is a temporary index.</td>
</tr>
</tbody>
</table>

Beginning with V4R5, new fields were added to include long names for the resources used:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QVQTBL</td>
<td>Queried table long name.</td>
</tr>
<tr>
<td>QVQLIB</td>
<td>Queried library long name.</td>
</tr>
<tr>
<td>QVPTBL</td>
<td>Base table long name.</td>
</tr>
</tbody>
</table>

Tip

If you are going to use the Operations Navigator Run SQL Script facility, change the ODBC Setup to force translation for CCSID 65535, because the QQJFLD has been defined as FOR BIT DATA. Otherwise this field will be shown in hexadecimal. Reset to default to use Visual Explain; otherwise it will fail.
Using AS/400 Database Monitor and Visual Explain To Identify and Tune SQL Queries

**QVPLIB** Base table library long name.

**QVINAM** Index used long name.

**QVILIB** Index library long name.

**How the query data is organized in DB Monitor**

The first occurrence of a unique query within the job always results in full open. A “unique” query is one that requires a new ODP. SQL has determined that there is no existing ODP that can be used.

The presence of optimization records indicates a full open for an open, select into, update, delete, or insert operation. Optimization records are immediately followed by SQL summary records (QQRID=1000) for that operation.

Subsequent occurrences of this query within same job either run in reusable ODP (Figure 4) or non-reusable ODP (Figure 5) mode. Non-reusable mode indicated by presence of optimization records each time a particular query is run (full open). Reusable ODP mode is indicated by 3010 and 1000 records each time the given query is run (no optimization records or full open).

```sql
SELECT qqrid AS "QQRID Record Type", qqucnt AS "QQUcnt Unique Counter", qqi5 AS "QQI5 Refresh Counter", qqjfld AS "QQJFLD Join Field", qqc21 AS "QQC21", qq1000 AS "QQ1000 Text" FROM team11.reusablodp ORDER BY qqtime OPTIMIZE FOR ALL ROWS;
```

**Figure 4. Example of a reusable ODP SQL statement**
Figure 5. Example of a non-reusable ODP SQL statement

**Linking query instances in Monitor data**

The data in the DB Monitor file is arranged chronologically. This can make it difficult to find all instances of a unique query. Use the QQJFLD, QQUCNT, and QQI5 fields to view specific query instances. Be aware of the following situations:

- **QQUCNT** is assigned during a full open and will be constant for all subsequent instances of a query.
- Non-ODP SQL operations (prepare, describe, commit) have QQUCNT = 0 and, therefore, cannot be linked to a query. If this is the case, you can use QQ1000 field that in the prepare or describe operation contains the prepared SQL text.
- Non-ODP 1000 records (commit, prepare, etc.) have QQI5 = 0.
A full open occurs when the SQL operation is either an update, insert, delete, or open and the QQI5 record is 0.

See Figure 4 and Figure 5 on page 7 for examples of QQJFLD/QQUCNT/QQI5 in reusable and non reusable ODPs.

**Reported DB Monitor record types**

Each of the DB Monitor record types provide a specific type of information gathered by the DB Monitor as it collects data during SQL performance runs.

You can build your own queries or use the predefined queries provided by Operations Navigator. To use the predefined Operations Navigator queries, select **Database -> SQL Performance Monitor** from the left window. A list of Performance Monitor data collections appears in the right window. Right-click a collection and select **analyze results** (see Figure 2). You will be able to select, modify, and execute summary, detailed and extended detailed results.

The summary reports give you information about the entire data collection and will help you find exceptions. For example, you may list jobs ordered by descending total run time in the Job Summary report.

You can also alter the predefined query by clicking the **Modify Selected Queries** button (see Figure 6).

**Tip**

Use predefined Summary Results over a summary DB Monitor to identify the most resource hungry and time-consuming jobs as your starting point for analysis.
Once you have detected jobs of interest, collect detailed DB Monitor data for those jobs. Use detailed and extended detailed predefined reports for further analysis.

The predefined reports are SQL queries built on the DB Monitor table, usually joining different types of records. The basic record used for each report is shown in Figure 7.

![Figure 7. Detailed, Extended Detailed Results and their main source of information](image)

1000 Record - SQL statement summary

The 1000 record is the basic record type for any SQL query analysis. One record exists for each SQL operation (open, update, close, commit, etc.).

The most commonly used fields

**QQ1000** Prepared text of SQL statement

Literals in the original SQL text may be replaced by parameter markers in prepared text if SQL was able to convert them during prepare (desired). For the original SQL text, use literal values from the matching 3010 record in the place of parameter markers or obtain the text from the step mode file using the QQRSTIM timestamp from this record.

**QQC21** Type of SQL operation (OP, FE, CL, UP, IN, DL,...): ‘MT’ in this field indicates a continuation record for SQL statements that exceed 1000 characters. FE (fetch) records are summary records, one per open.

**QQI2** Number of rows updated, inserted, or deleted

**QQI3** Number of rows fetched (only on FE records): The QQI3 field shows the actual number of rows fetched, not the number of fetch attempts.

**QQI5** Refresh counter (use in conjunction with QQUCNT). A full open occurs when QQI5=0 and QQC21 is UP, DL, IN or OP.

**QQI4** Elapsed time for this operation in milliseconds.
QQI6  Elapsed time for this operation in microseconds.
QQSTIM  Timestamp for start of SQL operation (microsecond granularity).
QQETIM  Timestamp for end of SQL operation (microsecond granularity).

QQETIM-QQSTIM shows the elapsed time for an operation in seconds down to microsecond granularity. For FE records, use QQI6 or QQI4; QQETIM-QQSTIM is not valid.

The other commonly used fields
QQC22  Access plan rebuild reason code.
QQC103 and
QQC104  Package name and package library name.
QQC181  Cursor name.
QQC16  Data conversion reason code.
QVC11  ALWCPYDTA setting (Yes, No, Optimize).
QVC14  ODP implementation (R: Reusable N:Nonreusable).
QVC41  Commitment control level.

In Figure 8, you can find an example of some of the fields commonly used on type 1000 records.
Figure 8. Example of type 1000 records with a user-defined query

3000 Record - Arrival sequence (table scan)

The 3000 record points out queries in which the entire table is scanned without using an index. A table scan is generally acceptable in cases where a large portion of the file will be selected or the selected file contains a very small number of records. Otherwise, using an index usually provides better performance.

An insert with a subselect will have a 3000 record for the file being inserted into, but this is not a performance problem on its own. This does not indicate that the ODP is non-reusable. The record data may contain useful index advisor data.

The most commonly used fields
QQTNLN, QQTFN, QQPTLN, QQPTFN, QVQTBL, QVQLIB, QVPTBL, and QVPLIB describe the scanned table as documented in “Global DB Monitor data fields” on page 4.

QQTOTR Number of rows in table. Use QQTOTR to determine if the table scan was done for a significant number of rows.

QQRCOD Reason code; why the arrival sequence was chosen.

QQIDX A Index advised (Y or N). If the QQIDX field is ‘N’, QQI2 and QQIDX will not contain data.
QQI2 Number of primary (key positioning) keys in the QQIDXD field.

QQIDXD Suggested keys for index (selection only).

The QQIDXD field can contain both primary and secondary keys. Starting from the left, QQI2 tells how many keys are considered primary. Other keys are considered less selective (secondary). It is important to include the most selective keys since the database optimizer is estimating. Even if an index is not advised, it is still best to determine if a good index can be created for this table by looking at the selection in the SQL text. This is especially important if the cumulative time for this query is significant. This can be measured by the total of QQI6 or QQI4 values for the 1000 FE records for this query.

**3001 Record - Using existing index**

The 3001 record shows the index that will be used to access the file and why it was chosen. If the index was chosen for join, additional information is given to help determine how the file “fits” in the join. The order of the 3001 records indicates the join order chosen by the optimizer.

**The most commonly used fields**

QQTLN, QQTFN, QQPTLN, QQPTFN, QQILNM, QQIFNM, QVQTBL, QVQLIB, QVPTBL, QVPLIB, QVINAM, and QVILIB describe the table and index used as documented in “Global DB Monitor data fields” on page 4. QQILNM and QQIFNM contain *N if the index was a temporary index.

**QQRCOD** Reason the index was selected

I1 Selection only
I2 Ordering or grouping
I3 Selection and ordering/grouping
I4 Nested loop join
I5 Record selection using bitmap

Bitmap selection has a DB Monitor record sequence of 3007, 3000, and 3001.

**QQTOTR** Number of rows in the table.

**QQIDXA** Index advised (Y or N).

**QQI2** Number of primary (key positioning) keys in QQIDXD field.

**QQIDXD** Recommended primary and secondary keys for index on this table.

**Important**

The keys listed can *possibly* be better than those in the selected index, but not always. Generally, the selected index will be fairly good, or the database optimizer has chosen to build a temporary index that fits well. These keys are for selection only and the user also needs to consider join/order by/group by clause criteria.

Figure 12 on page 20 shows some examples of 3001 records.
3002 Record - Index create

The 3002 record shows instances in which the database optimizer decided that existing indexes are too costly or do not have the right key order for join, group by, or order by clauses. Refer to the 3007 record for this file to see why existing indexes were not selected. The newly created indexes are temporary and are not usable by other ODPs.

A temporary index build does not mean that the ODP is non-reusable. The database optimizer tries to reuse the temporary index for each execution of the specific query but in some cases cannot. For example, if the selection built into the temporary index changes with each run of the query, the temporary index may not be reused.

If the temporary index build is done during the full open for this query, but the query goes into reusable ODP mode, then the temporary index is reusable.

If a particular query is run multiple times and a temporary index is built each time, a permanent index must be created to avoid the index build and to make the ODP reusable.

Indexes are never built for selection alone; they always involve a join or a group by or order by clause. No name is given to the temporary index. *TEMP is used in subsequent monitor records.

The most commonly used fields
QQTLN, QQTFN, QQPTLN, QQPTFN, QQILNM, QQIFNM, QVQTBL, QVQLIB, QVPTBL, QVPLIB, QVINAM, and QVILIB describe the table and index used as documented in “Global DB Monitor data fields” on page 4. QQILNM and QQIFNM contains blanks if the index was built from a table. They will contain the library and index name used if an index from index build occurred.

QQRCOD Reason for the index name build.
I2 Ordering or grouping
I3 Selection and ordering/grouping
I4 Nested loop join
I1 Listed in the DB Monitor guide, but will not appear for temporary index builds.

QQTOTR Number of rows in table.
QQRIDX Number of entries in temporary index.
QQSTIM Timestamp for start of index build.
QQETIM Timestamp for end of index build

QQETIM-QQSTIM shows the elapsed time for the index build. Long running builds or builds that are repeated many times and result in a fair number of full opens are prime candidates for being replaced with permanent indexes.

QQ1000 Contains join, order by or group by keys used in index build. This field indicates whether the key is ASC or DESC (this is important for permanent indexes). It does not include additional selection that may have been used to build the index. If *MAP is one of the keys listed, the index build cannot be avoided, but this does not necessarily mean that the ODP is non-reusable.
If QQTOTR = QQRIDX, then the selection is probably not built in, and the permanent index can generally be built using only the keys from the QQ1000 field.

If QQTOTR > QQRIDX, then the selection was built in. If so, it is necessary to use a combination of selection keys and the keys listed in the QQ1000 field to build a permanent index. It may be necessary to look in the query text for keys in ANDed equals predicates and other selective comparisons.

If QQIDXA = Y, QQIDXD will contain good selection keys. However, it still may be necessary to look at the query text if there are problems getting the database optimizer to choose a created index. It is generally best to build a permanent index with good selection keys first, followed by join, order by, or group by keys.

QQIDXA  Index advised (Y or N). If 'N', QQI2 and QQIDXD will not contain data.

QQI2  Number of primary keys in QQIDXD. The QQI2 field contains the number of keys over which key positioning can be used.

QQIDXD  Suggested primary and secondary keys for index (selection only).

The QQIDXD field can contain both primary and secondary keys. Starting from the left, QQI2 tells how many keys are considered primary. Other keys are considered less selective (secondary). This field will be blank if an existing index contains most or all of the recommended selection.

If keys are listed, use the most selective ones combined with keys from the QQ1000 field. It is still important to include the most selective keys since the optimizer is estimating. Even if an index is not advised, it may still be best to try to create an index using the selection from the SQL statement, if it is a good selection.

Remember that building a permanent index that the optimizer will use is an iterative process, but it often has significant payback.

Note the following points in Figure 9:

- The most expensive temporary index was built over ITEM_FACT with key CUSTKEY ascending. Even if this index was not advised, it is possible to get a benefit in creating this index, depending on how frequently the index is created.
- In the data sampled here, we found some indexes created for join, recognized by the reason code (QQRCOD) of I4. Others were created for ordering or grouping (reason code I2).
- Indexes over CUST_DIM were advised.
- Row 9 shows an index from index. Note the presence of information in fields QQILNM and QQIFNM. Note also the value on QVC16 (Index from Index).
- Row 7 shows an index with built in selection.
- Row 9 shows an index with a mapped key.
Figure 9. Type 3002 records - Index created

3003 Record - Query sort

The 3003 record shows that the database optimizer has decided to put selected records into a temporary space and sort them. This is either cheaper than alternative indexed methods or it is forced to do so, for example: UNION or order by on fields from more than one file.

Indexes can still be used to select or join records before the sort occurs. This does not indicate that the ODP is non-reusable.

The 1000 SQL summary record for the open may have a high elapsed time (QQI6 or QQI4). Sort buffers are refilled and sorted at open time, even in reusable ODP mode. However, high elapsed times may indicate a large answer set. In this case, the sort will outperform index usage (this is the situation in most cases).

If sort seems slow and using an index might be better, try to influence the optimizer away from the sort with better selection indexes. For example, if the answer set is small, but the optimizer does not have the right indexes available to

```
SELECT qptln as "QOPTLN Base Library", qptn as "QOPTPN Base Table", qplnm as "QPLNM Lib for Idx Used", qplfn as "QPLPNM Name for Idx Used", qrcood as "QRROOD Reason Code", qptotr as "QPTOTR Rows in Table", qridx as "QRIDX Entries in Temp Idx", qptim - qptim as "Idx Build Time", qqq1000 as "QQ1000 Index Created Key", qqq1000a as "QQ1000A Index Advised" FROM qpt0000898 WHERE qptid = 3002 ORDER BY "Idx Build Time" desc;
```
know that, creating these indexes can help. This is possible only if the optimizer is not forced to use the sort.

The most commonly used fields

**QQSTIM**  Timestamp for start of refill and sort.
**QQETIM**  Timestamp for end of refill and sort
**QQRCOD** Reason for choosing query sort.

The QQRCOD field helps to determine whether a sort was required or if it was “costed” this way. Refer to the DB Monitor guide for the reason codes.

**QQRSS**  Number of rows in sort space.

The QQRSS field can be used, along with reason code, to determine if the indexed approach is possible and possibly cheaper (for a small result set). Use the QQI3 value from the corresponding 1000 FE record for this open to determine how many rows were fetched from the sort space. If the QQRSS value is large, but the actual number of rows fetched is small, consider adding OPTIMIZE FOR n ROWS to the query to help the optimizer.

Figure 10 contains an example of the predefined query for sort information.

![Figure 10. Sort information predefined report (some columns are shown as truncated)](image)

**3004 Record - Temporary file**

The 3004 record show that the database optimizer is forced to store intermediate results/rows in a temporary file due to the nature of the query. Examples are group by on fields from >1 file or materializing view results.

The most commonly used fields

**QQSTIM**  Timestamp for start of fill temporary results table.
**QQETIM**  Timestamp for end of fill temporary results table.
**QQTMPR**  Number of rows in a temporary table.
**QQRCOD** Reason for building temporary index.

Refer to the DB Monitor guide for the specific reason codes.

**3006 Record - Access plan rebuild**

The QQRCOD field lists the reason the rebuild of the plan is occurring. It is not present on every full open. It only occurs in cases where the access plan already exists but for some reason it must change. This field can help determine the reason for a full open if other DB Monitor records do not show why.
3007 Record - Summary optimization

The 3007 record shows a summary of the optimization process, including all indexes evaluated for each given file, including which one, if any, was selected for use in this query, which were not selected, and why. Reason codes are listed next to each index. A reason code of 0 indicates that the index was selected. Other codes are the same as those in the second level text of CPI432C and CPI432D messages.

This record indicates whether the optimizer timed out while evaluating the indexes. Indexes are evaluated in order from newest to oldest, in the same order as shown by DSPDBR for the file, excluding views. To ensure an index is evaluated, delete and recreate it. Then it will be first on the list. The record will not appear if indexes do not exist or if only one index exists and it was selected (see 3001 record for this file).

The most commonly used fields

QQC11 Optimizer timed out (Y or N).

QQ1000 Contains library qualified index names, each with a reason code.

An index from index build may still occur for the index that was selected (look for the 3002 record). If a time-out occurred, only those indexes that were evaluated will be listed.

3010 Record - Host variable and ODP implementation

The 3010 record shows substitution values for host variables or parameter markers in the query text (refer to QQ1000 field in the 1000 record). This record appears just prior to each instance of an open, update, delete, or insert with subselect. This record does not appear for insert with values. Data may not match exactly for updates with parameter markers in the SET clause.

The most commonly used fields

QQ1000 Host variable or parameter marker values.

Values (separated by commas) correspond left to right with host variables/parameter markers in the corresponding SQL statement. All values show up as characters; no quotes or other indicators denote the value type. All floating point values show up as *F.

QQC11 ODP implementation (Reusable, Non-reusable).

QQC12 Host variable implementation (interface supplied values, host variable treated has constants, table management row positioning).

Figure 11 on page 18 shows a result set for the corresponding predefined query, Data conversion information. Note that in the predefined query, fields like QQC11 - Statement function and QQC12 - Host variable implementation are decoded for clarity.
Using AS/400 Database Monitor and Visual Explain To Identify and Tune SQL Queries

Figure 11. Data Conversion predefined report

Note: In Figure 11, some columns are truncated to show Statement Text and Host Variable Values.

<table>
<thead>
<tr>
<th>Record</th>
<th>Description</th>
<th>SQL Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>3021</td>
<td>Bitmap created</td>
<td>UPDATE DB2.UQAISDMVX_DB2_SET_Jobs_Monitored = H, &quot;Number of Jobs&quot; = H, &quot;H-related Name&quot; = H AND &quot;Data Started&quot; = H AND &quot;Time Started&quot; = H WITH NO REFERENCE TO A PREVIOUSLY CONSTRUCTED BITMAP.</td>
</tr>
<tr>
<td>3022</td>
<td>Bitmap merge</td>
<td>(SELECT SUMMARY *) FROM GUSRSYS.GAUSDBMFMD.DISTINCT6, (SELECT DISTINCT *) FROM GUSRSYS.GAUSDBMFMD.DISTINCT7 ORDER BY 1, 11, 12.</td>
</tr>
<tr>
<td>3023</td>
<td>Temporal hash table created</td>
<td>QQI2 Number of hash table entries. QQI3 Hash table size. QQI4 Hash table row size.</td>
</tr>
</tbody>
</table>

3021 Record - Bitmap created

The 3021 record shows that the optimizer decided to construct an index bitmap. To see the index that has been used to construct the bitmap, it is necessary to look at the previous record, which must be a 3001 record, for example:

```sql
SELECT B.QQIINM, B.QQIFNM, A.QQI2
FROM monitor-file-name A, monitor-file-name B
WHERE A.QQRID = 3021 AND RRN(B) = RRN(A) - 1
```

3022 Record - Bitmap merge

After one or more combinations of 3001 and 3021 records, there will be a 3022 record, which will indicate a bitmap merge. The QVC3001 field identifies the bitmaps that were merged together.

3023 Record - Temporal hash table created

Hash tables are used in DB2 UDB for AS/400 in two ways: for grouping and for implementing the hash join algorithm. In many cases, using hash tables gives you better performance than index grouping or joining, particularly if you are interested in the total turnaround and not the first fetched row time. In V4R5, DB Monitor captures information about hash tables used for grouping, but it does not capture data for hash join tables.

Some useful fields are:

- QQI2 Number of hash table entries.
- QQI3 Hash table size.
- QQI4 Hash table row size.
3028 Record - Grouping

The 3028 record shows you that a grouping was performed and explains which fields were used for grouping, the type of grouping (index or hash) and the index used, where applicable.

Some useful fields are:

- **QQC11** Grouping implementation (Hash or Index).
- **QQC101** Index used (or constraint).
- **QQC102** Library name.
- **QVINAM** Long name for the index (or constraint).
- **QVILIB** Long name for the index or constraint library.
- **QQI2** Estimated number of groups.
- **QQI3** Average of rows in each group.
- **QVC3001** Grouping columns.
- **QVC3002** to **QVC3008** Min, max, sum, count, avg, stddev, and var columns respectively.
- **QQC12** Having selection exist (Y/N).
- **QQC13** Having to Where conversion (Y/N).

Analyzing SQL statements involving JOINs

SQL statements with JOIN are among the most sensitive areas in tuning SQL queries. Unfortunately, there is not a specific record for JOIN information in DB Monitor collected data. The information is contained in the access method records used for the tables that are used in the query. To see the join information, you need to look for the following fields in records 3000 to 3004, 3007, 3021 to 3023, 3027 to 3029, among others:

- **QQJNP** Join position
- **QQC21** Join method (NL - Nested loop, MF - Nested loop with selection, HJ - Hash join). Currently DB Monitor can only capture data related to nested loop and nested loop with selection.
- **QQC22** Join type (IN - inner join, PO - Left partial outer join, EX - Exception join). Also present in the 3014 record.
- **QQC23** Join operator: When it uses the nested loop with selection, it is going to show a Cartesian product, even when it is not.
- **QVJFANO** Join fan out (Normal, Distinct fanout, Unique fanout).

From the example in Figure 12 on page 20, you can see that this operation joins two tables, ITEM_FACT as the primary table and PART_DIM as the secondary table. From the QQCC22 field, you see that this is an inner join. Also note that,
after reviewing indexes ITEM_ID2 and ITEM_ID1, ITEM_FACT was accessed using key positioning, based on index ITEM_ID1 and the reason for using an index is I1, row selection. For the PART_DIM table, a temporary index was created on PARTKEY ascending taking 0.24 sec. No indexes were advised.

Is this good enough? There is not an absolute answer to this question. Even if an index is not advised, you can try an index on PART_DIM. Was it easy to analyze? This is a subjective matter, but you can compare it with the visual explanation in Figure 17.

```sql
SELECT qgrid AS "QGRID Record Type"
  ,qqc21 AS "QQC21 Join Method"
  ,qqjnp AS "QQJNP Join Position"
  ,qqc22 AS "QQC22 Join Type"
  ,qqptln AS "QQPTLN Base Library"
  ,qqptfn AS "QQPTFN Base Table"
  ,QQILNM AS "QQILNM Index Library"
  ,QQIFNM AS "QQIFNM Index Name"
  ,qq1000
  ,qqrcod
  ,qqetim - qqstim AS "Elapsed Time"
  ,qqidxa AS "QQIDX A Index Advised"
  ,qvc16 AS "QVC16 Index from Index"
  ,"..." AS "...
  ,rrn(a)
FROM qzg0000898 a
WHERE qqucnt = 20
AND qqjfld LIKE '%022086%'
ORDER BY RRN(a)
```

**Figure 12.** DB Monitor information for an SQL Query involving JOINing

### 3014 Record - Generic query information

The 3014 record appears with full open optimization records. In most cases, one 3014 record appears per full open. Multiple 3014 records can appear if the query consists of multiple separately run queries. For example, subqueries with grouping functions or views that need results materialized for use in outer query. Values in this field help identify what type of query this record represents and how long it took to open the cursor for this query. Refer to DB Monitor for other field values.

In V4R5, record 3014 has been enriched with many valuable fields that can be used for determining the values of many parameters that affect the behavior of the optimizer. It also has summary information for the query or subquery.
Some of the information you can find there include:

QQI1  Time spent to open the cursor (in milliseconds).

QQORDG  Ordering involved (Y/N).

QQGRPG  Grouping involved (Y/N).

QQJNG  Joining involved (Y/N).

QQC22  Join type (if it applies).

QQI5  Time spent in optimizing.

QQC102  Query optimization File (QAQQINI) Library. *N when it was unable to retrieve a query optimization file.

QVC16  ALWCPYDTA setting.

QVC81  Parallel degree.

QVC18  Force join order (Y/N).

Plus, there are many others.

Other DB Monitor records

The following records are not used as much for performance tuning, but can provide other interesting data:

3005 record  Table locked.

3008 record  Subquery processing.

3018 record  STRDBMON/ENDDBMON information/option.

3019 record  Records retrieved detail record (only occurs with TYPE(*DETAIL).
Operations Navigator Visual Explain

Visual Explain is a new tool introduced in V4R5. It provides a graphical way of identifying and analyzing SQL performance, providing a graphical representation of the access plan the optimizer generates for a given SQL query. It will identify the tables involved and indexes considered for the query execution as well as join order, type of sort (if required), type of grouping (if required), and other valuable information.

Using Visual Explain

There are two ways to use Visual Explain:

- Interactively from an Operations Navigator Run SQL Scripts window
- From SQL Performance Monitor

Using Visual Explain interactively

In the Run SQL Scripts window of Operations Navigator, you will find a Visual Explain option on the main menu bar, with three options. Explain will explain the SQL statement where the cursor is positioned, without executing it. Run and Explain will run and then explain the SQL statement where the cursor is positioned. Graphical buttons are also present (see Figure 13).

![Figure 13. Visual Explain menu bar](image)

An interesting variation for this method, in a kind of “help desk approach”, involves right-clicking Database in the Operations Navigator left-hand window and selecting Current SQL for a Job. Follow the instructions to find the current (or last executed) SQL statement for that specific job (see Figure 14 on page 24). Clicking the Edit SQL button opens a Run SQL Scripts window with this statement, ready to be executed or just analyzed. Remember that this Run SQL Scripts is a new job, and may have different SQL runtime environmental properties to the original job.
Using Visual Explain over DB Monitor collected data

Visual Explain can also be used to analyze SQL statements from DB Monitor. It can analyze data captured with Operations Navigator SQL Performance Monitor, detailed option. It can also analyze DB Monitor information captured through the STRDBMON command, both summary or detailed. In this case, the data must be imported to SQL Performance Monitor as explained in “Importing an SQL Performance Monitor” on page 3.

Once the SQL Performance Monitor of interest is located in the Operations Navigator window, right-click **List Explainable Statements** (Figure 15). A window with the list of explainable statements appears. An SQL statement can be selected by clicking the **Date** column. The SQL statement appears in the bottom half of the window (see Figure 16). The visual explanation can be obtained by clicking the **Run Visual Explain** button.
Once an SQL statement has been selected for analysis, Visual Explain can be used to graphically describe its access plan, as shown in Figure 17 on page 26. The Visual Explain window is divided into two parts. The left-hand side of the window shows the graphical explanation, while the right-hand side shows the attributes for the highlighted icon. The following example shows a select statement involving a join. Highlighting each icon and reviewing their attributes, we can learn that, in this join, ITEM_FACT is the primary table and is accessed by key using ITEM_ID. PART_DIM is the secondary table in the join and a
temporary index was built to satisfy the join needs. The estimated number of rows to be selected from ITEM_FACT was 678 and for PART_DIM was 20,000, which explains the join order. This example corresponds to the one used in “Analyzing SQL statements involving JOINs” on page 19.

Figure 17. Visual Explain over an SQL statement involving JOINing

We now have a complete set of tools for tuning SQL statements, with SQL Performance Monitor summary and its predefined reports helping us detect the jobs that require further analysis, SQL Performance Monitor detailed and its predefined reports helping us to detect the SQL statements requiring further analysis and Visual Explain giving us a readable and easy to understand graph of the access plan for a particular SQL query.
Database performance analysis

There are many different methods to identify problems and tune troublesome database queries. One of the most common methods is to identify the most dominating, time-consuming queries and work on each of them individually. Another method is to leverage global information and to use this information to look for indexes that are “begging” to be created.

A set of queries have been included in this section. These queries have been designed to help produce useful results in most situations. Using these queries will help you to understand the data and to learn to construct other queries as well. Query analysis is iterative in nature. Try something running the job (with the DB Monitor active) or the individual query (using Operations Navigator Run SQL Script with Include Debug Messages in Job Log set on or STRSQL using debug messages) to see if it worked. Try it again if it did not work.

When using the STRSQL command, ensure that the appropriate settings are used for the “Data refresh” and “Allow copy data” options. These settings can be changed by using the Change Session Attributes option in STRSQL. For example, JDBC uses *FORWARD for Data refresh and *OPTIMIZE for Allow copy data. The defaults are *ALWAYS for Data refresh and *YES for Allow copy data.

It is usually best to first concentrate on repetitious non-reusable ODPs, table scans, and long index builds. Also, look for repetitious short-running queries that are not optimized well. Joins and sorts can be more difficult to analyze. If joins and sorts are accounting for a significant portion of run time they need to be addressed as well. Fine tuning smaller problems should be done after large problems have been addressed. Generally, indexes will be used to tune most performance problem areas.

Database performance problems that are not easily avoidable

Problems that are generally not simple to rectify include:

- Full opens due to repetitive query using new literal values each time it's invoked:

  If the prepared statement text differs in any way from any previous SQL statement in that job, a full open and new ODP will be needed. SQL will, in most cases, attempt to convert literals to parameter markers to make the repetitive statement appear identical each time. However, literals cannot be converted in the following cases:
  - Parameter marker conversion turned off
  - Original SQL statement contains both parameter markers and literals
  - Statement uses special registers (CURRENT DATE, CURRENT TIME)
  - Expressions used in SET or SELECT clause of statement

- Non-reusable ODPs due to:
  - Before V4R5, temporary results table created for ODP (3004 record appears in the full open). Now temporary tables are cleared between statement executions to allow reuse of the ODP.
  - Group by fields on more than one file or view with grouping (materialization).
Although the full opens listed here are not avoidable, they can still possibly be improved. For example, an open might contain a 3004 record but might also contain a costly temporary index build that can be avoided by building a new permanent index. Look at the opens to see what can be done.

- Time-consuming queries due to:
  - Correlated subqueries run excessive number of times within a query.
  - NOT EXISTS forcing poor join order.
  - Updates or deletes with poor join performance because the file being changed is forced to be first in join.

---

**Remember**

In V4R5, the modern join syntax does not force join order as in previous releases. Two new parameters were added to the QAQQINI file for controlling join implementation: FORCE_JOIN_ORDER (with values *DEFAULT or *FORCE) and STAR_JOIN (with values *DEFAULT or *FORCE).

---

- Long running index builds with mapped keys in the build (the 3002 record has *MAP in QQ1000). Some programming practices cause unnecessary data conversion that derived in *MAP fields that can be avoided with small changes in the SQL statement.

---

**Problems that are generally avoidable**

Problems that are relatively easy to resolve include:

- Temporary indexes created for join, order by or group by:
  - Individual long-running or repeated index creates (non-reusable ODPs).
  - A 3002 record exists in the full open (without *MAP in QQ1000 field).

- Time-consuming queries due to:
  - Table scans - single long-running or repetitious short-running
    A 3000 record exists in the full open (unless the open is done for a file that is being inserted into).
  - Poor join order
    Need to be familiar with what a better join order would be and how to influence optimizer.
  - Temporary sort being done when index approach feasible and faster
    A 3003 record exists in the full open and possibly longer opens in reusable ODP mode.
  - Reusable ODPs with a good initial access plan that become worse due to table growth
    Watch the QQI6 or the QQI4 value in the 1000 records to see if the value grows. Try to influence the optimizer to an initial access plan that is not affected by table growth.
Creating indexes

It is helpful to know how data in the table is populated and how selective certain key fields are. This information can be used to help create indexes that will be used for a large number of queries. It can also help to know why existing indexes were not used in some situations. For example, if a query has WHERE A =? AND B =? AND C =? and there is an index over A, B and C but the database optimizer decides not to use it, it may be because these fields are not very selective. Knowing the data can help to quickly detect this. If the selective of certain fields are not known, the fields can be queried to find out:

```
SELECT A, B, C, COUNT(*)
FROM TABLEA
GROUP BY A, B, C
ORDER BY 4 DESC
```

Try to create indexes that are used more globally. Use selective fields that are commonly used in WHEREs, and where applicable, use them in combination with common join, order by, and group by keys. If index is uniquely keyed, create the index as UNIQUE since this is useful to the database optimizer.

Do not create a lot of permanent indexes trying to cover every combination. Create one or two that are potentially good, run the job again or run the STRSQL command for a single query and see if they are used. If the indexes are used and the run time is noticeably better, consider deploying the indexes for permanent use. If the indexes are not used, delete them and try a different combination. Do not create indexes just to solve a single instance of a full open or query unless it accounts for a significant amount of time.

Each additional index created for a table will cause overhead when:

- Updates to the table include the index keys
- Rows are inserted or deleted for the table
- Full opens occur for that file (index evaluation)

For faster analysis

The DB Monitor file often is large and contains information on many jobs. Therefore, running queries on the data can sometimes be slower than desired. Users can try to help this by collecting only the job they want. However, sometimes this is not possible and, even if it is, batch jobs can generate a lot of DB Monitor output. Also, using interactive tools such as STRSQL, can result in longer run times on server models. If the response time is slow during the analysis, consider the following tips:

- Create a smaller DB Monitor file with only the records you are interested in:
  - Build and run a query to select only those jobs you’re interested in. Then, copy the records for those jobs into a separate DB Monitor file.
  - CRTDUPOBJ and INSERT with subselect specifying the QQJNUM value or values.
- Create indexes on the DB Monitor file over the common selection and grouping/order by clauses.
Examples of some key combinations to use are:
- QQRID, QQC21
- QQJFLD, QQC21 & QQI5
- QQRID & QQ1000
- QQRID & QQC21

Try other combinations as needed. Remember, combine the selection and grouping/order by clauses.

- When you use Operations Navigator to start a detailed SQL Performance Monitor, as soon as you end the data collection, it will provide you with a couple of indexes, one based on QQJFLD, QQC21 & QQI5 and another based on QQRID & QQC21.

**Altering insert with subselect data in a Monitor file**

Within the DB Monitor file, inserts with subselects actually contain two QQC21s. The first one is listed for subselect optimization messages during a full open. The second one is for the actual insert statement and each instance of the reusable ODP after that.

This can be a problem when trying to look at the optimization messages for the subselect using the QQC21 record for the insert operation itself. There is no easy way to view this. The problem is being addressed, but for now, use the following method to correct this problem.

```sql
UPDATE monitor-file-name
SET QQC21 = QQC21 + 1
WHERE QQJFLD || DIGITS(QQC21) IN
    (SELECT DISTINCT QQJFLD || DIGITS(QQC21)
     FROM monitor-file-name
     WHERE QQC21 = 'IN' AND UPPER(QQ1000) NOT LIKE '%VALUES%' AND
     QQI5 = 0)
```

**DB Monitor query examples 1 through 13**

Here we present a series of queries to help solve specific questions in the detection and resolution of SQL performance issues. Most of these queries have more elaborate equivalents in the SQL Performance Monitor predefined reports as we indicate below. However, it is still useful to be familiar with the DB Monitor table.

**Example 1: Identify the specific job(s) to analyze**

From the Job Summary predefined report (see “Reported DB Monitor record types” on page 8), pick out the jobs that use the most run time. Use the Job Number value as selection in other analysis steps. If only one job is monitored or one job completely dominates the others, Job Number is not needed. If the job cannot be identified from this information, then work with the customer or use other tools, like WRKSYSACT, to determine the job.

**Example 2: Which type of SQL operations account for the most run time**

The Operation Summary predefined report shows the number of SQL operations (Open, Prepare... Describe, Close, Call, Fetch,...) used and how much elapsed time is spent by each type of operation. The results from this query can help to
know what to concentrate on for the most potential payback (selects, updates, etc.).

Example 3: Which SQL operations account for the most run time (list by text)
The Basic Statement Information into Detail Result tab shows the text of the SQL operations that account for the most run time and the number of times they were run. It also shows valuable information, such as type of ODP (reusable or non-reusable), number of table scans, number of indexes created, number of indexes used, number of indexes, etc.

Example 4: Which queries account for the most run time (list by QQUCNT)

```sql
SELECT SUM(QQI4), COUNT(*), QQUCNT, QQFLD
FROM monitor-file-name
WHERE QQRID = 1000 AND QQC21 <> 'MT' AND QQUCNT <> 0
GROUP BY QQUCNT, QQFLD ORDER BY 1 DESC
```

This query shows individual query instances and how much time they took, sorted by the run time (largest to smallest). It also includes fetch and close time for opens. It does not include non-ODP SQL operations such as prepare or describe (QQUCNT=0). This provides a way to quickly find a query that is taking a large amount of time without having to know the text or anything else about that query. Consider limiting QQC21 values to look at certain query type. If there is not a single dominating query or set of queries, and if Example 3 does not provide the information that is needed, go to Example 5.

The equivalent predefined query is Basic Statement Information from the Extended Detailed Result tab.

Example 5: Which queries account for the most run time

```sql
SELECT QQFLD, QQUCNT, QQ1000, SUM(QQI4), COUNT(*)
FROM monitor-file-name
WHERE QQRID = 1000 AND QQC21 <> 'MT' AND QQUCNT <> 0
GROUP BY QQ1000, QQUCNT, QQFLD ORDER BY QQJFLD, QQUCNT, 1 DESC
```

The results of this query show all parts of a query grouped together and listed by QQUCNT. Fetches and closes are listed together with their corresponding opens. It does not include non-ODP operations (prepare, describe, commit, etc.). Scroll through the unique queries to see the most expensive ones, and use the QQUCNT record to further analyze them (see Example 6). The query can be changed to sort by text and run time (4, 1 DESC) to group similar queries together and see which are the most costly or just by run time to see the most costly overall. It may also be useful to add QQC21 values to select a certain query type.

Example 6: How do I determine how an individual query was run

```sql
SELECT QQRID, QQPTFN, QQIFNM, QQC21, QQI4, QQ1000
FROM monitor-file-name
WHERE QQJNUM = 'job-number' AND QQUCNT = query-number
```

The fields used in this query are a good starting point, but there are other fields that can help to determine what the query is doing and the access plan that was used. For example, to find how long an index build within this query is taking, replace QQI4 with QQETIM-QQSTIM. If full open data was collected, optimization...
Using AS/400 Database Monitor and Visual Explain to Identify and Tune SQL Queries

Records will appear first followed by the 1000 records, then repeated instances of the query if the ODP was reusable. The text of the query along with the host variable values from the 3010 record can be used to reconstruct the query for debug purposes.

For this purpose, Visual Explain represents an outstanding tool. Under the covers, Visual Explain interprets the contents of DB Monitor table and represents it graphically. Figure 18 and Figure 19 show a parallel between the way Visual Explain graphs a specific query and the records on DB Monitor for the same query.

Figure 18. Visual Explain to DB Monitor icon mapping on a single Table Scan selection

---

32 Using AS/400 Database Monitor and Visual Explain To Identify and Tune SQL Queries
Example 7: Which queries are significantly affected by full opens

The Open Information from the Detailed Results tab points out queries whose run time is noticeably affected by full opens. Open Information from the Extended Detailed Results tab also includes the SQL statement text. Use Example 8 to find any individual queries that are doing full opens.

Example 8: List all queries (sorted by text) that are doing full opens

```sql
SELECT QQI4, QQUCNT, QQ1000 FROM monitor-file-name
WHERE QQJNUM = 'job-number' AND QQRID = 1000 AND QQI5 = 0
```

Note

For mapping each icon with its respective field in the DB Monitor table, it is necessary to show more fields than are shown.
AND QQC21 IN ('OP', 'DL', 'IN', 'UP')
ORDER BY 3, 1 DESC

This query groups together queries that are doing full opens. Scroll through the list to find repetitive queries. Select QQUCNT from one of these and use it to look at the query to understand why it is doing a full open. Remember to look closely at the text. If it is a "new" SQL statement, the open cannot be avoided since SQL must create an ODP the first time. Look at certain query types by limiting the QQC21 list and sort by QQI4 to see the most costly opens. Remember that the first full open for each unique query is unavoidable, although it may be possible to improve them in some cases.

Example 9: List all queries with full opens that contain a temporary index build

```
SELECT QQ1000, QQUCNT
FROM monitor-file-name
WHERE QQJNUM = 'job-number' AND QQRID = 1000 AND QQI5 = 0
AND QQC21 IN ('OP', 'DL', 'IN', 'UP') AND QQUCNT IN (
    SELECT DISTINCT QQUCNT
    FROM monitor-file-name
    WHERE QQJNUM = 'job-number' AND QQRID = 3002 AND
    QQ1000 NOT LIKE '%*MAP%')
ORDER BY 3, 1 DESC
```

This query helps to quickly locate queries that are doing temporary index builds. It provides the QQUCNT values that can be used to look at the query optimization to find more information. First, sort by QQI4 first to find the most expensive opens. This does not necessarily mean that the open can be avoided. There may be other reasons the full open is occurring besides the temporary index build.

Example 10: Which index builds are done most often

```
SELECT QQUCNT, QQETIM-QQSTIM, QQPTFN, QQTOTR, QQRIDX,
QQRCOD, QQIDXA, SUBSTR(QQ1000, 1, 100),
SUBSTR(QQIDXD, 1, 100)
FROM monitor-file-name
WHERE QQRID = 3002
ORDER BY 8
```

This query points out commonly occurring index creates and how to create permanent indexes to avoid them. Look for repeated index builds or long index builds first. If an index build only occurs once and is not costly, it may be best to let it occur. The SUBSTR for QQ1000 and QQIDXD should cover most key lists, but it may be necessary to increase the respective values if 100 bytes is not enough.

Use the following methods to determine what keys to use to build a permanent index:

- If QQTOTR = QQRIDX, the keys from QQ1000 for the index build will probably be acceptable.
• If QQTOTR > QQRIDX, additional selection keys should be in the index along with the join, order by, or group by keys from the QQ1000 field. The fields from QQIDX can be used, if available. However, it may be desirable to add QQI2 to the selection to know the number of primary keys.

• If there is no data in QQIDX or the keys from QQIDX do not seem to work, get the most selective keys from the query text itself. Use QQUCNT to find the query text.

Example 11: Which full opens are not avoidable due to temporary table results

```sql
WITH AA AS
(SELECT DISTINCT QQJFLD, QQUCNT, QQRCOD
 FROM monitor-file-name
 WHERE QQRID = 3004)
SELECT A.QQUCNT, A.QQC21, A.QQI4, A.QVC14, B.QQRCOD, A.QQ1000
FROM monitor-file-name A, AA B
WHERE
-- Selection condition
A.QQRID = 1000 AND A.QQI5 = 0 AND A.QQC21 IN ('OP', 'IN', 'UP', 'DL') AND
-- Join condition
A.QQUCNT = B.QQUCNT AND B.QQJFLD = B.QQJFLD
ORDER BY 4, 3 DESC
```

This query shows all full opens that are unavoidable due to temporary results table and groups them by text and run time. The QQRCOD field shows the reason code for the temporary table. Often it is useful to sum on QQI4 and group on QQ1000 to determine the overall cost. It still may be possible to improve the cost of the open. Use QQUCNT to see if the query can be optimized better or if other tunable items, such as index build, exist.

Example 12: Which queries involve an arrival sequence

```sql
WITH R3000 AS
(SELECT DISTINCT QQUCNT, QQJFLD, QQRCOD
 FROM monitor-file-name
 WHERE QQRID = 3000 AND QQJNUM = 'job-number')
SELECT SUM(QQI4), COUNT(*), A.QQUCNT, B.QQRCOD, QQ1000
FROM monitor-file-name A, R3000 B
WHERE A.QQJFLD = B.QQJFLD AND A.QQUCNT = B.QQUCNT AND
A.QQRID = 1000
GROUP BY A.QQUCNT, B.QQRCOD, QQ1000 ORDER BY 1 DESC
```

This query lists the most expensive SQL operations that are using arrival sequence. The presence of a 3000 record does not necessarily cause bad performance (inserts, bit maps, and scans of small files). Use the QQUCNT record to look at the most expensive operations to see how much the file using arrival sequence is affecting the query. Look for fetches that are costly per fetch since this can suggest that a table scan is being done on a large file – prime candidate for creating a new index. It may be useful to order by QQUCNT since this lists the specific operations (for example, fetch) with the corresponding open/close.

The equivalent predefined query is Table Scan Information into Detail or Extended Detail Result tabs.
Example 13: Which queries involve the use of a query sort

WITH R3003 AS (  
    SELECT DISTINCT QQUCNT, QQJFLD, QQRCOD  
    FROM monitor-file-name  
    WHERE QQRID = 3003 AND QQJNUM = 'job-number')  
SELECT SUM(QQI4), COUNT(*), A.QQUCNT, B.QQRCOD, QQ1000  
FROM monitor-file-name A, R3003 B  
WHERE A.QQJFLD = B.QQJFLD AND A.QQUCNT = B.QQUCNT AND  
    A.QQRID = 1000  
GROUP BY A.QQUCNT, B.QQRCOD, QQ1000 ORDER BY 1 DESC

This query lists the most expensive SQL queries that are using a query sort. The presence of a 3003 record does not necessarily indicate poor performance. Use QQUCNT to look at the most expensive opens (selects). Look for opens (selects) that are costly per open. A sort may be required. See the QQRCOD field in the 3003 record. If it is not required, determine if an index would help. It may be useful to order by QQUCNT since this lists the specific operations (for example, fetch) with the corresponding open/close.

End user query tools

Once a basic understanding of the queries that can be used to query the Database Monitor output is gained, the next step is to become efficient at using the queries. A variety of query tools exist that can help access the data from the Database Monitor:

- Operations Navigator: Run SQL Script
- Interactive SQL
- Query/400
- DB2 UDB for AS/400 Query Manager

Of the query tools listed above, DB2 UDB for AS/400 Query Manager (LLP 5769-ST1) is the most flexible for composing, executing, and managing reports from queries. The DB2 UDB for AS/400 Query Manager allows user variables in queries so that items like the database filename can be selected when the query is run, instead of when the query is written. DB2 UDB for AS/400 Query Manager Report Forms make formatting output of queries easier by generating column headings, spacing, and field wrapping for more readable end-used reports.

You can locate *DB2 UDB for AS/400 Query Manager User Guide*, SC41-5212, on the Web at:


Summary

Tuning SQL statements and database performance can be a very demanding task. Performance analysis of database problems can be difficult and time-consuming. Now, with the tools introduced in V4R5, such as Visual Explain and SQL Performance Monitor predefined reports, it is becoming more accessible.

Performance tuning, particularly when dealing with database operations, is an iterative process but the availability and knowledge of powerful tools allow the performance analyst to find a solution quickly.
Knowledge and judicious usage of the OS/400 Database Monitor tool, its predefined queries, Visual Explain and the detailed queries provided in this document allow the analyst to gain more information about how the application SQL statements are performing and what can be done to improve their behavior. The time it takes for the performance analyst to understand the database problems and solve them can be reduced significantly by using this tool.

**Additional information and author contacts**

Special thanks goes to Dale Weber for the usage of his SQL class foils used to develop this redpaper.

IBM Global Services at Rochester, MN, offer technical support for Performance Analysis & Capacity Planning of AS/400 products. Their services are described at: [http://www.as400.ibm.com/service/igs/pss.htm](http://www.as400.ibm.com/service/igs/pss.htm)

Additional information regarding AS/400 and UDB DB2/400 and the Database Performance Monitor is available on the AS/400 home page beginning at: [http://www.as400.ibm.com/db2](http://www.as400.ibm.com/db2)

Questions regarding the conclusion reached in this redpaper should be addressed to Rick Peterson (rickmp@us.ibm.com) or Richard Odell (rjodell@us.ibm.com).

© Copyright International Business Machines, 2000. All Rights Reserved.
Appendix A. Special notices

This publication is intended to help programmers, analysts, and database administrators to identify and tune SQL queries. The information in this publication is not intended as the specification of any programming interfaces that are provided by DB2 UDB for AS/400. See the PUBLICATIONS section of the IBM Programming Announcement for DB2 UDB for AS/400 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

e (logo)®
IBM ©
AS/400
CT
DB2

Redbooks
Redbooks Logo
AT
Current
DB2 Universal Database
The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.