





Reading Sample

This chapter describes methods for analyzing individual ABAP programs, for example, using tools such as performance trace and ABAP runtime analysis, debugger, and code inspector. It also provides information on optimizing ABAP code.

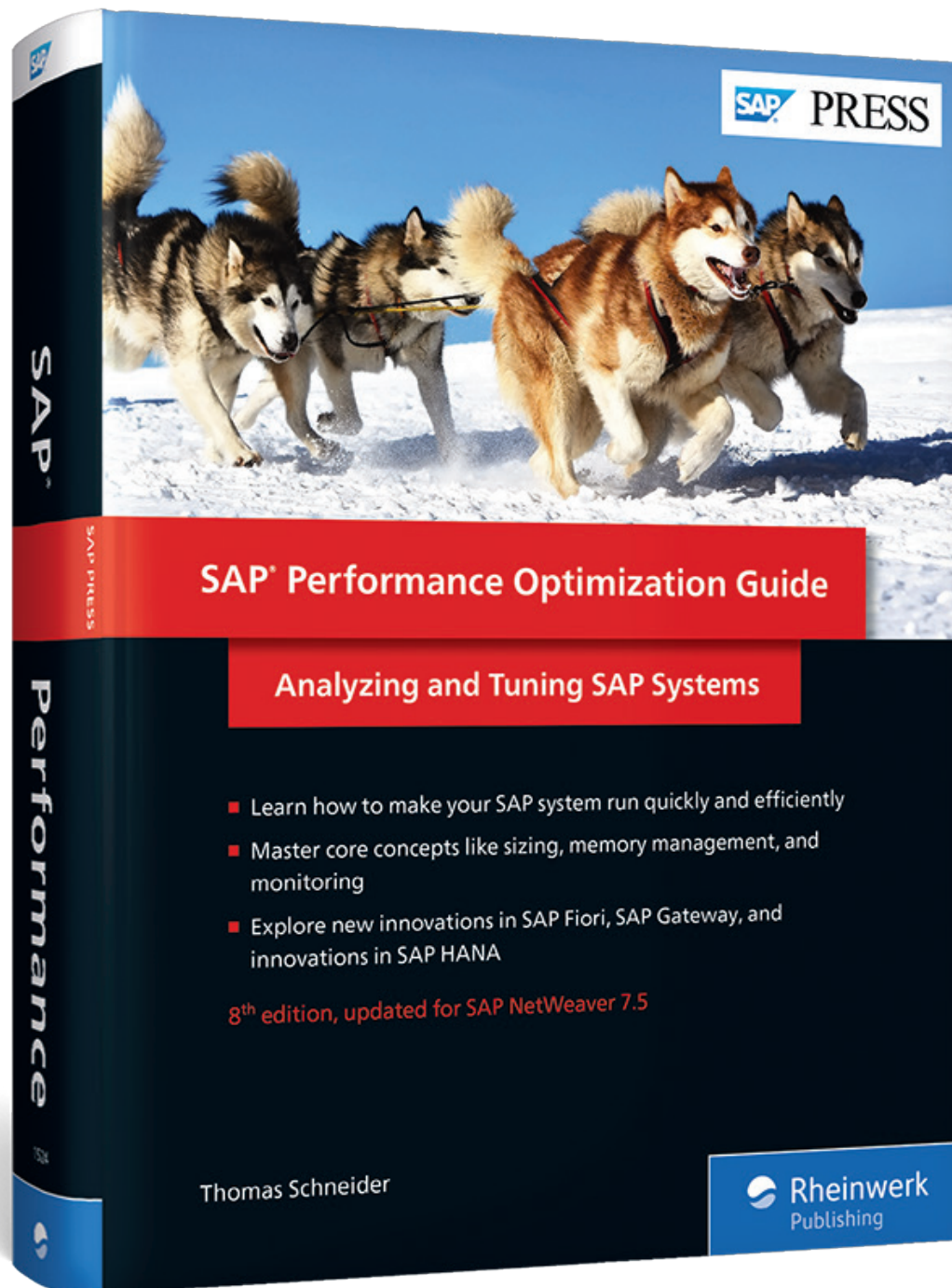
-  **“Optimization of ABAP Programs”**
-  **Contents**
-  **Index**
-  **The Authors**

Thomas Schneider

SAP Performance Optimization Guide: Analyzing and Tuning SAP Systems

892 Pages, 2018, \$89.95
ISBN 978-1-4932-1524-9

 www.sap-press.com/4370



Chapter 5

Optimization of ABAP Programs

This chapter explains how to perform a detailed performance analysis of programs and transactions that you've already identified as expensive. In other words, you've performed a workload analysis, consulted users, and discovered that the performance of these programs isn't satisfactory.

To begin the analysis of expensive ABAP programs, examine the *single statistical records*, which will give you an overview of the response times of a transaction. For more in-depth analysis, use SAP performance trace for detailed analysis of database accesses, remote function calls (RFCs), and lock operations (enqueues). If you still can't find the problem after using these methods, you can use ABAP trace and ABAP debugger as additional analysis methods. You should also routinely use the Code Inspector, which has numerous statistical performance checks.

When Should You Read This Chapter?

You should read this chapter if you've identified a program or transaction as being critical for performance and you now want to perform a detailed analysis of it.



5.1 Performance Trace

Performance trace is a powerful tool for analyzing the runtime of ABAP programs. It enables you to record a program runtime for the following operations: database access (i.e., SQL user statements), RFCs, enqueue operations, and accesses to SAP buffers. Performance trace is a tool that was developed by SAP and is thus identical for all database systems, except in the fine details.

Runtime analysis of
ABAP programs

To navigate to the initial performance trace screen, select **System • Utilities • Performance Trace** or use Transaction ST05. In this view, you'll find buttons to start, stop, and evaluate performance trace. You can also find checkboxes for selecting the trace modes **SQL-Trace**, **Enqueue Trace**, **RFC Trace**, **HTTP Trace** (as of SAP Basis version 7.10), and **Buffer Trace**. Only the field for the SQL trace is marked by default. For standard analysis of a program, we recommend that you activate the SQL trace, enqueue trace, HTTP trace, and RFC trace.

5.1.1 Activating a Performance Trace

You can start and stop a performance trace by clicking the **Trace On** and **Trace Off** buttons in Transaction ST05. You can only create one performance trace per application server at a time. In the **State of Trace** field, you can see whether a trace is already activated and which user has activated the trace. When you start a trace, a selection screen appears where you can enter users for whom the trace should be activated. The name with which you logged on is usually the user name entered here. Use a different name if you want to trace the actions of another user. The user who activates the trace doesn't have to be the same one whose actions are being traced.

Information on activating a trace

Keep in mind the following points when activating a trace:

- Ensure that the user whose actions are to be recorded only carries out one action during the trace; otherwise, the trace won't be clear. You should also ensure that no background jobs or update requests are running for this user.
- The performance trace is activated in the application server. For each database operation, data is written to a trace file in the file system on the application server. You must therefore ensure that you've logged on to the same application server as the user to be monitored. This is particularly important if you want to record an update request or a background job and are working in a system with distributed updating or distributed background processing. In this case, you won't know where the request will be started, and, as a result, you'll have to start the trace on all application servers with update or background work processes.

- The SQL trace only displays accesses to the database. SQL statements that can be satisfied from data in the SAP buffer don't appear in the trace. If you want to analyze SAP buffer accesses, activate the SAP buffer trace.
- However, buffer load processes are also recorded in the SQL trace. Because you're normally not interested in recording the buffer load process in the SQL trace, first execute a program once without activating the trace to allow the buffers to be loaded (i.e., the SAP buffers and database buffers). Then run the program again with the SQL trace activated, and use the results of this trace for evaluation.
- During the trace, look at the following monitors: the work process overview (for general monitoring), the operating system monitor of the database server (for monitoring possible CPU bottlenecks on the database server), and the database process monitor for direct monitoring of the executed SQL statements. It makes no sense to watch these monitors during the trace if you're logged on as the user being traced. The SQL statements of the monitors would appear in the trace and thus make the trace unreadable.
- The default trace file name is set with the SAP profile parameter `rstr/file`. In the initial screen, you can assign a different name to the trace file. Writing to the trace file is cyclical in the sense that when the file is full, the oldest entries are deleted to make room for new entries. The size of the trace file (in bytes) is specified by the SAP profile parameter `rstr/max_diskspace`, for which the default value is 16,384,000 bytes (16 MB).
- You can also store recorded traces. For this purpose, select **Performance Trace • Save Trace** or **Performance Trace • Display Saved Trace** to retrieve a saved trace again.

5.1.2 Evaluating an SQL Trace

To evaluate a performance trace, select **Display Trace** in the initial screen. A selection screen is displayed, and you can specify the part of the trace which you want to analyze in the **Trace Type** field. In this and the following sections, we'll discuss the evaluation of each of the trace types separately. In practice, you can analyze all trace modes together. Table 5.1 lists other fields you can use to restrict SQL trace analysis.

Field	Explanation
Trace Filename	Name of the trace file. Normally this name should not be changed.
Trace Type	The default trace mode setting is SQL-Trace . To analyze an RFC trace, enqueue trace, HTTP trace, and buffer trace, select the corresponding checkboxes.
Trace Period	Period in which the trace runs.
User Name	User whose actions have been traced.
Object Name	Names of specific tables to which the display of trace results is to be restricted. Note that, by default, the tables D010*, D020*, and DDLOG aren't shown in the trace results. These tables contain the ABAP coding and the buffer synchronization data.
Execution time	Restricts the display to SQL statements that have a certain execution time.
Operation	Restricts the trace data to particular database operations.

Table 5.1 Fields in the Dialog Box for Evaluating a Trace

Executing trace

Next, click the **Execute** button. The basic SQL trace list is displayed. Figure 5.1 shows an example of a basic trace list. Table 5.2 explains the fields displayed in an SQL trace.

Field	Explanation
hh:mm:ss.ms	Time stamp in the form <i>hour:minute:second:millisecond</i> .
Duration	Runtime of an SQL statement in microseconds (μ s). If the runtime is more than 150,000 μ s, the corresponding row is red to identify that the SQL statement has a long runtime. However, the value 150,000 μ s is a somewhat random boundary.
Program Name	Name of the program from which the SQL statement originates.
Object Name	Name of the database table or database view.

Table 5.2 Fields in an SQL Trace

Field	Explanation
Operation	The operation executed on the database, for example, PREPARE (preparation “parsing”) of a statement), OPEN (open a database cursor), FETCH (transfer of data from the database), and so on.
Curs	Database cursor number.
Records	Number of records read from the database.
RC	Database system-specific return code.
Statement	Short form of the executed SQL statement. The complete statement can be displayed by double-clicking the corresponding row.

Table 5.2 Fields in an SQL Trace (Cont.)

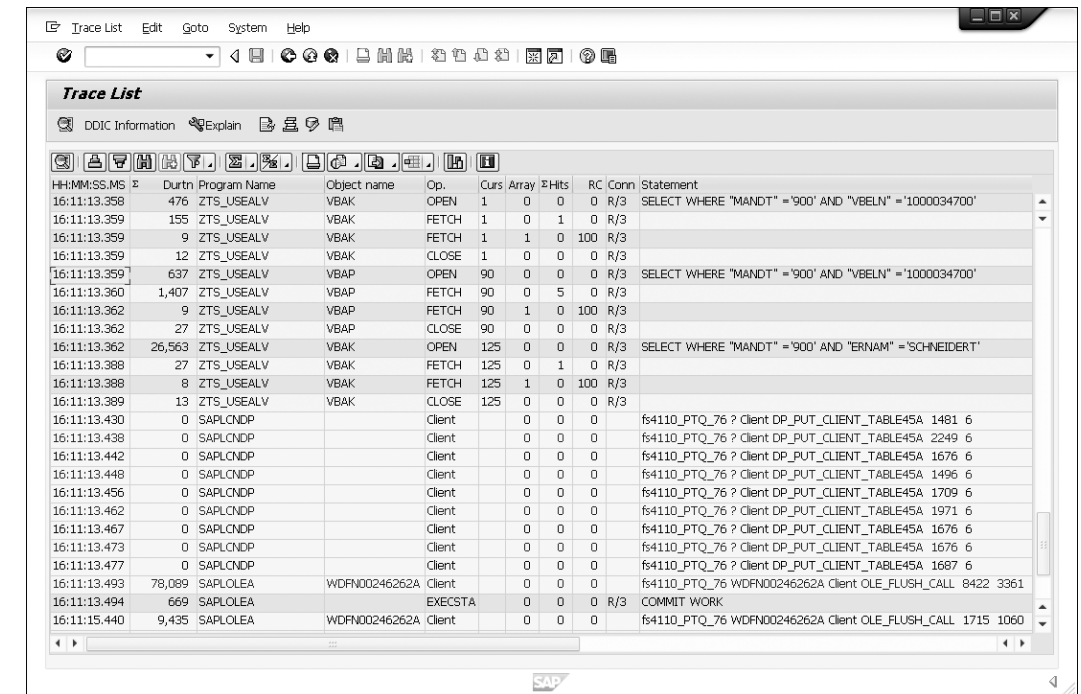


Figure 5.1 Basic Performance Trace List with Entries from SQL Trace and RFC Trace

Direct read An SQL statement that appears in Figure 5.1 accesses table VBAK. The fields specified in the WHERE clause are key fields in the table. The result of the request can therefore only be either one record (Rec = 1) or no record (Rec = 0), depending on whether a table entry exists for the specified key. SQL statements in which all fields of the key of the respective table are specified as “same” are called *fully qualified accesses* or *direct reads*. A fully qualified database access should not take more than about 2 to 10 ms. However, in individual cases, an access may last up to 10 times longer, such as when blocks can't be found in the database buffer and must be retrieved from the hard drive.

The database access consists of two *database operations*, an OPEN/REOPEN operation and a FETCH operation. The REOPEN operation transfers the concrete values in the WHERE clause to the database. The FETCH operation locates the database data and transfers it to the application server.

Sequential read A second access in Figure 5.1 takes place in table VBAP. Not all key fields in the WHERE clause are clearly specified with this access. As a result, multiple records can be transferred. However, in our example, five records are transferred (Rec = 5). The data records are transferred to the application server in packets, in one or more fetches (*array fetch*). An array fetch offers better performance for applications than transferring individual records in a client/server environment.

The second access takes place via an efficient index, thus the duration of execution also remains significantly less than 10 ms. The third access (again in table VBAK) takes place via a field for which there is no efficient index. Thus, the duration of this statement is significantly longer than that of the previous statement.

Maximum number of records The maximum number of records that can be transferred in a FETCH operation is determined by the SAP database interface as follows: every SAP work process has an I/O buffer for transferring data to or from the database. The SAP profile parameter `dbs/io_buf_size` specifies the size of this buffer. The number of records transferred from the database by a fetch is calculated as follows:

$$\text{Number of records} = \text{dbs/io_buf_size} \div \text{length of 1 record in bytes}$$

The number of records per fetch depends on the SELECT clause of the SQL statement. If the number of fields to be transferred from the database is

restricted by a SELECT list, more records fit into a single fetch than when SELECT * is used. The default value for the SAP profile parameter `dbs/io_buf_size` is 33,792 bytes and shouldn't be changed unless recommended explicitly by SAP.

Guideline Value for Array Fetch

The guideline response time for optimal array fetches is less than 10 ms per selected record. The actual runtime greatly depends on the WHERE clause, the index used, and how effectively the data is stored.

Other database operations that may be listed in the SQL trace are DECLARE, PREPARE, and OPEN. The DECLARE operation defines what is known as a *cursor* to manage data transfer between ABAP programs and a database, and also assigns an ID number to the cursor. This cursor ID is used for communication between SAP work processes and the database system.

In the subsequent PREPARE operation, the database process determines the access strategy for the statement. In the **Statement** field, the statement is to be seen with a variable (INSTANCE =:AO, not shown in Figure 5.1). To reduce the number of relatively time-consuming PREPARE operations, each work process of an application server retains a certain number of already parsed SQL statements in a special buffer (*SAP cursor cache*). Each SAP work process buffers the operations DECLARE, PREPARE, OPEN, and EXEC in its SAP cursor cache. After the work process has opened a cursor for a DECLARE operation, the same cursor can be used repeatedly until it's displaced from the SAP cursor cache after a specified time because the size of the cache is limited.

The database doesn't receive the concrete values of the WHERE clause (MANDT = 100, etc.) until the OPEN operation is used. A PREPARE operation is only necessary for the first execution of a statement, as long as that statement hasn't been displaced from the SAP cursor cache. Subsequently, the statement, which has already been prepared (parsed), can always be reaccessed with OPEN or REOPEN.

Figure 5.1 shows the SQL trace for the second run of the same report. Because the DECLARE and PREPARE operations are executed in the report's first run, our example shows only the OPEN operation.



Declare, prepare, open operations

Prepare operation

First and subsequent executions

Network problems If you've identified an SQL statement with a long runtime, you should activate the trace again for further analysis. It's useful to perform the trace at a time of high system load and again at a time of low system load. If you find that the response times for database accesses are high only at particular times, this indicates throughput problems in the network or in database access (e.g., an I/O bottleneck). For detailed information, see Chapter 2, Section 2.2.2. If, on the other hand, the response times for database access are poor in general (not only at particular times), the cause is probably an inefficient SQL statement, which should be optimized.

When evaluating database response times, remember that the processing times of SQL statements are measured on the application server. The runtime shown in the trace includes not only the time required by the database to furnish the requested data but also the time required to transfer data between the database and the application server. If there is a performance problem in network communication, the runtimes of SQL statements will increase.



Trace Comparison

You can best recognize network problems between the database and the application server by comparing traces as follows: First, execute the same SQL trace at least twice—once on the application server that is on the same computer as the database and is directly connected to the database, and once on an application server that is connected to the database via the TCP/IP network. Compare the two SQL traces. If there are significantly higher response times (greater by 50% or more) on the application server connected via the network, you have a network problem. Perform this test at a time of low system load, and repeat it several times to rule out runtime differences due to the buffer-load process on the database and application servers. This test works only when your application server is connected to the database via IPC.

5.1.3 Other Functions in the SQL Trace

Compression Using the **Summary** function of the SQL trace, you can get an overview of the most expensive SQL accesses. If you navigate via **Trace List • Compress Trace according to SQL Statements**, a list is displayed, which shows the data listed in Table 5.3 for each statement. Sort the list according to the runtimes

of the SQL statements. The SQL statements with the longest runtimes should be optimized first.

Field	Explanation
Executions	Number of executions
Redundancy	Number of redundant (identical) executions
Duration	Runtime for all executions of an SQL statement in microseconds
Object Name	Name of the table
Statement	Short form of the executed SQL statement

Table 5.3 Fields in the Compressed Summary of an SQL Trace

Inefficient programs are often characterized by the fact that they read identical data from the database several times in succession. To help you identify these identical SQL statements, choose **Trace list • Display Identical Selects** to see a list of identical selects that tells you how often each identical select was executed. By using this function in conjunction with the compressed data, you can roughly see how much of an improvement in performance can be gained by avoiding identical SQL statements via more skillful programming.

Identical selects

After this preliminary evaluation using the SQL trace, you have all of the information necessary for a more detailed analysis:

Other functions

- **Program name and transaction of the executed program**
Using the **ABAP Display** button in the trace list, you can jump directly to the code location that executes the SQL statement.
- **Table name**
The **DDIC Info** button gives a summary of the most important dictionary information for this table.
- **WHERE clause in the SQL statement.**
- **Analysis**
Detailed analysis of the SQL statement, for example, **EXPLAIN** function.

For more information on these functions, see Chapter 11.

Procedure for quality control For customer-developed ABAP programs, perform (at least) the following checks as a form of program quality control:

1. For each customer-developed ABAP program, perform an SQL trace either on the production system or on a system with a representative volume of test data.
2. From the basic trace list display, create a compressed summary to find the SQL statements with the longest runtimes: **Trace List • Compress Trace according to SQL Statements**.
3. Display a list of identical accesses to find SQL statements that are executed several times in succession: **Trace List • Display Identical Selects**.
4. Use these lists to decide whether the program should be approved or whether it needs to be improved by the responsible ABAP developer.
5. Save a copy of the lists along with the program documentation. If program performance diminishes at a later date (whether due to a modification or due to the growing data volume), perform another SQL trace, and compare it to the earlier one. Monitor performance in this way after each significant program modification.

These measures can be used not only to monitor customer-developed programs but also to regularly monitor frequently used, standard SAP transactions that are critical to performance. If the runtimes of particular SQL statements increase over time, you may need to archive the corresponding table.

5.1.4 Evaluating a Buffer Trace

Table 5.4 contains and explains the fields that are displayed in a buffer trace. The last three columns are only displayed if you click the **More Info** button. You can find more detailed explanations on SAP table buffering in Chapter 12.

Field	Explanation
hh:mm:ss.ms	Time stamp in the form <i>hour:minute:second:millisecond</i>
Duration	Runtime of an SQL statement in microseconds

Table 5.4 Fields of a Buffer Trace

Field	Explanation
Program Name	Name of the program that has triggered the buffer call
Object Name	Name of the database table or database view
Operation	Name of the function that was executed on the buffer
Records	Number of records that were read from the buffer
RC	Return code: <ul style="list-style-type: none"> ■ 0: Function executed correctly ■ 64: No further records available ■ 256: Records not available in the buffer ■ 1024: Records not available in the buffer, buffer is loaded (for parallel read accesses)
Statement	In the OPEN operation, the buffering type is indicated: <ul style="list-style-type: none"> ■ G: Generic buffering ■ P: Single-record buffering ■ R: Full buffering ■ S: Export/import buffer ■ C: CUA buffer ■ O: OTR buffer ■ E: Export/import buffering in shared memory

Table 5.4 Fields of a Buffer Trace (Cont.)

5.1.5 Evaluating a Remote Function Call Trace

Table 5.5 lists the fields displayed in an RFC trace.

Field	Explanation
hh:mm:ss.ms	Time stamp in the form <i>hour:minute:second:millisecond</i> .
Duration	RFC runtime in microseconds.
Program Name	Name of the program from which the RFC originates.
Object Name	Name of the recipient, for example, the SAP instance or computer that was called.

Table 5.5 Fields in an RFC Trace

Field	Explanation
Operation	For entry Client : RFC sent, that is, the instance on which the trace is executed is the client (sender). For entry Server : RFC received, that is, the instance on which the trace is executed is the server (recipient).
Records	Not used.
RC	Return code (for successful execution: 0 [zero]).
Statement	Additional information on the RFC, including the names of the sender and recipient; name of the RFC module; and amount of data transferred. You can view all information on the RFC by double-clicking the corresponding row in the RFC trace.

Table 5.5 Fields in an RFC Trace (Cont.)

Figure 5.1 shows an example of an RFC (OLE_FLUSH_CALL).

Detailed analysis in the RFC trace

Like the SQL trace, the RFC trace provides several detailed analysis functions:

- By double-clicking a row of the RFC trace or by clicking the **Details** button, you can obtain complete information on the RFC, including the names and IP addresses of the sender and recipient, the name of the RFC module, and the transferred data quantity.
- By clicking the **ABAP Display** button, you can view the source text of the corresponding ABAP program.

5.1.6 Evaluating an HTTP Trace

Table 5.6 contains and explains the fields that are displayed in an HTTP trace. You'll find more detailed descriptions of HTTP calls in Chapter 8.

Field	Explanation
hh:mm:ss.ms	Time stamp in the form <i>hour:minute:second:millisecond</i> .
Duration	Runtime of the HTTP call in microseconds.

Table 5.6 Fields of an HTTP Trace

Field	Explanation
Program Name	Name (logical path) of the executed HTTP call.
Object Name	Name of the recipient, for example, the SAP instance called.
Object Name	For entry Client : HTTP sent, that is, the instance on which the trace is executed is the client (sender). For entry Server : HTTP received, that is, the instance on which the trace is executed is the server (recipient).
RC	HTTP return code (200 for successful execution, 401 for failed authentication, etc.).
Statement	Additional information on the HTTP, including the names of the sender and recipient, name of the HTTP call, and amount of data transferred. You can view all information on the HTTP call by double-clicking the corresponding row in the HTTP trace.

Table 5.6 Fields of an HTTP Trace (Cont.)

5.1.7 Evaluating an Enqueue Trace

Table 5.7 contains and explains the fields that are displayed in an enqueue trace. The last three columns are only displayed if you click the **More Info** button. You can find more detailed explanations of the SAP enqueue concept in Chapter 10.

Field	Explanation
hh:mm:ss.ms	Time stamp in the form <i>hour:minute:second:millisecond</i> .
Duration	Runtime of the enqueue/dequeue statement in microseconds.
Program Name	Name of the program from which the enqueue statement originates.
Object Name	Name of the enqueue object. To see details of the corresponding object, call the ABAP Data Dictionary (using Transaction SE12) and enter the object name in the field under Lock Objects . To view the properties of the objects, select Display .

Table 5.7 Fields in an Enqueue Trace

Field	Explanation
Operation	<p>Operation ENQUEUE: Enqueue(s) set.</p> <p>Operation DEQUEUE: Individual enqueues are released.</p> <p>Operation DEQ ALL with specific object entry in the Object column: All enqueues for the object in question are released.</p> <p>Operation DEQ ALL with no object entry in the Object column (entry {{{{{{{{{{{}}}}}}}}): All enqueues for the transaction in question are released (end of a transaction).</p> <p>Operation ENQPERM: At the end of the dialog part of a transaction, enqueues are passed on to update management.</p>
Records	Number of enqueues that are set or released.
RC	Return code (for successful execution: 0 [zero]).
Statement	<p>More detailed information on the enqueue:</p> <p>Entries Excl or Shared: Exclusive or “shared” locks; name of locked unit, (e.g., “MARC 900SD000002”: Material SD000002 is locked in table MARC, in client 900). The entries in this row correspond with the properties of the enqueue object defined in the ABAP Data Dictionary.</p>

Table 5.7 Fields in an Enqueue Trace (Cont.)

5.2 Performance Analysis with ABAP Trace (Runtime Analysis)

You should use an ABAP trace, also referred to as ABAP runtime analysis, when the runtime of the programs to be analyzed consists mainly of CPU time. During an ABAP trace, not only the runtime of database accesses (SELECT, EXEC SQL, etc.) is measured but also the time required for individual modularization units (MODULE, PERFORM, CALL FUNCTION, SUBMIT, etc.), internal table operations (APPEND, COLLECT, SORT, READ TABLE), and other ABAP statements.

User interfaces are available in SAP GUI (Transaction SAT) and in the Eclipse-based ABAP development environment—the underlying tracing technology is identical.

5.2.1 Activating an ABAP Trace

You activate an ABAP trace for an SAP GUI transaction as follows:

For SAP GUI transactions

1. Use the following menu path to access the initial screen of the ABAP trace: **System • Utilities • Runtime Analysis • Execute**. Alternatively, select Transaction SAT.
2. In the upper part of the screen, under **Measure**, enter a transaction, program name, or function module, and then select **Execute** to start the measuring process. You can also click the **In Parallel Mode** button to access a process list, where you can activate the ABAP trace for the currently active work process.
3. The system starts to measure the runtime and creates a file with the resulting measurement data.
4. When you want to return to the initial runtime analysis screen, simply exit the transaction, function module, or program as normal, or start the runtime analysis again.

If a runtime measurement is to be performed for a transaction that is started via a web browser or an external system, follow these steps:

For web browser or external systems

1. Start the ABAP runtime analysis transaction as previously described.
2. Select the **Schedule** button under **User/Service**. The **Overview of Scheduled Measurements** screen is displayed.
3. Select **Schedule Measurement**, and enter the following data in the dialog box:
 - User, client, and server
 - Process type, for example, dialog, background processing, or HTTP
 - Object type (e.g., transaction, report, or URL) and object name
 - Maximum number of scheduled measurements
 - Note here that an end-to-end transaction may involve multiple measurements. To ensure that all parts are recorded, you should specify multiple measurements here.
 - Expiry date, time, and description
4. Close the window with the confirmation **Schedule Measurement**. The measurement is then scheduled and appears in the overview list.

5. Start the transaction to be analyzed. While the transaction is running, the columns **Started** and **Status** indicate whether measurement files are created.
6. When the transaction has ended, you evaluate the measurement files as described in the next section. If a measurement is still open because, for example, the web browser wasn't properly closed, you can stop a measurement by clicking **Stop Active Measurement** (on the **Evaluate** tab, as discussed later).

For BSP and Web Dynpro ABAP applications

To start a runtime analysis for Business Server Pages (BSPs) and Web Dynpro ABAP applications, call Transaction SICF (Service Maintenance). Select the service to analyze in the navigation tree, and activate the runtime analysis via **Edit • Runtime Analysis • Activate**.

As of version 6.40, you can restrict the procedure to a user name during activation and specify a variant to use for recording. Here, it's also possible to specify the measurement accuracy.



Information on Measurement Accuracy

In the initial screen, a traffic light on the **Measure** tab indicates whether the runtime measurement can determine reliable times. If the traffic light is red, it means this isn't possible because of nonsynchronized CPUs in a computer with multiple processors. Therefore, you must perform the time measurements with low accuracy (**Settings • Measurement Accuracy • Low**). You can find further operating system-specific information in SAP Notes 20097 and 87447. If you want to perform a runtime analysis in parallel mode, refer to SAP Note 729520.

When you're activating an ABAP trace, filter functions enable you to restrict the trace to a particular function module or group of ABAP statements or to adjust an aggregation. Section 5.2.3 explains how to use these options.

Activating traces in Eclipse

To activate a trace in an Eclipse-based ABAP development environment, select the **ABAP Profiling** perspective in Eclipse first. This perspective contains the **ABAP Trace Requests** and **ABAP Traces** views. In the **ABAP Trace Requests** view, select a system and choose **Create Trace Requests** in the context menu. You can now define in a dialog box which actions are supposed

to be recorded. The options are the same as described for the SAP GUI interface.

You can activate the runtime analysis either in SAP GUI or in Eclipse. The analyses recorded can be evaluated with both interfaces.

When you activate an ABAP trace, bear the following points in mind (which are similar to those that apply to an SQL trace):

- Because you're normally not interested in recording the buffer load process in the trace, you should first execute a program once without activating the trace, thereby allowing the buffers to be loaded (i.e., the SAP and database buffers). Then run the program again with the ABAP trace activated, and use the results of this trace for evaluation.
- Perform the trace at a time of low system load to ensure that the measured times aren't influenced by a temporary system overload (e.g., a CPU overload).
- To enable runtime analysis, the system requires the SAP profile parameters `abap/atrapath` and `abap/atrasizequota`. These parameters are set when the system is installed. The profile parameter `abap/atrapath` indicates in which directory the trace files are written. You can restrict the maximum size of all ABAP trace files by using the parameter `abap/atrasizequota`. The trace files are deleted after 30 days if you don't change the deletion date (**Evaluate** tab).

5.2.2 Evaluating an ABAP Trace

To display the results of an analysis, select the desired file under the **Evaluate** tab in Transaction SAT. In Eclipse, select the file in the **ABAP Traces** view.

The runtime analysis presents different views of the measurement results in list form or as graphics.

- Click the **Hit List** button to view a list that displays the execution time for each statement. This list is sorted in decreasing order of net times (see Figure 5.2).
- Via the **Hierarchy** button, you obtain a presentation of the chronological sequence of the recorded parts of the program.

Rules when activating traces

5

Evaluation views

- Other buttons will display specific evaluations that, for example, categorize database tables or modularization units.

If you've generated an ABAP trace, first display the **Hit List**. Sort the hit list according to net time to get an overview of statements with the highest net runtimes. If the ABAP trace was recorded in aggregate form, only the **Hit List** is available as an evaluation option.

Hits	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Statement/Event	Program Called	Calling Program
2	88,497	88,497	31.54	31.83	Wait for RFC	SAPLOLEA	SAPLOLEA
1	26,427	26,427	9.42	9.50	DB: Open VBAK	ZTS_USEALV	ZTS_USEALV
2	114,457	25,960	40.79	9.34	Rfc OLE_FLUSH_CALL	SAPLOLEA	SAPLOLEA
9	6,513	5,026	2.32	1.81	Rfc DP_PUT_CLIENT_TABLE45A	SAPLCNDP	SAPLCNDP
342	9,883	4,990	3.52	1.79	Perform CONVERT_TO_VARIANT	SAPLOLEA	SAPLOLEA
53	7,205	4,502	2.57	1.62	Call M. {O:6*CL_SALV_FUNCTIONS}...	CL_SALV_FUNCTIONS=...	CL_SALV_FUNCTIONS=====CP
342	4,297	4,297	1.53	1.55	Perform COLLECT_VARS	SAPLOLEA	SAPLOLEA
1	216,520	3,793	77.16	1.36	Call Screen 0500	SAPLSLVC_FULLSCREEN	SAPLSLVC_FULLSCREEN
242	3,705	3,705	1.32	1.33	Perform COLLECT_VARS	SAPLOLEA	SAPLOLEA
1	280,546	3,668	99.98	1.32	Submit Report ZTS_USEALV	SAPLS_ABAP_TRACE_D...	SAPLS_ABAP_TRACE_DATA
1,495	2,427	2,427	0.86	0.87	Call M. {O:7*CL_SALV_FUNCTIONS}...	CL_SALV_FUNCTION=...	CL_SALV_FUNCTIONS=====CP
2	5,318	2,369	1.90	0.85	Call M. LCL_MYXMLPARSER=>ABAP...	CL_GUI_DATAMANAGER...	CL_GUI_DATAMANAGER=====CP
62	6,843	2,335	2.44	0.84	Call Function AC_CALL_METHOD	SAPLOLEA	SAPLOLEA

Figure 5.2 ABAP Runtime Analysis: Aggregated Hit List of Call Locations Sorted according to Net Time

Gross and net time The runtime analysis establishes the gross and/or net times of individual program calls in microseconds (μ s). Gross time is the total time required for the call. This includes the times of all modularization units and ABAP statements in this call. The net time is the gross time minus the time required for the called modularization units (MODULE, PERFORM, CALL FUNCTION, CALL SCREEN, CALL TRANSACTION, CALL DIALOG, SUBMIT) and separately specified ABAP statements if logging was activated for them. For “elementary” statements such as APPEND or SORT, the gross time is the same as the net time. If the gross and net times for a call differ, this call contains other calls or modularization units. For example, if a subroutine shows a gross time of 100,000 μ s

and a net time of 80,000 μ s, this means 80,000 μ s are used for calling the routine itself, and 20,000 μ s are assigned to other statements in the routine, which were logged separately. The Eclipse-based evaluation interface uses the terms **Total Time** and **Own Time** as synonyms.

Figure 5.2 shows the analysis of a sample program. In this program, data records were read from tables VBAK and VBAP and were displayed via the ABAP List Viewer (ALV; SAP class CL_SALV_TABLE). The **Hit List** contains the ALV display function (program SAPLOLEA) with 39% net time and access to table VBAK at the fore with 8%.

Resource Requirement Due to Runtime Analysis

The runtime analysis involves a lot of work; generating the analysis can be as much as double the runtime of a program (compared to a program run without runtime analysis activated). The runtime analysis takes this into account and displays correspondingly adjusted runtimes in the lists. However, if you look at the statistical record created while the runtime analysis was active, you'll see that it's clearly distorted when compared to a program run without runtime analysis. In contrast, letting a performance trace run simultaneously doesn't involve much extra load—less than 5%.

5.2.3 Using Variations

When you execute the ABAP trace function, you can use variants to adjust how the trace is carried out. We highly recommend the use of variants. It's particularly advisable to try out these options when analyzing a complex program because data quantities of several megabytes can be generated very quickly, much of which is often completely irrelevant to the analysis. Using variants, you can determine more precisely what you want to analyze.

You can see the currently selected variant in the initial screen of the runtime analysis under **Measurement**. The **DEFAULT** variant is already set in the system. You can save your personal settings as your own variant.

The **Display Variants** or **Change Variants** button takes you to the screen **Settings** where you can enter the settings for a variant:

■ Aggregation type

On the **Duration and Type** tab, you can, among other things, set the type of aggregation. Aggregation is always relevant when a statement is called numerous times in a program, for example, an SQL statement within a loop. If aggregation isn't activated, an entry will be written to the trace measurement file each time the SQL statement is called. If aggregation is activated, only one entry will be recorded, in which the runtimes for each execution are added together. On the **Duration and Type** tab, you're offered two options:

– No Aggregation

An entry is written to the measurement file each time the statement is called.

– Aggregation According to Call Location

The runtimes for individual executions of a statement are added together in one entry. However, if a statement appears several times in a program at several locations in the program text, one entry is written for each time it appears in the program text. This option is activated in the default variant.

In general, activating aggregation dramatically reduces the size of the measurement file, and, in many cases, an analysis of long program sequences is only possible with aggregation. When you use aggregation, some evaluation functions (e.g., the hierarchy list) are lost and therefore no longer available for evaluation.

■ Trace on remote systems

Moreover, you specify on the **Duration and Type** tab whether RFCs and update calls are supposed to be recorded as well. If you activate the corresponding option, this information is forwarded to the corresponding processes. To be able to record an RFC in an external system, that system must accept external trace calls. You configure this using the `rstr/accept_remote_trace` parameter, which you must set to `true`. The trace file is then written to the external system. If you use the end-to-end (E2E) trace in SAP Solution Manager, this function is activated automatically, and the E2E trace automatically reads all trace files.

■ Filter operations

The settings on the **Statements** tab determine which operations will be monitored in the runtime analysis.



Tip

If you want to analyze operations on internal tables such as `Append`, `Loop`, or `Sort`, activate the **Read Operations** and **Change Operations** checkboxes under **Int. Tables**. These settings aren't activated in the default variant.

■ Filter program parts

On the **Program Parts** tab, you can set which parts of the program to analyze. If you select the **Restrict to Program Parts** option, you can restrict the trace to selected classes, function groups, and programs, and their subcomponents, methods, function modules, and form routines.



Procedure for More Complex Programs

How should you carry out an analysis of more complex programs? We recommend that you first carry out an analysis of the entire program with aggregation according to call location and without analyzing operations on internal tables (default variant settings). The objective of this analysis is to find the modularization units with the highest runtimes. After this initial analysis, sort the hit list according to net times and identify the modularization units or statements with high runtime.

If you can't deduce recommendations for optimizing the program from this first analysis, perform a more detailed analysis, setting variants to limit the analysis to these modularization units. Simultaneously, activate the trace for operations from internal tables and deactivate the aggregation.

5.2.4 Using Timeline Views

The Eclipse-based interface provides an exclusive view, which displays the program flow on a timeline. This view indicates the number of nesting levels of the program and the calls with a high net time.

To use the timeline view, follow these steps:

1. Choose the **ABAP Profiling** perspective and **ABAP Traces** view in the Eclipse-based ABAP development environment. Select a file, and open

the **Call Timeline** view. The system displays the time line of the runtime analysis (see Figure 5.3).

2. You can use navigation tools such as zoom settings and the preview to navigate in the lower part of the view. Calls with a high net time are displayed as long horizontal bars in the view.
3. Select one of the critical calls. A window shows the following information about the call:
 - Operation
 - Calling program and program that is called
 - Gross and net time
4. Using the functions in the context menu, you can navigate to other views, such as **Hit List** and **Call Hierarchy**.
5. To improve the display of the overview, define your own coloring schema. For this purpose, select **Edit Coloring Schemas** from the context menu. Assign specific colors to the program groups. For example, assign programs with the “Z” schema to the color “Red” to identify program parts that begin with a prefix. Figure 5.3 shows a colored timeline view. You can store coloring schemas as local files and upload them to the system.

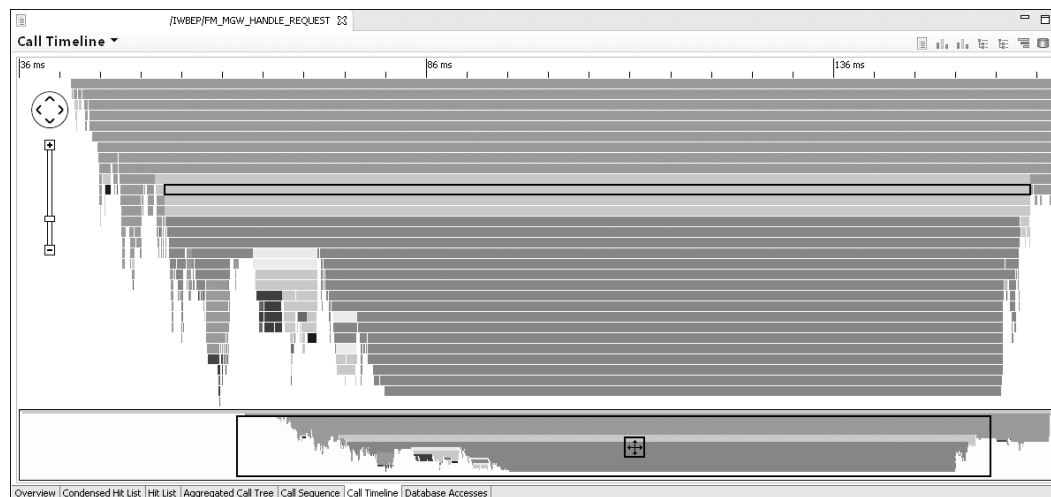


Figure 5.3 ABAP Runtime Analysis: Timeline View

5.3 Analyzing Memory Usage with ABAP Debugger and in the Memory Inspector

In addition to expensive SQL statements, one of the most important causes of performance problems is internal tables with many entries. Large internal tables consume massive amounts of memory and CPU, for example, during copy, sort, or search operations.

You can use the ABAP debugger to create an overview of all internal tables of a program. ABAP debugger is a tool for performing functional troubleshooting in programs. You can find more detailed descriptions of the debugger in SAP literature under ABAP programming. You'll find information on main memory usage both in the classic and the new debugger. You can set the debugger in the ABAP Editor (Transaction SE38) via **Utilities • Settings • Debugging**.

Performance analysis using the ABAP debugger isn't a standard procedure and is best performed by an ABAP developer.

Rules When Debugging

Take the following advice into account when working with the ABAP debugger. During the debugging process, the ABAP program may terminate and display the error message **Invalid interruption of a database selection**, or the system may automatically trigger a database commit. In either case, an *SAP Logical Unit of Work (LUW)* has been interrupted, and this may lead to inconsistencies in the application tables. Therefore, you should only debug on a test system or in the presence of someone who is very familiar with the program being analyzed and who can manually correct inconsistencies in the database tables if necessary. See “Debugging Programs in the Production Client” in SAP Online Help for the ABAP debugger.

You perform a performance analysis with the debugger as follows:

1. Start the program to be analyzed. Then open a second session. Here you can monitor the program to be analyzed in the work process overview (Transaction SM50). Enter the debugger from the work process overview by selecting the **Debugging** function. By using the debugger several times in succession, you can identify the parts of the program that cause



Steps for performance analysis

high CPU consumption. Often, these sections consist of LOOP ... ENLOOP statements that affect large internal tables.

2. To display the current memory requirements (in the “classic” debugger), select **Goto • Other Screens • Memory Use**.
3. Check for cases of unnecessary memory consumption that may have been caused by a nonoptimal program or inefficient use of a program. As a guideline, bear in mind that a program being used by several users in dialog mode should not allocate more than 100 MB.
4. As of SAP NetWeaver AS ABAP 6.20, you can use the classic debugger to create a list of program objects located in the memory by selecting **Goto • Status Display • Memory Use**. Under **Memory Consumption**, the **Ranking Lists** tab contains a list of objects and their memory consumption.

In SAP NetWeaver AS ABAP versions 4.6 und 6.10, you can obtain a memory consumption list by choosing **Goto • System • System Areas**. Enter “ITAB-TOP25” in the **Area** field. This way, you’ll obtain a list of the 25 largest internal tables.

5. In the “new” debugger, you first display the memory analysis tool by clicking the button for the new tool and then selecting **Memory Analysis** from the tools on offer in the **Memory Management** folder. The initial screen then displays how much memory is allocated or used by the analyzed internal session. Click the **Memory Objects** button to go to the list of the largest memory objects, which can be the internal tables, objects, anonymous data objects, or strings.

Memory Inspector

Moreover, you can create and then analyze a memory extract, that is, an overview of the objects that occupy memory space. You can create a memory extract in any transaction by selecting **System • Utilities • Memory Analysis • Create Memory Extract** or simply enter function code “/HMUSA”. The third option is to create a memory extract from program coding. Refer to SAP Help for a description of the system class CL_ABAP_MEMORY_UTILITIES.

Evaluating the memory extract

To evaluate the memory extract, start the Memory Inspector by selecting **System • Utilities • Memory Analysis • Compare Memory Extracts** in any transaction or via Transaction S_MEMORY_INSPECTOR. The Memory Inspector lists all memory extracts in the upper part of the screen. In the lower part of the screen, you can find details about the individual memory extract. Here, a distinction is made among the object types, programs, classes, dynamic memory request of a class, table bodies, strings, and types

of anonymous data objects. You’re provided with different ranking lists, according to which you can sort the objects. For each memory object, you’re provided with the values of bound allocated, bound used, referenced allocated, and referenced used memories. You can find a detailed description of the ranking lists and the displayed values in SAP Help.

Using the Memory Inspector

The Memory Inspector is particularly useful for examining transactions over a long period of time, as is the case in a customer interaction center. Here, users frequently enter a transaction at the beginning of their workday and exit it when they go home. In these “long-term” transactions, data often remains, and therefore memory consumption continuously increases.

Figure 5.4 shows an example of a memory extract. The dominator tree shows the hierarchical program structure and the memory used by the program parts. With a size of 494 MB, table LT_MEM is conspicuous. The next largest object is the CL_GUI_ALV_GRID class with a size of 250 KB. Below this class, 130 KB are used by table MT_DATA.

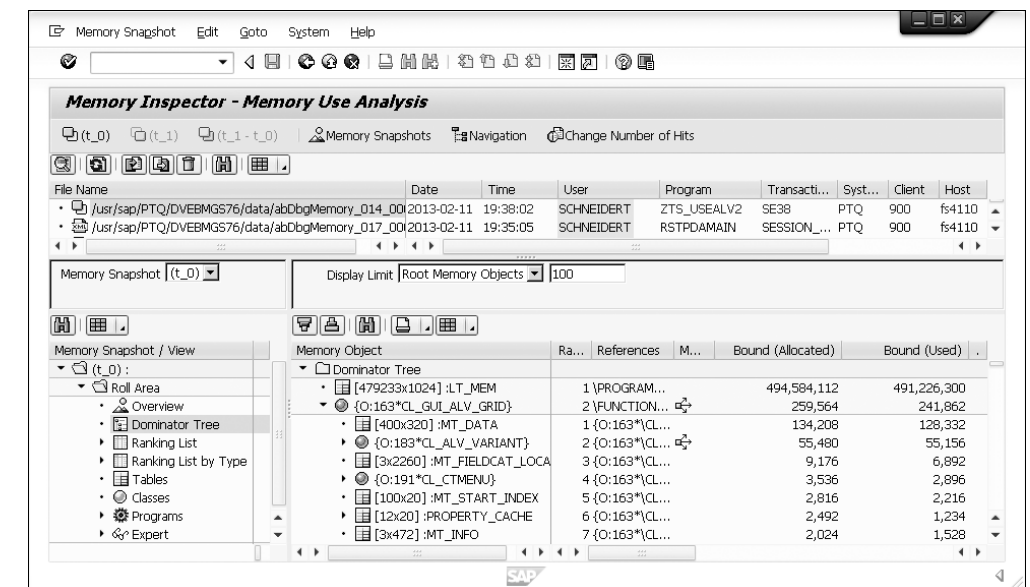


Figure 5.4 Memory Inspector: Dominator Tree

5.4 Code Inspector

The Code Inspector tool checks ABAP programs and other repository objects statically for problems. Its meaning for the quality analysis of ABAP programs goes far beyond the meaning for performance analysis, to which the following description is restricted. The Code Inspector performs a static analysis, that is, the code doesn't need to be executed and is therefore independent of test or production data. It has been generally available as of SAP Basis version 6.10.

You call the Code Inspector from the development tools of the ABAP Workbench for programs, function modules, or ABAP classes (Transactions SE38, SE37, or SE24) using the menu path **Program/Function Module/Class • Check • Code Inspector** or directly using Transaction SCII.

Performance checks in the Code Inspector

Table 5.8 summarizes which checks are performed in the **Performance** category and where you can find further information on the checks. The checks reveal standard errors and problems. They can't indicate how seriously they will affect the performance. For this purpose, you require the runtime checks described previously. The verification of these checks should nevertheless be part of the quality control of ABAP programs.

Check	Description
Analysis of the WHERE clause	Checks that the SQL statements include a WHERE clause and that they have index support. Buffered tables, joins, and views aren't covered by this check. For more information, see Chapter 11, Section 11.2.
Analysis of table buffer accesses	Checks whether SQL statements to buffered tables will access the database and pass the buffer. For more information, see Chapter 12, Section 12.1.2.
SELECT statements with CHECK	Checks whether SELECT/ENDSELECT loops include a CHECK statement. This statement can often be integrated in the WHERE clause and thus reduce the read data quantity from the start. For detailed information, see Chapter 11, Section 11.2.

Table 5.8 Performance Checks in the Code Inspector

Check	Description
SQL statements in loops	Checks whether SQL statements are in loops. These result in an increased communication effort and can possibly be replaced by bundled accesses. For more information on this, see Chapter 11, Section 11.2.
Nested loops	Checks the system for nested loops where nonlinear runtime behavior can occur.
Copying large data objects	Sends alerts on high copy costs for large data objects, for instance, nested internal tables. However, this check only refers to structures with a width of more than 1,000 bytes; it can't consider how long a table will be at runtime.
Inefficient operations on internal tables	Sends alerts on inefficient read accesses to internal tables.
Inefficient parameter transfers	Examines whether there is a better way to transfer parameters when you call a form, function module, method, or event.
EXIT or no ABAP commands in SELECT/ENDSELECT loop	Checks whether SELECT/ENDSELECT loops include an EXIT statement or no ABAP coding at all. With this type of code, you often check the existence of records in a database table, which can be designed to be more efficient. For detailed information, see Chapter 11, Section 11.2.
Instance generation of Business Add-Ins (BADIs)	As of SAP Basis version 7.0, you should use the statement GET BADI instead of the method call CALL METHOD cl_exithandler=>get_instance for performance reasons.
Check of table properties	Examines the technical settings such as transportability, buffering, and indices of database tables.

Table 5.8 Performance Checks in the Code Inspector (Cont.)



Check Variants in the Performance Environment

The Code Inspector offers *check variants* that bundle checks for specific areas. For performance analysis, the following variants are available:

- PERFORMANCE_CHECKLIST
Minimum checklist with performance checks.
- PERFORMANCE_CHECKLIST_HDB
Checklist with SAP HANA-specific performance checks.
- PERFORMANCE_CHECKLIST_XL
Advanced checklist with performance checks.
- PERFORMANCE_DB
Checklist of database-specific performance checks.

The following checklist can be used in the context of the migration from a traditional database to SAP HANA and comprise function checks (e.g., for code that might include programming on database-specific functions) and performance checks (code that might lead to performance issues):

- FUNCTIONAL_DB and FUNCTIONAL_DB_ADDITION
Classic and advanced checklist for the migration to SAP HANA.

5.5 Tips and Tricks for High-Performance ABAP Programs

Frequent performance problems

In the SAP online help for the ABAP programming environment and in the ABAP runtime environment (Transaction SAT), the “Tips and Tricks” section provides detailed information on high-performance ABAP programming, which you can use to get familiar with the sample code introduced there.

There are three common programming errors that cause large memory or CPU requirements for programs:

- **Missing REFRESH or FREE statements**
The ABAP statements REFRESH and FREE delete internal tables and release the memory that was allocated to them. If these statements are missing, memory resources may be unnecessarily tied up, and the operations being executed (READ or LOOP) will require an unnecessarily large amount of time.



What Effect Do CLEAR, REFRESH, FREE, and DELETE Have?

The ABAP statements CLEAR, REFRESH, FREE, and DELETE have the following effect on the allocated memory:

- A CLEAR or REFRESH statement causes the release of the table content and—if available—of the indexes. The header information of the table remains unchanged. Note that the CLEAR statement only deletes the header row in internal tables with a header row. Use this statement if the table is to be used again as the program proceeds.
- A FREE statement causes the complete release of the table content and—if available—of the indexes. Use this statement if the table isn't to be used again as the program proceeds.
- A DELETE statement in an internal table causes *no* release of memory! If you reduce an internal table with a large number of entries to very few entries using DELETE statements, the memory still remains allocated. When you copy such a table, the “memory image” is copied; that is, the copied table has the same size as the original table. To solve this problem, build a new small table instead of thinning out a large table using DELETE.

- **Inefficient reading in larger internal tables**

The ABAP statement READ TABLE ... WITH KEY ... enables you to search internal tables. If you use this statement by itself for a standard table, the search is sequential. For large tables, this is a time-consuming process. You can significantly improve search performance by adding the clause ... BINARY SEARCH, thereby specifying a binary search. However, the table must be sorted (see ABAP Help for the statement READ TABLE).

You can optimize the performance of operations on large tables by using sorted tables (SORTED TABLE) or hash tables (HASHED TABLE). If a READ statement is executed on a sorted table, the ABAP processor automatically performs a binary search. It's important that the key fields used for the search correspond to the sort criteria for the table. For a sorted table, the search effort increases logarithmically with the size of the table. For hash tables, constant access costs exist if the ABAP statement READ TABLE ... WITH TABLE KEY is used. However, efficient access to hash tables is only possible if you enter the complete key.

■ Nested loops

Nested loops are frequently used for processing dependent tables (e.g., header and position data):

```
LOOP AT HEADER INTO WA_HEADER.
  LOOP AT POSITION INTO WA_POSITION
    WHERE KEY = WA_HEADER-KEY.
    "Processing ...
  ENDLOOP.
ENDLOOP.
```

If `HEADER` and `POSITION` are standard tables in this case, then for each entry in the `HEADER` table, the ABAP processor loops across all entries in the `POSITION` table and checks if the `WHERE` clause is fulfilled for all entries. This is especially time-consuming if the tables `HEADER` and `POSITION` contain many entries. However, if you use `SORTED` tables, the ABAP processor determines the data to be processed by performing a binary search and only loops across those areas that fulfill the `WHERE` clause. Sorting is only useful, however, if the `WHERE` clause contains the first fields of the sort key.

As an alternative, you can use index operations that are much less time-consuming. To use index operations, it's necessary to sort the internal tables `HEADER` and `POSITION` by the `KEY` field as follows:

```
I = 1.
LOOP AT HEADER INTO WA_HEADER.
  LOOP AT POSITION INTO WA_POSITION FROM I.
    IF WA_POSITION-KEY <> WA_HEADER-KEY.
      I = SY-TABIX.
      EXIT.
    ENDIF.
    " ...
  ENDLOOP.
ENDLOOP.
```

Table 5.9 summarizes the most important performance properties of standard tables, sorted tables, and hash tables in ABAP. It provides information on how access costs increase when the size of the table increases (see also the “Scalability of a Program” section). Based on these properties, you can determine which table type is ideal for your specific application.

Operation	Standard Table	Sorted Table	Hash Table
Populate (mass operation)			
APPEND LINES ...	Depending on position and size due to movement costs	Depending on position and size due to movement costs (check required)	Depending on position and size due to movement costs
INSERT LINES ...	Depending on position and size due to movement costs	Depending on position and size due to movement costs (check required)	Depending on position and size due to movement costs
Populate (single record)			
APPEND	Constant	Constant (higher than standard as check required)	–
INSERT ... INTO ... INDEX	Depending on position and size due to movement costs	Depending on position and size due to movement costs (higher than standard as check required)	–
INSERT ... INTO ... TABLE	Constant	Logarithmic	Constant (higher than standard due to hash management)
Read (areas)			
LOOP ... ENDLOOP (all rows)	Linear	Linear	Linear

Table 5.9 Most Important Performance Properties of Standard Tables, Sorted Tables, and Hash Tables in ABAP

Operation	Standard Table	Sorted Table	Hash Table
LOOP ... WHERE END-LOOP (complete key)	Linear	Logarithmic	Constant
LOOP ... WHERE END-LOOP (incomplete key with initial section)	Linear	Logarithmic	Linear
LOOP ... ENDLOOP WHERE (incomplete key without initial section)	Linear	Linear	Linear
LOOP ... FROM ... TO	Constant	Constant	–
Read (individual rows)			
READ ... INDEX	Constant	Constant	–
READ ... WITH KEY (complete key)	Linear, logarithmic for binary search	Logarithmic	Constant
LOOP ... WHERE END-LOOP (incomplete key with initial section)	Linear, logarithmic for binary search	Logarithmic	Linear
LOOP ... WHERE END-LOOP (incomplete key without initial section)	Linear	Linear	Linear
Change (analogously, delete)			
MODIFY ... TRANSPORTING ... WHERE (complete key)	Linear	Logarithmic	Constant

Table 5.9 Most Important Performance Properties of Standard Tables, Sorted Tables, and Hash Tables in ABAP (Cont.)

Operation	Standard Table	Sorted Table	Hash Table
MODIFY ... TRANSPORTING ... WHERE (incomplete key with initial section)	Linear	Logarithmic	Linear
MODIFY ... TRANSPORTING ... WHERE (incomplete key without initial section)	Linear	Logarithmic	Linear
MODIFY ... FROM <wa> (index access)	Constant	Constant	–
MODIFY TABLE ... FROM <wa>	Linear	Logarithmic	Constant
COLLECT (table compression)	Linear	Logarithmic	Constant
SORT	Runtime intensive, depending on size and sorting	(already sorted)	Runtime intensive, depending on size and sorting

Table 5.9 Most Important Performance Properties of Standard Tables, Sorted Tables, and Hash Tables in ABAP (Cont.)

Scalability of a Program

Scalability of a program refers to the dependency of a program's runtime from the data quantity. Many operations depend linearly on the data quantity ($t = O(n)$); that is, the runtime increases linearly to the data quantity. Examples include the database selections in large tables without or with inappropriate index support and loops via internal tables in the program. Linear scalability is acceptable for the processing of medium data quantities. If they can't be avoided in programs that are supposed to process large data quantities, you must consider parallelization.



Of course, constant runtimes ($t = O(1)$) or a logarithmic dependency ($t = O(\log n)$) is better for performance than a linear scalability. Logarithmic dependencies occur, for example, for database selections in large tables with optimal index support or for read operations in internal tables with binary search. Because the logarithm function increases only very slowly, in real life, you don't need to differentiate between constant and logarithmically increasing runtimes.

Quadratic dependencies ($t = O(n \times n)$) and anything beyond are unacceptable for the processing of medium and large data quantities. However, problems with quadratic dependencies through intelligent programming can usually be traced back to dependencies of $t = O(n \times \log n)$. An example is the comparison of two tables that both grow with a factor of n . A comparison of the unsorted tables would result in a quadratic dependency, a comparison with sorted tables in dependency $t = O(n \times \log n)$. Because the logarithm function increases only very slowly, in real life, you don't need to differentiate between an increase of $t = O(n \times \log n)$ and a linear increase.

5.6 Summary

Performance trace This chapter discussed monitors that enable detailed analysis of individual ABAP programs. *SQL trace* is the recommended tool for analyzing SQL statements in ABAP programs. Evaluating the trace enables you to identify network problems or throughput bottlenecks in the database. You'll find further information on optimizing SQL statements in Chapter 11.

You use RFC trace to analyze the performance of sent and received RFCs. As of SAP NetWeaver AS ABAP 7.10, you also have an HTTP trace at hand to record HTTP statements. For more information, see Chapter 7 and Chapter 8.

The enqueue trace is a means for selecting analyses of lock operations (enqueue/dequeue operations). For detailed information, see Chapter 10.

You trace inefficient table buffering with the buffer trace. For detailed information, see Chapter 12.

ABAP trace and ABAP debugger For high CPU consumption problems, use an *ABAP trace*. In contrast to an SQL trace, an ABAP trace enables time measurements for operations on internal tables (LOOP, READ, SORT, etc.).

As an alternative, you can monitor CPU-consuming programs using the ABAP debugger, which you can call from the work process overview. However, only developers should perform this analysis.

You should examine ABAP programs proactively using the Code Inspector, which implements static checks of the program and sends alerts on standard performance errors and problems. **Code Inspector**



Important Concepts

After studying this chapter, you should be familiar with the following concepts:

- Performance trace: SQL trace, buffer trace, RFC trace, HTTP trace, enqueue trace
- ABAP trace
- ABAP debugger
- Code Inspector
- Internal tables and their performance attributes

5.7 Questions

Appendix C, Section C.1, provides the answers to these questions.

1. What do you have to consider when you perform an SQL trace?
 - a) There is only one trace file in each SAP system. Therefore, only one SQL trace can be created per SAP system.
 - b) The user whose actions are being traced should not run multiple programs concurrently.
 - c) You should perform the SQL trace on a second execution of a program because the relevant buffers will already have been loaded.
 - d) SQL traces are useful on the database server but not on application servers, which yield inexact results due to network times.
2. When should you perform an ABAP trace?
 - a) If a problem occurs with the table buffer
 - b) For programs with high CPU requirements
 - c) For analyzing I/O problems on hard drives

Contents

Preface and Acknowledgements	19
Introduction	21

1 Performance Management of an SAP Solution 35

1.1 SAP Solution Architecture	36
1.1.1 SAP Solutions and SAP Components	36
1.1.2 Application Scenarios for SAP HANA	39
1.1.3 Client/Server Architecture	43
1.2 Monitoring and Optimization Plan for an SAP Solution	52
1.2.1 Requirements of a Monitoring and Optimization Plan	52
1.2.2 Service Level Management	56
1.2.3 Plan for Continuous Performance Optimization	62
1.2.4 Tools and Methods for the Monitoring and Optimization Plan	68
1.2.5 SAP Solution Manager	71
1.3 Summary	73

2 Analysis of Hardware, Database, and ABAP Application Server 77

2.1 Basic Terms	78
2.2 Hardware Monitoring	79
2.2.1 Analysis of a Hardware Bottleneck (CPU and Main Memory)	81
2.2.2 Identifying Read/Write (Input/Output) Problems	88
2.2.3 Parameter Changes and Network Check	89

2.3 Database Monitoring	91
2.3.1 Performance Monitor in the Database Administration Cockpit	92
2.3.2 Analyzing the Database Main Memory	94
2.3.3 Identifying Expensive SQL Statements	96
2.3.4 Identifying Read/Write (Input/Output) Problems	103
2.3.5 Other Database Checks	105
2.4 Analyzing SAP Memory Configuration	113
2.4.1 Analyzing SAP Buffers	114
2.4.2 Analyzing SAP Extended Memory, SAP Heap Memory, and SAP Roll Memory	116
2.4.3 Displaying Allocated Memory	118
2.4.4 Other Monitors for Detailed Analysis	120
2.5 Analyzing SAP Work Processes	122
2.5.1 Work Process Overview Fields	122
2.5.2 Analyzing Work Processes	126
2.5.3 Monitoring the Dispatcher Queue	129
2.6 Analysis of the Internet Communication Manager	131
2.7 Summary	132
2.8 Questions	133
3 Workload Analysis	135
<hr/>	
3.1 Basics of Workload Analysis and Runtime Analysis	136
3.2 Workload Monitor	138
3.2.1 Working with the Workload Monitor	139
3.2.2 Technical Settings for the Workload Monitor	141
3.3 Workload Analysis	142
3.3.1 Transaction Step Cycle	142
3.3.2 Other Time Components	145
3.3.3 Interpreting Response Times	147
3.3.4 Activity, Throughput, and Load	149

3.4 Performing Workload Analyses	151
3.4.1 Analyzing General Performance Problems	151
3.4.2 Analyzing Specific Performance Problems	158
3.5 Single-Record Statistics	161
3.6 End-to-End Workload Analysis	166
3.6.1 Basics of End-to-End Workload Analysis	166
3.6.2 Central Workload Monitor	167
3.6.3 End-to-End Workload Monitor and Runtime Analysis in SAP Solution Manager	172
3.6.4 Central Single-Record Statistics	172
3.6.5 End-to-End Runtime Analysis in SAP Solution Manager	176
3.7 Summary	180
3.8 Questions	182
4 Hardware Sizing and Component Distribution	185
<hr/>	
4.1 Initial Hardware Sizing	186
4.1.1 Overview of the Project for Initial Sizing	187
4.1.2 Sizing Methods	192
4.1.3 Performing a Sizing Project in Detail	197
4.1.4 SAP Standard Application Benchmarks	200
4.2 Sizing to Deal with Increased Workload, Change of Release, or Migration	206
4.2.1 A Sizing Project in an Installation Environment Already in Use	206
4.2.2 Performing Sizing in Installation Environments Used in Production	208
4.2.3 SAP HANA Migration	215
4.3 Planning the System Landscape	217
4.3.1 Virtualization and Hardware Consolidation	218
4.3.2 Scaling at the Database Level for SAP HANA	221
4.3.3 Distribution of SAP Application Instances	224

4.4	Summary	225
4.5	Questions	225
5	Optimization of ABAP Programs	227
5.1	Performance Trace	227
5.1.1	Activating a Performance Trace	228
5.1.2	Evaluating an SQL Trace	229
5.1.3	Other Functions in the SQL Trace	234
5.1.4	Evaluating a Buffer Trace	236
5.1.5	Evaluating a Remote Function Call Trace	237
5.1.6	Evaluating an HTTP Trace	238
5.1.7	Evaluating an Enqueue Trace	239
5.2	Performance Analysis with ABAP Trace (Runtime Analysis)	240
5.2.1	Activating an ABAP Trace	241
5.2.2	Evaluating an ABAP Trace	243
5.2.3	Using Variations	245
5.2.4	Using Timeline Views	247
5.3	Analyzing Memory Usage with ABAP Debugger and in the Memory Inspector	249
5.4	Code Inspector	252
5.5	Tips and Tricks for High-Performance ABAP Programs	254
5.6	Summary	260
5.7	Questions	261
6	Memory Management	263
6.1	Memory Management Fundamentals	263
6.1.1	Basic Terms	263
6.1.2	Memory Areas of the SAP Instance	265
6.1.3	Zero Administration Memory Management	268
6.1.4	Technical Implementation in Detail	268

6.1.5	Summary	278
6.2	Configuring and Monitoring Memory Areas	279
6.2.1	Monitoring Swap Space	280
6.2.2	Configuring and Monitoring SAP Memory Areas	282
6.2.3	Assistance with Troubleshooting	285
6.3	Summary	292
6.4	Questions	294
7	Load Distribution, Remote Function Calls, and SAP GUI	297
7.1	Services of SAP NetWeaver Application Server	298
7.2	Load Distribution within the ABAP Instances	301
7.2.1	Distributing Message and Enqueue Services	303
7.2.2	Distributing Dialog, Background, and Spool Work Processes	303
7.2.3	Distributing Users and Work Processes over CPU Resources	304
7.2.4	Dynamic User Distribution: Configuring Logon Groups	307
7.2.5	Load Distribution Concept for Dialog Work Processes	310
7.2.6	Limiting Resources per User	313
7.2.7	Operation Modes	314
7.2.8	Configuring Dynamic Work Processes	315
7.2.9	Background Processing	315
7.2.10	Update	316
7.3	Remote Function Calls	323
7.3.1	Fundamentals and Concepts	324
7.3.2	Remote Function Call Cycle	327
7.3.3	Configuring and Testing Remote Function Call Destinations	330
7.3.4	Selecting Serialization	334
7.3.5	Monitoring Inbound and Outbound Loads	336
7.3.6	Configuring Parallel Processes with Asynchronous Remote Function Calls	341

7.3.7	Monitoring Data Transfer with Transactional Remote Function Calls	342
7.3.8	Background Remote Function Calls	343
7.4	SAP Graphical User Interface	344
7.4.1	Interaction Model and Performance Measurement	344
7.4.2	Analyzing and Optimizing the Performance of GUI Communication	347
7.5	Summary	352
7.6	Questions	355
8	Internet Connection and SAP Fiori	357
8.1	SAP Web Applications	358
8.2	Analyses on the Presentation Server	362
8.2.1	Performance Tools of Internet Browsers	365
8.2.2	SAP Statistics in the HTTP Request	366
8.2.3	Continuously Monitoring Web Applications	369
8.3	Buffering of Web Documents	370
8.3.1	Browser Buffer (Browser Cache)	370
8.3.2	Buffer in the Internet Communication Manager and SAP Web Dispatcher	373
8.3.3	Content Delivery Network	375
8.3.4	Summary: Using Buffers in Web Applications	376
8.4	Performance Analysis of Web Applications on SAP NetWeaver AS ABAP	376
8.4.1	HTTP Trace in the Internet Communication Manager	377
8.4.2	Performance Analysis of ABAP Web Applications	378
8.4.3	Monitoring Web Service Calls	381
8.5	BSPs, Web Dynpro for ABAP, and Integrated ITS	381
8.5.1	Business Server Pages and Web Dynpro ABAP	382
8.5.2	Integrated Internet Transaction Server	385
8.6	SAP Fiori, SAPUI5, and OData Services	386
8.6.1	Basic Principles of SAP Fiori, SAPUI5, and OData	387

8.6.2	Structure of an SAP Fiori System Landscape	388
8.6.3	General Performance Aspects of SAP Fiori, SAPUI5, and OData	392
8.6.4	SAP Fiori, SAPUI5, and OData on the SAP NetWeaver Application Server ABAP	395
8.7	Conclusion	402
8.8	Questions	403
9	Optimizing Java Programs	405
9.1	Garbage Collection	407
9.1.1	Construction of the Java Heap	407
9.1.2	Garbage Collection Algorithms	409
9.1.3	Selection of Collectors	410
9.1.4	Parameterization of Java Heap Memory and Garbage Collection	411
9.1.5	Logging Garbage Collections	413
9.2	Just-in-Time Compiler	413
9.3	SAP Management Console	415
9.4	SAP Java Virtual Machine Profiler	417
9.4.1	Architecture	417
9.4.2	Establishing the Connection	418
9.4.3	Allocation Analysis	421
9.4.4	Performance HotSpot Analysis	427
9.4.5	Method Parameter Analysis	431
9.4.6	Synchronization Analysis	434
9.4.7	File and Network Input/Output Analysis	436
9.4.8	Garbage Collection Analysis	438
9.4.9	Class Statistics and Heap Dump	441
9.5	Memory Analyzer	443
9.5.1	Creating HPROF Files	443
9.5.2	Analysis of the Java Heap with the Memory Analyzer	444

9.6	Other Tools for Performance Analysis on the SAP NetWeaver AS Java	446
9.7	Summary	448
9.8	Questions	449
10	Locks	451
10.1	Lock Concepts of Database System and SAP System	452
10.1.1	Database Locks	452
10.1.2	SAP Enqueues	453
10.1.3	Read Consistency	455
10.2	Monitoring Locks	457
10.2.1	Database Locks	457
10.2.2	SAP Enqueues	462
10.2.3	Read Consistency	465
10.3	Number Range Buffering	466
10.3.1	Fundamentals	466
10.3.2	Activating Number Range Buffering	471
10.3.3	Monitoring Number Range Buffering	472
10.4	Locking with Quantities and the Available-to-Promise Server	473
10.4.1	Fundamentals	474
10.4.2	Configuring the Available-to-Promise Server	475
10.4.3	Monitoring the Available-to-Promise Server	478
10.5	Summary	479
10.6	Questions	479
11	Optimizing SQL Statements	481
11.1	Identifying and Analyzing Expensive SQL Statements	483
11.1.1	Identification and Preliminary Analysis	483
11.1.2	Detailed Analysis	485

11.2	Optimizing SQL Statements in Programs	490
11.2.1	The Five Golden Rules for Efficient SQL Programming	490
11.2.2	Example of Optimizing an SQL Statement in an ABAP Program	496
11.2.3	Presetting Field Values in Report Transactions	506
11.3	Optimizing SQL Statements through Secondary Indexes	511
11.3.1	Database Organization Fundamentals	511
11.3.2	Administration for Indexes and Table Access Statistics	522
11.3.3	Rules for Creating or Changing Secondary Indexes	528
11.4	Summary	537
11.5	Questions	539
12	SAP Buffering	543
12.1	Table Buffering Fundamentals	545
12.1.1	Buffering Types	545
12.1.2	Buffer Accessing	548
12.1.3	Buffer Synchronization	550
12.1.4	Activating Buffering	553
12.1.5	Which Tables Should Be Buffered?	555
12.2	Monitoring Table Buffering on the ABAP Server	557
12.2.1	Table Access Statistics	558
12.2.2	Analyzing Buffered Tables	562
12.2.3	Analyzing Tables That Are Not Buffered Currently	566
12.2.4	Detailed Table Analysis	568
12.2.5	Monitoring Buffer Synchronization (Table DDLOG Entries)	571
12.2.6	Expensive SQL Statements Due to Incorrect Buffering	573
12.3	Monitoring Object-Oriented Application Buffers	574
12.4	Summary	580
12.5	Questions	581

13 Optimizing Database Queries with SAP HANA	583
13.1 Basic Principles of the Main Memory Database in SAP HANA	585
13.1.1 Column-Based Data Storage	587
13.1.2 Data Compression	588
13.1.3 Data Storage in Main Memory	592
13.1.4 Indexing	594
13.1.5 Text Search and Categorization	597
13.2 Architecture of SAP HANA	598
13.3 Data Modeling from a Performance View	601
13.3.1 Table and Index Design	601
13.3.2 Designing Database Views and SELECT Statements	603
13.3.3 Design of Writing Applications	613
13.4 SAP HANA Administration Tools for Performance Optimization	614
13.4.1 Analyzing Main Memory and CPU Resources	618
13.4.2 Identifying Expensive SQL Statements	625
13.4.3 Analyzing Expensive SQL Statements: Execution Plan and Plan Visualization	634
13.5 Technical Optimization Options in Detail	651
13.5.1 Monitoring Table Size, Loading, and Unloading	653
13.5.2 Monitoring Delta Indexes	659
13.5.3 Creating Indexes	665
13.5.4 SAP HANA Cached Views	668
13.5.5 Partitioning and Scaling	675
13.5.6 Reducing Data via Deletion or Displacement	680
13.5.7 Replication	688
13.6 Summary	691
13.7 Questions	693

14 Optimizing Queries in SAP Business Warehouse	695
14.1 Fundamentals of SAP Business Warehouse	696
14.1.1 Overview of the Most Important Concepts in SAP BW	697
14.1.2 InfoCubes and Extended Star Schemas	701
14.1.3 DataStore Objects	713
14.1.4 Data Modeling from a Performance View	716
14.1.5 Query Execution	719
14.2 SAP BW Administration Tools for Performance Optimization	721
14.2.1 Analysis of Expensive SAP BW Queries	722
14.2.2 SAP BW Statistics in the Workload Monitor	727
14.3 Technical Optimization Options in Detail	730
14.3.1 Managing Indexing and Database Statistics	732
14.3.2 Settings on Data Selection	739
14.3.3 Compressing the Fact Table	740
14.3.4 Monitoring the OLAP Cache	742
14.3.5 Optimizing Reports	746
14.3.6 Aggregates	747
14.4 Summary	757
14.5 Questions	759
Appendices	543
A Database Monitors	763
B Important Transaction Codes	825
C Review Questions and Answers	831
D Glossary	837
E Information Sources	849
F The Author	871
Index	873

Index

64-bit	264	Address space	264, 837
64-bit architecture	280	AGate and WGate	841
A			
ABAP	485, 488, 837	Agent	167, 784, 786
<i>application instance</i>	78	Agent private memory	784
<i>debugger</i>	227, 249, 260, 261	Aggregate	487, 583, 603, 747, 754
<i>instance</i>	353	<i>compress</i>	756
<i>objects</i>	577	<i>create</i>	751
<i>shared object</i>	288	<i>function</i>	491
<i>trace</i>	69, 227, 240	<i>indexing</i>	755
<i>trace variant</i>	245	<i>maintenance</i>	755
ABAP Class Builder	577	<i>suggestion</i>	753
ABAP Class Library	827	Aggregation	246
ABAP Data Dictionary 108, 522, 543, 574, 837		ALE	837
<i>display</i>	827	<i>administration</i>	825
<i>table</i>	827	Alert Monitor	837
ABAP Development Workbench	458	Allocation	
<i>BSP development</i>	384	<i>analysis</i>	421
ABAP Dictionary	498	<i>rate</i>	426
ABAP Editor	291, 508, 827	<i>trace</i>	448
ABAP List Viewer	245	ALV Control	344
ABAP program	490, 496	American National Standards	
<i>quality analysis</i>	252	Institute	837
<i>runtime analysis</i>	227	Analysis	
<i>termination</i>	286	<i>linguistic</i>	27, 597
ABAP runtime analysis 240, 243, 260, 490		Analytical applications	696
ABAP runtime analysis for Web		ANSI	837
Dynpro applications	242, 381	Appliance	222
ABAP Runtime Error	829	Application	
ABAP Server	137, 144	<i>analysis</i>	135, 829
ABAP Workbench	252, 510, 827	<i>error</i>	288
Access plan	786	<i>layer</i>	45
ACID principles	837	<i>optimization</i>	22
Action profile	169	<i>support layer</i>	784
Activation queue	715	<i>tuning</i>	32, 66
Active data	715	Application buffer	
Active session history	813	<i>object-oriented</i>	543, 574, 580
Activity → throughput		Application Link Enabling	837
Adaptive Server Enterprise → ASE		Application Server	78, 144, 234, 837
		Archiver stuck	109, 154
		Array fetch	232
		ASE	777

- ATP
logic 451, 473
server 473–475
service 298, 303
- Attribute 697, 717
display attribute 704
navigation attribute 704
vector 589, 595
- Automatic workload repository 813
- Auxiliary storage pool 801
- Availability 53, 169, 302
check 473, 577
- B**
- B* tree 735
- Background
load 154
processing 828, 837
program 195, 460
service 298, 303
- Background job
SAP_COLLECTOR_FOR_PERFORMANCE 142
Scheduling Monitor 826
- Backup 54, 58
- Bandwidth 594
- BAPI 837
- Batch input 195, 837
- Benchmark 201, 204, 845
- BEx Web Analyzer 362
- bgRFC monitor 344
- Big data 26
- Binary search 595, 766
- Blade server 222
- Block 95
- Bottleneck analysis 135, 306
- Browser 837
- BSP applications, runtime
analysis 242, 381
- Buffer 48, 94, 543
access 548
buffer hierarchy 49
catalog buffer 784
communication buffer 784
data buffer 95
- Buffer (Cont.)
database buffer 48
management 548
metadata buffer 779
object-oriented application buffer ... 48
operating system buffer 48
package buffer 784
process buffer 779
quality 95, 111
setting 115
single-record buffer 548
status 561
storage subsystem 48
synchronization 550, 825
synchronization monitor 571
- TABL 548
table buffer 548
TABLP 548
trace 228, 236
types 543
- Buffer pool 784
- Buffer trace 228
- Buffering → SAP buffering
- Business
hours 57
process 52
- Business Application Programming
Interface → BAPI
- Business Server Pages (BSPs) 45, 357, 359, 381
- BusinessObjects 838
- Button 838
- BW Administrator Workbench 826
- BW Aggregate Maintenance 826
- BW Check Report 826
- BW Query Monitor 826
- BWA → SAP BW Accelerator
- C**
- CA Wily Introscope 449
- Cache → buffer
- Calculation engine 600, 612
- CALL 266
- Catalog cache 768, 784
- CBO → Cost-Based Optimizer (CBO)

- CCMS 32, 37, 68, 526, 838
Monitor 334
System Component Repository (SCR) 168
- Central monitoring system 168
- Central SAP monitor 70, 827
- Central single-record statistics 172
- Change 559, 567
log 715
logical 154
request 553
run 757
- Change and Transport Organizing ... 838
- Change and Transport System 553
- Characteristic 697, 704, 717
- Characterizing parameter 136
- Checking group 476
- Checkpoint 95
- Class
loader 407
statistic 448
- CLEAR statement 255
- Client 838
destination statistics records 339
statistics records 339
- Client/server architecture 43
scalability 51
- Cloud applications 33
- Cluster coding 591
- Clustered index 822
- Code
cache 414
completion 626
- Code Inspector 227, 252, 261, 550
- Code push-down 583
- Column store → Column-based
data storage
- Column-based data storage 583, 585, 587, 600, 654
- COM routine 145
- Command monitor 770
- Common Programming Interface
Communication 838
- Compilation 413
- Compressibility 602
- Compression 585, 588
cluster coding 591
prefix coding 591
run-length coding 591
sparse coding 591
- Computer 78, 125
- Computer Center Management
System → CCMS
- Concordance 598
- Concurrent mark-sweep collector 410
- Condition table 556, 573
- Container 92
- Context switch 82, 265, 838
- Continuous performance
optimization 62
- Controls 297
- CO-PA Accelerator 41
- Core 82, 779
- Cost-Based Optimizer (CBO) 517
- Coupling
hard 324
soft 325
- Coupling, Application Link Enabling 325
- CPI-C 838
- CPU 51, 82, 838
Bottleneck 107
bottleneck 87, 155
Load 85
load 81, 156, 209
requirement 205, 352
resources 304
time 146, 149, 152, 157, 165, 305
trace 240
utilization 107
wait time 306
- CPU bottleneck 83
- CTO 838
- Cursor 233
- Cursor cache 264
- Cursor ID 233
- Customer interaction center 251
- Customizing 838
data 556

D

Data

archiving 838
buffer 95, 768, 784, 804, 810, 819
cache 95, 768, 819
locality 676
mining 27, 697
package 757
warehouse 697
 Data Control Language 839
 Data Definition Language 839
 Data Manipulation Language 839
 Data modeling
 SAP HANA 601, 603
 Data volume 774
 transferred 346
 Database 78
 access, fully qualified 232
 administration 111
 analysis 77, 91
 Analyzer 774
 buffer 94, 585
 error log file 106, 788
 global memory 784
 heap 784
 index, missing 108
 instance 78, 283, 838
 load 494
 monitor 69, 91, 119, 458, 763
 object, missing 825
 operation 232
 optimizer .. 108, 514, 516, 534, 634, 839
 optimizer, cost-based (CBO) 108
 procedure 145
 procedure call 145
 procedure subrecord 145
 procedure time 145
 process 86, 627
 process monitor 97, 125, 229, 446, 449, 769, 780, 793, 805, 811, 821
 processor 107
 query, parallelization 586
 response time
 long 126
 server 50, 78, 839

Database (Cont.)

service 352
 standstill 109
 table 695
 task 769
 time 144, 152, 156, 157
 tuning 32
 view . 504, 584, 766, 776, 783, 790, 808, 817, 823
 Database Administrator Cockpit
 → DBA Cockpit
 Database lock 105, 447, 452, 453, 457, 788, 838
 IBM DB2 for z/OS 805
 monitor 125, 457
 SAP MaxDB 773
 SQL Server 821
 Database optimizer 843
 Database performance
 monitor 92, 828
 problem 156
 Database system 452, 838
 parallel 78
 DatabaseDatabase layer 838
 Datafile 92
 DataStore Object
 indexing 738
 SAP HANA-optimized 715
 DataStore object 698, 713, 717
 DB2 for LUW 785
 DB2 → DB2
 DBA 839
 DBA Cockpit 70, 92, 125, 458, 523, 524
 DB2 for IBM i 763, 767, 793
 DBA log 825
 DBA Planning Calendar 524
 DBACOCKPIT 69, 764, 767
 DCL 839
 DDL 839
 Deadlock 461, 839
 Debugger 249, 414
 DELETE statement 255
 Delta index 592, 659, 665
 Demilitarized zone (DMZ) 46, 300
 Deoptimization 414
 Dequeue module 454

Destination 325, 330
 Developer log 290
 Developer trace 829
 Development Workbench 37
 DIAG protocol 839
 Dialog
 load 154
 response time 321
 service 298, 302, 303
 user 194, 314
 work process 144, 310, 839
 work process. 317
 Dialog step → Transaction step
 Dictionary 595, 596
 coding 588
 Difference coding 591
 Dimension ID 701
 Dimension table 701
 index 736
 Direct read 232, 766, 775, 782, 806
 Direct reading → Direct read
 disp+work 298
 Dispatcher 302, 839
 process 298
 queue 129, 143
 wait time 143, 152, 157, 306
 Dispatching 298
 Displacement (swap) 115
 Displacement → Swap
 Display attribute 704
 Distributed Statistic Record → *Statistics record* distributed
 Distributed statistics record (DSR) 166
 DML 839
 Documentation 66
 Dominator tree 445
 Drilldown 699
 Dump 286
 Dynamic statement cache 804
 Dynamic user distribution 307
 Dynpro 839

E

E fact table 741
 E2E trace → End-to-end trace

EarlyWatch Alert → SAP EarlyWatch Alert
 Easy Web Transaction 385, 839
 Eclipse 839
 Eden 407, 409
 EDI 839
 EDM
 DSC cache 804
 pool 804
 Electronic Data Interchange 839
 End User Experience Monitoring 59
 End-to-end
 diagnostics 71
 runtime analysis 172, 176, 178, 182
 trace 246, 378
 End-User Experience Monitoring 369
 Enqueue 452, 839
 module 454
 operation 227
 service 298, 301, 303
 trace 228, 239
 Enterprise JavaBean 46
 Entity 839
 Error code
 DBIF_RSQL_NO_MEMORY 292
 EXPORT_NO_SHARED_MEMORY 287
 EXSORT_NOT_ENOUGH_hMEMORY 292
 PXA_NO_SHARED_MEMORY 286
 SET_PARAMETER_MEMORY_OVERFLOW 287
 STORAGE_PARAMETERS_WRONG_SET 286, 289, 292
 SYSTEM_NO_MORE_PAGING 276
 SYSTEM_NO_ROLL 286
 TSV_TNEW_PAGE_ALLOC_hFAILED 286
 TSV_TNEW_PG_CREATE_FAILED .. 276
 Escalation procedure 61
 ETL process 701
 Event 384
 Event data 27
 EWT 839
 Exclusive database lock 105

Exclusive lock wait 105, 113, 134, 457, 479
SQL Server 130, 805, 821
 Execution plan 513, 514, 634, 765, 839
IBM DB2 for z/OS 806
Microsoft SQL Server 822
Oracle 815
 Executive Information System (EIS) . 695
 Expert monitor 68
 Export/import buffer 475, 544, 574
 Export/import SHM buffer 287, 544, 574
 Extended memory area (EM) 288, 289, 839
 Extended Storage server 599
 Extensible Markup Language 847
 Extraction 701

F

F fact table 741
 Fact table 701, 756
 compress 740
 index 736
 SAP HANA 710
 Failover solution 303
 FDDI 839
 FE Net Time 346
 FETCH operation 232, 500, 559
 Fiber Distributed Data Interchange .. 839
 File and network I/O analysis 436
 File operation 436
 File system cache 87
 Files statistics 436
 Firewall 839
 Flow 698
 FOR ALL ENTRIES 500, 504
 Fragmentation 569, 579
 FREE statement 254
 Frontend
 time 346
 trace 176
 Full table scan 515, 522, 530, 734, 765
 Function Builder 321
 Function module 320
 Functional trace → Single-record statistics, central

G

Garbage collection 406, 407, 448, 840, 842
 analysis 438, 448
 collector 410
 compacting 409
 full 408
 log 413
 mark and copy 409
 mark and sweep 409
 partial 408
 SAP JVM 407
 Gateway monitor 328
 Gateway service 298
 Generation 407
 Generation time 144
 Global cache hit ratio 805
 Globally Unique Identifier (GUID) 840
 GoingLive Check → SAP GoingLive Check
 GUI 840
 communication 347
 controls 344
 time 145, 165, 346, 349
 GUID 840

H

HANA → SAP HANA
 Hard disk 594
 Hard disk monitor 774, 821
 Hardware 51
 analysis 77
 bottleneck 84, 155
 capacity 156
 consolidation 217, 218
 landscape 302
 monitoring 79, 81
 partner 186, 188
 sizing 185, 186, 476
 tuning 32
 Hash table 255
 Heap 265
 Heap dump 444
 analysis 448
 Heap memory 840

Hierarchy 706, 718
 High availability 302, 303, 840
 High water mark 116
 Hint 534
 Hit ratio 95
 Hot spot 103, 428, 677, 821
 HotSpot Java Virtual Machine 409
 HPROF file 443
 HTML 840
 HTML business template 386
 HTTP 840
 HTTP call 381
 HTTP trace 238, 260, 377
 HybridProvider 698
 Hypertext Markup Language 840
 Hypertext Transfer Protocol 840

I

I/O bottleneck 88, 103, 234
 I/O operation 448
 IBM DB2
 for IBM i 791
 for Linux, UNIX, and Windows 784, 785
 for z/OS 803
 IBM i 791, 792, 801
 ICM monitor 69, 132, 377, 378
 IDES 840
 IDoc 840
 IMG 840
 Implementation Guide 744, 840
 Index 583, 603, 665, 749
 administration 522
 B index* 595
 B tree index* 738
 bitmap index 735, 738
 concatenated 595, 597
 create 522
 efficient 232
 fragmentation 527
 inverted 595, 596
 maintain 522
 missing 108
 primary index 109
 quality 527

Index (Cont.)
 range scan 515, 522
 reorganize 527
 scan 734
 server 599
 unique scan 514
 Indexing 732
 InfoCube 698, 701, 717, 750
 HANA migration 826
 optimization 730
 Overview 826
 tables 707
 transactional 736
 InfoObject 697
 InfoProvider 698, 717, 743
 InfoSet 698
 Initial sizing 191
 In-memory
 application 39
 computing 583, 586
 database 594
 Installation
 central 302
 distributed 302
 Instance → SAP instance.
 Integer ID 588
 Integrity 53
 Inter Process Communication 841
 Interaction model 344
 Interface 297
 Internal Document 840
 International Demo and Education System 840
 Internet Communication Framework (ICF) 382, 385
 Internet Communication Manager (ICM) 131, 298, 376, 377, 381, 382
 Internet Communication Manager, Monitor 828
 Internet connection 357
 Internet of things 26
 Internet Pricing and Configuration (IPC) 841

Internet Transaction Server (ITS) 309, 358, 385
 Integrated 275, 385
 integrated 828
 performance analysis 378
 Interpreter 413
 Intranet 841
 Introscope statistics 167
 Invalidation 115, 553
 IPC 841
 iSeries 801
 ITS monitor 69
 ITS → Internet Transaction Server (ITS)
 ITS status monitor 386

J

J2EE Engine → SAP NetWeaver
 Application Server Java
 Java 841
 application instance 78
 bytecode 413
 Dictionary 543
 heap 407, 412
 heap analysis 444, 448
 runtime 418
 servlet 45, 357
 statistics 167
 trace 69
 workload monitor 172
 Java Development Kit (JDK) 405
 Java Server Page 45
 Java Server Pages 357, 359
 Java Virtual Machine ... 405, 448, 841, 846
 memory area 407
 work process 415
 JavaScript 600
 JIT compiler 405, 413
 Job Analysis 828
 Job overview 756
 Join 718, 734
 engine 600, 612
 Just-in-time compiler 405, 406, 413, 428

K

Key figure 136, 697

L

LAN 51, 841
 check 89
 Landscape replication 601
 Large I/O pool 779
 Latency time 361, 437, 594
 Leaf 735
 Line item dimension 711, 717
 Linearity 196
 Linux 277
 liveCache 764
 Load 149, 159
 external system 169
 inbound 336
 outbound 336
 Load distribution 137, 155, 297
 ABAP 301
 incorrect 84, 128, 156
 Load distribution concept
 new 310
 Load profile 137
 Load time 144
 Loading 701
 Local Area Network (LAN) 841
 Local memory 264
 Lock 434, 448, 451
 database lock 452
 escalation 773
 handler 452
 list 784
 lock concept 452
 lock conflict 457
 lock object 453
 lock with quantities 476
 locks with quantities 475
 monitoring 457
 SAP enqueue 452, 473
 shared lock 475
 table lock 461
 wait situation 451
 Log area 109, 774

Logical analysis 65
 Logical Unit of Work (LUW) 342, 842
 Logistic Information
 System (LIS) 320, 695
 Logon group 302, 307, 334
 maintenance 828
 monitor 302
 Loop, nested 256
 Low speed connection 352
 LRU 841

M

Machine code 413
 Main memory 51, 583, 585, 594
 bottleneck 155
 buffering 467
 profile 158
 requirement 208, 282
 sizing 196, 215
 utilization 83
 virtual 293
 Main memory bottleneck 83, 84, 87
 Main memory configuration
 monitor 208
 Maintain Profile Parameters
 (By Profile) 826
 Maintenance view
 DBDIFFVIEW 524
 Mark-and-copy collector 410
 Master
 data 555
 data table, indexing 738
 service 678
 Memory
 allocated 118, 619
 allocation 271
 area 279, 292
 available 280
 configuration 113, 263
 extract 250
 fixed allocated (HEAP) 288
 Inspector 249, 250
 leak 443, 446, 448, 449
 local 264, 841
 management 842
 Memory (Cont.)
 physical 119, 619
 pipe 132, 384
 shared 264
 used 619
 virtual 263, 619, 846
 virtual required 280
 Memory Analyzer 443, 449
 Memory Configuration Monitor 113, 284, 289, 478, 566, 576, 579, 828
 Memory management 117, 157
 IBM i 791, 801
 integrated 407
 Linux 277
 Merge 592, 659
 auto merge 660
 critical merge 661
 hard merge 660
 smart merge 661
 Message server 297, 298
 Message service 298, 303, 353
 Metadata 94
 Method 413, 448
 memory requirement 423
 parameter trace 448
 Microsoft SQL Server 511
 Microsoft Windows 277, 306
 Migration 214
 Missing index 523
 Mode
 list 284
 PRIV mode 272, 284
 Model View Controller (MVC) 382
 Modularization unit 240
 Monitor RFC destination 334
 Monitoring 52, 60
 agent 172
 central 55
 plan 52, 55
 system, central 168
 Moore's law 842
 Multicore processor 82
 Multilingualism 718
 MultiProvider 698, 699, 717
 Multithreaded CPU 82
 MVC 382

N

Name server 599
 Nametab buffer 544
 Navigation attribute 704, 718
 Nested loop join 506, 734, 783
 Net Time 346
 Network 89, 234, 305
 graphic 826
 I/O and file I/O trace 448
 operation 436
 problem 234
 tuning 32
 Number Range
 buffering 828
 Number range 466
 buffer 474
 buffer mode 472
 buffering 466, 467, 471
 interval 466
 number range level 471
 object 466

O

Object Linking and Embedding (OLE) 843
 OLAP 205, 696, 843
 engine 600, 612
 processor 719
 OLAP cache 577, 719
 configuration 743, 826
 monitor 744, 746, 826
 monitoring 742
 Old generation 407, 411
 collectors 410
 OLE 843
 OLTP 205, 696, 843
 Open object 820
 OPEN operation 232
 Operating mode 314
 Operating system 843
 command 827
 configuration parameter 286
 file 825
 limit 289
 paging 265

Operating system (Cont.)
 parameter 89
 process 85
 swap space 286
 Operating system monitor 68, 79, 88,
 208, 229, 350, 792
 process overview 125
 Operation mode 828, 843
 Optimization
 plan 52, 55
 technical 22
 Optimizer → Database optimizer
 Oracle 810
 Oracle wait event 812
 OS Monitor 828

P

Package cache 784
 Package dimension 707
 Page-in 846
 Page-out 846
 Pages 95
 Paging 843
 rate 84, 87, 791
 Paging file → Swap space
 Parallelization 586, 675
 Parameter
 change 89
 characterizing 136
 maintenance 378
 rddb/obj/buffersize 476
 rddb/obj/max_objects 476
 Parsing 514
 Partitioning 318, 675, 679
 area-based 676
 round robin 676
 PASE runtime environment 802
 Passport 166, 172, 845
 Pending period 551, 560
 Performance 53, 843
 forum 75
 measurement 344
 trace 227, 228, 348, 729
 Performance HotSpot analysis 428
 Performance HotSpot trace 448

Performance management 35
 proactive 23
 Performance problem
 general 151
 permanent 152
 specific 151, 158
 temporary 152
 Performing sizing project 197
 Permanent generation 407, 412
 Physical main memory (RAM) .. 279, 293
 Pivoting 699
 Planning application 696
 Pop-up 843
 Position 698
 Prefix coding 591
 PREPARE statement 233
 Prepared statement 804
 Prepared template 386
 Preprocessor 599
 Presentation layer 44
 Presentation server 45
 analysis 362
 Primary index 512, 524, 766
 Priority class 311
 PRIV Mode 273
 PRIV mode 127
 Private mode 272
 Probe 447
 Procedure cache 819
 Process
 complete 128
 external 87
 stopped 127
 Process chain 701
 Process ID 125
 Processing time 146, 149, 152
 Processor 82, 214, 838
 Processor thread 354
 Profile parameter
 abap/atrapath 243
 abap/atrasizequota 243
 abap/heap_area_dia 271, 273,
 289, 291
 abap/heap_area_nondia 271, 273,
 289, 291
 abap/heap_area_total .. 271, 281, 289

Profile parameter (Cont.)
 abap/heaplimit 158, 271
 db/IO_buf_size 232
 em/address_space_MB 277
 em/blocksize_KB 270
 em/initial_size_MB 117, 127, 270,
 283, 289
 em/max_size_MB 277
 maintenance 827
 PHYS_MEMSIZE 117, 268, 283
 rdisp/atp_server 476
 rdisp/bufrefmode 552
 rdisp/bufreftime 552
 rdisp/enqname 303
 rdisp/max_wprun_time 308
 rdisp/mshost 303
 rdisp/Pg_MAXFS 276
 rdisp/Pg_SHM 276
 rdisp/ROLL_MAXFS 270, 289
 rdisp/ROLL_SHM 117, 270
 rdisp/vb_dispatching 319
 rdisp/vbstart 317
 rddb/max_blocking_factor 503
 rstr/file 229
 rstr/max_diskspace 229
 setting 290
 stat/dbprocrec 146
 ztta/roll_area 270, 273
 ztta/roll_extension 270, 273, 284,
 289, 293
 ztta/roll_extension_dia 270, 284
 ztta/roll_extension_nondia .. 270, 284
 ztta/roll_first 271–273
 Profile parameters 286
 Program
 buffer 286, 544
 counter 82
 error 286
 RSCOLLOO 142
 SAPOSCOL 81
 termination 286
 Program Global Area (PGA) 810
 Promotion 407
 PTF package 795

Q

Quadratic dependency	260
Quality analysis	252
Quantity structure	194
Query	
<i>analytical</i>	592
Quick Sizer	192, 293
Quick Sizer project	197

R

Radio Frequency Identification (RFID)	26
RAID	843
Ranking	598
RBO → Rule-Based Optimizer (RBO)	
RDBMS	843
Reaction time	59
Read	
<i>record-by-record</i>	521
<i>sequential</i>	521
Read access	473
<i>logical</i>	95
<i>physical</i>	95
Read random hit ratio	804
READ TABLE	255
READ TABLE ... WITH KEY	255
Read/write (I/O) problem	88, 103, 788
Recoverability	54
Recovery	58
Recursive call	811
Redo log file	154
Region, generic	547
Relational database	38
Relational Database Management System	843
Remote Function Call (RFC)	45, 195, 227, 310, 843
<i>ABAP coding</i>	325
<i>Asynchronous (aRFC)</i>	309
<i>asynchronous (aRFC)</i>	325, 341, 828
<i>background RFC (bgRFC)</i>	325, 343
<i>client profile</i>	339
<i>connection</i>	330, 828
<i>cycle</i>	327
<i>fundamentals</i>	323

Remote Function Call (RFC) (Cont.)	
<i>queued (qRFC)</i>	325
<i>RFC time</i>	330
<i>statistics</i>	338
<i>synchronous</i>	325
<i>trace</i>	165, 228
<i>transactional (tRFC)</i>	325, 342, 846
RemoteCube	743
REOPEN operation	232
Reorganization	679
Replication	601
Reporting	
<i>precalculation</i>	746
Repository browser	827
Repository Information System	827
Request	559, 567
Required field	507
Resource	
<i>monitor</i>	773
<i>requirements</i>	186, 196
Response time	59, 135, 147, 304, 305, 764
<i>average</i>	159
<i>distribution</i>	169
RFC → Remote Function Call (RFC)	
RFID	843
Roll buffer	283
Roll memory → SAP roll memory	
Roll wait time	145, 165, 329, 346
Roll-in	143, 267, 838
Roll-out	143, 267, 838
Rollup	699
Root recognition	598
Roundtrip	346, 438
Row	
<i>cache</i>	810
<i>ID</i>	513
<i>store</i>	600, 658
Row-based data storage	585
RSRCACHE	744
Rule-Based Optimizer (RBO)	517, 519
Run-length coding	591
Runtime	427
<i>analysis</i>	136, 227, 447
<i>constant</i>	260
<i>logarithmically increasing</i>	260

S

SAP application	
<i>architecture</i>	36
SAP application instance → SAP instance	
SAP Application Performance	
Standard	202
SAP ASE	
<i>data buffer</i>	778
<i>database process</i>	779
<i>engine</i>	779, 780
<i>execution plan</i>	782
SAP Basis	37
SAP BEx analyzer	700
SAP buffer	114, 145, 282, 292
<i>access</i>	227
<i>trace</i>	229
SAP buffering	495, 543, 546
<i>activate</i>	553
<i>full</i>	546
<i>generic</i>	546
<i>type</i>	545
SAP Business Suite	
<i>architecture</i>	36
<i>Introduction</i>	217
<i>SAP HANA</i>	39
SAP Business Warehouse	137, 362, 577, 695, 843
<i>administration tools</i>	721
<i>analysis</i>	722
<i>data selection</i>	739
<i>frontend</i>	700
<i>high database response times</i>	728
<i>indexing</i>	732
<i>load profiles</i>	728
<i>optimization</i>	730
<i>performance optimization</i>	716
<i>query</i>	696, 700
<i>SAP HANA</i>	39
<i>statistics</i>	724
<i>workload statistics</i>	721
SAP BusinessObjects	700
<i>analysis</i>	700
<i>Business Intelligence</i>	700
<i>Dashboards</i>	701
<i>Explorer</i>	701

SAP BusinessObjects (Cont.)	
<i>Web Intelligence</i>	700
SAP BW	38
SAP BW Accelerator	39, 584, 721, 843
SAP BW Accelerator Monitor	826
SAP BW Administrator	
<i>Workbench</i>	721, 722, 736, 740, 753
SAP client plug-in	177, 182
SAP component	36
SAP Crystal Reports	700
SAP cursor cache	233
SAP Customer Relationship Management (SAP CRM)	841
SAP EarlyWatch Alert	27, 61, 73, 207, 290
SAP Easy Access menu	352
SAP EG memory	275
SAP enqueue	453, 462, 473
SAP enqueue monitor	478
SAP Enterprise Portal	38
SAP ERP SD benchmark	201
SAP extended global (EG) memory	386
SAP extended memory	116, 270, 272, 275, 283, 292
SAP GoingLive Check	27, 187, 188, 206, 290
SAP GoingLive Functional	
<i>Upgrade Check</i>	207
SAP GoingLive Functional	
<i>Upgrade Service</i>	213
SAP GoingLive Migration Check	207
SAP GUI	360, 843
<i>controls</i>	297, 344
<i>for HTML</i>	360
<i>for Java environment</i>	44, 360
<i>for Windows</i>	44, 360
<i>transaction, end-to-end runtime analysis</i>	178
SAP HANA	496, 583, 843
<i>administration</i>	712
<i>compression</i>	588
<i>database platform</i>	584
<i>indexing</i>	594
<i>main memory area</i>	619
<i>scalability</i>	496
<i>scaling</i>	52
<i>sizing</i>	196

SAP HANA Studio 70, 614
 SAP HANA XS server 600
 SAP heap memory 116, 270, 289, 292
 SAP host agent 793
 SAP Implementation Guide (IMG) 828
 SAP Instance 841
 maintenance 826
 overview 828
 SAP instance 78, 140, 221, 224, 282
 SAP Internet Transaction Server
 (SAP ITS) 45, 841
 SAP Java Virtual Machine (JVM) 406
 SAP Java Virtual Machine Profiler 417,
 444, 448
 SAP kernel 290
 SAP List Viewer Control 344
 SAP liveCache 38, 137, 844
 SAP Logical Unit of Work 453
 SAP Management Console 448
 SAP MaxDB 511, 767
 command monitor 770
 execution plan 775
 resource monitor 773
 SAP memory area 282
 SAP memory configuration 113, 126
 SAP Memory Configuration
 Monitor 69, 113, 744
 SAP memory management 157
 SAP NetWeaver
 release 7.40 550
 Release 7.50 33
 SAP NetWeaver Administrator 68, 70
 SQL trace 447
 SAP NetWeaver Application
 Server 45, 358
 services 298
 SAP NetWeaver Application
 Server ABAP 78
 SAP NetWeaver Application Server
 Java 37, 78, 405
 SAP NetWeaver AS Java
 server node 418
 SQL trace 447
 SAP NetWeaver Enterprise
 Search 38, 844
 SAP paging 265
 memory 275, 579, 843
 SAP parameter, change 120
 SAP Performance Menu 829
 SAP performance trace 227
 SAP Process Integration 38
 SAP R/3 24
 SAP roll
 buffer 269
 file 269
 memory 116, 289, 292
 SAP roll area
 local 269
 shared 269
 SAP service 297, 302
 SAP Service Marketplace, service 72
 SAP solution 36
 SAP solution landscape 45
 SAP Solution Manager 138, 167, 176,
 182, 246, 369, 764
 analysis 71
 monitoring 71
 performance optimization 72
 SLM 61
 tracing 71
 workload analysis 71
 SAP Standard Application
 Benchmark 185, 200
 SAP Support Portal 293
 SAP system 45, 79
 service 844
 trace 828
 SAP System Identifier 845
 SAP transaction 453
 SAP Web Dispatcher 46, 297–299, 309
 SAP work process 85, 105, 122, 283,
 293, 768
 overview 122
 SAPCCMSR 172
 SAProuter 844
 SAPS 202, 844
 Savepoint 95
 Scalability 196, 678, 844
 horizontal 51, 844
 program 256, 259
 vertical 51, 844

Secondary index 511, 513, 522, 524
 change 528
 create 528
 Security 54
 SELECT
 ** clause* 500
 identical 235
 nested 499
 Selection screen 506
 Selectivity 518, 519, 529
 Semaphore 125
 Send System Message 827
 Sensor data 26
 Sequential read 232, 765, 775, 782,
 789, 806, 807, 816, 822
 Serialization 125
 Server 79, 844
 Server consolidation 218
 Server destination statistics records 339
 Server profile 156
 Service 301
 Service Level Agreement 56
 Service Level Management
 (SLM) 56, 74, 844
 Service maintenance 242, 377, 381
 Service-level report 56, 58, 61, 74
 Session 266, 845
 external 266
 internal 266
 monitor 811
 Shadow process 811, 812
 Shared cursor cache 810, 813
 Shared memory 264, 286, 384, 474,
 577, 784, 845
 Shared objects 574
 Area configuration 578
 Area monitor 578
 buffering 544, 577
 Shared pool 810
 Shared SQL
 area 813
 cache 813
 SID 705, 845
 table 705
 Simulation application 696
 Single point of failure (SPOF) 303
 Single-core processor 82
 Single-level storage 801
 Single-record buffer 543, 545
 Single-record statistics 69, 136, 141,
 181, 227, 329, 347, 558, 722, 829
 central 168, 172
 Sizing 185, 196
 initial 187
 process 186
 sizing plausibility check 188
 throughput-based 194
 T-shirt sizing 192
 user-based 193
 Skeleton 804
 Slave service 679
 Slice & dice 699
 Social network 26
 Socket statistics 436
 Solution monitoring 52
 Sparse coding 591
 Spool service 298, 303
 SQL 845
 editor 626
 native SQL 536
 OpenSQL 536
 plan cache 629, 632, 633, 635
 trace ... 69, 163, 228, 229, 234, 260, 449,
 483, 484, 487, 491, 496, 573
 SQL code 499
 SQL programming
 efficient 490
 golden rules 490, 588, 691
 SQL Server 819
 primary index 822
 SQL statement 72, 98, 227, 481, 490,
 511, 529, 531, 634, 765, 770, 810, 812
 buffer 779
 dynamic 804
 expensive 96, 483, 788, 829
 nested 500
 optimize 481
 SQL statistics 98, 447, 483, 485, 488,
 537, 573, 629, 813, 828
 analysis 102
 DB2 786, 787
 IBM DB2 for z/OS 805

SQL statistics (Cont.)	
<i>SAP MaxDB</i>	770
<i>SQL Server</i>	821
SQL TraceSQL trace	828
SQLScript	603
Stack pointer	82
Star join execution plan	733
Star schema	590, 698
<i>extended</i>	701
Statement cache	819, 821
Statement string	804
Statistics	
<i>derived</i>	426
<i>record</i>	845
<i>server</i>	679
<i>single record</i>	181, 227
Statistics record	161, 166, 349
<i>distributed</i>	166, 172
Stemming	597
Stock	698
Stop-the-world	409
Stored procedure	584
Stored Procedures	845
Structured Query Language	845
SUBMIT	266
Subrecord	137, 145
Suite Accelerator	40
Support package	845
Survivor	409
Swap	553, 557
Swap space	83, 117, 263, 279, 280, 293, 801
Sybase	846
Synchronization analysis	434
Synchronization trace	448
System availability	303
System buffer	544
System Global Area (SGA)	810
System landscape	217, 846
System load analysis, global	168
System Log	827
System monitoring	54
T	
Table	
<i>analysis</i>	568, 825
<i>ARFCSDATA</i>	342
<i>ARFCSSTATE</i>	342
<i>ATAB</i>	574
<i>buffered</i>	562, 566
<i>condition table</i>	556
<i>created by SAP</i>	564, 568
<i>custom</i>	564, 568
<i>DOIO*</i>	574
<i>DOIOS</i>	573
<i>DO20*</i>	574
<i>DBDIFF</i>	524
<i>DBSTATC</i>	525
<i>DDLOG</i>	551, 571, 577
<i>DDNTF</i>	567, 574
<i>DDNTT</i>	567, 574
<i>hash table</i>	255
<i>internal</i>	249, 856
<i>KAPOL</i>	561, 574
<i>NRIV</i>	457, 466, 472
<i>NRIV_LOKAL</i>	468
<i>reorganize</i>	527
<i>RESB</i>	66, 473, 527
<i>sorted</i>	255
<i>SWNCMONI</i>	142
<i>TCURR</i>	569
<i>update table</i>	318, 322
<i>VBBE</i>	473
<i>VBDATA</i>	318
<i>VBHDR</i>	318, 567
<i>VBMOD</i>	318
Table Access Statistics	828
Table access statistics	108, 519, 524, 558, 562, 567, 826
Table buffer	164, 543
<i>generic (TABL)</i>	543
<i>partial (TABLP)</i>	543
Table buffering	545
<i>ABAP server</i>	557
Table maintenance	524, 668
Table operation	240
Table pool	
<i>ATAB</i>	573

Table pool (Cont.)	
<i>KAPOL</i>	573
Table size	567
Tablespace	92
Tag	383
Task type	140
TCP/IP	846
Technical analysis	65
Teraspace	802
Text dictionary	591
Text edit control	344
Think time	304, 305
Thread	81, 82, 448, 627, 768, 779
<i>dump</i>	416
<i>hardware</i>	82
<i>monitor</i>	629
<i>software</i>	82
Throughput	149, 159, 451
Time dependency	705, 718
Time dimension	707
Time profile	169
Time-based sampling	428
TMS	846
Trace	
<i>frontend</i>	176
<i>level</i>	178
Training	850
Transaction	453, 846
<i>AL11</i>	825
<i>AL12</i>	825
<i>BALE</i>	825
<i>code</i>	846
<i>critical</i>	58
<i>data</i>	555
<i>DBO1</i>	93, 105, 457, 479
<i>DBO2</i>	93, 519, 825
<i>DBO5</i>	569, 825
<i>DB12</i>	93, 825
<i>DB13</i>	93, 825
<i>DB13C</i>	93
<i>DB20</i>	825
<i>DB21</i>	525
<i>DB24</i>	93
<i>DBACOCKPIT</i>	70, 92, 125, 458, 523, 524, 614, 825, 839
<i>DWDM</i>	345
Transaction (Cont.)	
<i>LISTCUBE</i>	722, 826
<i>Lock</i>	827
<i>MES7</i>	507
<i>OSO6</i>	79
<i>OSO6N</i>	79
<i>OSO7</i>	79
<i>OSO7N</i>	79
<i>OSS1</i>	826
<i>profile</i>	158
<i>response time</i>	227
<i>RSA1</i>	721, 722, 724, 736, 753, 826
<i>RSCACHE</i>	826
<i>RSCUSTV14</i>	826
<i>RSDDBWAMON</i>	826
<i>RSDDV</i>	721, 826
<i>RSMIGRHANADB</i>	712, 826
<i>RSODSO_SETTINGS</i>	826
<i>RSRT</i>	721, 723, 729, 743, 826
<i>RSRV</i>	826
<i>RZ01</i>	826
<i>RZ02</i>	826
<i>RZ03</i>	826
<i>RZ04</i>	826
<i>RZ10</i>	826
<i>RZ11</i>	378, 826
<i>RZ12</i>	827
<i>RZ20</i>	70, 334, 827
<i>SAT</i>	240, 254
<i>SBGRFCCONF</i>	344
<i>SBGRFCMON</i>	344
<i>SC11</i>	252
<i>SE11</i>	498, 522, 524, 827
<i>SE12</i>	239
<i>SE14</i>	523, 827
<i>SE15</i>	827
<i>SE16</i>	724, 827
<i>SE24</i>	252, 827
<i>SE30</i>	69
<i>SE37</i>	252, 321
<i>SE38</i>	252, 291, 458, 508, 827
<i>SE80</i>	384, 510, 827
<i>SEU</i>	827
<i>SHMA</i>	578
<i>SHMM</i>	578
<i>SICF</i>	242, 376, 377, 381

- Transaction (Cont.)
- SITSMON* 827
 - SITSPMON* 69, 386
 - SMO1* 827
 - SMO2* 827
 - SMO4* 328, 827
 - SMI2* 464, 478, 827
 - SMI3* 110, 126, 318, 827
 - SM21* 110, 126, 271, 827
 - SM24* 577
 - SM30* 524
 - SM36* 827
 - SM37* 738, 756, 827
 - SM49* 828
 - SM50* 69, 122, 249, 290, 302, 321, 327, 378, 458, 769, 828
 - SM51* 79, 122, 129, 828
 - SM56* 471, 828
 - SM58* 342, 828
 - SM59* 330, 331, 334, 343, 828
 - SM63* 828
 - SM65* 828
 - SM66* 122, 126, 302, 321, 327, 378, 769, 828
 - SM69* 828
 - SMGW* 328
 - SMICM* 69, 132, 377, 378, 828
 - SMLG* 128, 302, 308, 828
 - SNRO* 471
 - SPRO* 476, 828
 - SQLM* 828
 - STO1* 828
 - STO2* .. 69, 113, 208, 280, 284, 287, 289, 478, 566, 576, 828
 - STO3* .. 69, 137, 138, 168, 302, 329, 336, 378, 721, 724, 727, 828
 - STO3G* 69, 167, 168
 - STO3N* 138, 210
 - STO4* 93, 119, 828
 - STO5* 228, 348, 483, 828
 - STO6* 68, 79, 85, 88, 103, 119, 125, 208, 280, 350, 791, 828
 - STO6N* 79
 - STO7* 828
 - ST10* 828
 - ST11* 828
- Transaction (Cont.)
- ST14* 829
 - ST22* 286, 829
 - STAD* 137, 161, 168, 329, 338, 347
 - STAD/STATS* 829
 - STAT* 558
 - STATTRACE* 167, 168
 - step* 141, 142, 265
 - STMS* 829
 - STUN* 68, 829
 - SWLT* 103, 829
 - SXMB_MONI* 829
 - TREXADMIN* 125, 829
 - TUO2* 829
 - VAO1* 159, 162
 - variant* 509
 - VLOIN* 162
 - Transformation 701
 - Transmission Control Protocol/Internet Protocol 846
 - Transport 846
 - Transport Management System 829, 846
 - Tree 735
 - Tree control 344
 - TREX 38, 843, 846
 - Administration* 829
 - compression* 588
 - search functions* 597
 - tRFC table 343
 - Troubleshooting 59
 - T-shirt sizing 192
 - Tune summary 113
 - Tuning 539
 - application* 74
 - application tuning* 62
 - program optimization* 63
 - technical* 62, 66
 - Tuple reconstruction 588
- U**
- Unicode 212, 264, 282
 - conversion* 213
 - Uniform Resource Locator 846
 - Unit dimension 707
 - UNIX 306

- UNIX standard implementation 278
 - Update 110, 316, 353
 - asynchronous* 317
 - deactivated* 126
 - dispatching* 319
 - local* 321
 - priority decision* 320
 - records* 827
 - request* 317
 - service* 298, 302
 - synchronous* 321
 - table* 317
 - update time* 322
 - update type* 319
 - V1 update* 319
 - V2 update* 319
 - V3 update* 319
 - work process* 317, 318
 - Upgrade 212
 - URL 846
 - User
 - active* 150
 - call* 811
 - context* 143, 265, 311, 846
 - exit* 192
 - monitor* 328
 - occasional* 150
 - power user* 151
 - profile* 151, 169
 - transactional* 150
 - User interface, graphical 840
 - User kernel thread (UKT) 768
 - User List 827
 - User mode
 - Expert* 139
 - Service Engineer* 140
- V**
- Value per unit 698
 - Variant 245
 - Verification 66
 - Virtual
 - main memory* 293
 - memory* 263
 - provider* 698
 - Virtual Machine Container (VMC) 298, 846
 - VMC service 298
- W**
- WAN 847
 - Web application 360, 369
 - Web browser 837
 - Web Dynpro 847
 - Web Dynpro ABAP 45, 242, 357, 359
 - runtime analysis* 382
 - Web Dynpro for Java 357, 359
 - Web GUI 360
 - Web reporting 700
 - Web service 45, 195, 359
 - monitor calls* 381
 - Web transaction 176
 - WHERE clause 493
 - Wide Area Network (WAN) 847
 - Wide area network (WAN) 51
 - Wily Introscope 69
 - Windows 277, 306
 - Work group 307
 - Work process 126, 128, 267, 298, 304, 352
 - ABAP trace* 241
 - increasing the number* 354
 - Java Virtual Machine* 415
 - monitor* 321, 378
 - nondialog work process* 274
 - overview* 69, 97, 122, 229, 249, 290, 302, 327, 458, 769, 828
 - overview, global* 828
 - Work process type 140
 - Workload analysis .. 27, 62, 135, 136, 142, 151, 447, 449
 - Workload monitor 69, 137, 139, 142, 158, 210, 302, 329, 378, 722, 828
 - Central* 168
 - central* 167
 - Java* 172
 - role* 139
 - SAP BW system workload* 727
 - Workload overview 169
 - World Wide Web 847
 - WWW 847

X

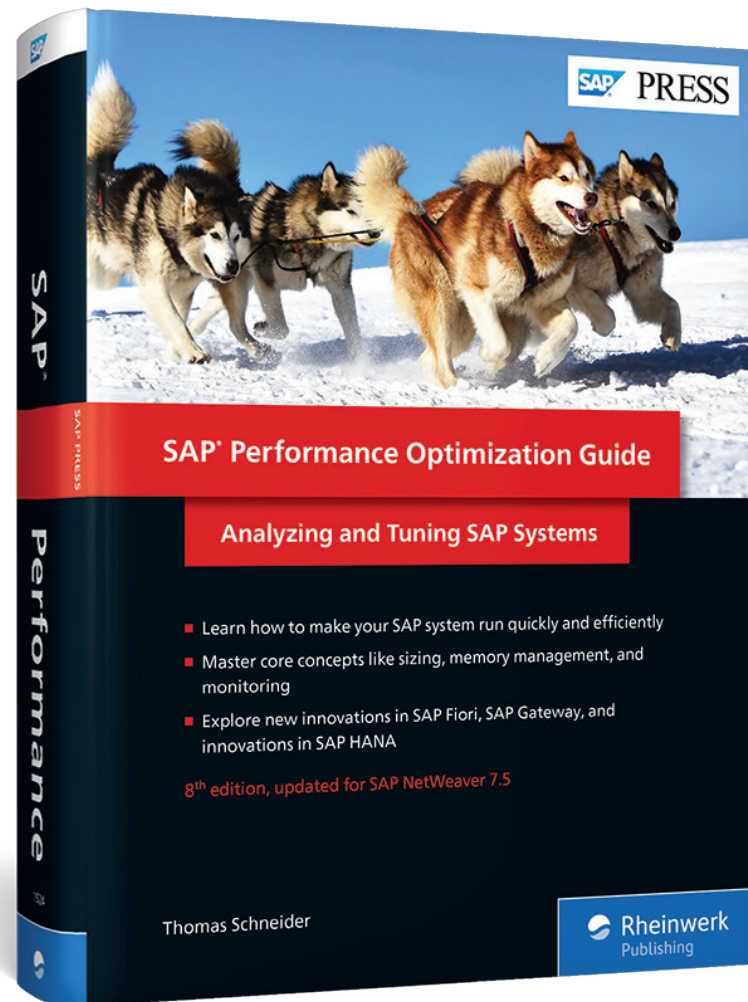
XML 847

Y

Young generation 407, 411

Z

Zero Administration Memory
Management 117, 268, 280, 293



Thomas Schneider

SAP Performance Optimization Guide: Analyzing and Tuning SAP Systems

892 Pages, 2018, \$89.95

ISBN 978-1-4932-1524-9

 www.sap-press.com/4370

Dr. Thomas Schneider started his career at SAP AG in 1996. Among other things, he was in charge of the Center of Expertise for Performance in the Service & Support group and the responsible Support Alliance manager for key accounts. Since 2004 he has worked in the Research & Breakthrough Innovation group, where he first was responsible for IT Service & Application Management. Since 2009 he has focused on the Partner Development Infrastructure in the SAP Cloud organization as the responsible architect.

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.