SQL

COMMAND

ANALYSIS

User's

Guide

Software
Product
Research

SQL Command Analysis
Release 3.4

© Copyright Software Product Research 1996 - 2000

All product names, mentioned in this manual, are trademarks of their respective owners.

Using this manual

Getting started

Page 3 and following explain shortly how to use SQL/Command Analysis.

Functional Description

Turn to page 7 for a full description of SQL/Command Analysis.

Using SQL/CA

For complete information on SQL/Command Analysis usage, read page 41 and following.

Interpreting the analysis report

Page 87 and following give a detailed description of the command analysis report and its clauses.

Glossary

The glossary explains DB2/VM related terminology. See page 133 and following.

Warning messages

Go to page 149 for a full explanation of the warning messages found in the analysis report.

1 Getting started

1.1 Invoking the SQL/Command Analysis Menu

Type **SQLCA** on the CMS prompt. You get the following menu screen.

SQL Command Analysis Menu

1 Analyze CMS Source Program
2 Analyze CSP Application
3 Analyze REXX Application
4 Analyze NATURAL Application
5 Analyze Stored SQL Query
6 Analyze SQL Package
7 Interactive Command Analysis
8 Table Oriented Analysis
9 Query Analysis Archive
10 SQL/CA Utilities Menu

Select Menu Option

Choose the desired option, for example **1** if you are going to analyze a source program. Proceed with ***Specifying Analysis Options*** on page 4.

1.2 Invoking SQL/Command Analysis from a CMS filelist

When you have a CMS filelist displayed on your screen, type **SQLCA** on the line for the source file that you want to analyze. Proceed with ***Specifying Analysis Options*** on page 4.

1.3 Invoking SQL/Command Analysis from an XEDIT session

When you are in an XEDIT session and you want to analyze the program you are editing, type **SQLCAP** on the XEDIT command line. Proceed with ***Specifying Analysis Options*** on page 4.

1.4 Specifying analysis options

Command analysis options are specified on the following screen:

```
        Analyze CMS Source Program

           Program name to analyze :    ........ ........ A1
          Archive analysis results :    YES
              Edit analysis report :    YES
             Print analysis report :    NO
                   Uppercase print :    NO
      Save previous analysis report :    NO
  Update statistics before analysis :    FIRST
  Update statistics for all columns :    NO
         Delayed analysis by userid :
     Copy analysis report to userid :
      Copy source extract to userid :
                Connect to database :
                        as SQL user :
                      with password :
```

When analyzing a CMS source file, type its name (for example **TEST2 COBOL)**. When analyzing CSP applications, QMF queries or packages, type the corresponding name. When you invoked analysis from a CMS Filelist or from XEDIT, the name of the source file will already be on the screen.

The remaining screen fields define **execution options**. Initially, they contain the system defaults, defined when SQL/CA was installed. Overtype them if necessary. For an in-depth description of the execution options, please refer to page 46.

Press **ENTER** to start the analysis.

1.5 Viewing the analysis report

When analysis is complete, the analysis report is displayed on the terminal, using **XEDIT**. Initially, you see the last page of the report, the **analysis summary**.

```
ANALYSIS SUMMARY

     Highest command cost : 1169
            DBspace scans : 1
Non-selective index scans : 4
     Average access score : 2.8
      Severity 3 warnings : 10
      Severity 2 warnings : 5
      Severity 1 warnings : 8
```

Use XEDIT commands to browse the report or to search for words or phrases. For example: type the **TOP** command to view the first page of the analysis report.

```
Cust : CATEST
File : SQLCADMO PLIOPT   A1    Line nr 2                          User=SQLAF
Date : 94/04/19  16:10:55      ArchQno = 501                     Dbase=CATEST
 COMMAND EXECUTION STRUCTURE BY BLOCK AND PARENT
 BLK Par  Command Text

   1   0  SELECT * FROM SQLDBA.INVENTORY FOR UPDATE OF PARTNO,DESCRIPTION,
          QONHAND

          COMMAND COST SUMMARY
          Total Command Cost : 18 (ISQL like Cost : 1)

          COMMAND EXECUTION STRUCTURE

          Estimated number of rows SELECTed : 22 out of 22
          Estimated global command filter factor : 1
 >>> AW86  Estimated times predicate conditions satisfied : 22

          COMMAND EXECUTION DETAIL

          Plan : 1
 >>> AW80     Access : Scan of DBspace SAMPLE
                      DBspace pages scanned : 8
                      DBspace scan productivity : 13%

1 Glossary     2 -> Warning   3 Quit         4 <- Command   5 <- CmdBlock
6 ->> Warning  7 PageUp       8 PageDown     9 Detail      10 -> Command
```

1.6 Examining the analysis report

In addition to the usual XEDIT commands, you can use a number of PF-keys to speed up your work. This is a list of those keys, with their meaning.

PF1 Displays the SQL/CA glossary. The glossary is an online book, that contains the following sections of this manual:

- INTERPRETING THE ANALYSIS REPORT
- SQL/CA GLOSSARY
- SQL/CA WARNING MESSAGES
- SQL/CA OBJECT NOTES
- SQL/CA MESSAGES

If the cursor is located on an analysis warning message when you press PF1, the glossary is automatically positioned to the description of the warning in the "Warning Messages" section. (A warning message line starts with one or more > signs and usually has a message number in the form **AWxx**). The glossary too is displayed using XEDIT. You can use XEDIT commands to browse the book or to search for specific items.

PF2 Locates the next analysis warning on the current page.

PF3 Quits the report editor.

PF4 Locates the begin of the current SQL command being analyzed.

PF5 Locates the begin of the current SQL subquery, if the command contains subqueries.

PF6 Locates the next analysis warning on the following page.

PF7 Displays the previous report page.

PF8 Displays the next report page.

PF9 Switches to condensed or non-condensed report format, in a flip-flop manner.

A condensed report shows the SQL commands and the generated warnings only. A non-condensed report shows the entire report. You might choose to start with condensed mode, so that you have all warnings for a command on the same screen. If you want to see the analysis for that command in detail, you can switch to the non-condensed format.

Note that your last format choice is remembered by SQL/CA across sessions. It will be used automatically when you display your next report.

PF10 Locates the next SQL command in the report.

2 Functional description

SQL Command Analysis **('SQL/CA')** is a software tool for DB2/VM (previously known as SQL/DS).

Its primary purpose is to assist SQL programmers in producing efficient and well performing SQL applications, by providing analysis services during program **development**. SQL/CA encourages SQL developers to systematically analyze their applications in view of optimal SQL coding. This approach will detect poorly designed SQL at an early stage and **prevent** many performance problems, that otherwise would appear only when the program is moved into production.

SQL/CA examines the *source text* of the program, and therefore is able to signal SQL performance deviations that cannot be detected by means of the traditional DB2 tuning procedures.

SQL/CA presents its findings in the form of an **analysis report**. The report is easy to read: it does not require a highly technical background to be understood.

SQL/CA operates in the VM/CMS environment and is capable of analysing:

- Assembler, Cobol, Fortran or PL/1 source programs, residing on a CMS minidisk
- CSP applications residing in a CSP MSL
- REXX/SQL applications[1]
- NATURAL applications
- EASYTRIEVE applications
- ISQL routines and QMF procedures
- CMS files containing SQL commands in DB2 Database Services format
- Any program text file forwarded to the SQL/CA analysis server
- Designated DB2 packages in the currently connected database
- DB2 packages that contain references to a designated table or view.

Except when analyzing DB2 packages, SQL/CA operates on the **text of the source program**. DB2 packages are *not* referenced during command analysis: application programs need *not* to be prepped in order to be analyzed. However, the SQL commands must have correct syntax and the DB2 objects (tables etc.) used by the application, must exist in the database connected during analysis.

During analysis, some simulation facilities are provided, for example, to create an index and observe its effects on access path selection.

Although SQL/CA is designed for DB2/VM and executes in the CMS environment, it may also be helpful for VSE users, provided that:

- the source programs are available as CMS files or in a CMS CSP/MSL
- the VSE system accesses a DB2/VM database through IBM's "Guest Sharing" facility.

[1]Support is provided for both the traditional RXSQL and the newer EXECSQL interface.

When analyzing a program, SQL/CA extracts the SQL commands, invokes DB2 **EXPLAIN** and performs **text analysis**.

2.1 Command extraction

All SQL commands that can be explained and the related host variable specifications are extracted from the program source text or from the unloaded DB2 package and stored into an *extract* file. SQL commands that can be explained are: SELECT, INSERT, UPDATE, DELETE and DECLARE CURSOR FOR SELECT | INSERT.

2.2 SQL EXPLAIN

The SQL **EXPLAIN** command is issued for all the extracted SQL commands and the resulting SQL explain table data are converted from their encoded and numeric format into a textual and easily readable **analysis report**. This report is stored on the user's A-disk as a CMS file with filetype "SQLCA" and with a filename identical to the filename of the analyzed source.

For each SQL command, the analysis report provides following data obtained from the SQL Explain tables:

2.2.1 Command text

Prints the SQL command in a manner that reflects the logical execution tree structure. If the command contains subqueries, they are formatted by command block and calling block. The indention level used when printing a given subquery, corresponds to the nesting level of the query within the logical execution tree.

2.2.2 Command Cost

Reports the cost of command execution, as estimated by DB2. If the command consists of multiple queries, a number of calculations are carried out in the **Cost** paragraph, in order to show the cost associated with each subquery, taking into account its execution characteristics within the logical execution tree. SQL/CA will also indicate the subquery with the highest cost. If our SQL/MF monitor program product has been installed[2] and if the user has access to the monitor tables during analysis, certain **real-time execution statistics** will be included in the report.

[2]See *SQL/MF Integration* on page 41.

2.2.3 Command Structure

Reports:
- from which query block a dependent block is called, if the command contains embedded queries
- the estimated number of rows processed
- the estimated number of execution iterations

  If the command consists of multiple queries, the **Structure** paragraph will compute the projected execution properties of each subquery, taking into account the execution properties of all its logical predecessors within the execution tree.

2.2.4 Command Plans

Reports:
- whether the command is executed
  - by DBspace scan
  - by index-only scan
  - by fully-qualified index scan
  - by selective or non-selective index scan
  - by view materialization
- the name of the plan index, if any
- the number of matching index-key columns
- the join method if the command implies joining
- the number of rows in the inner and outer join table
- whether sort operations will be performed

If the command is executed by DBspace scan, the **Plan** paragraph will compute the DBspace scan productivity as the estimated percentage of table data accessed during the relational scan. If indexed access is used, the column definitions of the index and its characteristics (first or not, clustered or not, unique or not) are shown.

2.2.5 Command Reference

Reports:
- the column names appearing in the WHERE or UPDATE SET clauses
- the selectivity (filter factor) of those columns
- the columns appearing in a sargable predicate
- the columns used during a join
- the columns used for ORDER or GROUP BY
- the columns updated by the command

The **Reference** paragraph shows the explain column numbers as names. If a view is being explained, the base table name and column names from explain are related to the corresponding viewname and view column names, as used in the application.

2.2.6 Explain Warnings

In addition, an SQL command will be flagged with the warning character > and a warning message code:

- if it is executed by DBspace scan
- if it is executed by a non-selective index scan
- if data pages must be accessed to resolve the predicate conditions
- if no indexes exist for the table
- if no highly clustered indexes or unique indexes exist for the table
- if no indexes exist created using the current DB2 release
- if view materialization occurs (DB2 Version 3 only)
- if all rows of the table are being accessed
- if a subquery is executed at each invocation of its parent block
- if the outer join table is larger than the inner join table

A severity code 1, 2 or 3 is associated with each of the warnings and printed as 1, 2 or 3 > signs. The highest severity code 3 is assigned to conditions that are assumed to increase database I/O during command execution.

Note

If the command contains **view references**, SQL EXPLAIN operates using the underlying real tables. Therefore, SQL/CA expands such commands by inserting the view definition(s) until the resulting command contains real table references only. The original and the expanded commands are printed and the expanded command is used for SQL/CA text analysis.[3]

[3]If the original command is a SELECT *, the * is not expanded, due to the maximal command length restrictions.

2.3 Text analysis

SQL/CA text analysis examines the application program for adherence to the SQL coding and performance rules described in the IBM "performance tuning" manuals and detects command specifications that lead to suboptimal performance. Text analysis will specifically warn the following performance exposures:

- Indexing columns are being updated by the command.

- Table indexes do exist, but the command predicate does not contain index column references.

- Table indexes do exist, but the command predicate contains incomplete index column references, that is, one or more indexing columns are not specified for the **plan** index (the index that is effectively used during execution).

- The predicate uses indexing columns in expressions such as "BALANCE*2 > 1000". This may disqualify index use.

- The datatype compatibility rules are violated in a combination of two syntactical elements. For example: the datatype of a host variable is not compatible with the datatype of the corresponding table column. See “Datatype evaluation table” on page 185. Moreover, when the column is DECIMAL and the host variable DECIMAL, INTEGER or SMALLINT, the scale compatibility rules must be observed, as described on page 185.

- The data length compatibility rules are being violated for non-decimal columns. The lengths of the syntactical elements A and B are considered compatible when length(A) >= length(B), except for VARCHAR columns with length < 254 and VARGRAPHIC columns with length < 127, which are compatible with all character values, fixed or variable of any length.

- The data precision compatibility rules are being violated for decimal columns. The precisions of the decimal elements A and B are considered compatible when precision(A) >= precision(B).

- The data scale compatibility rules are being violated for decimal columns. The scales of the decimal elements A and B are considered compatible when scale(A) = scale(B) when B is not a constant. When B is a constant the compatibility rule is scale(A) >= scale(B).

- SQL predicate operators used in the command are no index keymatching candidates, that means, the key values cannot be used to directly fetch an index entry. For instance: an = operator is key-matching, a ¬ = operator is not. For a complete list of index keymatching operators, see the table on page 186.

- The command predicate specifications are not "sargable", that is, they cause the predicate to be evaluated by RDS (the DB2 Relational Data System) and not by DBSS (the DB2 Database Subsystem). This involves additional CPU time consumption. A number of command operators is not sargable. For a list of them, see the table on page 186.

- The predicate is not sargable because an indicator variable is used with a host variable in an EQUALS predicate for a column that does not permit nulls.

- The left and right hand expressions of the predicate refer to columns of the same table (T1.COL1=T1.COL2). Such predicates are not sargable and not keymatching.

- The command contains suboptimal SQL verbs such as "OR" or "NOT".

- The entire command predicate has become non-sargable, due to a non-sargable predicate connected by OR.

- The default filter factor is used for **range** operators (such as > , LIKE or BETWEEN) because the predicate contains host variables. Due to the default selectivity rules applied by the DB2 optimizer, such specifications may lead to index disqualification. The default filter factor used for the predicate is shown: it depends on both the range operator type and the number of distinct column values. For more information, see the table on page 187.

- Range predicates such as >, >=, <, <= are used. These are less selective than BETWEEN.

- A predicates uses indicator variables. Such predicates are neither key-matching or sargable.

- A join predicate violates the datatype and data length rules. Contrarily to normal predicates (which must be compatible), join predicates require that the datatypes, lengths, precisions and scales of the join columns be identical.

- A join command omits necessary search conditions. If, in addition to the join predicate, the command states more "local" predicates, the latter should normally be stated for both join columns. The DB2 optimizer will automatically add a missing "local" condition, but this will be done only for an equijoin and when the local condition is sargable.

Many of the above conditions disqualify selective index use, that is, DB2 may decide to read the entire table, using either an index scan or a DBspace scan (in the latter case, all tables in the DBspace are scanned). For each of the above conditions, a specific warning message is inserted in the report.

These warning message can be searched by code in the SQL/CA **Glossary**, while examining the analysis report on the screen. The Glossary describes the detected performance exposure in full detail and suggests corrective action. It contains a number of tables that illustrate the rules followed by DB2 in evaluating the command predicate. The Glossary can be found on page 133 of this publication.

A severity code is associated with each warning issued, to indicate its importance. For warnings involving predicate columns, a distinction is made between indexing and non-indexing columns. Since a performance exposure concerning an indexing column will have a more considerable performance impact, a severity 3 code is signalled. For a non-indexing column, a severity code 1 is associated with the warning.

2.4 Object lists

Following the analysis report, lists are produced to describe the physical characteristics of the tables, table columns and indexes used by the program. These characteristics and statistical data are taken from the DB2 catalogs. If no statistics are available for a table (no UPDATE STATISTICS command executed), the corresponding statistic values are unknown and displayed as **?** (a question mark).

The **table section** shows for each table accessed by the application:

- the name of the table's DBspace
- the average rowlength
- the number of table rows
- the **effective** number of rows per DBspace page, taking into account the effective freespace class
- the number of table pages
- the percentage of DBspace pages occupied by the table
- the number of dependent and parent tables in a referential integrity environment

The **index section** shows for all indexes defined on all tables in the table section:

- the "clustered" attribute
- the clustering ratio
- the "first" attribute
- the "unique" attribute
- the first key count
- the full key count
- the number of leaf pages in the index
- the number of levels in the index

The **index column section shows** for all columns of all indexes appearing in the index section:

- the percentage of distinct column values
- the first eight bytes of the second lowest value in the column.
- the first eight bytes of the second highest value in the column.
- the value of the column at the 10th percentile[4]
- the value of the column at the 50th percentile
- the value of the column at the 90th percentile
- the most frequent column value and its percent frequency
- the second most frequent column value and its percent frequency

[4] If a table has N rows and all N values of the column are arranged in ascending order, the value shown is at position **0.1*N** (for the 10th percentile), **0.5*N** (for the 50th percentile) or **0.9*N** (for the 90th percentile).

2.5 Object notes

SQL/CA analyses each DBspace, table or index used by the application and (if applicable) issues following notes at the end of the analysis report:

- the freespace in a DBspace may not be used during insert activity
- a small DBspace does not have the "row" locklevel
- the DBspace is located in the same storage pool as the DB2 catalogs
- a table has more than 10% overflow rows
- a small table has indexes other than those required for referential integrity or primary keys (small tables are best processed by DBspace scan)
- VARCHAR or VARGRAPHIC columns are part of an index definition: such indexes will never be used for an index-only scan
- the first column of a composite index is not the most selective one

2.6 Analysis summary

At the end of the report, a summary is printed that shows the highest SQL cost value encountered in the program, the number of DBspace and index scans and the number of SQL/CA warnings, by warning severity.

2.7 User interface

SQL/CA functions are normally invoked using a menu driven interface. Analysis of sources residing in CMS files can also be requested from a CMS Filelist or from a user CMS EXEC.

2.8 SQL/CA help

Online help is provided at all levels of the SQL/CA user interface. The product also provides an online **Glossary** that explains the messages issued during analysis and defines most of the DB2 technical terminology and the SQL/CA specific terms. This glossary may be printed or edited. It can be also displayed by function key while the analysis report is being examined.

2.9 Additional processing options

A number of **additional processing options** can be entered on the SQL/CA analysis control screen.

2.9.1 Archiving Analyzed Commands and Results

Enabling the option causes the extracted SQL commands, their EXPLAIN results and eventual SQL/CA warnings to be stored into the SQL/CA **archive tables**. As more applications are being analyzed, the archive tables will contain a record of all application SQL commands with their execution characteristics, as obtained by SQL Explain and SQL/CA text analysis. The archive may then prove useful for the performance management of SQL applications.

The "key" of the archive tables is composed of the filename of the application program, the connected database name and the analysing userid. Thus, the same command analysed in different databases or by different users, will generate multiple archive entries.

When analysis results are stored for a given command, an **archive querynumber** is assigned to the command and printed on the analysis report.

A number of archive queries is provided with the product in order to report SQL commands

- by cost
- by access type (DBspace scan, index scan, index access)
- by access method (join, sort)
- by SQL/CA warning code
- by SQL command
- by subquery
- by response size
- by cross reference between programs and tables, indexes or columns referred to

Each database may have its own archive tables. Alternatively, a common database may be designated to hold the archived data. In the latter case, SQL/CA will automatically perform all necessary database switching.

An archived SQL command can be imported into a CMS file and re-analyzed.

Notes:

1. The archive query and archive import functions require that the user has explicit SQL SELECT privileges on the archive tables.

2. When the archive tables are shared between databases, the user performing analysis must be able to connect to the archive database under his current DB2 userid and must have run privilege on the SQL/CA packages in that database.

2.9.2 Editing the Analysis Report

If requested, the analysis report will be XEDITed after command analysis. While the report is being edited, following PFkeys have a specific meaning:

PF1 displays the SQL/CA glossary; if the cursor is located on an analysis warning message, the glossary is positioned to the description of the warning
PF2 locates the next analysis warning on the current page
PF3 quits report editor
PF4 positions to the begin of the current SQL command being analyzed
PF5 positions to the begin of the current SQL subquery if the command contains subqueries.
PF6 locates the next analysis warning on the following page
PF7 displays the previous report page
PF8 displays the next report page
PF9 switches to condensed/non-condensed report display, in a flip-flop manner
PF10 locates the next SQL command

2.9.3 Printing the Analysis Report

If requested, the analysis report will be printed after analysis using the current spool options of the virtual printer. If the report should be printed by RSCS, it is the user's responsibility to setup RSCS routing before invoking analysis.

Uppercase Printing

When used in conjunction with the print option, the analysis report will be translated to uppercase in view of processing by printer devices without lowercase font.

2.9.4 Saving the Analysis Report

When an application program is being analyzed and an analysis report already exists, the existing report will be replaced by the new one. The save option will rename the existing report before starting analysis.

2.9.5 Updating Statistics before analysis

The SQL Explain command and the DB2 Optimizer rely on statistical data recorded in the DB2 catalog tables. To obtain reliable explain results, it is important that these statistics be up to date. If enabled, the update statistics option will automatically perform an "UPDATE STATISTICS" command prior to command explain for all tables referenced by the application. Auto-statistics may be unconditional or be requested on a "first time" basis, that is, when statistics have never been updated for the table.

Updating statistics for all columns

When used with Auto-Statistics, the option will perform an UPDATE STATISTICS for all table columns. By default, statistics are updated for the indexing table columns only. Updating all column statistics requires an additional DBspace scan, but provides statistical information for all table columns.

Note:

Performing update statistics for large tables may be time consuming and lock out other DB2 users trying to access the table. SQL/CA contains an automated update statistics facility which can be run during off-shift.

2.9.6 Command Analysis in server mode

Analyzing applications containing many SQL commands may be time consuming and cause additional contention on the DB2 catalog tables. Therefore, an option has been provided to forward the analysis request to a server machine, executing the SQL/CA analysis program in disconnected mode. Moreover, the server can be tailored to execute the analysis requests at specific times (during offshift for instance).

Requests for analysis may be sent by non-VM clients, capable of storing data into the VM reader queue. Server-mode analysis can be done under the userid (and DB2 privileges) of the sending user or under the userid of the server, which may have broader DB2 privileges.[5] After server-mode analysis, the report is returned to the virtual reader of the requesting user.

Note:

SQL/CA uses CMS SENDFILE facilities when forwarding the source to the server. This implies that the specified servername may be either a VM machine name or a CMS "nickname", defined in a CMS "NAMES" file. In the latter case, a network node can be defined for the nickname and the server may reside in any node of the network.

[5]Since the DB2 authorization schemes also apply to the EXPLAIN command, developers can analyze only against the tables they are authorized on. On the other hand, it may be desirable to analyze SQL commands against tables in the operational environment. By granting operational database access to the SQL/CA batch userid, developers are able to analyze in a life environment, without having privileges themselves in the operational database.

2.9.7 Copying the Analysis Report for another user

This option allows a developer to forward the analysis report to another user (the DBA for instance) for further examination. The copy is sent only when warnings of severity 2 or 3 have been issued.

2.9.8 Copying the Source Extract for another user

This option allows the user to send the SQL commands and host variables extracted from his application, to another user for analysis by SQL/CA in that user's machine (and with that user's DB2 privileges).

2.9.9 Connecting a DB2 database or userid

By default, analysis is performed in the current database and with the default DB2 userid. Connection to another database and/or DB2 userid may be requested during analysis. When connecting another userid, the password of that user must be supplied. Please note that the connected database server should be a DB2/VM server.

If server-mode analysis has been requested, analysis is performed using the default connections of the server. If these are not satisfactory, the desired database and/or userid must be supplied when the server-mode analysis request is being made. You can overwrite the databasename, the username or both.

The last databasename, userid and password supplied are kept across SQL/CA sessions.

2.10 Simulation facilities during analysis

When analyzing a CMS application, XEDIT can be invoked from the initial selection screen. This allows to modify the application SQL and verify the result by requesting a new analysis.

When analyzing CMS applications or DB2 packages, the initial selection screen allows to create or drop an index. After creating an index, its effect on access path selection can be verified by requesting a new analysis.

2.11 Automated table statistics

SQL/CA provides a program that automatically updates the statistics for designated tables in a database, either on a periodical basis or depending on the growth rate of the tables. When statistics have been updated, packages depending on the affected tables will be automatically invalidated and reprepped. This ensures that these packages take advantage of the optimal access paths available as a result of the statistics update process. Following statistic updating, the program will list the state of the first index, the current number of table rows, the current overflow row percentage and the table growth rate if applicable. If defined, a user exit will be invoked after updating the statistics and before reprepping the packages. The arguments to the exit include the table and DBspace name and some statistical data from the DB2 catalogs, such as the clustering state of the first index, the number of table rows, the number of table overflow rows and the table growth rate. Depending on this information, the exit may start specific processing, such as reorganizing the table. After statistics, users may request **automatic index reorganization** in order to remove empty index pages for example. However, to recluster an unclustered index, a table reorganization must be performed.

The statistics update routine is usually invoked from the batch SQL/CA server and may be tailored to execute during specific periods of time (off-shift).

2.12 DB2 program monitoring

DB2/VM may automatically change the data access strategy of a stored package depending on the state of the database objects used by the program. For instance, indexed access in a given program command may be changed **without warning** to DBspace scan access, due to an increase in table rows or changes in index definitions. Such event may have a serious effect on program response time. The SQL COMMAND ANALYSIS **program monitoring** facility will detect and report such unforeseen changes in access method and cost, so that database administration may proceed to problem determination - using the monitor analysis results - before users are affected by the response time impact. Indirectly, program monitoring allows to perform command analysis for packages in the database, without manual intervention. The facility should run on a periodical basis, preferably in conjunction with the auto-statistics function described above.

2.13 DB2 data modelling facility

It is desirable that the development databases resemble the production databases as much as possible. This ensures that access paths chosen by DB2 for the test programs are identical to those in the production system. However, completely replicating the operational data in the test database usually is not practical or feasible. In fact, it is not necessary : DB2/VM allows database administrators to modify certain columns in the catalog tables, in order to create a **model** of a production database on a test system. SQL statements in the test system are then executed using the access path that would have been chosen in the production system. Manually modelling the catalogs is time-consuming and requires knowledge about DB2 internals. Therefore, SQL/CA offers a utility program that can be used to quickly copy the catalog statistics from one system to another.

The data modelling facility can also be used to model **future** production systems and to test *what if* scenario's. For example: how will a program behave when a given table becomes very large?

2.14 SQL/CA software prerequisites

- VM/SP Release 5.0(and later) or VM/XA Release 2.0 (and later)
- VM/ESA Release 1.0 (and later)
- VM/CMS Utilities (if the REPORTWAIT option is used)
- VM/REXX (if REXX/SQL applications are analyzed)
- DB2/VM Version 3 and later

3 SQL/CA INSTALLATION

3.1 Installing the CMS product material

This installation step loads the SQL/CA modules, execs, helpfiles and other material from the distribution medium to the public disk or directory chosen as residence for SQL/CA.

The same installation procedure is used for a first-time installation and for installing updated SQL/CA versions. The DB2/VM related installation must be performed for each database where SQL/CA will be executed.

3.1.1 Prerequisites

1. Ensure that the VM userid performing the installation has write access to the public minidisk or directory that will contain SQL/CA.
2. Ensure that enough space is available on that minidisk or directory. See "SQL/CA Size Estimates" on page 33.
3. Ensure that you have access to the DB2/VM production minidisk.
4. If SQL/CA has been received on tape, ensure that a tape or cardtridge device has been attached to the virtual machine with virtual address 181.

3.1.2 Installation

1. Access the public SQL/CA minidisk or directory with CMS filemode A.
2. If you received the product on tape, issue the command: **TAPE LOAD * * A**.When the TAPE LOAD command has completed, you do no longer need the distribution tape.
3. If you received the product on another medium, use its **INSTALL** procedure to load the product code on the SQL/CA minidisk.

3.1.3 Linking the SQL/CA Modules

1. With the public SQL/CA minidisk or directory accessed as "A" enter **SQLCAPI** on the CMS prompt.
2. On the next panel, choose "**CMS Installation**".
3. The SQL/CA modules are now linked.
4. On a first-time installation, the default SQL/CA processing options file is stored on the SQL/CA minidisk or directory as SQLCA OPTIONS. When SQL/CA has been installed previously, the existing SQLCA OPTIONS file (which may have been customized) will not be affected. Instead, the distributed options file will be stored on the minidisk as SQLCA STDOPT.
5. The message "*SQL/CA CMS Install complete*" appears at the end of the installation.

3.1.4 Customizing the SQL/CA Default Processing Options

After a first-time installation, you may wish to update (using XEDIT) the default SQL/CA processing options as recorded in the CMS file "SQLCA OPTIONS".

For a description of the control statements used in the OPTIONS file, please refer to the chapter **System Processing Options File** of the SQL/CA User's Guide.

3.2 DB2/VM related installation

The DB2/VM installation procedure will perform one or more of the following steps in one or more named databases:

1. Connect the target database.
2. Acquire the required DYNPRG DBspace and create the DYNPRG table
3. Optionally, acquire the ARCHIVE DBspace, create the archive tables and load the ISQL archive query routines.
4. Optionally, acquire the process control DBspace and create the process control table.
5. Load the SQL/CA packages and load the dynamic SQL/CA program statements in the DYNPRG table.
6. Grant EXECUTE authority on the SQL/CA packages to one or more users.

The installation procedures allows you to bypass one or more of the above steps.

When installation has been completed, you may wish to verify the installation process. You can do so by invoking command analysis for the sample program **SQLCADMO PLIOPT** that is part of the distributed software material. (You don’t need the PL/1 compiler for the analysis).

3.2.1 Prerequisites

1. You should have access to the SQL/CA minidisk or directory in any filemode, preferably different from A. If access is done using filemode A, the installation steps 4, 5 and 6 will permanently alter the DBspace acquisition model files SQLCADB1 DBS, SQLCADB2 DBS and SQLCADB3 DBS on the SQL/CA installation disk.
2. You should have access to the DB2/VM production minidisk.
3. Enough DBspaces of the appropriate size should be available for creating the SQL/CA tables. Please refer to the section "SQL/CA Size Estimates" on page 33.
4. You should be able to connect as DB2/VM user 'SQLDBA'.

3.2.2 Installation

Type **SQLCAPI** on the CMS prompt and choose "**SQL Installation**" on the next panel.

On the following panel, press PF3 to terminate installation or specify the following installation controls:

1. The name of the database where SQL/CA should be installed.

2. The password of the DB2/VM userid SQLDBA in that database. (DBA authority is required to acquire the SQL/CA DBspaces.)

3. If the **mandatory** DYNPRG DBspace has not been acquired yet in the current database, reply **Y** after the prompt ***"Create required SQLCA_DYNPRG?".***

4. If you intend to use the SQL/CA archiving or program monitoring facility and if the archive tables should reside in the current database (as specified in the DATABASE statement of the SQLCA OPTIONS file), reply **Y** after the prompt ***"Create optional SQLCA_ARCHIVE?".***

5. If you intend to use the **SQLCABAS** program for performing automated table statistics or program monitoring, reply **Y** after the prompt ***"Create optional SQLCA_CONTROL?".*** Contrarily to the archive DBspace, the process control DBspace cannot be shared between databases. Therefore, every database which will be connected to SQLCABAS (as a result of the CONNECT statement in the SQLCABAS control file) needs a process control DBspace.

6. To load the SQL/CA packages in the current database and to create the SQL/CA dynamic programs in the DYNPRG table, reply **Y** after the prompt ***"Load SQL/CA packages?"***

7. To grant EXECUTE authority on the SQL/CA packages in the current database, enter a DB2 userid after the prompt ***"Grant EXECUTE on SQL/CA to user :"***

Installation Procedure

1 **You replied Y after the prompt *"Create required SQLCA_DYNPRG".***
The DBspace acquisition command file SQLCADB1 DBS is copied to your A-disk and XEDITed. Specify the **STORPOOL** parameter and issue an XEDIT **file** command to start the DBspace acquisition and table creation. Since a minimal DBspace is sufficient for the DYNPRG table, the PAGES=128 parameter need not to be modified. PCTFREE=0 is specified because the DYNPRG is a read-only table. If SQL errors occur during this step, refer to paragraph 6 **"SQLDBSU Errors".**

2 **You replied Y after the prompt *"Create optional SQLCA_ARCHIVE".***
The DBspace acquisition command file SQLCADB2 DBS is copied to your A-disk and XEDITed. Specify the **STORPOOL** parameter and perform an XEDIT **file** command to start the DBspace acquisition and table creation. The PAGES parameter reflects the installation defaults (see the section "SQL/CA Size Estimates" on page 33 if you wish to modify this parameter). The DB2/VM default PCTFREE, PCTINDEX and LOCK parameters should be appropriate. This step will also store the ISQL routines for the SQL/CA archive queries in the SQLDBA.ROUTINE table. If SQL errors occur during this step, refer to paragraph 6 **"SQLDBSU Errors".**

3 **You replied Y after the prompt *"Create optional SQLCA_CONTROL".***
The DBspace acquisition command file SQLCADB3 DBS is copied to your A-disk and XEDITed. Specify the **STORPOOL** parameter and perform an XEDIT **file** command to start the DBspace acquisition and table creation. The PAGES parameter reflects the system default (see the section "SQL/CA Size Estimates" on page 33). The DB2/VM default PCTFREE, PCTINDEX and LOCK parameters should be appropriate. If SQL errors occur during this step, refer to paragraph 6 **"SQLDBSU Errors".**

4 **You replied Y after the prompt *"Load SQLCA packages".***
The SQL/CA packages are loaded in the current database and the SQL/CA dynamic programs are stored in the DYNPRG table.

5 **You entered a DB2 userid after the prompt *"Grant EXECUTE on SQL/CA to user".***
EXECUTE authority is granted on the SQL/CA packages to the named user.

6 After completion of the above steps, control is transferred to the initial installation panel, to allow for installation in another database.

7 **SQLDBSU Errors.**
The installation procedure performs DBspace acquisition and table creation by calling SQLDBSU. If an error is detected by SQLDBSU, the SQLDBSU control listing is XEDITed on the terminal. After inspecting it, you will be asked how this error should be handled.

Enter **A** to abort the installation procedure
Enter **C** to correct the error and XEDIT the SQLDBSU stream for correction.
Enter **I** to ignore the error and continue with the next instalation step.

3.3 VM/CSP considerations

When analyzing a CSP application, SQL/CA invokes CSP with an EZECIN command file that contains a print command for the named CSP application. The CSP invocation procedure is a REXX EXEC, called **SQLCACSP**. SQLCACSP on its turn calls another EXEC, named **SQLCACSI** for opening ('EXEC SQLCACSI OPEN') and closing ('EXEC SQLCACSI CLOSE') the CSP environment. The SQLCACSI EXEC initiates a standard CSP environment by executing the IBM EXEC's CSPDDLBL and USERDLBL on the CSPUSER 193 minidisk. This results in the definition of a read/write MSL on the user's 503 minidisk. The standard CSP invocation procedure then allows to analyze applications from that primary MSL. If you have a differently customized CSP environment (with project MSL's for instance), you may have to alter the **SQLCACSI** EXEC, in order to suit your CSP setup.

Alternatively, you can write a CMS EXEC which invokes a customized CSP environment. When a user invokes CSP application analysis, the name of this EXEC can be entered as a function parameter when performing CSP analysis.

Invocation EXEC entry values

When invoking the customized EXEC, SQL/CA passes 3 parameters:

- the filename and filetype of a CMS file built by SQL/CA; that file contains following command: **PRINT MEMBER(application_name);**

- the name of the CSP application to be analyzed

Prior to invocation, SQL/CA has issued following VM commands:

```
CHANGE RDR CLASS S HOLD
SPOOL 00E * CLASS S CONT
```

Invocation EXEC exit values

On exit from the customized invocation EXEC, SQL/CA expects the listing of the application to be analyzed in its reader or as a CMS file with filetype LISTING on minidisk A.

3.4 VSE/CSP considerations

There is no explicit support for VSE/CSP in SQL/CA. If applications to be analyzed reside in a VSE/CSP MSL, there are two solutions:

- If VM/CSP has been installed, the SQLCACSP EXEC may be customized to link to the VSE minidisk containing the VSAM catalog and the CSP library. Normal VM/CMS facilities can be used to achieve this.

- If VM/CSP has not been installed, the VSE/CSP application should be obtained by submitting a VSE job to print the application into the reader of the virtual machine requesting analysis. A skeleton job, named **SQLCACSV EXEC,** is provided with SQL/CA. You can use it as a base for installing your own support.

3.5 NATURAL considerations

NATURAL application analysis is controlled by the SQL/CA EXEC **SQLCAM12**.

Before starting analysis, an EXEC named **SQLCANT0** is invoked for defining the following NATURAL related global variables:

- NATURAL module name; if NATURAL is called by an exec, enter **EXEC nnn** as assignment value
- the CMS userid, the virtual address and the password for the minidisk that contains the NATURAL software material; if this minidisk is public, set the above 3 parameters to blank
- the NATURAL command separator (default is a comma)

Following default values for these variables are stored as REXX assignments at the beginning of SQLCANT0. Change the assignments to meet your installation requirements.

```
NATURAL_programname       = 'NAT22'
NATURAL_userid            = 'SAGDBA'
NATURAL_mdisk             = '196'
NATURAL_mdisk_password    = 'PASS196'
NATURAL_command_separator= ','
```

3.6 SQL/CA size estimates

3.6.1 CMS Minidisk or directory Requirements

The SQL/CA CMS material requires about 10 3380 cylinders on the public product disk or directory.

3.6.2 DYNPRG DBspace Size

The DBspace SQLCA_DYNPRG contains a table **required** during SQL/CA execution. The table is very small and read-only. The minimal DB2/VM space allocation (128 pages) is appropriate.

3.6.3 ARCHIVE DBspace Size

The DBspace SQLCA_ARCHIVE is **optional** and is used only when the analysis archiving option is enabled. It contains following tables:

SQLCA_Index Contains the archived SQL command text with redundant clauses (such as "INTO") omitted. An average command length of 512 characters is assumed in view of the size estimate.

SQLCA_Cost Contains the SQL explain COST data for the archived commands.

SQLCA_Plan Contains the SQL explain PLAN data for the archived commands.

SQLCA_Reference Contains the SQL explain REFERENCE data for the archived commands.

SQLCA_Structure Contains the SQL explain STRUCTURE data for the archived commands.

The total size of the ARCHIVE DBspace is 3328 pages in the distributed SQLDBSU command stream. This provides for about 4096 archived SQL commands, assuming that the average SQL command is 512 characters long, contains 2 subqueries, 3 plans and 8 columns references. The following table shows the detail of the estimated space parameters.

Table name	Rowlen	Rows/ Page	Nr of Rows	Nr of Pages	Pages + Indexes	Round to 128
INDEX	680	4	4096	1024	1362	1408
COST	14	220	8192	37	49	128
PLAN	67	46	12288	267	355	384
REFERENCE	85	37	32768	885	1177	1280
STRUCTURE	21	147	8192	55	71	128

Note:

- The calculations are based on PCTFREE=15 and PCTINDEX=33 (the DB2/VM defaults).

- When estimating the archive table size, be aware that SQL commands contained in a given program may appear more than once in the archive. This will be the case if the same program is analyzed in several databases or by different users.

3.6.4 PROCESS CONTROL DBspace Size

The DBspace SQLCA_CONTROL is **optional** and is used only when the automatic statistics or the program monitoring options are enabled. It contains the table PROCESS_CONTROL. The SQLCABAS auto-statistics facility maintains one row for each table that has been selected for automatic statistics. The SQLCABAS program monitoring facility maintains one row for each package selected for monitoring. Since a DBspace page with the default PCTFREE contains 72 process control table rows, the minimal DBspace allocation provides for more than 4096 table entries.

3.7 Granting SQL/CA related privileges

3.7.1 Grant RUN privilege on SQL/CA

To enable a user to execute the SQL Command Analysis program in a given database, EXECUTE authority is required on the DB2/VM packages **SQLCAXC, SQLCAXC2** and **SQLCPEX**. Querying the archive tables by SQL/CA warning code also requires EXECUTE authority on the package **SQLCAXM**.

Use the GRANT EXECUTE option of the SQLCAPI menu to perform the above GRANTs.

If the SQL/CA archive tables are shared between databases, SQL/CA will automatically switch between the user's database and the archive database. Therefore, each SQL/CA user should have CONNECT authority to the archive database under his own DB2/VM userid and RUN privilege on the above packages in the archive database.

3.7.2 Granting SQLCABAS

When the SQLCABAS **INVALIDATE** option is used to re-prep dependent packages, DBA authority is required during execution, since it implies updating the SYSTEM.SYSACCESS.VALID column. However, since the SYSACCESS UPDATE is done in dynamic mode, SQLCABAS can be granted to non-DBA users, as long as they do not invoke the INVALIDATE function.

No DBA authority is required when the **REBIND** option is used for reprepping dependent packages, as long as the user rebinds his own packages. DBA authority is required to rebind packages for other users.

3.7.3 Grant SELECT Privilege on SQL/CA Archive Tables

To enable a user to execute the ISQL routines that implement the SQL/CA archive queries, access to the SQLDBA.ROUTINE table should be granted and SELECT authority should be given for the following tables:

- SQLDBA.SQLCA_INDEX
- SQLDBA.SQLCA_COST
- SQLDBA.SQLCA_PLAN
- SQLDBA.SQLCA_REFERENCE
- SQLDBA.SQLCA_STRUCTURE
- SQLDBA.ROUTINE

To enable a user to import an archived SQL command, SELECT authority should be granted for the table SQLDBA.SQLCA_INDEX.

3.8 Installing an SQL/CA Server

To install an SQL/Command Analysis server, perform the following:

1. Setup a new VM machine with a name of your choice, for example SQLCASRV. If you choose another name, replace SQLCASRV with that name in all the following steps.

2. In the profile EXEC of SQLCASRV, insert the command **EXEC SQLCABAT 00:00 NONE**, if your intention is to process analysis requests only. If you wish to use the virtual machine to perform auto-statistics as well, additional parameters are required on the SQLCABAT call, as described in the **SQL/CA User's Guide** in the chapter **SQL/CA Batch Facility**.

3. Ensure that the SQLCASRV machine has access to the SQL/CA product minidisk.

4. Create the DB2/VM userid SQLCASRV in all databases to which the server will connect on behalf of a client request. In most cases, you will want this userid to perform analysis of all applications. Therefore, it must also have the privileges to access all the tables referenced by these applications (this is a requirement of the SQL EXPLAIN command). The easiest way to achieve this is to give DBA authority to SQLCASRV.

5. Ensure that you have a private or public Dbspace in which to create the EXPLAIN tables. For optimal EXPLAIN performance, use a dedicated Dbspace for the EXPLAIN tables.

6. Create the EXPLAIN tables for user SQLCASRV. To do so, type **SQLCA** at the CMS prompt. Invoke the **SQL/CA Utilities Menu** and choose the **Create Explain Tables** function of that menu. If the server will connect to multiple databases, perform this step in each of these databases.

7. Ensure that SQLCASRV is auto-logged during VM IPL.

8. If server-mode analysis will be the default analysis procedure, update the **SQLCA OPTIONS** file and insert the statement **DELAY SQLCASRV.** This ensures that all user analysis requests are forwarded to SQLCASRV by default.

9. Logon the server and create the DB2/VM bootstrap module on its A-disk, by performing an **SQLINIT**.

3.9 SQL/CA SERVER User exit

If the SQL/CA server finds the **SQLCAMSG** exec on its CMS search chain, a call will be made to that EXEC at the start and the end of an analysis.

On the call, following arguments are passed to SQLCAMSG:

- call type as:
- **START** at the begin of an analysis
- **END** at the normal completion of an analysis
- **ABEND** at the abnormal completion of an analysis
- userid submitting the request
- name of the analysis request file
- filetype of the analysis request file

The main purpose of the SQLCAMSG exec is to send messages and inform the submitting user about the progress of analysis.

3.10 EXPLAIN table usage by SQL/CA

SQL/CA must have **exclusive control** over the EXPLAIN tables used during analysis. Which EXPLAIN tables are used depends on the DB2/VM userid active during EXPLAIN. This userid becomes the creator of the explain tables: it can be specified in different ways on the SQL/CA menus.

Due to the method used by SQL/CA for accessing the EXPLAIN tables, it is **not possible** for the user to maintain its own explain output in those tables. These data will be cleared by SQL/CA at the next analysis request.

Generally, the EXPLAIN tables have very few rows and are best processed by a DBspace scan. For optimal performance, the user's 4 EXPLAIN tables should be created in a dedicated DBspace and no indexes should be defined on them.

After issuing the EXPLAIN command, SQL/CA has to process the EXPLAIN tables of the user. This cannot be achieved in static mode, since the creator of the EXPLAIN tables is unknown at preprocessing time. Instead, SQL/CA creates an extended dynamic package, the first time a user invokes Command Analysis. The package is named **userid.SQLCAXU2**. This package is managed in the same manner as static packages. If a package is dropped, it is automatically recreated by SQL/CA at the next analysis.

3.11 ALIAS USERNAMES

3.11.1 Functional Description

In some cases, it may be desirable that users perform program analysis using a DB2/VM authorization ID, other than their own. For example, several developers within a team may use the same DB2/VM userid when preprocessing their programs. This facilitates authorization management and allows developers to work on each other's programs and use each other's tables. In such cases, analysis must also be done under the common DB2/VM username. The alias facility allows the system administrator to define each of these users in the **SQLCA CONNECT** file and specify the DB2/VM username and password to be used by SQL/CA when the user forwards an analysis request. Since the CONNECT file contains sensitive data (the user password), it is kept in encrypted format by SQL/CA. The CONNECT file should be on a minidisk or in a directory that is accessible to the users during analysis. It may be placed on the disk containing the SQL/CA software material for instance.

3.11.2 Defining an alias

To define (or remove) an alias, access the minidisk or directory containing the SQLCA CONNECT file in filemode A. Issue **SQLCALIA** on the CMS prompt. The program saves the current SQLCA CONNECT as SQLCA CONNECTS. Then, the SQLCA CONNECT file is decrypted into a file named **CONNECT INPUT**. The CONNECT INPUT file is then XEDITed. Use XEDIT commands to insert (or remove) records that define the alias names. Issuing XEDIT **file** will encrypt and copy the CONNECT INPUT file to SQLCA CONNECT. The alias definitions take effect from this moment on.

Each alias description line has the following syntax:

CONNECT user_name TO database_name AS alias_name alias_password

Meaning: when **user_name** connects to **database_name**, connect should be done automatically by SQL/CA, using **alias_name** and **alias_password**. If **user_name** is specified as *****, all users connect to the database under the alias name.

SQL/CA scans the SQLCA CONNECT file sequentially and stops its search for alias replacements when a line is found where both **user_name** and **database_name** are matching.

For example: to allow a number of DB2/VM developers named SQLDEV1 thru SQLDEVn to connect under the common SQLDEV username, following lines should be coded in the CONNECT file:

- **CONNECT SQLDEV1 TO DBTEST AS SQLDEV SQLDEVPW**
- **other CONNECTs**

- **CONNECT SQLDEVn TO DBTEST AS SQLDEV SQLDEVPW**

4 Using SQL/CA

4.1 Prerequisites

Prior to using SQL/CA, you should ensure that following prerequisites are met:

- You should have access (in any CMS filemode) to the public minidisk, containing the SQL/CA software material.
- You should have access to the DB2 production minidisk.
- You should be able to connect to the DB2 database where you want analysis to occur.
- You should have the DB2 bootstrap modules on your A-disk, that is, an SQLINIT should have been executed previously..
- You should have access to the DB2 explain tables, created under your DB2 userid.[6]

4.2 SQL/CA - SQL/MF integration

If our SQL/MF monitor program product has been installed, real-time execution statistics will be included for each analyzed application, under the following conditions:

- The VM userid performing analysis should have a read link to the SQL/MF product disk.
- The DB2 userid active during analysis should be able to connect to the SQL/MF **Log** database.
- The DB2 userid should have been granted the SELECT privilege on the monitor table SQLCS_SQL_STMNTS.

If analyses are executed in server-mode, the above conditions can be met easily. Since the server must have DBA authority to explain other user's packages, it is sufficient to grant DBA authority to the analysis server in the SQL/MF log database and to provide a link to the SQL/MF disk.

[6]One of the SQL/CA menu options allows you to create these tables. You do not need explain tables if you perform analysis in server mode only.

4.3 Using the SQL/CA command menu

To invoke the SQL/CA command menu, enter **SQLCA** at the CMS prompt[7]. Following menu screen will be presented:

```
               *--------------------------------------*
               *       SQL Command Analysis Menu      *
               *--------------------------------------*

      Analyze CMS Source Program : 1
         Analyze CSP Application : 2
        Analyze REXX Application : 3
     Analyze NATURAL Application : 4
        Analyze Stored SQL Query : 5
             Analyze SQL Package : 6
    Interactive Command Analysis : 7
         Table Oriented Analysis : 8
          Query Analysis Archive : 9
           SQL/CA Utilities Menu : 10

              Select Menu Option : _

Database : DBTEST
PF1 = Help                                              PF3 =Quit
        (C) Copyright Software Product Research - 1996
```

Option codes:

1 Analyses a CMS source program (Assembler, Cobol, Fortran, PL/1,Easytrieve)
2 Analyses a CSP application.
3 Analyses a REXX application.
4 Analyses a NATURAL application.
5 Analyses a stored ISQL or QMF query.
6 Analyses a DB2 package.
7 Analyses a command interactively.
8 Analyses DB2 packages for a designated table name.
9 Queries the SQL/CA archive tables.
10 Invokes the SQL/CA Utilities menu

The desired function is selected by entering its numeric code and pressing **ENTER**.

[7]All SQL/CA components execute as CMS nucleus extensions. Therefore, the SQL/CA menu may also be invoked from the CMS subset of any other program.

4.3.1 CMS source program analysis

The following function screen will request the name of the application source file to be analyzed and the non-default analysis processing options.

```
                *--------------------------------------*
                *       Analyze CMS Source Program     *
                *--------------------------------------*


           Program name to analyze :    _
          Archive analysis results :    YES
              Edit analysis report :    YES
             Print analysis report :    NO
                   Uppercase print :    NO
     Save previous analysis report :    NO
 Update statistics before analysis :    FIRST
 Update statistics for all columns :    NO
        Delayed analysis by userid :
    Copy analysis report to userid :
     Copy source extract to userid :
               Connect to database :
                        as SQL user :
                      with password :

PF1=Help    PF2=Reports     PF3=Quit     PF4=XEDIT     PF5=Indexing
```

PFkey Assignments

- To display online SQL/CA help, press **PF1**.
- To display the analysis reports sent by an analysis server, press **PF2**.
- To exit analysis, press **PF3**.
- To edit the analyzed source, press **PF4**. This will start an XEDIT session. After modifying the source, the menu panel is shown again, so that you may request analysis of the modified source
- Using **PF5**, you can create or drop an index while remaining in the SQL/CA environment. After creating a new index, you may re-analyze the application, to note the effect of the index on access path selection.

Creating or dropping an index from SQL/CA

If you wish to create an index, enter the following data:

- The code **C** in the first screen field to request index creation.
- The name of the database where the index should be created. Before creating the index, the interface will connect your default DB2_id to the named database. This userid should have the required authority to create the index.
- The index creator name. (The default is your VM userid).
- The name of the index.
- The unicity of the index (**Y** for a unique index, **N** for a non-unique index).
- The creator of the table to be indexed.
- The name of the table to be indexed.
- The name(s) of the indexing column(s). Up to 10 columns can be specified, each columnname being entered in its own screen field. By default **ASC** ordering is assumed. If the column must be indexed on descending value, enter the word **DESC** after the column name.

If you wish to drop an index, enter the following data:

- The code **D** in the first screen field to request index drop.
- The name of the database where the index should be dropped. Before creating the index, the interface will connect your default DB2_id to the named database.
- The index creator name. (The default is your VM userid).
- The name of the index.

Note:

The meaning of the ENTER, the PF1, PF2 and PF3 keys is identical in all SQL/CA menu functions. The online help texts are displayed using the CMS HELP facility and the standard PFkey settings apply when browsing through the help text.

4.3.1.1 Program name to analyze

Enter the name of the CMS file containing the source program to be analyzed as filename, filetype and filemode (filemode defaults to A). Filename and filetype may contain a generic CMS name such as ABC* COBOL, if several programs should be analyzed in one run.

Since SQL/CA extracts SQL commands and host variable declarations, it must know the language the application is written in. SQL/CA derives the language from the CMS filetype as follows:

Language	**If filetype starts with:**
Assembler	ASS or ASM
Cobol	COB
Fortran	FORT
PL/1	PLI

For COBOL, Easytrieve and PL/1 programs, the language will be determined from the source text, if the filetype does not start with “COB”, “EZT” or “PLI” respectively. A COBOL program is recognized by the occurrence of the statement ID[ENTIFICATION]-DIVISION. A PL/1 program is recognized by the occurrence of the statement PROC[EDURE] OPTIONS(MAIN). An Easytrieve application is assumed if the first line in the source equals * PARM.

If the CMS filetype equals **SQLACMOD** or **SQLPACK**, you may analyze an package, unloaded outside SQL/CA, using DB2 Database Services. The unload file may contain multiple packages.

If the filetype is not equal to any of the above, a source containing SQL commands in Data Base Services format is assumed. In this case, a semicolon (;) is expected as delimiter of the individual SQL commands. Parameter markers (?) or variable names (&name) may be used to designate variable syntactical elements. Host variables should not be present, since the file format does not allow for an SQL DECLARE SECTION.

If another user did send his source extract to your A-disk (using one of the processing options), that source will have filetype EXTRACT. SQL/CA determines the original language from the extract.

The application source may contain external text embedded by means of the SQL INCLUDE command. The external text is inserted by SQL/CA during analysis.

The record length of the source file should not exceed 8192 characters.

4.3.1.2 Additional processing options

The following fields on the screen allow to specify analysis processing options. The default options (as stored in the SQL/CA system OPTIONS file) are displayed and may be overridden for the current run.

For a complete functional description of the processing options, refer to the paragraph "Additional Processing Options" in the Functional Description on page 15.

Archive analysis results

Specify YES if the extracted SQL command and its EXPLAIN results should be archived into the SQL/CA archive tables.

Edit analysis report

Specify YES if the analysis report should be displayed after analysis, using XEDIT. Specify W1, W2 or W3 if the report should be edited only when analysis has generated warnings of severity 1, 2 or 3.

Print analysis report

Specify YES if the analysis report should be printed after analysis using the current spool options of the virtual printer. Specify W1, W2 or W3 if the report should be printed after analysis warnings of severity 1, 2 or 3. Specify SHORT if the report should be printed in condensed format. A condensed report contains the command text and the related warnings only. All other print options will print the report in non-condensed format.

Uppercase print

Specify YES in conjunction with PRINT YES to force uppercase translation for printer devices without lowercase font.

Save previous analysis report

If you already have an analysis report (<filename> SQLCA) for the application on your A-disk and you want to preserve it, specify YES. This will rename the existing report to a file named <filename> SQLCAXXX, where XXX is a sequence number assigned by SQL/CA. If no saved reports exist, the sequence number will be set to 001. If saved reports do exist, the sequence number is incremented by 1 for each newly saved report.

Update statistics before analysis

Specify FIRST if you want an automatic update statistics during command analysis for all tables referenced by the application, if an update statistics has never been performed for the table. Specify YES if you want an unconditional automatic update statistics during command analysis for all tables referenced by the application. Specify NO if you do not want automatic update statistics.

Update statistics for all columns

Specify YES in conjunction with Auto-Statistics FIRST or YES to perform an UPDATE **ALL** STATISTICS. NO will update the statistics for the indexing table columns only.

Delayed analysis by userid

If you do not want to analyze your program interactively but in "server mode", specify the name of the userid running the SQL/CA analysis server facility. After analysis the report is returned to your virtual reader with a filetype of 'SQLCA'. Eventual error messages issued to the virtual console of the server during analysis, will be found in a reader entry with filetype 'SQLCACON'. When you request server-mode analysis, other processing options do not apply, except for the DB2 connection parameters database, userid and password. The analysis server always uses the default processing options from the SQLCA OPTIONS file. The report EDIT option is always reset during batch processing.

Copy analysis report to userid

Specify "userid" if you want to send a copy of your analysis report to another user when analysis completes. The option has a different meaning when server-mode analysis is requested. In this case the analysis report is sent to "userid" and the user requesting analysis does not receive the report.

Copy source extract to userid

If specified, a copy of the extract and the hostvar files is sent to the named userid. This userid may then request analysis for your application.

Connect to database

Specify the name of the DB2 database to be connected before analysis starts. If none is specified, analysis is done in the currently connected database.

as SQL user

Specify the DB2 userid to be connected before analysis starts. If not specified, analysis is done under the default userid (your VM userid).

If server-mode analysis has been requested, analysis is done under the default userid of the server machine. If you wish connection of another userid, that userid and its corresponding password have to be specified here, even if it is your own userid.

with password

Specify the password for the userid named above. This argument is required if connection of a given DB2 user has been requested.

4.3.2 CSP application analysis

The function screen allows to designate the CSP application to be analyzed and to specify the non-default analysis processing options.

4.3.2.1 CSP application name

Enter the name of the CSP application to be analyzed. Unless your installation modified the SQL/CA CSP invocation EXEC, that application must reside on your **primary read/write MSL**. SQL/CA invokes CSP with an EZECIN command file and requests a print of the named application. As a rule, analysis of applications should be requested. Although analysis of single CSP processes is provided for, analysis results for processes may be less accurate.[8]

SQL/CA extracts all SQL processes from the CSP application and stores them into a CMS file named <applicationname CSPAPPL> on your A-disk. It is possible to analyze this file later on, using the compiled application analysis option. This may be useful if you want to experiment with alternative SQL command specifications, without altering the CSP processes.

4.3.2.2 CSP invocation EXEC name

If your installation did setup an alternative interface between SQL/CA and CSP, enter the name of the corresponding CMS EXEC here. The name you enter will be saved as a global CMS variable and will be presented the next time CSP analysis is requested. To use the default CSP interface, specify a blank name or **SQLCACSP**.

4.3.2.3 Additional processing options

The following fields on the screen allow to specify analysis processing options. The default options (as stored in the SQL/CA system OPTIONS file) are displayed and may be overridden for the current run.

These options are identical to the options specified when analysing a CMS source program. Please refer to the paragraph **Additional processing options** on page 46.

[8]When printing a process, CSP does not print the process object (the SQL row definition) and the CSP working storage. Both may be needed by SQL/CA during analysis.

4.3.3 REXX/SQL program analysis

SQL/CA provides for the analysis of REXX/SQL **PREP** (dynamic) and **XPREP** (extended dynamic) requests.

Analyzing REXX/SQL programs presents a typical problem, because REXX EXECs usually build the SQL command during execution before issuing the PREP or XPREP commands. Therefore, the SQLCA/REXX support will execute the named EXEC file, intercept the SQL commands specified on the PREP and XPREP requests and store them into a CMS file for analysis.

After choosing REXX analysis on the main menu, following screen is presented, allowing to specify:

- *EXEC name* the CMS filename of the REXX program to analyze (the filetype EXEC and the filemode are implied)

- *EXEC args* the invocation parameters of the EXEC, if any

Your EXEC is then executed and its SQL commands intercepted. After interception, following screen is presented.

4.3.3.1 RXSQL filename

The name of the CMS file built by SQLCA/REXX is presented on the option screen as **"exec_name RXSQL"** and should not be modified.

4.3.3.2 Additional processing options

The following fields on the screen allow to specify analysis processing options. The default options (as stored in the SQL/CA system OPTIONS file) are displayed and may be overridden for the current run.

These options are identical to the options specified when analysing a CMS source program. Please refer to the paragraph **Additional processing options** on page 46.

4.3.4 NATURAL application analysis

After choosing NATURAL analysis on the main menu, a screen is presented that allows to specify:

Library name

Specify the name of the Natural library where the program resides. Your latest library specification will be kept by SQL/CA in a CMS global variable and presented as the default library, the next time you request program analysis.

Application name

Specify the name of the Natural application.

SQL/CA extracts the Natural application into a CMS file and presents the following panel.

4.3.4.1 NATURAL filename

The name of the CMS file built by SQL/CA is presented on the option screen as **"program_name NATURAL"** and should not be modified.

4.3.4.2 Additional processing options

The following fields on the screen allow to specify analysis processing options. The default options (as stored in the SQL/CA system OPTIONS file) are displayed and may be overridden for the current run.

These options are identical to the options specified when analysing a CMS source program. Please refer to the paragraph **Additional processing options** on page 46.

4.3.4.3 Technical Notes

1. To perform analysis of NATURAL applications, SQL/CA obtains the application source by means of a NATURAL **UNLOAD** command and the application's SQL by means of a **CREATE DBRM** request.

2. For better readability, SQL/CA replaces the host variable names (:Vxxxyyyy) generated by the CREATE DBRM function with the name of the corresponding variables in the source application. Since analysis is mainly concerned with examination of the command predicates, name substitution occurs in the predicate only, that is, in the command section following the SQL clause **WHERE** (SQL SELECT/UPDATE/DELETE) or the NATURAL clause **WITH** (NATURAL FIND).

3. During command analysis a file called <program_name ASSEMBLE> is created on your A-disk. This file contains all SQL commands in the application, with the corresponding NATURAL command inserted as comments. This ASSEMBLE file can be used as input for subsequent command analysis (if you wish to experiment with the SQL without altering the NATURAL source). It also shows what SQL is generated by your NATURAL commands.

4.3.5 SQL query analysis

On the function screen, name the stored ISQL or QMF query to be analyzed. SQL/CA will access the named query on the ISQL or QMF query table and extract all SQL commands that can be analyzed.

A list of all variable symbols (symbols starting with &) appearing in the query is then presented and you may assign substitution values for each variable. This allows to analyze the command using realistic predicate values. If a substitution value is not assigned, the name of the variable symbol will be used as its value. This however may bias the analysis results and is generally not recommendable.

The analyzable SQL commands are stored into a CMS file named <query_name query_processor> and analysis continues as for a CMS source program, as described on page 43.

4.3.5.1 Name of the query processor

Specify **ISQL** or **QMF**.
Default is ISQL.

4.3.5.2 Name of the query owner

Specify the name of the owner (creator) of the stored query that you want to analyze.
If not specified, the current DB2 userid is taken as the owner.

4.3.5.3 Name of the query

Specify the name of the stored query that you want to analyze.

4.3.5.4 Connect parameters

Specify the databasename, DB2 userid (or alias) and password to be used for analyzing the query.

Note: All the above parameters, except "name of query", are stored by SQL/CA as global CMS variables, lasting across CMS sessions. Your latest parameter assignments will be re-displayed, the next time query analysis is requested.

4.3.6 Package analysis

The function screen will request the name of the DB2 package to be analyzed and the non-default analysis processing options.

4.3.6.1 Package name to analyze

Enter the creator of the package and the name of the package. The package should reside in the currently connected database or in the database whose connection is requested in the **Connect to Database** option described on page 48. The current DB2 userid or the userid connected by the corresponding processing option, determines whether the package can be analyzed. Unless you have DBA authority, you can analyze only packages that you own.

Both package creator and package name can be generic (using the DB2 conventions). If a generic package name is used, multiple packages will be analyzed in the same run. In such cases, a single analysis report file (filetype SQLCA) will be created. The name of the analysis report will be taken from the **first** package name effectively analyzed.

4.3.6.2 Additional processing options

The following fields on the screen allow to specify analysis processing options. The default options (as stored in the SQL/CA system OPTIONS file) are displayed and may be overridden for the current run.

These options are identical to the options specified when analysing a CMS source program. Please refer to the paragraph **Additional processing options** on page 46.

Notes

1. To analyze multiple packages in a single SQL/CA run, you can also use the **SQLCAGPA** utility program, which is described on page 82.

2. Packages generated by CSP do not contain host variable names but parameter markers. Therefore, SQL/CA cannot perform those text analysis procedures that examine host variable specifications, because the datatype and length of the hostvars cannot be derived from the package. The source text of the CSP application is required for full analysis.

3. Since DB2 version 3.2, all host variable names in an package are replaced with the **:H** host variable marker during preprocessing. SQL/CA replaces the **:H** marker with an internally generated name in the format **TTTLLSS**, where

TTT =	**When datatype equals**
CHAR	CHARACTER
VCHAR	VARCHAR
LVCHAR	LONG VARCHAR
GRAPH	GRAPHIC
VGRAPH	VARGRAPHIC
LVGRAPH	LONG VARGRAPHIC
DEC	DECIMAL
FLOAT	FLOAT
INT	INTEGER
SMINT	SMALL INTEGER

LL is the length of the host variable (if not float, integer or smallint)
SS is the precision of a decimal hostvar

For example:

CHAR3 represents a CHAR(3)
DEC70 represents a DECIMAL(7,0)

4.3.7 Interactive command analysis

This function allows to enter an SQL command on the terminal for analysis.

Enter the SQL command to be analyzed. SQL/CA appends a trailing ; to conform to SQLDBSU format and stores the command into a CMS file named "INTACT COMMAND" on your A-disk. The CMS source program analysis screen is then presented with the filename filled in (see page 43).

The INTACT COMMAND file is retained after analysis and may be updated and re-submitted for analysis.

4.3.8 Table oriented analysis

When this option is selected, a selection screen is presented for specifying the following arguments:

Table Creator / Table Name

Specify the creator and the name of the table that should be used as base for table oriented analysis. That is: all DB2 packages that contain a reference to this tablename, will be analyzed in a single run. This table name must be specific. Generic names cannot be used.

Package Creator / Package Name

By default, all DB2 packages are inspected for dependency on the above table name. If you want to limit that search to a given subset of packages, name the subset by specifying a creator and package name. Both the creator and the package name can be generic. Follow the DB2 conventions for stating generic names.

Connect parameters

Specify the databasename, DB2 userid (or alias) and password to be used for analyzing the query.

Usage Notes

- Table oriented analysis usually requires the DBA privilege, since unrestricted access to all DB2 packages is necessary.

- All selected package statements are stored into a CMS file called **SQLCAPUO ACCMOD**, which is subsequently submitted for analysis. Consequently, the analysis report will be named **SQLCAPUO SQLCA**.

4.3.9 Editing the analysis report

After analysis, the analysis report is displayed using XEDIT and a number of XEDIT macros, provided with SQL/CA. During report editing, following Pfkey-based functions are available:

PF1 Displays the SQL/CA glossary. If the cursor is pointing to an analysis warning message, the glossary is positioned to the description of that warning.

PF2 Locates the next analysis warning on the current page.

PF3 Quits the report editor.

PF4 Positions to the begin of the current SQL command being analyzed.

PF5 Positions to the begin of the current SQL subquery, if the command contains subqueries.

PF6 Locates the next analysis warning on the following page.

PF7 Displays the previous report page.

PF8 Displays the next report page.

PF9 Switches to condensed/non-condensed report format, in a flip-flop manner.

PF10 Locates the next SQL command.

Notes:

- When the SQL/CA glossary is being displayed, use the standard XEDIT PFkeys to browse through the document. Press PF3 to exit the glossary. Alternatively, use the **X** XEDIT command to switch between the glossary and the analysis report.

- When the condensed report format has been chosen (using PF9), following items of the report are displayed:

  a. the command text
  b. the SQL command cost value, if the latter exceeds 1000
  c. all severity 2 and 3 warnings issued for the command
  d. the entire **Predicate Analysis Warnings** paragraph

- To return to the non-condensed report, press PF9 again.

- The latest "condens" choice will be used when editing future analysis reports.

4.3.10 Summarizing the analysis report

The **SQLCASUM** EXEC can be used to create a summary report from an existing analysis report. Summarizing may be convenient before printing a large analysis report.

The summary report will contain:

- the SQL command text
- the SQL command cost
- the command access path
- all SQL/CA warnings issued

In addition, SQLCASUM can be used to extract only those SQL commands that exceed a specified SQL cost.

Invocation:

SQLCASUM filename [cost]

where:

- **filename** is the name of the report to summarize (<filename> **SQLCA**)
- **cost** is an optional numerical value; if specified, only the SQL commands greater than or equal to the specified cost will be included in the summary report

The summary report will be written on the A-disk as <filename> **SQLCASUM**.

4.3.11 Printing the analysis report

The analysis request is printed automatically, when you specify one of the print options on the analysis request screen.

You can use the CMS PRINT command to print an existing analysis report (which is a CMS file with filetype **SQLCA**.)

You can print an analysis report in condensed format by typing **SQLCAPRD** on a filelist or by issuing **<SQLCAPRD** filename SQLCA> on the CMS prompt.

4.3.12 Waiting on analysis reports in server mode

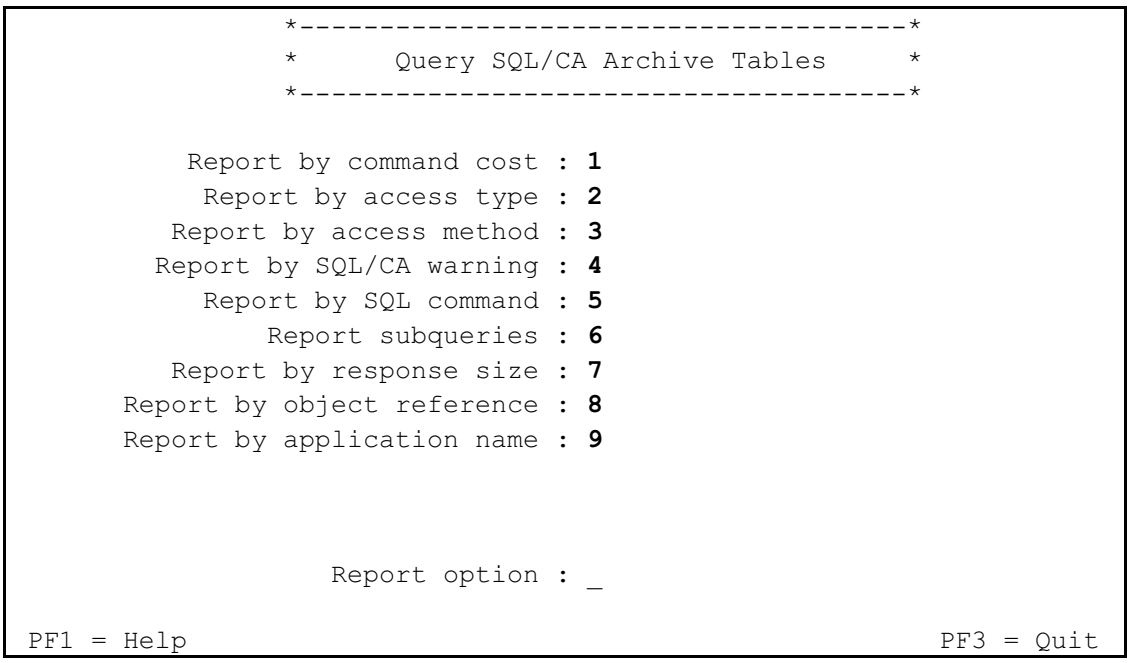
When you send an analysis request to an analysis server and the REPORTWAIT YES option has been enabled in the system processing options file (described on page 129), you may wait for the analysis report. After sending the request, you will get the message:

Waiting for analysis report ...
To interrupt the wait, press ENTER.

If you wait for the report, it will be edited (and printed if requested), as soon as it is returned to you by the server. Press the ENTER key to end waiting.

Note In order to use the wait facility, your VM system must include the CMS **WAKEUP** command, which is part of the CMS Utilities.

4.3.13 Query the SQL/CA archive tables



This option of the Analysis Menu allows to execute a number of queries on the SQL/CA analysis archive tables. These queries are executed as ISQL routines.

When selected, archive query will request a query option which should be entered as

1 to query by command cost
2 to query by access type (relational scan, index scan, view materialization)
3 to query by access method (join, sort)
4 to query by SQL/CA analysis warning
5 to query by SQL command code (SELECT, UPDATE, INSERT ..)
6 to query commands containing subqueries
7 to query by response size
8 to query by command object names (tables, indexes, columns)

The screenfield "report applications named" is common to all archive queries and restricts archive searching to the named application(s). The application name is generic by default, that is, all applications beginning with your name specification will be processed. There is one exception to this rule: if a query for all SQL/CA warnings issued for an application is requested (using query option 4, with the warning code set to 0) and a generic name is required, it must be stated explicitly, using the % marker.

If no application name is entered, all application commands present in the archive tables will be searched, which will obviously increase archive query response time.

To terminate the ISQL routine doing the archive access, press PF3. All other ISQL PFkey assignments do apply.

4.3.13.1 Query by Command Cost

Enter a non-fractional **numeric value** on the option menu. All commands with an execution cost greater than or equal to this value will be reported.

4.3.13.2 Query by Access Type

On the option menu enter:

X to query all commands executed by selective index-only scan
Y to query all commands executed by non-selective index-only scan
I to query all commands executed by selective index scan
W to query all commands executed by non-selective index scan
R to query all commands executed by DBspace scan
L to query the commands for which view materialization (DB2 version 3) will be performed

Note:

- the access types (X,Y,I,W,R) are listed above in decreasing order of access performance, 'selective index-only scan' being the best and 'DBspace scan' the worst access.

- index-only scans are detected only when analysis has been done using DB2 version 3.4 or later.

4.3.13.3 Query by Access Method

On the option menu enter:

1 to query the commands performing a nested loop JOIN
2 to query the commands performing a merge scan JOIN
3 to query the commands for which an additional sort plan will be executed. If an ORDERing or GROUPing clause is executed implicitly using the index order, the command will not appear in this query.

4.3.13.4 Query by SQL/CA Warning

When asked for "SQL/CA warning code", enter **0** to obtain for the named application(s), a list of **all** SQL/CA warnings issued during text analysis. Contrarily to other query options, where a generic application name is the default, you must enter a generic name, if multiple applications should be reported.

Alternatively, enter a message class code **between 1 and 24** as "SQL/CA warning code", if you wish to obtain all SQL commands in the specified application(s) for which the particular warning has been issued. Following is a list of the SQL/CA warning class codes and their meaning:

1 expression on indexing column used
2 data type incompatibility detected
3 data length incompatibility detected
4 data precision incompatibility detected
5 data scale incompatibility detected
6 indicator used with NOT NULL column
7 logical connector OR used
8 logical connector NOT used
9 operator used is no index keymatching candidate
10 operator used is not sargable
11 default filter factor used in predicate
12 range predicate used with host variable
13 plan index columns not or not completely specified in predicate
14 no indexing columns used in predicate at all
15 join columns have mismatching datatype
16 missing search condition in join command
17 indexing columns updated by command
18 expression residual due to mismatching datatype
19 expression residual due to mismatching precision/scale
20 no indexes exist for the table
21 no highly clustered indexes exist for the table
22 table index created using a back-level DB2 release
23 materialization of view performed
24 child block executed at each open of its parent block
25 suboptimal range operator used instead of BETWEEN

```
Package : SQLDBA.ARIDSQL Line nr 7          ARCHQNO     804
Database=DBTEST                 Userid=SQLAF
Timestamp=1996-09-03-12.43.30.192995

SELECT CREATOR , TNAME FROM SYSTEM.SYSCATALOG WHERE DBSPACENO =
:DBSPNO AND TABLETYPE = 'R'

 W14: No index columns used in predicate.
 W21: No highly clustered indexes exist for table.


Press ENTER to continue; type END to terminate.
```

The query shows the first 7 lines of the SQL commands concerned. To get the warnings in full detail, it may be necessary to re-analyze the application.

4.3.13.5 Query by SQL Command Code

On the option menu enter **SELECT**, **INSERT**, **UPDATE** or **DELETE** to obtain a report of the SQL commands in the designated application(s) containing the specified command code. A DECLARE CURSOR for SELECT or INSERT is treated as SELECT or INSERT.

4.3.13.6 Query by Subqueries

This query shows all SQL commands containing subqueries. The only parameter requested is the [generic] name of the application(s) to be examined.

4.3.13.7 Query by Response Size

On the command menu enter a non-fractional **numeric value**. The query will show all SQL commands with an estimated response set equal to or exceeding this value.

The query uses the 'ROWCOUNT' column in the Explain Structure table. This column indicates the estimated number of rows that will be returned to the query as a result of its predicate specifications.

4.3.13.8 Query by Command Object Names

When selected, the query option will request following parameters:

- reference type: specify
  - T to obtain the SQL commands using a given (generic) table name
  - I to obtain the SQL commands using a given (generic) index name (as shown in the explain plan)
  - C to obtain the SQL commands using a given (generic) table column name in their predicate (as shown in the explain reference table)

- name of the table creator (type T and C) or the index creator (type I)
- name of the table (type T and C) or index (type I)
- name of the table column (type C)

The creator name and the name of the table, index or column may be a generic SQL name. If a given name is omitted, it will be set to '%', causing all names to be examined.

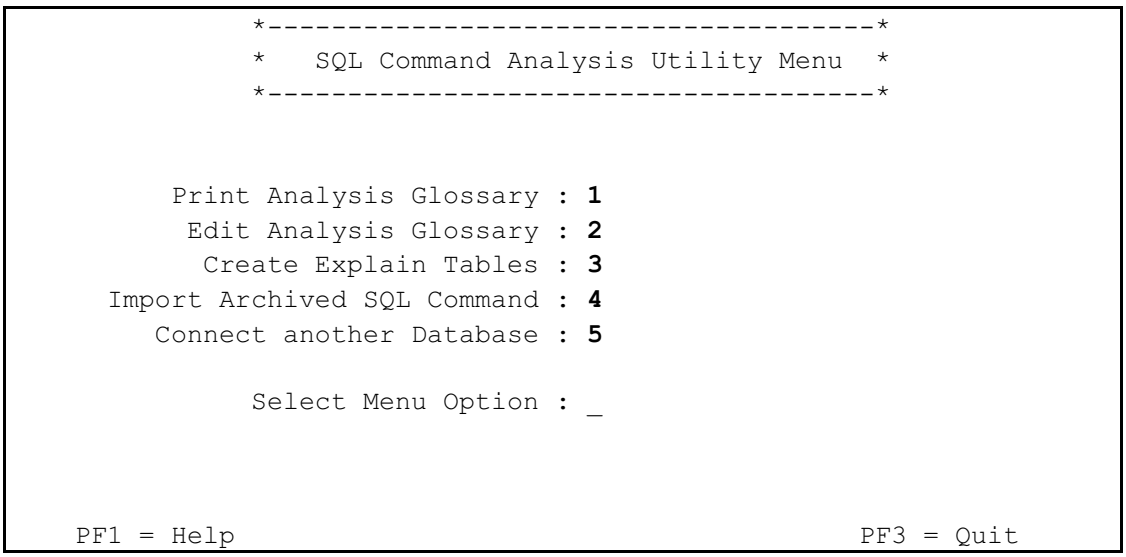
The sample report shows the applications referring to the table SQLDBA.SQLCA_INDEX.

```
File: SQLCACOL ISQLQRY  A  Line nr 1   SQLDBA   SQLCA_INDEX
File: SQLCACST ISQLQRY  A  Line nr 1   SQLDBA   SQLCA_INDEX
File: SQLCAWNG ISQLQRY  A  Line nr 1   SQLDBA   SQLCA_INDEX
```

4.3.13.9 Query by Application name

This query reports the archive entries for a given application. The application name specified is automatically considered as generic.

4.3.14 SQL/CA utility menu



The utility menu allows to select one of the following functions, which are selected by entering the corresponding numeric code.

1 Print the SQL/CA glossary.
2 Edit the SQL/CA glossary.
3 Create the DB2 explain tables.
4 Import an archived query.
5 Connect another database.

4.3.15 Print the SQL/CA glossary

This menu option prints the Glossary using the current spool characteristics of the virtual printer.

4.3.16 Edit the SQL/CA glossary

This menu option displays the Glossary using XEDIT and the SQL/CA XEDIT profile. The same function can be invoked when viewing the analysis report by pressing PF1.

4.3.17 Create the DB2 EXPLAIN tables

The SQL EXPLAIN command, issued during analysis, requires 4 explain tables which can be created using this menu function.

The function will request:

- The **name of the Dbspace** where the explain tables should be created. If you have a private DBspace, leave the name blank, which will create the explain tables in your private DBspace. Else, enter the name of the public DBspace where the tables should be created.

- The **creator name** for the explain tables, which defaults to your VM-id. Using this option allows you to create explain tables for other people, provided you have the DB2 authority to do so.

The function issues a DROP TABLE followed by a CREATE TABLE for the 4 explain tables. When an error is detected by DBS, the DBS listing is displayed on the terminal.

Notes:

- The explain tables should be reserved for exclusive use by SQL/CA. It is not possible to keep user explain output in these tables, since SQL/CA deletes all explain table rows before starting analysis.

- The create explain table function is DB2 release sensitive. If it is executed under DB2 3.4 and above, the explain tables will be created in the extended format.

- For performance reasons, you should provide a small dedicated DBspace for the explain tables and not create indexes on them, since they are best accessed by DBspace scan.

4.3.18 Import an archived query

Each archived SQL command receives an **archive query number** when stored in the archive tables. This query number is shown on the analysis report and during archive query. To import a command from the archive, enter its query number when asked. A CMS file named "ARCHQNO <query number>" will be created on your A-disk. That file will then be submitted to SQL/CA for analysis, using the compiled application analysis function described on page 43.

4.3.19 Connect another database

This function allows to **permanently** connect another database. The function will request the name of the desired database and issue an **SQLINIT** to that database.

4.4 Invoking command analysis at the CMS prompt

4.4.1 Program Command Analysis

Command analysis can be invoked for an application program stored in a CMS file, by typing **SQLCA** followed by the CMS filename, filetype and filemode, if the latter is not 'A'.

For example: *SQLCA XYZ PLIOPT*

The CMS Source Program analysis screen is then presented with the filename filled in. You may eventually change processing options and start analysis by pressing ENTER. (For a description of the processing options please refer to the paragraph **Additional processing options** on page 46.)

4.4.2 Analysis Report Editing

You may edit (with the SQL/CA XEDIT profile) an existing analysis report for a given application XYZ by typing the CMS command:

SQLCA XYZ SQLCA

4.5 Invoking command analysis from XEDIT

When you are editing a CMS source program using XEDIT, you may invoke SQL Command Analysis by entering the XEDIT subcommand **SQLCAP**.

The CMS Source Program analysis screen is then presented with the filename filled in. You may eventually change processing options and start analysis by pressing ENTER. (For a description of the processing options please refer to the paragraph **Additional processing options** on page 46.)

When analysis completes, the analysis report is edited as a new file in the XEDIT ring. This allows you to switch between the analysis report and your source program using XEDIT commands.

Please note that changes to the source program should be saved (written to the disk file) before invoking command analysis, which reads the source file from disk.

4.6 Using the SQLCA command on a CMS filelist

4.6.1 Program Command Analysis

Command analysis can also be invoked for a program appearing on a CMS Filelist, by typing **SQLCA** on the corresponding filelist line. The CMS Source Program analysis screen is then presented with the filename filled in. You may eventually change processing options and start analysis by pressing ENTER. (For a description of the processing options please refer to the paragraph **Additional processing options** on page 46.)

4.6.2 Analysis Report Editing

When typing **SQLCA** on a Filelist line for an analysis report (filetype = SQLCA), the SQL/CA report will be edited using the SQL/CA XEDIT profile.

4.7 Using the SQLCAX EXEC

By invoking the **SQLCAX** EXEC, an application EXEC may initiate SQL/CA processing for a single source program, stored in a CMS file.[9]

The invocation syntax of the SQLCAX EXEC is as follows:

```
SQLCAX   FN FT FM
         'NONE'
         [ALLSTAT { YES | NO }]
         [ARCHIVE { YES | NO }]
         [AUTOSTAT { FIRST | YES | NO }]
         [CONDB database]
         [CONUSR userid]
         [CONPSW password]
         [COPY  { * | userid }]
         [DELAY { * | userid }]
         [DEST  { * | userid }]
         [EDIT { YES | NO | W1 | W2 | W3 }]
         [PRINT { YES | NO | W1 | W2 | W3 | SHORT }]
         [SAVE { YES | NO }]
         [UPCASE { YES | NO }]
```

FN FT FM

Name of the CMS source file to be analyzed.

'NONE'

Only used when **fn ft fm** is an unloaded DB2 package. In this case the package creator name is specified.

ALLSTAT { YES | NO }

Specify YES in conjunction with AUTOSTAT FIRST|YES to force an UPDATE **ALL** STATISTICS. ALLSTAT NO will update the statistics for the indexing table columns only.

ARCHIVE { YES | NO }

Specify YES if analysis results should be archived.

[9]All SQL/CA components execute as CMS nucleus extensions. Therefore, the SQLCAX EXEC may be invoked from a program executing in the CMS User Area.

AUTOSTAT { FIRST | YES | NO }

The AUTOSTAT option controls automatic statistics update during command analysis for all tables referenced by the application.
Specify AUTOSTAT FIRST to request statistics, if never been issued for the table.
Specify AUTOSTAT YES to unconditionally update the statistics.
Specify AUTOSTAT NO to disable automatic update statistics.

CONDB database

Specify the name of the DB2 database to be connected before analysis starts. If not specified, execution is in the currently connected database.

CONUSR userid

Specify the DB2 userid to be connected before analysis starts. If not specified, analysis is done under the default userid (the VM userid).

CONPSW password

Specify the password for the userid named in CONUSR. This argument is required if CONUSR is specified.

COPY userid

If a copy of the extract and the hostvar files is to be sent to the named VM userid or to the nickname defined in a CMS "NAMES" file , specify that userid. Else specify *.

DELAY { userid | * }

If server-mode analysis is desired, specify the name of the VM userid or the CMS nickname running the SQL/CA server. Specify * to request interactive analysis.

DEST { userid | * }

If a copy of the analysis report is to be sent to the named VM userid or to the CMS nickname, specify the userid. Specify * to reset the option. If the DEST option is specified in conjunction with DELAY, the analysis report will be sent to the DEST user and no report will be available for the user requesting server-mode analysis.

EDIT { YES | NO | W1 | W2 | W3 }]

Specify YES if the analysis report should be XEDITed after analysis.

Specify NO if the report should not be edited.

Specify W1, W2 or W3 if the report should be edited after analysis warnings with a severity code of 1, 2 or 3 respectively.

PRINT { YES | NO | W1 | W2 | W3 | SHORT}]

Specify YES if the analysis report should be printed after analysis using the current spool options of the virtual printer.

Specify NO if the report should not be printed.

Specify W1, W2 or W3 if the report should be printed after analysis warnings with a severity code of 1, 2 or 3 respectively.

Specify SHORT if the report should be printed in condensed format. A condensed report contains the command text and the related warnings only. All other print options will print the report in non-condensed format.

SAVE { YES | NO }

If an existing analysis report (<filename> SQLCA) for the application should be preserved, specify YES. This will rename the existing report to a CMS file named <filename> SQLCA-XXX, where XXX is a sequence number assigned by SQL/CA. If no saved reports exist, the sequence number will be set to 001. If saved reports do exist, the sequence number is incremented by 1 for each newly saved report.

UPCASE { YES | NO }

Specify YES in conjunction with PRINT YES to force uppercase translation for printer devices without lowercase font.

Note

- Unless stated otherwise in the above argument description, the default analysis processing options, as coded in the SQLCA OPTIONS file, will take effect if the corresponding arguments have been omitted on the SQLCAX call.
- The SQLCAX EXEC sets a returncode of 0 if no warnings were produced during analysis. A returncode of 4, 8 or 12 is presented if warnings of severity 1,2 resp. 3 were issued. A returncode 16 indicates that errors were detected during EXPLAIN (for instance due to incorrect SQL command syntax).

4.8 Using the SQLCAGPA utility

The SQLCAGPA package analysis utility allows to analyze multiple packages in a single analysis run. Each package analyzed will have its own analysis report file (filetype SQLCA).[10]

The SQLCAGPA command is entered at the CMS prompt and has the following syntax:

```
SQLCAGPA   DATABASE n
           CREATOR n
           PACKAGE n
           USER n
           PASSWORD n
```

The default values for the command arguments are as follows:

DATABASE : default database
CREATOR : %
PACKAGE : %
USER / PASSWORD : default DB2 userid

The DELAY argument in the SQLCA OPTIONS file will determine whether interactive or server-mode analysis is performed.

When analyzing a large number of packages, a considerable amount of space will be required on your A-disk if interactive analysis is done, or in the VM reader spool if server-mode analysis is performed.

In server-mode analysis, the analyis reports will be returned as reader files. To get them on your A-disk (filetype SQLCA), use the **Retrieve** function of the CMS RDRLIST command. To display the analysis report using the SQL/CA macro, type the **SQLCA** command on the Filelist of the reports.

[10]The package analysis function of the SQL/CA menu also allows multiple package analysis by specifying a generic package name. However, all packages will be analyzed into a single analysis report.

4.9 Stopping the SQL/CA server

The stop the analysis server, issue the command **SMSG** <servername> **STOP**.
On receipt of the command, the server will perform a CP LOGOFF when the current analysis request, if any, has been completed. Non-processed requests remain in the server's reader queue.

Note

The STOP command is available only when the WAKEUP command is available (WAKEUP is part of the CMS Utilities feature).

4.10 VM/CSP considerations

When analyzing a CSP application, SQL/CA invokes CSP with an EZECIN command file that contains a print command for the named CSP application. The CSP invocation procedure is a REXX EXEC, called **SQLCACSP**. SQLCACSP on its turn calls another EXEC, named **SQLCACSI** for opening ('EXEC SQLCACSI OPEN') and closing ('EXEC SQLCACSI CLOSE') the CSP environment. The SQLCACSI EXEC initiates a standard CSP environment by executing the IBM EXEC's CSPDDLBL and USERDLBL on the CSPUSER 193 minidisk. This results in the definition of a read/write MSL on the user's 503 minidisk. The standard CSP invocation procedure then allows to analyze applications from that primary MSL. If you have a differently customized CSP environment (with project MSL's for instance), you may have to alter the **SQLCACSI** EXEC, in order to suit your CSP setup.

Alternatively, you can write a CMS EXEC which invokes a customized CSP environment. When a user invokes CSP application analysis, the name of this EXEC can be entered as a function parameter when performing CSP analysis.

Invocation EXEC entry values

When invoking the customized EXEC, SQL/CA passes 3 parameters:

- the filename and filetype of a CMS file built by SQL/CA; that file contains following command: **PRINT MEMBER(application_name);**

- the name of the CSP application to be analyzed

Prior to invocation, SQL/CA has issued following VM commands:

```
CHANGE RDR CLASS S HOLD
SPOOL 00E * CLASS S CONT
```

Invocation EXEC exit values

On exit from the customized invocation EXEC, SQL/CA expects the listing of the application to be analyzed in its reader or as a CMS file with filetype LISTING on minidisk A.

4.11 VSE/CSP considerations

There is no explicit support for VSE/CSP in SQL/CA. If applications to be analyzed reside in a VSE/CSP MSL, there are two solutions:

- If VM/CSP has been installed, the SQLCACSP EXEC may be customized to link to the VSE minidisk containing the VSAM catalog and the CSP library. Normal VM/CMS facilities can be used to achieve this.

- If VM/CSP has not been installed, the VSE/CSP application should be obtained by submitting a VSE job to print the application into the reader of the virtual machine requesting analysis. A skeleton job, named **SQLCACSV EXEC,** is provided with SQL/CA. You can use it as a base for installing your own support.

4.12 NATURAL considerations

NATURAL application analysis is controlled by the SQL/CA EXEC **SQLCAM12**.

Before starting analysis, an EXEC named **SQLCANT0** is invoked for defining the following NATURAL related global variables:

– NATURAL module name; if NATURAL is called by an exec, enter **EXEC nnn** as assignment value
- the CMS userid, the virtual address and the password for the minidisk that contains the NATURAL software material; if this minidisk is public, set the above 3 parameters to blank
- the NATURAL command separator (default is a comma)

Following default values for these variables are stored as REXX assignments at the beginning of SQLCANT0. Change the assignments to meet your installation requirements.

```
NATURAL_programname       = 'NAT22'
NATURAL_userid             = 'SAGDBA'
NATURAL_mdisk              = '196'
NATURAL_mdisk_password   = 'PASS196'
NATURAL_command_separator = ','
```

5 Interpreting the analysis report

For each SQL command in the application program, following paragraphs are printed in the analysis report.

5.1 Command Execution Structure by block and parent

SQL/CA formats the SQL command in such a way, that the execution structure for a multiple query command becomes apparent. Each subquery is indented according to its nesting level within the execution tree. The outer (first) query is printed at indent level 1; a subquery called from the outer query is printed at indent level 2; a subquery called by another subquery is printed at the indent level following that of its parent query. For each subquery, the query blocnumber (**BLK**) and the number of the invoking parent block (**Par**) is provided.

Note that the execution hierarchy is derived by SQL/CA from the explain tables and that in some cases the actual execution structure may differ from the source command structure. The DB2 optimizer may effectively execute some subqueries earlier, at the opening of an ancestor block, when there is no correlation to the tables in the intermediate query block.

5.2 Structure of Referential Constraint Commands

A command modifying a table that is a member of a referential integrity constraint, generates additional commands that enforce the integrity. A delete on a parent table for instance, will generate delete or update commands for all its dependent tables. The execution structure of these internal commands is provided by SQL/CA in separate paragraphs that are titled **Structure of referential constraint command**, followed by the internal command number (1 to N). The text of the internal command is derived from the EXPLAIN data.

5.3 Command Cost Summary

The cost paragraph applies to the command as a whole. For a single query command, 2 cost estimates are provided:

- *Total Command Cost* The query cost estimate from the explain table. The value is zero for INSERT commands.

- *ISQL like Cost* (total_command_cost / 1000) + 1. Interactive SQL systems such as ISQL present the Query Cost Estimate in this format.

For a multiple query command, following values are computed:

- *Single_Cost* The cost of a single execution of the query block, not including the cost of the dependent and the ancestor blocks.

- *Exec_Times* The estimated number of invocations of a query from **all** its parent blocks. Eventual multiple parents as well as their "ATOPEN" attribute are taken into account. (The ATOPEN flag tells whether the subquery is executed once or multiple times by its immediate parent block).

- *Multi_Cost* Single_Cost multiplied by Exec_Times.

- *SQL_Cost* The cost from the DB2 explain table i.e. the cost of the query plus the cost of all its dependent blocks.

- *Highest_Subquery_Cost* The highest Multi_Cost found for a subquery in the command. Performance tuning should concentrate on this subquery.

If our SQL/MF monitor program product has been installed and if the monitor tables are accessible during analysis[11], following execution-time program statistics are included:

N_Rows number of rows processed
DBSScall number of calls to the DBSS component
Buflook number of buffer lookups performed
Tot_IO number of I/O's or dataspace transfers performed
SQL_Cost effective command cost as TOT_IO + (DBSSCALL/3)

These statistics were recorded during the last execution of the application. They can be used to compare the cost estimated by DB2/VM with the real execution cost.

[11]See *SQL/MF Integration* on page 41.

5.4 Command Plan Summary

If the analyzed command has more than one row in the plan table, the **plan summary** is printed.

For each entry in the plan table, following data is provided:

- the query number starting from 1 up to the number of nested queries in the command
- the plan number within the query (1 to N)[12]
- the access descriptor for the plan entry

For primary plan entries, one of the following access descriptors is provided:

- **scan of DBspace N for table N**
- **selective index[-only] scan of table N**
- **non-selective index[-only] scan of table N**
- **fully qualified index[-only] scan of table N**
- **literal based index[-only] scan of table N**

For secondary plan entries, one of the following access descriptors is provided:

- **nested loop join**
- **merge scan join**
- **sort at end of query**

For join plans, the access path taken during join is also shown, using one of the primary plan descriptors. For example: "Nested loop join using selective index scan of table N".

Each plan is described in full detail in the **Command execution detail** paragraph.

[12]In DB2 releases prior to 3.4, the same plannumber may occur more than once, if the primary command generates internal commands that enforce referential integrity.

5.5 Command Execution Structure

The paragraph is printed for each subquery in the SQL command and provides following data obtained from the DB2 explain tables:

Estimated number of rows processed ... out of ...

The ROWCOUNT column from the EXPLAIN table. It indicates the estimated number of rows returned for the table(s) used in this command. SQL/CA computes the total number of table rows from which the estimated number of rows is selected and prints this number following **out of**. For join commands, this total counts the rows for all tables participating in the join.

Estimated global command filter factor

Represents the fraction of table rows estimated to satisfy the command predicate as

(estimated number of satisfying rows) / (sum of all rows of all tables accessed)

The filter value ranges from 0 to 1. The lower the value, the better the selectivity of the command.

Estimated times predicate conditions satisfied

The TIMES column from the EXPLAIN table. It estimates the probability that the predicate conditions of the command will be true. It also shows how many times eventual dependent blocks will be executed. If the estimated value equals the number of table rows, SQL/CA assumes a sequential table scan and issues a warning.

Estimated execution iteration by ancestor blocks

Using the TIMES column from the EXPLAIN Structure table for **all** parent blocks, SQL/CA computes the total number of times this query block will actually be executed. The ATOPEN attribute of each ancestor block is also taken into account. (The ATOPEN flag tells whether the subquery is executed once or multiple times by its immediate parent block).

Block executed ... times by parent block ..

The TIMES column from the EXPLAIN Structure table. It estimates the number of times the subquery is invoked by its **immediate** parent, which is represented by its blocknumber.

5.6 Command Execution Detail

The following information is printed for each plan in the query. JOIN queries or commands requesting a sort operation, will have more than one plan.

5.6.1 Plan Detail

The execution plan is detailled by providing the following items:

METHOD

The information is available for JOIN and sort plans only. It takes following values:

Merge scan JOIN

DB2 merges the response set obtained thus far with the new table to be joined in the order of the join column and joins rows with matching columns. A sort operation may be necessary to access the table to be merged in the required order. A merge scan join may require work dataspace.

Nested loop JOIN

For each row of the response set obtained thus far, the new table to be joined is searched for matching rows and the matching rows are joined. An index may be used to access the new table.

Additional sort at end of query block

The plan executes a final sort operation in order to satisfy ORDER, GROUP or DISTINCT command clauses.

ACCESS

Using scan of DBspace

Rows are accessed by scanning the named DBspace. Data belonging to other tables in the same DBspace will also be scanned. SQL/CA adds the following informations:

DBspace pages scanned

Provides the number of active pages in the scanned DBspace, that is the number of pages that actually will be read.

DBspace scan productivity

If multiple tables share the scanned DBspace, the productivity value shows the percentage of DBspace data scanned belonging to the table accessed in the plan. For critical tables productivity should normally be 100%, that is, the table should have its dedicated DBspace. Otherwise, a DBspace scan will imply unnecessary I/O by reading pages of other tables and unproductive locks on pages that contain rows belonging to other tables.

Fully-qualified index scan

The table will be accessed using all columns of the named unique index. The name of the index and its columns are printed. The characteristics of the index are specified as **first** or **non-first, highly clustered** or **weakly clustered, unique** or **not unique.**

Fully-qualified index-only scan

The table will be accessed using all columns the named unique index. Moreover, all selected data will be retrieved using the index, that is, data pages will not be accessed. The name of the index and its columns are printed. The characteristics of the index are specified as **first** or **non-first, highly clustered** or **weakly clustered, unique** or **not unique.**

Fully-qualified index scan

The table will be accessed using all columns of the named unique index. The name of the index and its columns are printed. The characteristics of the index are specified as **first** or **non-first, highly clustered** or **weakly clustered, unique** or **not unique.**

Literal based index scan

The table will be accessed using the named index and literal values from an **IN** list. The name of the index and its columns are printed. The characteristics of the index are specified as **first** or **non-first, highly clustered** or **weakly clustered, unique** or **not unique.**

Literal based index-only scan

The table will be accessed using the named index and with values from an **IN** list. Moreover, all selected data will be retrieved using the index, that is, data pages will not be accessed. The name of the index and its columns are printed. The characteristics of the index are specified as **first** or **non-first, highly clustered** or **weakly clustered, unique** or **not unique.**

Non-selective index only scan

The table will be accessed using the named index without key values, which means that a sequential index scan will be performed. However, all selected data will be retrieved using the index, that is, data pages will not be accessed. The name of the index and its columns are printed. The characteristics of the index are specified as **first** or **non-first, highly clustered** or **weakly clustered, unique** or **not unique.**

Selective index scan

The table will be accessed using the named index with specific key values. The name of the index and its columns are printed. The characteristics of the index are specified as **first** or **non-first, highly clustered** or **weakly clustered, unique** or **not unique.**

Non-selective index scan

The table will be accessed using the named index without key values, which means that a sequential index scan will be performed. Data pages may be accessed if the predicate references columns that are not available in the index. The name of the index and its columns are printed. The characteristics of the index are specified as **first** or **non-first, highly clustered** or **weakly clustered, unique** or **not unique.**

Materialization of view

View materialization is a technique used by DB2 version 3 in order to remove restrictions on the processing of views. The feature implies storing of intermediate select results in internal tables. To access these intermediate results, indexed access is never used by DB2. Hence the possibly negative performance implications of view materialization.

Score

The access score is a value computed by SQL/CA in order to represent the efficiency of the DB2 access path chosen. The higher the score, the better the path. One point is added to the score each time one of the following conditions is true:

- index access is being performed
- fully-qualified index access is being performed
- index-only access is being performed
- selective index access is being performed
- index access is using a highly-clustered index
- index access is using a unique index

The highest score is achieved when all the above conditions are true. The highest score is 6 (DB2 Version 3.4 and later) or 4 (DB2 versions before 3.4) The lowest score is 0 and indicates a DBspace scan.

AW91 Data pages accessed for predicate and data

Data pages must be accessed to resolve the predicate and to retrieve data. If the warning is not provided and the access is **index-only**, both predicate and data are resolved using the index only. If the warning is not provided and the access is not **index-only**, the predicate is resolved using the index and data pages are accessed to retrieve data and predicate columns not available in the index.

Key-matching index columns: n out of m

The number of index keys (**n**) that have key-matching predicates used in an index scan. The number of columns in the index is provided by **m**. If n < m, warning **AW92** is inserted.

No indexes for table

No indexes have been defined for the named table.

No highly clustered first index for table

Indexes do exist for the table, but none of them is highly clustered.

No unique indexes for table

The table has no unique indexes. This is a warning only, since the structure of the data may be such that non-unique indexes are acceptable.

No indexes found created using current DB2 release

All indexes for the table were created using a backlevel DB2 release.

SORT

New table [not] sorted

The new table[13] accessed in this command plan needs [does not need] to be sorted.

New table sorted and duplicates removed

The new table accessed in this command plan needs to be sorted and duplicates to be removed, for instance, in order to satisfy a DISTINCT request.

New table sorted for JOIN purposes

The new table accessed in this command plan needs to be sorted as part of a JOIN plan.

New table sorted due to ORDER BY

The new table accessed in this command plan needs to be sorted to satisfy the command's ORDER BY clause.

New table sorted due to GROUP BY

The new table accessed in this command plan needs to be sorted to satisfy the command's GROUP BY clause.

Composite [not] sorted

When a command is performed in several steps, the DB2 term "composite" refers to the response set obtained thus far, as the result of execution in previous steps. For a join operation it is the "input" table. The composite had [not] to be sorted at the initiation of the plan, for example, to prepare the composite for a merge scan join.

Composite sorted and duplicates removed

The composite has to be sorted and duplicates removed.

Composite sorted for JOIN purposes

The composite table accessed in this command plan needs to be sorted as part of a JOIN plan.

[13]See the Glossary on page 133 for a definition of "new table".

Composite sorted due to ORDER BY

The composite table accessed in this command plan needs to be sorted to satisfy the command's ORDER BY clause.

Composite sorted due to GROUP BY

The composite table accessed in this command plan needs to be sorted to satisfy the command's GROUP BY clause.

Additional information provided for a JOIN

Number of rows in inner table: the number of rows in the table from which rows are joined in the current plan (the *new* table). Not available if no statistics exist for the table.

Number of rows in outer table: the number of rows in the table to which the join is done (the *composite* table). Not available if no statistics exist for the table.

The above *number of rows* represents the effective tablesize during the join, that is, local non-join predicates are applied when computing the value.

5.6.2 Column Reference Details

The table and columns intervening in the current command execution plan are obtained from the DB2 EXPLAIN Reference table. For each column, following items are reported:

Filtering factor

For each column referenced in the command predicate, DB2 determines a filter factor, which represents the fraction of table rows estimated to satisfy the predicate as:

estimated_number_of_rows / number_of_table_rows

The filter is a value between 0.0 and 1.0. The lower the value, the better the column's selectivity. A filter factor of 1 means the column has no selectivity at all.

The filter factor depends on the distribution of data values for the column. A column having a high number of unique values within the table, will have a good selectivity, that is, a small filter factor. If an SQL command supplies a search value for such a column, the DB2 optimizer then knows that only a few number of rows will meet the condition and that the response set will be small, which in turn affects the command execution cost.

Estimated table rows filtered

This value is obtained by multiplying the number of table rows by the above filter factor. It shows how many table rows are estimated to be filtered through this column.

Sargable predicate associated with column

A sargable predicate is associated with this column. To receive the qualification:

- the column must be in a predicate that is connected by AND to the rest of the WHERE clause, or be the only WHERE predicate
- the predicate in which the column appears must be in the format: "column operator expression"

Sargable equi-JOIN predicate associated with column

The column appears in a join predicate using the = operator and the join predicate is sargable.

Appears in ORDER BY clause on position ..

The column is used on position .. of an ORDER BY clause.

Appears in GROUP BY clause on position ..

The column is used on position .. of a GROUP BY clause.

Updated by literal expression

The column appears in the SET clause of an UPDATE statement that updates the column with a constant value.

Updated by column or expression

The column appears in the SET clause of an UPDATE statement that updates the column with a column value or an expression

Resolved using index page

The column occurs in the plan index and is fixed-length. No data page must be accessed when fetching this column.

Resolved using index and data page

The column occurs in the plan index but is VARCHAR or VARGRAPHIC. A data page must be accessed when fetching this column.

Resolved using data page

The column is not in the plan index. A data page must be accessed when fetching this column.

5.6.3 Column Reference Details (DB2 versions before 3.4)

The table and columns intervening in the current command execution plan are obtained from the DB2 EXPLAIN Reference table. For each column, following items are reported:

Filtering factor

For each column referenced in the command predicate, DB2 determines a filter factor, which represents the fraction of table rows estimated to satisfy the predicate as:

estimated_number_of_rows / number_of_table_rows

The filter is a value between 0.0 and 1.0. The lower the value, the better the column's selectivity. A filter factor of 1 means the column has no selectivity at all.

The filter factor depends on the distribution of data values for the column. A column having a high number of unique values within the table, will have a good selectivity, that is, a small filter factor. If an SQL command supplies a search value for such a column, the DB2 optimizer then knows that only a few number of rows will meet the condition and that the response set will be small, which in turn affects the command execution cost.

Estimated table rows filtered

This value is obtained by multiplying the number of table rows by the above filter factor. It shows how many table rows are estimated to be filtered through this column.

Appears in WHERE clause such that index can be used

A column receives the above qualification

- when it is the only predicate in the WHERE clause
- when it appears in a sargable[14] predicate connected by means of AND with the remainder of the WHERE clause

Used as a JOIN column

The column appears in a join predicate.

[14]See the Glossary on page 133 for a definition of "sargable".

Column plausible for use in index scan

The qualification is given for a column if both the percentage of DBspace used by the table and the count of distinct values for the column are small. In these circumstances an index scan, even using a non-clustered index, may be preferable to a DBspace scan.

Updated by literal expression

The column appears in the SET clause of an UPDATE statement which updates the column with a constant value.

Updated by non-literal expression

The column appears in the SET clause of an UPDATE statement which updates the column with a column value or an expression.

Appears in ORDER BY clause on position ..

The column is used on position .. of an ORDER BY clause.

Appears in GROUP BY clause on position ..

The column is used on position .. of a GROUP BY clause.

Predicate column resolved using index page

The predicate column occurs in the plan index and is fixed-length. No data page must be accessed for resolving the predicate search condition on this column.

Predicate column resolved using index and data page

The predicate column occurs in the plan index but is VARCHAR or VARGRAPHIC. The data page must be accessed for resolving the predicate search condition on this column.

Predicate column resolved using data page

The predicate column is not in the plan index. The data page must be accessed for resolving the predicate search condition on this column.

5.7 Predicate Analysis Warnings

Predicate analysis warnings are issued by the Text Analysis component of SQL/CA. They all have a message number starting with **AW** followed by a message code. Please refer to page 149 for a detailled description of the warning messages.

Unless the warning applies to the SQL command as a whole, the warning will be preceded by the part of the command predicate that is causing the warning.

6 SQL/CA batch facility

To start the batch facility in a disconnected server machine, the SQLCABAT EXEC should be called (usually from the PROFILE EXEC). The SQLCABAT EXEC has following syntax:

SQLCABAT

analysis_starttime
statistics_starttime
statistics_control_filename
statistics_control_filetype
statistics_control_filemode
[statistics_stoptime]

ANALYSIS_STARTTIME:

The time at which SQL/CA analysis requests forwarded by online users, should be processed, in the format HH:MM[:SS].

If no time limit for analysis is required, specify **00:00**. This will ensure that analysis requests are processed as soon as they arrive.

If no batch command analysis should be performed, specify **NONE**.

STATISTICS_STARTTIME:

The time at which SQL/CA automatic statistics updating and/or program monitoring should start, in the format HH:MM[:SS]. When that time has been reached, the EXEC will issue the EXEC SQLCABAS command with the name of the statistics control file. When execution of the program SQLCABAS terminates, SQLCABAT resumes the "wait for a user analysis request" sequence (unless the analysis time has been set to "NONE").

Notes

- If the statistics_starttime is greater than the current time when the SQLCABAT command is issued, the starttime is assumed to be in the next day.

- If no time limit for statistics is provided, specify **00:00**. This will start statistics rightaway.

- If automatic statistics should not be performed, the argument should be set to **NONE**.

STATISTICS_CONTROL_FILENAME, -FILETYPE, -FILEMODE:

If a statistics starttime has been specified, specify the name of the CMS file containing the statistics control statements, as described on page 109 and following. Please note that filename, filetype and filemode are all required parameters.

STATISTICS_STOPTIME:

The time at which automatic statistics updating and/or program monitoring should stop, in the format HH:MM[:SS]. When that time has been reached, statistics will terminate after processing of the current control command.

The stoptime may also be specified as a duration, expressed as +HH:MM. The effective stoptime is calculated as (start_time + duration), unless starttime equals 00:00, in which case the stoptime is calculated as (current_time + duration).

If stoptime is omitted, statistics terminate at the end of the control file.

Note

- If online users do not specify a userid when sending an analysis request, the server processes the request using his own default DB2 userid. In this case that userid should usually have DBA authority to be able to successfully issue the EXPLAIN command for tables it does not own.

- The batch facility always uses the default processing options from the SQL/CA OPTIONS file. The report EDIT option however is always reset during batch processing.

- All the above times may be specified as HH:MM:SS or as HH:MM.

Examples

EXEC SQLCABAT 00:00 20:00 SQLCABAS CONTROL1 A 23:00

Requests immediate processing of analysis requests and start auto-statistics at 8 p.m. using the control file SQLCABAS CONTROL1. Statistics terminate at 11 p.m. or at the end of the control file, whichever event occurs first.

EXEC SQLCABAT NONE 00:00 SQLCABAS CONTROL1 A +03:00

Do not provide analysis server facilities. Start statistics immediately and stop 3 hours later or at the end of the control file.

7 Analysis requests issued from non-VM environments

In a VM environment, the SQL/CA menu handles the forwarding of the analysis request to the analysis server. CMS SENDFILE (and RSCS) is used to send the source file.

Analysis requests may also be issued from non-VM environments that are capable of a storing a source file into the reader queue of the VM server machine.

The source file forwarded should be sent with spool class A and be identified in the VM reader queue by a CMS-like filename and filetype (as is achieved when issuing the VM CLOSE command with the NAME parameter). The filename is the name of the source program and the filetype indicates the language of the program (COBOL, PLIOPT etc).

If the server should perform analysis under a database or DB2 userid other than its own, an additional control file must be forwarded **before** the source file. That control file must be identified by the filename of the source program and a filetype of **SQLCACTL**.

The control file consists of 1 or 3 records, containing:

- the name of the database to be connected during analysis
- the name of the DB2 userid to be connected during analysis
- the password of the above DB2 userid

If one record only is supplied, it is assumed to be the databasename and analysis will be performed using the server's DB2 userid.

When analysis is complete, the analysis report is returned by the server to the requestor using CMS SENDFILE (and RSCS). The analysis report is a file, identified by the original filename and a filetype of **SQLCA**. If errors occurred during analysis, a second file with filetype **SQLCACON** may also be returned. Facilities of the non-VM guest should be used to retrieve the returned file(s).

8 SQL/CA program monitoring facility

The program monitoring facility is implemented by the **SQLCABAS** program, described in the next chapter. The facility is governed by the **MONITOR** command, which is included in the SQLCABAS control file among the other commands, described below. The monitoring report is sent to the virtual printer.

Note:

- The program monitoring facility relies on the SQL/CA archiving facility; those tables should have been created during SQL/CA installation.

- The facility allows to obtain a regular and automated command analysis for all the designated packages. The results of analysis are stored in the archive tables and analysis reports are produced.

- Program monitoring should be done under a DB2 userid that has the DBA privilege.

9 SQL/CA automatic table statistics facility

The **SQLCABAS** program implements the SQL/CA **auto-statistics** and the **program monitoring** facilities. It is usually called from the SQLCABAT EXEC described in chapter 4. It can also be invoked by any user having DBA authority, by issuing the command EXEC SQLCABAS.

The SQLCABAS auto-statistics facility is governed by a control file whose name is specified in CMS format (filename, filetype, filemode) when issuing the SQLCABAS command.

The control file should be a fixed or varying length CMS file, with a recordlength not exceeding 512 characters. It is processed sequentially and SQLCABAS processing terminates at the end of the control file.

SQLCABAS issues informatory messages, such as the current table characteristics after update statistics, to the virtual console. The console should be spooled to a user or printer in order to retrieve these messages. SQLCABAS closes the console at the end of control file processing.

Following control statements are recognized by SQLCABAS. All of them are free format, each command consisting of a number of operand words separated by a variable number of blanks. No end of command indicator is necessary and the order of the operand words within the command is not significant. However, each command must be fully contained on one line. Full blank lines in the control file are ignored.

9.1 Comment statement

Specify an asterisk (*) as the first word on the line.

9.2 CONNECT statement

CONNECT [DATABASE database] [USER user PASSWORD password]

Connects the specified DB2 database and/or userid. If the USER keyword is specified, the PASSWORD keyword is also required.

If DATABASE is specified alone, the current DB2 userid is connected to the named database.

If USER is specified alone, the designated userid is connected to the current database.

If both DATABASE and USER are specified, connection is to both the database and userid.

The CONNECT command remains in effect until a new CONNECT is encountered.

Note: Connect is done using the SQL CONNECT command, not using SQLINIT. This means that eventual user SQL programs invoked from SQLCABAS will have a connection to the default database and userid.

9.3 STATISTICS statement

STATISTICS [ALL]
[EVERY n]
[EXIT n]
[GROWTH {n|ANY}]
[INValidate]
[IREORG]
[NOREPORT]
[REBIND]
[ON day]

The STATISTICS statement defines how automatic statistics should be performed for the DB2 tables named in the subsequent TABLES or DBSPACE command(s). When a STATISTICS statement is encountered, all command parameters are set to their default values, except for the EXIT parameter which remains active if previously specified.

The parameters EVERY, GROWTH and ON define the conditions for statistics. Several conditions may be specified on the same STATISTICS statement. All of them are examined and statistics occur as soon as one condition is satisfied. If no conditions are stated, unconditional statistics is performed.

ALL

Specify ALL if an UPDATE **ALL** STATISTICS command should be issued, instead of the default UPDATE STATISTICS.

EVERY n

Specify the auto-statistics interval as a number of days. The date of the last statistics for a table is kept in an SQL/CA control table. When the difference in days between that date and the current date is greater than or equal to the EVERY parameter, statistics are updated.

EXIT

Specify the name of a user REXX EXEC to be invoked after updating the table statistics and before reprepping the dependent packages. Following parameters are passed to the user exit:

- database name
- DBspace owner and DBspace name
- table creator name and table name
- status of the first table index as **blank** if no table indexes exist, **F** if the first table index is clustered or **W** if the first index is no longer clustered
- current number of table rows
- current percentage of table overflow rows
- table growth percentage (when statistics were triggered by the GROWTH parameter)

On exit the user should pass following returncode:

0 normal exit
4 never call userexit again
8 do not call userexit again for current database
12 do not call userexit again for current dbspace

A sample user exit named SQLCABAX EXEC, is part of the product material.

GROWTH n

When specified, SQLCABAS counts the rows in the selected tables by means of a SELECT COUNT(*). When the absolute difference between the actual rowcount and the rowcount recorded at the last statistics exceeds the growth percentage specified, statistics are updated. The GROWTH **ANY** specification causes statistics when any change in the number of table rows is detected.

INValidate

If this parameter is specified, auto-statistics for a table will cause invalidation and reprepping of all packages dependent on that table. Invalidating packages implies updating the column VALID in the SYSACCESS catalog. **Therefore this command option requires DBA authority**. If the parameter is omitted, no automatic invalidation occurs.

IREORG

Specify this parameter to request automatic reorganization of all table indexes after statistics. If the parameter is omitted, no index reorganization is performed. Specify this parameter only if a DB2 version with support for the REORGANIZE INDEX command has been installed. Note that this command does **not** alter the clustering state of the index: a table or DBspace reorganization is required to achieve this. However, index-only reorganization may be used to reclaim empty index pages or to reorder fragmented indexes.

NOREPORT

After statistics, SQLCABAS displays on the console:

- the current number of table rows
- the current percentage of overflow rows
- the state of the table's first index

Specify **NOREPORT** to suppress the above output.

REBIND

This option is a better alternative for the INVALIDATE option, if you have DB2 Version 3 with support for the REBIND PACKAGE command. **REBIND** causes the dependent package(s) to be re-preprocessed by DB2. Contrarily to INVALIDATE, REBIND does not require the DBA privilege, if the packages are rebound by its creator. A DBA may rebind all packages.

ON day

If auto-statistics must be executed on a given day, specify the name of that day as sunday, monday etc.

9.4 TABLES statement

TABLES creator.tablename

Specify the name of the tables(s) for which automatic statistics should be performed, based on the criteria from the last STATISTICS command.

Both creator and tablename may contain the generic % specification, in order to select multiple tables: %.% will select all tables in the database.

9.5 DBSPACE statement

DBSPACE [owner.]dbspacename

Specify the name of the dbspace(s) for whose tables automatic statistics should be performed, based on the criteria from the last STATISTICS command.

If owner is omitted, a PUBLIC dbspace is assumed. The dbspacename may be contain the generic % specification in order to select multiple dbspaces.

9.6 MONITOR statement

MONITOR [creator.]package_name [NOREPORT]

The MONITOR commands causes the named packages to be inspected for changes in their access method or cost. The creator and package name may be generic, thus allowing monitoring of multiple packages in one command. If the monitor command is part of a control file that contains auto-statistics control commands, the command should normally be specified as the last command executed against a given database. This ensures that automatic package invalidations, forced by autostatistics, are taken into account during monitoring. Program monitoring implies invocation of the command analysis program, which stores its results in the archive tables.

An analysis report is produced for each of the packages analysed, unless the **NOREPORT** option has been specified. The analysis report is stored on the A-disk of the executing machine as a CMS file named **<fn> SQLCA**, where <fn> equals the name of the package.

The main purpose of package change monitoring, is to report transitions from indexed to DBspace scan access.

The monitor facility should execute with a DB2 userid that has the DBA privilege.

For all packages named in the MONITOR command, following actions will be taken:

- If the package has not been [automatically] reprepped since the last monitoring run, no action is taken.

- Else, a new analysis is started, an analysis report is produced (unless NOREPORT is in effect) and the analysis results are stored in the SQL/CA archive tables. The new analysis results are then compared with the previous analysis results.

- If a command in the package has a changed access method or a higher execution cost, it is printed with the new access method[15] and cost on the monitoring report. Any warnings issued during analysis are printed as well.

- For package commands that have been inserted or changed manually, the old cost and old access method will not appear on the change report.

[15]indexed access with or without key values, DBspace scan access or view materialization

9.7 VM or CMS command

A control file command not recognized as an SQLCABAS command, is considered to be a VM or CMS command, that is, the command line will be executed by SQLCABAS using the CMS SVC 202 interface, with both a standard (tokenized) and extended (non-tokenized) parameterlist.

- If a VM command has to be executed, the command line should start with the characters **CP**. For example: **CP SPOOL CONSOLE START**

- In order to execute a CMS EXEC file, start the command line with the word **EXEC**. The EXECname and all EXEC arguments should be contained within the same control file record. For example: **EXEC XXX arg_1 arg_2**

- A CMS command needs no prefix. For example: **FILEDEF PRINT** .....

Note:

Since SQLCABAS runs as a CMS nucleus extension, all CMS commands can be executed, even those that need the CMS user area.

9.8 Sample SQLCABAS Control File

```
CP SPOOL CONSOLE START *
CONNECT DATABASE DB1 USER SQLDBA PASSWORD XXXX
STATISTICS EXIT BASEXIT
STATISTICS EVERY 7 GROWTH 10 IREORG REBIND
TABLES %.%
STATISTICS GROWTH 5 IREORG REBIND
TABLES SQLDBA.ACCOUNT
MONITOR SQLDBA.%
CP SPOOL CONSOLE STOP SYSTEM
```

The above control file ensures that the statistics for all tables in the database DB1 are updated every week (**EVERY 7**) or at a 10% table growth (**GROWTH 10**), whichever occurs first. For the table "SQLDBA.ACCOUNT" statistics are updated at a 5% growth (**GROWTH 5**).

After statistics, indexes will be reorganized automatically (**IREORG**). Then the user exit **BASEXIT** is invoked. and all dependent packages reprepped (**REBIND**).

Finally, the access strategy and cost of all newly [auto]prepped packages owned by SQLDBA are compared with the access strategy and cost recorded during previous analysis (**MONITOR**). Eventual access changes are reported.

9.9 Sample Auto-statistics Exit

The function of the sample exit BASEXIT EXEC is to automatically reorganize tables

- with a weakly clustered primary index
- with more than 10% overflow rows
- with a table growth of 15% or more

Reorganization is done by calling an installation provided REORG exec.

If multiple tables reside in the same DBspace, they are all reorganized in the same run and the exit is no longer invoked for tables in that DBspace.

```
    /*   Sample auto_statistics user_exit    */

Arg database dbowner dbspace tcreator tname istat rows orows growth
If (istat = 'w') ! (orows > 10) ! (growth > 15) then do
    Dbspn = strip(dbowner) !! '.' !! Dbspace
    'EXEC SQLREORG' dbspn
    Exit 12                          /* signal dbspace processed */
End
Exit 0
```

10 SQL/CA data modelling facility

The data modelling facility of SQL/CA can be used to transfer catalog statistics for one or more Dbspaces between two DB2 systems, for example between a production system (source) and a development system (target). After the copy, DB2 access strategies for the table(s) in the Dbspace(s) processed, will be identical in both systems.

Copying is done by:

- Executing the **OBTAIN** function of the modelling facility program **SQLXDMF** in the source system. This results in the creation of a CMS *extract file* which contains the catalog data to copy.

- Executing the **SQLXDMF INSTALL** function in the target database. The INSTALL function reads the specified CMS extract file and stores its data in the target DB2 catalogs by means of an UPDATE command. Please note that DBA authority is required to update the catalogs.

The data modelling facility may also be used to modify the statistics within the same system, for simulation purposes. In this case the OBTAIN and INSTALL functions are executed against the same DB2 database and the new catalog data are inserted manually in the extract file, using XEDIT.

Both the OBTAIN and INSTALL functions request CONNECT information, as a database name, a username and a user password. This allows you to specify the source database for an OBTAIN or the target database for an INSTALL operation. The CONNECT function displays the name of the current database as a default. If the current database and user connection are correct, press ENTER on the connection screen. Otherwise, enter the new databasename and the new username and password.

The catalog statistics for a **DBspace** are not installed automatically (to avoid problems during DBspace reorganization utilities that use the NPAGES column). The DBspace section of the generated SQLXDMF SQL file is prefixed with a comment (asterisk) sign. To effectively update the DBspace statistics, the asterisk must be removed before INSTALL, that is, the keyword ***DBSPACE*** should be changed to ***DBSPACE***.

10.1 Obtain catalog statistics from source system

The OBTAIN function is invoked using the following command:

SQLXDMF OBTAIN [Dbspace_owner.]DBspacename [CMS filename]

Dbspace owner

May be omitted, in which case **PUBLIC** is assumed; a generic name should not be used here.

Dbspace name

Is required and designates the Dbspace for which statistical data should be stored in the extract file; a generic name, containing the % pattern character can be used to obtain data for several Dbspaces.

CMS filename

- Specify the name of the extract file to be created, as filename, filetype and filemode.
- If filemode is omitted, it defaults to A.
- If no filename is specified, the extract file will be named **SQLXDMF SQL A**.
- If a filename is specified, and that file already exists, the newly extracted data will be appended to the end of the file.

The following is a sample extract file:

```
*DBSPACE  PUBLIC.SQLCA_CONTROL
          NACTIVE=2
          NPAGES=512
TABLE     SQLDBA.SQLCA_CONTROL
          ROWCOUNT=53
          NPAGES=1
COLUMN    SQLDBA.SQLCA_CONTROL.CREATION_DATE
          COLCOUNT=6
          HIGH2KEY=1996-05-11
          LOW2KEY=1996-03-27
          AVGCOLLEN=5
COLUMN    SQLDBA.SQLCA_CONTROL.CREATOR
          COLCOUNT=6
          HIGH2KEY=SQLAF
          LOW2KEY=CMSVA
          AVGCOLLEN=9
          VAL10=CMSVA
          VAL50=SQLAF
          VAL90=SQLDBA
          FREQ1VAL=SQLAF
          FREQ1PCT=30
          FREQ2VAL=SQLDBA
          FREQ2PCT=26
INDEX     SQLDBA.SQLCA_CONTROL_IX
          FULLKEYCOUNT=53
          FIRSTKEYCOUNT=6
          NLEAF=1
          NLEVELS=1
          CLUSTER=F
          CLUSTERRATIO=10000
```

10.2 Modifying catalog statistics on the extract file

Between the OBTAIN and the INSTALL steps, the extract file may be altered using XEDIT. This may be required for modifying the catalog statistics in a database, when testing *what if* scenario's, such as:

- what access strategy will be used for a very large table (ROWCOUNT)?
- what if a table is not in a dedicated Dbspace (NTABS > 1)?
- what will be the effect of changing the clustering ratio of an index?

The extract file supplies following updateable information:

DBSPACE (comment * inserted by default)
- NACTIVE Number of active pages in the dbspace (INTEGER)
- NPAGES Number of pages defined in the dbspace (INTEGER)

TABLE
- ROWCOUNT Total number of rows for this table (INTEGER)
- NPAGES Number of pages in the dbspace that contain rows of this table (INTEGER)

COLUMN
- COLCOUNT Number of distinct values in this column (INTEGER)
- HIGH2KEY Second highest value in this column.
- LOW2KEY Second lowest value in this column.
- AVGCOLLEN Average length of the column (SMALLINT)

Column statistics for non-uniformly distributed column values
- VAL10 The column value at the tenth percentile.
- VAL50 The column value at the fiftieth percentile.
- VAL90 The column value at the ninetieth percentile.
- FREQ1VAL The most frequent value in the column.
- FREQ1PCT Number of rows that contain the FREQ1VAL column value, given as a percentage of the total number of rows (SMALLINT)
- FREQ2VAL The second most frequent value in the column.
- FREQ2PCT Number of rows that contain the FREQ2VAL column value, given as a percentage of the total number of rows (SMALLINT)

INDEX
- FULLKEYCOUNT Number of distinct values of the full key (INTEGER)
- FIRSTKEYCOUNT Number of distinct values of the first column of the key. Equals COLCOUNT for the index column (INTEGER)
- NLEAF Number of leaf pages in the index (INTEGER)
- NLEVELS Number of levels in the index (SMALLINT)
- CLUSTER The index clustering attribute(CHAR(1))
- CLUSTERRATIO Measures how clustered an index is (10000 for a completely clustered index) (SMALLINT)

When modifying the extract file, the following should be taken into account:

- The majority of the statistical data are positive numbers. During update, the maximum value allowed depends on the data type (integer or smallint).

- Special coding rules apply for the statistics **HIGH2KEY, LOW2KEY, VAL10, VAL50, VAL90, FREQ1VAL, FREQ2VAL**. These statistics contain **column data distribution** information. They are stored in the catalogs in an internal SQL-coded format and converted to conventional notation by SQLXDMF. A date field for instance, is stored internally in 3 bytes; the external format used by SQLXDMF is 10 characters (YYYY-MM-DD). When INSTALLing an updated value, SQLXDMF converts it back to the DB2 format before issuing the SQL UPDATE.[16] When manually modifying the above data, the following rules should be observed:

  - for [VAR]CHAR columns the update value should be supplied without enclosing quotes
  - for numeric columns a negative update value should be supplied as -nnn
  - for DECIMAL columns with a scale, no decimal point should be inserted; all digits in the fractional part of the number should be specified immediately following the non-fractional digits. For example, specify 400 as the value 4.0 of a DECIMAL(3,2) column.
  - for DATE columns the update value should be in the form YYYY-MM-DD
  - for TIME columns the update value should be in the form HH.MM.SS
  - for TIMESTAMP columns the update value should be in the form YYYY-MM-DD-HH.MM.SS.MMMMMM (where MMMMMM is the microsecond value)

- If the COLCOUNT value for an index column is supplied, the HIGH2KEY and LOW2KEY should also be supplied. If the data is not uniformly distributed, also supply values for the statistics VAL10, VAL50, VAL90, FREQ1VAL, FREQ1PCT, FREQ2VAL and FREQ2PCT.

[16]SQL-coded format is used for following data types: DATE, TIME, TIMESTAMP, DECIMAL, SMALLINT, INTEGER.

10.3 Install catalog statistics in target system

The INSTALL function is invoked using the following command:

SQLXDMF INSTALL [CMS filename]

CMS filename:
Specify the name of the extract file containing the data to be installed, as filename, filetype and filemode; if filemode is omitted, it defaults to A; if no filename is specified, the extract filename defaults to **SQLXDMF SQL A**.

By default, only table and column statistics are updated, which is usually sufficient for an install. To update the DBspace statistics as well, remove the asterisk in the keyword ***DBSPACE*** in the extract file.

If an SQL error occurs during install, the failing command and the SQLCODE are displayed. SQLCODE 100 indicates that the DB2 object copied from the source database has not been defined in the target. For example: the source system has a table index, not present in the target system.

To undo an INSTALL for a given Dbspace, run an UPDATE STATISTICS for that Dbspace. This will insert the real table statistics into the catalogs and remove the simulated ones.

10.4 Running the SQLXDMF TRANSFER function

The SQLXDMF **transfer** function provides for a combined operation of the OBTAIN and INSTALL functions. Moreover, it provides some functional enhancements

- by taking a copy of the catalog data in the target database, before performing the install
- by calling XEDIT for the SQLXDMF SQL transfer file between obtain and install. This allows for manual modifications to the catalog data before installing them.

The TRANSFER function is invoked using the following command:

SQLXDMF TRANSFER [Dbspace_owner.]DBspacename

Dbspace owner:
May be omitted, in which case **PUBLIC** is assumed; a generic name should not be used here.

Dbspace name:
Is required and designates the Dbspace for which statistical data should be stored in the extract file; a generic name, containing the % pattern character can be used to obtain data for several Dbspaces.

The function performs the following functions:

- OBTAIN the current catalog data for the designated Dbspace(s) of the target database into a CMS file named **SQLXDMF SAVE A**.

- OBTAIN the current catalog data for the designated Dbspace(s) of the source database into a CMS file named **SQLXDMF SQL A**.

- call XEDIT for the **SQLXDMF SQL A** file

- INSTALL the catalog data for the designated Dbspace(s) in the target database using CMS file **SQLXDMF SQL A**.

10.5 Running SQLXDMF in non-interactive mode

The SQLXDMF EXEC can be invoked from user-written procedures. However, the SQLXDMF program requests connection parameters in fullscreen mode to obtain the target database and a DBA userid. During non-interactive execution, these data should be stacked before invoking SQLXDMF. The databasename, the userid and its password should be stacked using separate "queue" commands. To terminate the simulated 3270 screen input, the string **<ENTER>** should be queued. The database and username may be omitted, if the default connection is adequate. The <ENTER> string must always be queued.

For example:

```
queue "DB1"
queue "SQLDBA"
queue "SQLPASS"
queue "<ENTER>"
'EXEC SQLXDMF OBTAIN" database_name
```

11 SQL/CA system processing OPTIONS file

The default SQL/CA processing options are kept in a CMS file named "SQLCA OPTIONS". During product installation, the file - with the initial defaults - is placed on the public minidisk containing the other product material. The initial options may be changed using XEDIT. The options present in the file are displayed when command analysis is invoked and may be overridden for each analysis run. When searching for the SQLCA OPTIONS file, SQL/CA uses the default CMS search order, by searching the minidisks from filemode A to Z. This allows to provide one or more users with an individual copy of the options file.

The SQLCA OPTIONS file contains the following control statements. Each statement should be coded on a separate line. The statements are free format and a variable number of blanks may appear between the words. The order of the statements in the file is not relevant.

ALLSTAT { YES | NO }

Specify YES if UPDATE **ALL** STATISTICS should be performed.

System default: NO

ARCHIVE { YES | NO }

Specify YES if analysis results should be archived.

System default: YES

AUTOSTAT { FIRST | YES | NO }

Specify AUTOSTAT FIRST if command analysis should perform an automatic update statistics for all tables referenced by the application, if an update statistics has never been issued for the table.

Specify AUTOSTAT YES if command analysis should unconditionally perform an automatic update statistics for all tables referenced by the application.

Specify AUTOSTAT NO to disable automatic update statistics.

System default: FIRST

COPY { * | userid }

Specify userid, if a copy of the extract and the hostvar files should sent to the named user. Specify * in the other case.

System default: *

DATABASE databasename

If the SQL/CA archive tables reside in a common database, state the name of that database. Omit the parameter when each database has its own archive tables.

System default: omitted

DELAY { * | userid }

If disconnected (batch) mode execution is standard, specify the userid running the SQL/CA batch facility. Specify * if the execution mode defaults to interactive.

System default: *

DEST { * | userid }

Specify userid, if a copy of the analysis report should sent after command analysis. Specify * in the other case.

System default: *

EDIT { YES | NO | W1 | W2 | W3 }]

Specify YES if the analysis report should be XEDITed after analysis.

Specify NO if the report should not be edited.

Specify W1, W2 or W3 if the report should be edited when analysis warnings have been issued with a severity code of 1, 2 or 3 respectively.

System default: YES

NOMONSTATS

Disables integration of the statistics gathered for the package by the SQL/Monitoring Facility. Specify if SQL/MF is installed but integration is not wanted.

REPORTWAIT { YES | NO}]

When analysis requests are forwarded to a server and REPORTWAIT=YES is specified, the user will have the opportunity to wait for the analysis report.

PRINT { YES | NO | W1 | W2 | W3 | SHORT}]

Specify YES if the analysis report should be printed after analysis using the current spool options of the virtual printer.

Specify NO if the report should not be printed.

Specify W1, W2 or W3 if the report should be printed when analysis warnings have been issued with a severity code of 1, 2 or 3 respectively.

Specify SHORT if the report should be printed in condensed format. A condensed report contains the command text and the related warnings only. All other print options will print the report in non-condensed format.

System default: NO

SAVE { YES | NO }

If an existing analysis report (<filename> SQLCA) for the application should be preserved, specify YES. This will rename the existing report to a CMS file named <filename> SQLCAXXX, where XXX is a sequence number assigned by SQL/CA. If no saved reports exist, the sequence number will be set to 001. If saved reports do exist, the sequence number is incremented by 1 for each newly saved report.

System default: NO

UNLOADP {SERVER |CLIENT}

This option controls the unloading of DB2 packages during their analysis in server mode.

If UNLOADP is set to CLIENT, the package is unloaded using the client's privileges and the client can unload only packages owned by him.

If UNLOADP is set to SERVER or omitted, the package is unloaded using the server's privileges and the client can indirectly unload packages not owned by him, if the server has the necessary DB2 authority.

System default: SERVER

UPCASE { YES | NO }

Specify YES if the analysis report should be converted to uppercase before printing.

System default: NO

12 SQL/CA Glossary

ACCESS

DB2 has following ways of accessing a table:

1 DBSPACE SCAN (also termed RELATIONAL SCAN)

The entire DBspace containing the table is scanned. This will also read (and lock) pages of other tables residing in the same DBspace.

In most cases, a DBspace scan is a problem. However, for small tables residing in a dedicated DBspace, a DBspace scan may be the most appropriate access strategy.

2 SELECTIVE INDEX SCAN

Selected rows of the table can be accessed using an index and specific key values, because the command predicate specifies a string that can be used to directly access an index page. Data pages will be accessed selectively during predicate resolution, when the predicate contains search conditions on columns that are not in the search index, or when the search index has VARCHAR or VARGRAPHIC columns.

3 NON-SELECTIVE INDEX SCAN

The table is accessed using an index without specific key values, because because the command predicate specifies a string that cannot be used to directly access an index page.

This happens:

- when the predicate operator does not allow direct index access (such operators are called **non key-matching**; they are listed in the table **Predicate operator evaluation** on page 186), for example : WHERE C1 IS NOT NULL

- when the predicate omits leading columns in a composite index, for example : WHERE C2 = x (and the index is on C1,C2)

- when there is some other coding inefficiency in the predicate, such as: WHERE C1=:HOSTVAR+10 or WHERE C1=:HOSTVAR (and the datatype of hostvar is not compatible with that of C1)

A non-selective index scan accesses all the pages of the index. It will also access table data pages:

- when the predicate contains search conditions on columns not available in the search index
- when a column of the search index has the VARCHAR or VARGRAPHIC datatype

An index scan may be chosen by DB2 as an alternative for a DBspace scan, for example, when the relational scan productivity is low (many pages not belonging to the table would be scanned because the table's SYSCATALOG.PCTPAGES value is low). A true DBspace scan might be performed under different circumstances, for instance when the table gets larger or when less tables share the same DBspace. In some cases, a non-selective index scan may be worse than a DBspace scan, since both index and data pages must be accessed, whereas a DBspace scan accesses data pages only.

4 SELECTIVE OR NON-SELECTIVE INDEX-ONLY SCAN

All columns in the SELECT list and in the predicate are indexing columns and can be retrieved without accessing the data pages. A selective index-only scan is the most efficient access path. In DB2 versions before 3.4, EXPLAIN does not signal an index-only scan.

5 FULLY-QUALIFIED INDEX SCAN

All columns of a unique index can be used for the index scan.

ADDITIONAL SORT

DB2 performs an additional sort at the end of the query block. This may be caused by ORDER BY, GROUP BY or SELECT DISTINCT clauses. Note that a small number of rows will be sorted by DB2 in storage with no I/O involved. Larger response sets will be sorted using internal DBspaces.

ATOPEN

A dependent block may be executed once when the corresponding parent block is initiated (**ATOPEN=YES**) or it may be executed for each iteration in the parent block (**ATOPEN=NO**). In the latter case, the iteration count of the dependent block is the product of its own estimated iteration and that of its parent(s). If a block has multiple ancestors and each invocation has ATOPEN=NO, the iteration count and the execution cost may become very high. Therefore SQL/CA will issue a warning. However, high iteration rates may exist, even with an ATOPEN=YES block, because DB2 may save the result of the dependent block in internal DBspaces and iteratively search these Dbspaces, instead of executing the dependent query. This situation is not visible in the EXPLAIN results. In any case, you should try to keep the number of matching subquery rows as low as possible. Using a correlated subquery may reduce the size of the subquery result.

CHILD BLOCK

The opposite of parent block. See *PARENT BLOCK*.

CLUSTERED INDEX

An index is clustered if the sequence of its entries reflects the physical ordering of the table rows. The clustering attribute of a given index is stored in SYSINDEXES as a flag and as a clustering ratio. The latter is a value ranging from 0 to 10000 and represents the degree of clustering, where 10000 is the optimal clustering ratio.

An index is termed "weak" by SQL/CA, if the SYSINDEXES flag says so, or if the clustering ratio drops below 6000. The clusterratio on the SQL/CA index report equals the DB2 clusterratio divided by 100.

COMMAND COST SUMMARY

The cost value from the DB2 explain cost table indicates for a given query block the total execution cost estimate. This estimate includes the cost of eventual dependent blocks. For commands containing multiple queries, SQL/CA provides a more refined cost computation method. For each query block, following values are computed:

Single_Cost
The cost of the query block, not including the cost of the dependent and the ancestor blocks.

Exec_Times
The estimated number of invocations from all parent blocks. The number of invocations of the parent blocks themselves and the ATOPEN flags are taken into account.

Multi_Cost
Single_Cost multiplied by Exec_Times.

SQL_Cost
The cost from the DB2 explain table i.e. the command cost plus the cost of all its dependent blocks.

Highest_Subquery_Cost
The highest Multi_Cost found for a subquery in the command.

For both single and multiple query commands, following cost information is provided:

Total Command Cost
The cost estimate associated by DB2 with the command as a whole.

ISQL-like Cost
(total_command_cost / 1000) + 1. Interactive SQL systems such as ISQL present the Query Cost Estimate in this format.

COMPOSITE

When a command is performed in several steps, the DB2 term "composite" refers to the response set obtained thus far, as the result of execution in previous steps. In joins, the composite is also termed the *outer* table.

DBSPACE SCAN

A synonym of RELATIONAL SCAN. Refer to the keyword *ACCESS*.

DBSPACE SCAN PRODUCTIVITY

A value computed by SQL/CA to estimate the percentage of table data actually accessed during a DBspace scan. The value is taken from the PCTPAGES column in the SYSCATALOG row for the table.

For critical, operational tables productivity should normally be 100%, that is, the table should have its dedicated DBspace. Otherwise, a DBspace scan will imply unnecessary I/O by reading pages of other tables and unproductive locks on pages that contain rows belonging to other tables.

DEFAULT FILTER FACTOR

The DB2 Optimizer may not be able to use significant filter factors during path selection. Default filter factors are then adopted.

This will be the case when:

- no catalog statistics are available to compute the filter factor
- the predicate contains host variables (which do not have a value when the access strategy is determined during program preprocessing)
- the predicate contains expressions
- disjunctive predicates are used, by means of the OR connector

When no catalog statistics are available, the default filter factor is as follows:

- for the = operator 0.040 (4% qualifying rows)
- for LIKE and BETWEEN 0.100 (10% qualifying rows)
- for all other operators 0.333 (30% qualifying rows)
- for the <> operator 0.960 (96% qualifying rows)

When statistics are available, the default filter is as follows:

- for the = operator 1/COLCOUNT (where COLCOUNT is the estimated number of distinct values in the column)

- for all other operators see the default filter factor table on page 187.

ESTIMATED GLOBAL COMMAND FILTER FACTOR

Represents the fraction of table rows estimated to satisfy the command predicate as

(estimated number of satisfying rows) / (sum of all rows of all tables accessed)

The filter value ranges from 0 to 1. The lower the value, the better the selectivity of the command predicate.

ESTIMATED EXECUTION ITERATION BY ANCESTOR BLOCKS

Using the TIMES column for all ancestor ("parent") blocks, SQL/CA computes the total number of times this query block will actually be executed. The ATOPEN attribute of each ancestor block is also taken into account.

ESTIMATED NUMBER OF ROWS PROCESSED

The ROWCOUNT column from the EXPLAIN Structure table. It indicates the estimated number of rows returned for the table(s) used in this command. The filtering factors associated with the columns referenced in the command predicate, intervene in estimating the size of the response set. The ROWCOUNT column may be zero for small response sets. SQL/CA computes the total number of table rows out of which the estimated number of rows is selected. For join commands, the total counts the rows for all tables participating in the join. If the estimated value equals the number of table rows, SQL/CA assumes a sequential table scan and issues a warning.

ESTIMATED TIMES PREDICATE CONDITIONS MET

The TIMES column from the EXPLAIN Structure table. It estimates the probability that the predicate conditions of the command will be true. It also shows how many times eventual dependent blocks will be executed.

FILTER FACTOR

For each column reference in the command predicate, DB2 determines a filter factor, which represents the fraction of table rows estimated to satisfy the predicate as:

estimated_number_of_rows / number_of_table_rows.

The filter is a value between 0.0 and 1.0. The lower the value, the better the column's selectivity. A filter factor of 1 means the column has no selectivity at all. The filter factor depends on the distribution of data values for the column. A column having a high number of unique values within the table, will have a good selectivity, that is, a small filter factor. If an SQL command supplies a search value for such a column, the Optimizer knows that only a few number of rows will meet the condition and that the response set will be small. This will reduce the command execution cost and determine the access strategy. A column with a small filter is also a good index candidate.

The filter factor chosen depends on the predicate operator used, as shown on page ?.

FULLY-QUALIFIED INDEX SCAN

Refer to the keyword *ACCESS.*

INDEX DISQUALIFIED

An index is present but not used by the optimizer, because the application command specifications are inhibiting it. The optimizer will use a DBspace scan or an index scan (scanning the entire table using the index). Whether a DBspace or an index scan is chosen, depends on factors such as the index clustering ratio, the percentage of table pages in the DBspace etc.

INDEX SCAN

Refer to the keyword *ACCESS.*

INDEX-ONLY SCAN

Refer to the keyword *ACCESS.*

KEYMATCHING

A predicate is called keymatching, when the columns used in the predicate can be formed into a string that matches existing entries of the table index. Such a predicate allows direct retrieval of a qualifying key and row. **To be a candidate for keymatchinq, the predicate must be sargable.**

Examples of key-matching predicates:

```
C1 = 1
C1 > 1
```

Examples of non key-matching predicates:

```
C1 <> 1
C2 NOT LIKE :VAR
```

Given a composite index on (C1,C2):

C1 = 1 is key-matching
C2 = 1 is not key-matching

MATERIALIZATION

View materialization is a technique used by DB2 version 3 in order to remove restrictions on the processing of views. The feature implies storing of intermediate select results in internal tables. To access these intermediate results, indexed access is never used by DB2. Hence the possibly negative performance implications of view materialization.

MERGE SCAN JOIN

DB2 scans the composite and the new table in the order of the join column and joins rows with matching columns. A sort operation may be necessary to access the new table and or the composite in the required order and additional work dataspace (internal DBspaces) may be required. Therefore, a merge scan join is usually less performant than a nested loop join.

METHOD

Represents the action performed by DB2 for a given plan: a merge scan join, a nested loop join, an additional sort plan or a view materialization.

NESTED LOOP JOIN

For each row of the composite, matching rows of the new table are located and joined. An index may be used to access the new table.

NEW TABLE

When a join is being executed, this DB2 term refers to the new table that is being accessed in the current step (plan) and joined to the data resulting from previous steps (the composite table). The new table is also called the *inner* join table.

NO EXPLICIT SORT PLAN

The ordering clause requested by the command can be satisfied using the index order without requiring a specific sort operation.

NON-SELECTIVE INDEX SCAN

Refer to the keyword *ACCESS*.

PARENT BLOCK

In a multiple query structure, a parent or ancestor block is the subquery that initiates another subquery, which in turn is called a child or dependent block.

PLAN

An SQL command may be executed in several steps. Each step is called a plan by SQL EXPLAIN and receives a plan number, representing the order in which the command's plans are executed. There are specific plans for join, sort and view materialization operations.

PREDICATE

The search condition in the WHERE clause of an SQL statement.

There are four types of predicates. They are, in decreasing order of performance:

Keymatching

A predicate is keymatching, when it can be applied against an index for direct retrieval of qualifying keys. The predicate is applied by the DB2 Database Subsystem (DBSS).

Index sarg

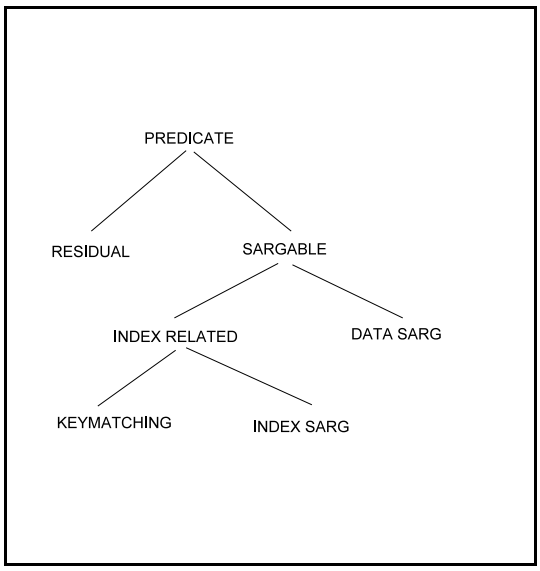
An index sarg is a predicate that can by applied by the DB2 Database Subsystem (DBSS) against keys in index leaf pages.

Data sarg

A data sarg is a predicate that can by applied by the DB2 Database Subsystem (DBSS) against column values in data pages.

Residual

A residual predicate cannot be applied by the DB2 Database Subsystem (DBSS), but is evaluated by the DB2 Relational Data System (RDS). This implies that DBSS must obtain all rows that are to be evaluated by RDS.



RANGE PREDICATE

A range predicate contains the SQL verbs LIKE or BETWEEN or one of the range operators >, >=, <, <=.

A *DEFAULT FILTER FACTOR* is used for a range predicate, when it has the format:

- Column range_operator Host variable
- Column range_operator Column
- Column LIKE expression
- Column BETWEEN expression AND expression

RESIDUAL

The opposite of *SARGABLE.* See SARGABLE.

SARGABLE

An expression is sargable if it can be evaluated by the DB2/VM Database Subsystem (DBSS), that is, while data is being retrieved from the I/O system. If the expression is not sargable (also called a RESIDUAL expression) it is evaluated by the DB2/VM Relational Data System (RDS) after calling the Database Subsystem to obtain data. Filtering the data (testing whether they satisfy the predicate specifications) at the RDS level causes additional DBSS calls and increases the processing overhead.

Only sargable expressions are keymatching candidates. A predicate containing residual expressions only, is resolved using an index or dbspace scan.

SCORE

An attempt to measure the efficiency of the DB2 access path. The higher the score, the better the path.

One "point" is added to the score each time one of the following conditions is true:

- index access is being performed
- fully-qualified index access is being performed
- index-only access is being performed
- selective index access is being performed
- index access uses a highly-clustered index
- index access uses a unique index

The highest score is achieved, when all the above conditions are true. The highest score is 6 (DB2 Version 3.4 and later) or 4 (DB2 versions before 3.4) The lowest score 0 indicates a DBspace scan.

SELECTIVE INDEX SCAN

Refer to the keyword *ACCESS.*

UNCLUSTERED INDEX

The opposite of a clustered index. If the first index is unclustered, index reorganization is needed to make the index clustered again. For a non-first index, the indexing column definition may be such that the index will always be unclustered. A non-clustered index is less efficient than a clustered index.

WEAKLY CLUSTERED INDEX

DB2 maintains a clustering ratio for each index as a value between 0 and 10000, where 10000 is the best clustering ratio. If DB2 still considers the index as clustered, SQL/CA considers the index as weakly clustered, if its clustering ratio drops below 6000. Also see *UNCLUSTERED INDEX* .

13 SQL/CA archive tables

If you intend to write your own queries against the SQL/CA Archive Tables, following description of the archive tables may be useful. The tables Archive_Cost, Plan, Reference and Structure contain the data from the DB2 Explain tables when the program was analyzed. The Archive_Index table contains SQL/CA specific data. When an SQL command is stored in the archive tables, a unique "archive querynumber" is assigned. The QUERYNO column may be used to join the rows of the different tables pertaining to a particular command.

SQLCA_INDEX one row per command

QUERYNO	INTEGER	
FILENAME	CHAR(8)	CMS filename of source program
FILETYPE	CHAR(8)	CMS filetype (**CSPAPPL** for a CSP application, **ACCMOD** for an)
FILEMODE	CHAR(2)	CMS filemode
DBASE	CHAR(8)	databasename where analysis has been done
USERID	CHAR(8)	DB2 userid performing analysis
STAMP	TIMESTAMP	time of analysis
SQLCA_WARNING	SMALLINT	highest SQL/CA warning severity
SQLCA_W1	CHAR(1),	if an SQL/CA warning of class 1-19 has been issued
SQLCA_W2	CHAR(1),	the corresponding SQLCA_Wx column
SQLCA_W3	CHAR(1),	contains **Y** or blank otherwise. For a list of the
SQLCA_W4	CHAR(1),	warning message classes,
SQLCA_W5	CHAR(1),	please refer to page 67.
SQLCA_W6	CHAR(1),	
SQLCA_W7	CHAR(1),	
SQLCA_W8	CHAR(1),	
SQLCA_W9	CHAR(1),	
SQLCA_W10	CHAR(1),	
SQLCA_W11	CHAR(1),	
SQLCA_W12	CHAR(1),	
SQLCA_W13	CHAR(1),	
SQLCA_W14	CHAR(1),	
SQLCA_W15	CHAR(1),	
SQLCA_W16	CHAR(1),	
SQLCA_W17	CHAR(1),	
SQLCA_W18	CHAR(1),	
SQLCA_W19	CHAR(1),	
SQLCA_W20	CHAR(1),	
SQLCA_W21	CHAR(1),	
SQLCA_W22	CHAR(1),	
SQLCA_W23	CHAR(1),	
SQLCA_W24	CHAR(1),	
SQLCA_W25	CHAR(1),	

```
SQLCA_W26    CHAR(1),
SQLCA_W27    CHAR(1),
SQLCA_W28    CHAR(1),
SQLCA_W29    CHAR(1),
SQLCA_W30    CHAR(1),
SQLCA_W31    CHAR(1),
SQLCA_W32    CHAR(1),
REMARK         CHAR(46)        filename   &   line   or   CSP
                                application & processname
SHORT_COMMAND  CHAR(72)        first characters of SQL command
COMMAND        VARCHAR(8192)   full SQL command text
```

The columns FILENAME, FILETYPE, FILEMODE, DBASE and USERID constitute the logical and unique key for an archived command.

SQLCA_COST one row per command query

QUERYNO	INTEGER	
QBLOCKNO	SMALLINT	query number: significant if command contains subqueries
COST	DECIMAL(15)	DB2 cost of the query

SQLCA_PLAN one row per query plan

QUERYNO	INTEGER	
QBLOCKNO	SMALLINT	query number
PLANNO	SMALLINT	plan number: significant for joins and sorts
METHOD	SMALLINT	classifies plan as join|sort
CREATOR	CHAR(8)	creator of table
TNAME	CHAR(18)	tablename
TABNO	SMALLINT	internal table number
ACCESSTYPE	CHAR(1)	DBspace scan|index usage
ACCESSCREATOR	CHAR(8)	creator of index used
ACCESSNAME	CHAR(18)	name of index used
SORTNEW	CHAR(1)	Y if new table is sorted
SORTCOMP	CHAR(1)	Y if composite is sorted

SQLCA_REFERENCE one row per column referenced in the command query

QUERYNO	INTEGER	
QBLOCKNO	SMALLINT	
CREATOR	CHAR(8)	
TNAME	CHAR(18)	
TABNO	SMALLINT	
COLNO	SMALLINT	SYSCOLUMNS.COLNO
COLNAME	CHAR(18)	columnname (inserted by SQL/CA)
FILTER	DECIMAL(4,3)	column filter factor
DETAIL	CHAR(28)	details for reference

SQLCA_STRUCTURE one row per command query

QUERYNO	INTEGER	
QBLOCKNO	SMALLINT	
ROWCOUNT	INTEGER	estimated nr of rows returned
TIMES	DECIMAL(15)	estimated nr of times executed
PARENT	SMALLINT	blocknumber of parent query
ATOPEN	CHAR(1)	Y if query executed once at open of parent

14 SQL/CA analysis warning messages

The following messages are issued by the Text Analysis component of SQL/CA. The part of the command predicate the message is referring to, will be printed before the message.

The periods in the message text are replaced by the column, hostvar or constant name, length or datatype, whichever applies.

The following descriptive texts assume a predicate expression with the format "column operator expression" (eg. column=constant). The inverse format (constant=column) is also handled during analysis.

Most of the questionable conditions causing a warning, are fully described in the IBM publication **Performance Tuning Handbook** for DB2/VM (Document Number SH09-8111-00).

A severity code 1, 2 or 3 is associated with each warning. The higher the severity code, the higher the assumed impact on command performance. For some warnings, a higher severity code is issued when the warning is related to an indexing column. The same warning has a lower severity if issued for a non-indexing column.

AW01 Expression on index column .. is suboptimal

Reason

The predicate contains an arithmetic expression or a builtin function reference involving an indexing column, such as **colname-100 = 500** or **colname =:hostvar+100**. Such predicates are not key-matching and not sargable.

Action

Transfer the expression to the other member of the predicate expression (in the first example: "colname > 600") or compute the host variable expression (the second example) using host language commands.

AW02 Datatype .. of column .. not compatible with datatype .. of column ..

Reason

An incompatibility has been found between the datatypes of the designated predicate columns. Such predicates are not key-matching and not sargable. If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Action

The easiest solution is to use identical datatypes. If this is not possible, use compatible datatypes.

Refer to the datatype evaluation table on page 185.

AW03 Datatype .. of column . not compatible with datatype .. of hostvar ..

Reason

An incompatibility has been found between the datatypes of the designated predicate column and the host variable. Such predicates are not key-matching and not sargable. If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Action

Refer to the datatype evaluation table on page 185.

AW04 Length .. of column .. exceeds length .. of column ..

Reason

An incompatibility has been found between the lengths of the designated predicate columns. Such predicates are not key-matching and not sargable.If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Lengths A and B are considered compatible if length(A) >= length(B), except for VARCHAR columns with length < 254 and VARGRAPHIC columns with length < 127, which are compatible with all character values, fixed or variable of any length.

For instance, if column A has been defined as CHAR(20), column B specified as CHAR(25) violates the above rule.

Action

Ensure that the length of the right hand predicate expression is less than or equal to the length of the left hand expression.

AW05 Length .. of hostvar .. exceeds length .. of column ..

Reason

An incompatibility has been found between the lengths of the designated host variable and the predicate column. Such predicates are not key-matching and not sargable.If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Lengths A and B are considered compatible if length(A) >= length(B), except for VARCHAR columns with length < 254 and VARGRAPHIC columns with length < 127, which are compatible with all character values, fixed or variable of any length.

For instance, if a table column has been defined as CHAR(20), a host variable specification of CHAR(25) violates the above rule.

Action

Ensure that the length of the right hand predicate expression is less than or equal to the length of the left hand expression.

AW06 Length .. of constant .. exceeds length .. of column ..

Reason

An incompatibility has been found between the lengths of the designated constant and the predicate column. Such predicates are not key-matching and not sargable. If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Lengths A and B are considered compatible if length(A) >= length(B), except for VARCHAR columns with length < 254 and VARGRAPHIC columns with length < 127, which are compatible with all character values, fixed or variable of any length.

For instance, if a table column has been defined as CHAR(20), a constant specification of length 25 violates the above rule.

Action

Ensure that the length of the right hand predicate expression is less than or equal to the length of the left hand expression.

AW07 Precision .. of column .. exceeds precision .. of column ..

Reason

An incompatibility has been found between the precisions of the designated predicate columns. Such predicates are not key-matching and not sargable. If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Precisions A and B are considered compatible if precision(A) <= precision(B).

Precision applies to DECIMAL columns only and represents the total number of digits, including the digits following the decimal point. For instance, if a table column A has been defined as DEC(2), a column B specification of DEC(3) violates the above rule.

Action

Ensure that the precision of the right hand predicate expression is less than or equal to the precision of the left hand expression.

AW08 Precision .. of hostvar .. exceeds precision .. of column ..

Reason

An incompatibility has been found between the precisions of the designated host variable and the predicate column. Such predicates are not key-matching and not sargable. If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Precisions A and B are considered compatible if precision(A) >= precision(B).

Precision applies to DECIMAL columns only and represents the total number of digits, including the digits following the decimal point. For instance, if a table column has been defined as DEC(2), a host variable specification of DEC(3) violates the above rule.

Action

Ensure that the precision of the right hand predicate expression is less than or equal to the precision of the left hand expression.

AW09 Precision .. of constant .. exceeds precision .. of column ..

Reason

An incompatibility has been found between the precisions of the designated constant and the predicate column. Such predicates are not key-matching and not sargable. If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Precisions A and B are considered compatible if precision(A) >= precision(B).

Precision applies to DECIMAL columns only and represents the total number of digits, including the digits following the decimal point. For instance, if a table column has been defined as DEC(2), a constant specified as DEC(3) violates the above rule.

Action

Ensure that the precision of the right hand predicate expression is less than or equal to the precision of the left hand expression.

AW10 Scale .. of column .. does not match scale .. of column ..

Reason

An incompatibility has been found between the scales of the designated predicate columns. Such predicates are not key-matching and not sargable. If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Scales A and B are considered compatible if scale(A) = scale(B) where B is not a constant. Scale applies to DECIMAL columns only and represents the total number of digits following the decimal point.

Action

Ensure that the scale of the right hand predicate expression is equal to the scale of the left hand expression or that the scale of a constant does not exceed the scale of the left hand expression.

AW11 Scale .. of column .. does not match scale .. of hostvar ..

Reason

An incompatibility has been found between the scales of the designated host variable and the predicate column. Such predicates are not key-matching and not sargable. If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Scales A and B are considered compatible if scale(A) = scale(B) where B is not a constant. Scale applies to DECIMAL columns only and represents the total number of digits following the decimal point.

Action

Ensure that the scale of the right hand predicate expression is equal to the scale of the left hand expression or that the scale of a constant does not exceed the scale of the left hand expression.

AW12 Scale .. of column .. does not match scale .. of constant ..

Reason

An incompatibility has been found between the scales of the designated constant and the predicate column. Such predicates are not key-matching and not sargable. If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Scales A and B are considered compatible if scale(A) >= scale(B). Scale applies to DECIMAL columns only and represents the total number of digits following the decimal point.

Action

Ensure that the scale of the right hand predicate expression is equal to the scale of the left hand expression or that the scale of a constant does not exceed the scale of the left hand expression.

AW13 Indicator variable used with NOT NULL column ..

Reason

An EQUALS predicate involves an indexing column defined as NOT NULL and a host variable followed by an indicator variable. This causes the expression to become non-sargable. If an indexing column is involved, a warning severity code 3 is issued, as this condition may prevent the optimizer from using an index. For incompatible, non-indexing columns the severity code is set to 1.

Action

Omit the indicator variable, since the column can never be null.

AW14 Logical connector OR is suboptimal

Reason

A predicate is connected to a previous predicate in the same command by means of an outer OR connector (inner OR connectors are not warned). OR prevents the use of key-matching predicates. Moreover, if an Or-ed predicate is not sargable, the entire command predicate becomes not sargable.

Action

Use alternative predicate specifications for the OR connector, such as UNION, IN with a value list etc.

For example:

SELECT ... WHERE A=value_1 OR A=value_2 OR A=value_3 should be stated as:
SELECT ... WHERE A IN (value_1,value_2,value_3)

AW15 Logical connector NOT is suboptimal

Reason

A predicate of the type "NOT operand_1 operator operand_2" has been detected. Such predicates are not sargable.

Action

Most of these predicates can be specified without the NOT connector.

"WHERE A >= B" is a better performer than "WHERE NOT A < B".

AW16 Predicate operator .. is not a key-matching candidate

Reason

The predicate operator displayed cannot be used for direct index access. This will cause additional I/O requests during command execution.

Action

If the command logic allows it, select a key-matching operator. See the table **Predicate operator evaluation** on page 186 and the Glossary on page 133 under *KEY-MATCHING.*

AW17 Predicate operator .. is not sargable

Reason

The predicate operator displayed will be evaluated by RDS and not by DBSS. This will cause additional CPU consumption during execution.

Action

If if the command logic allows it, select a sargable operator. See the table **Predicate operator evaluation** on page 186 and the Glossary on page 133 under *SARGABLE*.

AW18 Default filter factor used for range predicate is ..

Reason

The default filter factor shown is used for the predicate, because it has the format

- COL range_operator HOSTVAR
- COL range_operator COL ("range_operator" is one of the operators >, >=, <, <=)
- COL LIKE expression
- COL BETWEEN expression AND expression

The default filter factor depends on the number of distinct values for the table column, as stated in the table **Default filter factor** on page 187.

Action

You should try to keep the filter factor as small as possible, since it determines the estimated command's response set.

Use LIKE/BETWEEN instead of >, >=, <, <=. The default filter factor table shows that LIKE/BETWEEN is three times more selective.

Ensure that a COLCOUNT is available: update the table statistics if it is not.

AW19 Range predicates used with hostvars may disqualify the index

Reason

Predicate operators such as BETWEEN and LIKE, but also >, >=, <, <= are called "range operators". When used with host variables, the selectivity of the predicate (i.e. the number of table rows that will satisfy the predicate conditions) is unknown, since the values contained in the host variables are not known at prep time. Therefore, DB2 applies default selectivity rules. These defaults are rather pessimistic (a selectivity ranging from 20-30%) and may effectively disqualify the index, especially on large tables, whereas the same SQL command executed dynamically using ISQL or QMF, with constants specified instead of hostvars, will use the index.

If an indexing column is involved, a warning severity code 3 is issued. For non-indexing columns the severity code is set to 1.

Action

If range predicates cannot be avoided, specifying additional command predicates may help the optimizer to determine a more adequate selectivity. You can try to make an eventual DBspace scan less productive and hence less attractive to the optimizer, by ensuring that the table's PCTPAGE value is low, for example by putting a large infrequently accessed table in the same DBspace. Dynamic command execution can be considered, but this involves a serious programming effort (except in CSP).

AW20 Datatypes .. (column ..) and constant .. incompatible
Expression not sargable.

Reason:

The predicate is not sargable due to the datatype of the column and the implied datatype of the constant.

Action

The warning can be avoided be re-specifying the items datatype.

Please refer to the table **Decimal precision evaluation** on page 185.

AW21 Datatypes .. (column ..) and .. (hostvar ..) incompatible
Expression not sargable.

Reason:

The datatypes of the involved column and host variable are such that their evaluation is not sargable. Please refer to the table **Decimal precision evaluation** on page 185.

Action

The warning can be avoided be re-specifying the items datatype.

AW22 No predicate specifications for leading column(s) .... of plan index ..

Reason

The predicate specifies one or more columns that are part of a multicolumn index but omits leading indexing columns. For instance: the index is defined on columns C1 and C2 and the predicate references C2 only.

Action

Omitting leading index columns will lead to a non-selective index scan. Since the predicate is incomplete, it cannot be used by the optimizer to directly access an index entry. Instead, the index will be scanned sequentially to evaluate the predicate. During evaluation, data pages may also be scanned.

AW23 No index columns referenced in the command predicate for table (..) ..

Reason

For the designated table, no predicate is stated or the predicate states non-indexing columns only. This may lead to a DBspace or an index scan.

Action

If the command logic allows it, specify a predicate with indexing columns, or create an index for one of the predicate columns.

AW24 JOIN predicate not index eligible

Reason

A join predicate has been found, but index use has been disqualified because the join columns do not have the same datatype, length or scale. Note that the warning applies to the predicate as a whole and that the offending predicate clause may not be displayed before the warning.

Action

Ensure that the join columns have identical characteristics.

AW25 Missing search condition on JOIN command

Reason

A join predicate (T1.COL1=T2.COL1) specifies additional predicates on the join columns. There should be an equal number of additional predicates for both join columns. Omissions are completed by the DB2 Optimizer only when the missing condition is sargable and an equijoin is being performed. This was not the case for the command predicate warned.

Action

Repeat the local predicate specified for the first join column as a local predicate for the other join column.

A command such as

*SELECT * FROM TABA T1,TABB T2 WHERE T1.COL=T2.COL AND T1.COL>5*

will result in AW25. Add the local predicate AND T2.COL>5. This gives the Optimizer a better choice when determining the JOIN inner / outer table.

NOTE

- In general, try to specify as many local predicates as possible (that is, predicates other than the join predicate). DB2 evaluates the local predicates before performing the join. Therefore, local predicates reduce the size of the data to be joined, which has a beneficial performance impact.

- During a nested loop join, DB2 attempts to take the smallest table as the first one in the join (the "outer" table). Specifying all local predicates helps the Optimizer during determination of the inner or outer table.

AW26 Missing search condition on JOIN command will be added by Optimizer

Reason

A join predicate (T1.COL1=T2.COL1) specifies additional predicates on the join columns. There should be an equal number of additional predicates for both join columns. Omissions are completed by the DB2 Optimizer[17] only when the missing condition is sargable and an equijoin. This was the case for the command predicate warned.

Action See the Notes under AW25 above.

The command SELECT * FROM TABA T1, TABB T2 WHERE T1.COL=T2.COL AND T1.COL=5 will result in AW26. The Optimizer adds the implied clause AND T2.COL=5.

AW27 A decimal scale (column ..) is incompatible with .. (column ..)
Expression not sargable.

Reason:

One of the predicate columns is scaled DECIMAL and the other column is SMALLINT or INTEGER. Therefore, the expression is residual. Please refer to the table **Decimal precision evaluation** on page 185.

Action:

The warning can be avoided be re-specifying the datatype or scale of the items involved.

AW28 A decimal scale (column ..) is incompatible with .. (hostvar ..). Expression not sargable.

Reason:

One of the predicate elements is a scaled DECIMAL column and the other element is a SMALLINT or INTEGER hostvar. Therefore, the expression is residual. Please refer to the table **Decimal precision evaluation** on page 185.

Action:

The warning can be avoided be re-specifying the datatype or scale of the items involved.

[17] in a process called *transitive closure*

AW29 Decimal precision < 4 (column ..) incompatible with SMALLINT (column ..). Expression not sargable.

Reason:

One of the predicate columns is DECIMAL with precision < 4 and the other column is SMALLINT. Therefore, the expression is residual. Please refer to the table **Decimal precision evaluation** on page 185.

Action:

The warning can be avoided be re-specifying the datatype or precision of the items involved.

AW30 Decimal precision < 4 (column ..) incompatible with SMALLINT (hostvar ..). Expression not sargable.

Reason:

One of the predicate elements is a DECIMAL column with precision < 4 and the other element is a SMALLINT hostvar. Therefore, the expression is residual. Please refer to the table **Decimal precision evaluation** on page 185.

Action:

The warning can be avoided be re-specifying the datatype or precision of the items involved.

AW31 Decimal precision < 10 (column ..) incompatible with INTEGER (column ..). Expression not sargable.

Reason:

One of the predicate columns is DECIMAL with precision < 10 and the other column is INTEGER. Therefore, the expression is residual. Please refer to the table **Decimal precision evaluation** on page 185.

Action:

The warning can be avoided be re-specifying the datatype or precision of the items involved.

AW32 Decimal precision < 10 (column ..) incompatible with INTEGER (hostvar ..). Expression not sargable.

Reason:

One of the predicate elements is a DECIMAL column with precision < 10 and the other element is an INTEGER hostvar. Therefore, the expression is residual. Please refer to the table **Decimal precision evaluation** on page 185.

Action:

The warning can be avoided be re-specifying the datatype or precision of the items involved.

AW34 No predicate specifications for trailing column(s) .... of plan index ..

Reason

The plan index is a composite index and the predicate does not contain references for trailing index columns. For instance: the index is defined on columns C1 and C2 and the predicate references C1 but not C2.

Action

If the omitted columns are not significant in the conditions stated by the predicate, this warning should be ignored. Otherwise, specify the missing columns.

AW35 No predicate specifications for column(s) .... of plan index ..

Reason

The predicate does contain indexing column references (otherwise message **AW23** would have been produced), but the DB2 optimizer has opted for another table index (the "plan index"). No columns of this index are referenced in the predicate.

Action

In most cases, you should examine why the intended index is not used. If that index is composite, you probably do not specify all the indexing columns. The index could also have been discarded due to its unclustered state or its weak clustering ratio.

AW36 BETWEEN is a more selective operator than ..

Reason

The predicate contains a range operator such as >, <, >= or <=. Such predicates are less selective than BETWEEN. See the table **Default filter factor** on page 187.

Action

Replace the range operator indicated in the message with a BETWEEN or LIKE clause.

AW37 Predicate uses columns of the same table

Reason

The predicate has the form T1.column_1 <operator> T1.column_2. Such predicates are no key-matching candidates and they are not sargable.

Action

Try to formulate the query so that different tables are used within the same predicate expression. Consider replacing the COL=COL form with a COL=HOSTVAR specification.

AW38 Predicate using indicator variables is not sargable

Reason

Predicates with indicators are resolved by the Relational Subsystem, not by the Database Subsystem. Additional processing overhead will be involved.

Action

There is usually no reason for using indicator variables in predicates, since the "unknown" predicate state is functionally equivalent to the "false" state, that is, both are not "true".

AW39 JOIN is usually more efficient than a subquery

Reason

A join access often outperforms a nested quey.

Action

Many nested queries can be formulated as a JOIN.

AW40 All predicates residual due to residual OR-ed predicate

Reason

A query contains multiple predicates and at least one of them is connected (at the outer level) by means of the OR connector. All non-OR-ed predicates are sargable, but one or more of the OR-ed predicates is not. This makes the whole predicate non-sargable.

For example :

WHERE C1=5 OR C2=C3

If C1, C2 and C3 are columns of the same table:

C1=5 is sargable
C2=C3 is residual (and you get warning AW37)

Due to the OR connector however, the entire predicate is residual.

Action

Try to avoid OR-ed predicates. If you can't, try to make the OR-ed predicate residual. In most cases, you will have an SQL/CA warning indicating why that predicate is residual.

AW41 UNION ALL may avoid unnecessary sort

Reason

Multiple UNION's are specified. Since UNION eliminates duplicate rows, multiple sorts are possible. These sorts can be avoided by replacing all UNION's, except for the last one, by UNION ALL. UNION ALL does **not** eliminate duplicates.

Action

Specify UNION ALL except for the last UNION.

For example:

SELECT_1
UNION
SELECT_2
UNION
SELECT_3

may cause 2 sorts

The above command can be replaced with:

SELECT_1
UNION ALL
SELECT_2
UNION
SELECT_3

and will perform 1 sort

AW42 One or more predicates should be specified before the subquery.

Reason:

The command contains a subquery and other predicates are specified AFTER the subquery.

For example :

SELECT ... FROM T
WHERE C1 = 1
AND C2 IN (SELECT... FROM T2)
AND C3 = 3

Action:

It is best to specify all normal predicates BEFORE the subquery. This will reduce the number of qualifying rows that is passed to the subquery. In the above example C3 = 3 should be placed before the subquery.

AW43 MIN/MAX function is less efficient due to presence of WHERE clause

Reason:

The MIN/MAX functions can be executed using an index-only access. Moreover, the MIN function can retrieve the required value directly from the first index entry. The MAX function can retrieve the value directly from the last index entry.

The presence of a WHERE clause requires a complete index scan and possibly access to data pages when the predicate column is not in the index.

Action:

Omit the WHERE clause, if possible.

AW44 MIN/MAX function is less efficient because <column-name> if not the first column in the index

Reason:

The MIN function can retrieve the required value directly from the first index entry. The MAX function can retrieve the value directly from the last index entry. In both cases, the named column must be the first or only column in the index.

Action:

Specify MIN/MAX for the first index column, if possible.

AW45 MAX function is less efficient because index column <column-name> allows NULL values

Reason:

The MAX function can retrieve the MAX value directly from the last index entry, provided the column has been defined as **not null**. If the column allows nulls, the null index entries appear in the index **after** the highest not null entries. Hence, a fast "locate last" cannot be performed.

Action:

Redefine the column with the NOT NULL attribute, if possible.

AW46 MIN/MAX function is less efficient because <column-name> is not a column in the plan index

Reason:

The index used in the SQL command does not have the named column as the first index column. Therefore, fast index locate is not used.

Action:

Attempt to rewrite the command, so that the index containing the column specified in the MIN/MAX function, is used for access.

AW47 Full table join performed because no local join predicates specified

Reason:

The join command does not contain predicates other than the join predicate.

Action:

Unless a full table join is intended, specify other predicates in the WHERE clause, to decrease the number of table rows participating in the join.

AW48 A sort can be avoided if the ORDER BY clause specifies all columns of the plan index <index-name> in the same sequence

Reason:

The plan index used allows to avoid ORDER BY processing (additional sort). However, **all** columns of the plan index must appear in the ORDER BY clause in the **same sequence** as defined for the index. If not, a sort will be performed at the end of the query.

For example: if the plan index is on C1,C2

```
SELECT C1,C2 FROM table
WHERE C1 = ...
ORDER BY C2
```

will **not** avoid a sort

```
SELECT C1,C2 FROM table
WHERE C1 = ...
ORDER BY C1,C2
```

will avoid a sort

Action:

Specify all index columns on the ORDER BY clause (unless another sequence is desired).

AW49 <column-name> missing

Reason:

When not all index columns are key-matching (warning AW92 has been issued), the AW49 warning indicates that the index column <column-name> is not keymatching, because it is not referenced in the predicate.

For example:

Given an index C1,C2,C3,

the predicate WHERE C1 = x AND C2 = y will result in warning AW49 for column C3.

Action:

Specify the missing column if possible.

AW50 <column-name1> discarded by missing column <column-name2>

Reason:

When not all index columns are key-matching (warning AW92 has been issued), the AW50 warning indicates that the index column <column-name1> is not key-matching because the column <column-name2> that precedes it in the index definition, is missing.

For example:

Given an index C1,C2,C3

the predicate WHERE C1 = x AND C3 = y

will result in warning AW50, indicating that C3 cannot be used for key-matching because C2 is missing.

Action:

Specify the missing column if possible.

AW51 <column-name1> discarded by non-keymatching column <column-name2>

Reason:

When not all index columns are key-matching (warning AW92 has been issued), the AW51 warning indicates that the index column <column-name1> is not keymatching because the column <column-name2> that precedes it in the index definition, is not key-matching. Warning AW54 will have been issued for column-name2.

For example:

Given an index C1,C2,C3

the predicate WHERE C1 <> x AND C2 = y

will result in warning AW51, indicating that C2 (although a keymatching expression itself) cannot be used for keymatching because it follows a non-keymatching expression on C1.

Action:

If possible, make the expression on column-name2 keymatching (cfr AW54).

AW52 <column-name1> discarded by non-sargable column <column-name2>

Reason:

When not all index columns are key-matching (warning AW92 has been issued), the AW52 warning indicates that the index column <column-name1> is not keymatching because the column <column-name2> that precedes it in the index definition, is not sargable and therefore not key-matching. Warning AW55 will have been issued for column-name2.

For example:

Given an index C1,C2,C3

the predicate WHERE C1 = (:V*2) AND C2 = y

will result in warning AW52, indicating that C2 (although a keymatching expression itself) cannot be used for key-matching because it follows a non-sargable and therefore non-keymatching expression on C1.

Action:

If possible, make the expression on column-name2 sargable (cfr AW55).

AW53 <column-name1> discarded by range predicate on <column-name2>

Reason:

When not all index columns are key-matching (warning AW92 has been issued), the AW53 warning indicates that the index column <column-name1> is not keymatching, because a range predicate has been coded for column-name2.

For search conditions on composite indexes to be key-matching, all but the last column must be matched with equal predicates; the last predicate can be either an equal or a range predicate.

For example:

Given an index C1,C2,C3

the predicate WHERE C1 > x AND C2 = y AND C3 = z

will result in warning AW53, indicating that the expressions on C2 and C3 (although themselves keymatching) have been discarded as keymatching candidates, because a range expression has been coded for C1.

Action:

Avoid the range predicate, if the application logic allows it.

The performance impact of AW53 may be severe, if the discarded columns are the most selective ones.

In the above example: if C1 is non-selective, but C2 and C3 are, a large number of rows will be retrieved for C1 > x, resulting in a high number of I/O's. The predicates on C2 and C3 will be applied by RDS after receiving the rows qualifying for C1 > x.

In such cases, it may be necessary to define a new index, or to alter an existing one. In the above example, an index on C2,C3,C1 would perform much better.

AW54 <column-name> non-keymatching

When not all index columns are key-matching (warning AW92 has been issued), the AW54 warning indicates that the index column <column-name> is not key-matching.

For example:

Given an index C1,C2,C3

the predicate WHERE C1<> x AND C2 = y

will result in warning AW54, indicating that C1 is a non-keymatching expression.

Action:

If possible, make the expression on column-name key-matching (cfr AW16).

AW55 <column-name> non-sargable and non-keymatching

Reason:

When not all index columns are key-matching (warning AW92 has been issued), the AW55 warning indicates that the index column <column-name> is not sargable and therefore not key-matching.

For example:

Given an index C1,C2,C3

the predicate WHERE C1 = (:V*2) AND C2 = y

will result in warning AW55, indicating that the expression on C1 is not sargable and therefore not key-matching.

Action:

If possible, make the expression on column-name sargable. In the above example, the application should execute the *2 on the hostvariable and specify the predicate as C1 = :V.

AW56 Multiple IN operators disable keymatching

Reason:

When not all index columns are key-matching (warning AW92 has been issued), the AW56 warning indicates that some expressions, although key-matching themselves, may have been discarded, because more than one IN operator appears in the predicate.

For example:

Given an index C1,C2,C3 and the predicate

WHERE C1 IN (1,2) AND C2 IN (1,2) AND C3 = x

the expression on C3 will not be considered for key-matching, because more than one IN precedes.

Action:

If possible, avoid using more than one IN operator in a WHERE clause.

AW57 Avoid fetching columns in an EXISTS SELECT with an index-only predicate

Reason:

If the SELECT that follows the (NOT) EXISTS clause, uses indexing columns only to perform the EXISTS check, there is no reason to fetch columns in this SELECT.

If you do, DB2/VM will unnecessarily access data pages, with a performance degradation as a result.

Action:

Replace the column names with a constant or a special register such as CURRENT DATE. This ensures that DB2 accesses index pages only when executing the SELECT.

For example:

If both T1 and T2 have an index on C1, the statement:

SELECT COUNT(*) FROM T1 WHERE EXISTS (SELECT * FROM T2 WHERE T1.C1 = T2.C1)

should be coded as:

SELECT COUNT(*) FROM T1 WHERE EXISTS (SELECT 'XXX' FROM T2 WHERE T1.C1 = T2.C1)

AW80 Using scan of DBspace .....

Reason:

The DB2 explain data show that the command plan will be executed using a scan of the named DBspace. SQL/CA provides following additional data regarding the DBspace scan:

DBspace pages scanned:
the number of DBspace pages that actually wil be read during the scan, that is the number of active pages in the DBspace, as found in the SYSDBSPACES catalog.

DBspace scan productivity:
a percentage computed to represent the number of DBspace pages read that belong to the table referred to by the command plan.

Action:

While in some cases a DBspace (or relational) scan may be the most plausible access method (a small table residing in a dedicated DBspace for instance), it is problematical in most cases. It may be due to various reasons such as:

- the absence of table indexes
- the unclustered state of the table indexes
- a high estimate for the number of rows satisfying the command predicate
- syntactical expressions prohibiting index eligibility
- a high PCTPAGES value for the table, meaning that the table occupies a large percentage of pages in the DBspace, so that a DBspace scan seems productive and hence attractive to the Optimizer

In most cases, other SQL/CA warning messages will have been issued. Use them to determine the reason of the DBspace scan.

AW81 No indexes for table .....
AW81 Insert using default rules

Reason:

No indexes have been defined for the table. Therefore, a DBspace scan will be used to access the table and insert will be done in the last datapage of the table.

Action:

Create a table index, unless the table is small and residing in a dedicated DBspace.

AW82 No highly clustered first index for table .....

Reason:

The first index of the table is either unclustered or weakly clustered. An index is considered weak by SQL/CA, when its clustering ratio drops below 6000. (10000 is the highest and best clustering ratio).

Action:

Reorganize the first index or the entire table.

AW83 No indexes found created using current DB2 release

Reason:

All existing table indexes were created by a backlevel release of DB2.

Action:

For best performance, the indexes should be upgraded to the latest release level, by doing a table reorganization.

AW84 Non-selective index scan

Reason :

All rows of the table are accessed using an index without specific key values, because the first column of the index key is not specified in the predicate.

In some cases an index scan may be a masked relational scan which may turn into a true DBspace scan (when the table gets larger for instance).

Access with specific key values is obviously better for performance because less I/O will be done. See the Glossary on page 133 under *SELECTIVE INDEX SCAN*.

Action :

Specify the leading columns of the index key, wherever possible. Also check whether enough indexes are available for the columns appearing in the predicate.

AW85 Materialization of view .....

Reason:

View materialization is a technique used by DB2 since version 3, in order to remove certain restrictions regarding views. Processing a materialized view implies storing intermediate results into temporary tables using internal DBspaces. These temporary tables are processed without using indexes.

Action:

Functionality has to be weighted against performance. An attempt should be made to keep the response set small.

AW86 Estimated times predicate conditions satisfied

Reason:

The number of times the command predicate is true equals the number of table rows. This means that the entire table is processed. The message may occur in conjunction with the messages AW80 and AW84.

Action:

See the suggestions at AW80 and AW84.

AW87 Block executed N times by parent block

Reason:

A child block may be executed once, when the parent block is initiated or it may be executed each time the parent block is invoked. The latter case is signalled by this warning which computes in N, the total estimated number of times this block is executed, taking into account its immediate and remote parent block invocation counts.

Action:

The actual execution time of the block is the product of all preceding parent block execution iterations. As the number of iterations can become very high, care should be taken. Alternative predicate statements may produce better results.

For example:

in the command:

*SELECT * FROM T1 WHERE C1 IN (SELECT C1 FROM T2)*

the SELECT FROM T2 is executed for each row of T1. For a large T1, such SELECT commands may take hours!

Alternatives would be:

- a JOIN of T1 and T2 (the preferred solution)
- SELECT * FROM T1 X WHERE EXISTS (SELECT C1 FROM T2 WHERE C1=X.C1)
- a correlated query such as SELECT * FROM T1 X WHERE C1 IN (SELECT C1 FROM T2 WHERE C1=X.C1)

AW88 Indexing columns updated by command

Reason:

The column listed is member of an index definition. Updating a column of the primary table or plan index, using an UPDATE or a SELECT FOR UPDATE command will disqualify indexed access.

Action:

If the named indexing column is not part of the primary table or plan index, a severity 1 warning will be issued and the warning may be ignored. In the other case, the severity of the warning will be 3 and the update should be replaced with a DELETE and INSERT sequence. Primary indexing columns should not appear in a SELECT FOR UPDATE clause, even if the column is not updated actually. In CSP terms, define indexing columns with the read-only attribute in the SQL record. Alternatively, remove the indexing column from the FOR UPDATE clause in the CSP process.

AW89 Merge scan Join

Reason:

A merge scan join is performed as part of a join plan. Merge scan joins are usually worse for performance than nested loop joins, since a merge scan join often implies the use of work dataspace and additional sorting.

Action:

Try to obtain a nested loop join. For example: define an index that allows the Optimizer to access the inner join table in the required order. Ensure that the available indexes are clustered.

AW90 Outer table larger than inner table.

Reason:

A nested loop join is performed as part of a join plan. Best performance is generally achieved when the outer table (the first or "composite table") is smaller than the inner table (the second or "new table"). The tablesizes considered take into account the local (non-join) predicates, since these are applied by DB2 before initiating the join.

Action:

Specify additional local (non-join) predicates for the outer table in the command's WHERE clause.

AW91 Data pages accessed for predicate and data

Reason:

The predicate cannot be resolved using index page access and all data is retrieved from data pages.

Action:

If the selected data are not available from an index, retrieving data from the data pages is normal. You should investigate why no index is used for resolving the predicate. If the index contains VARCHAR columns, the data pages must be accessed.

AW92 Key-matching columns: N out of M

Reason:

The predicate does not specify all columns of the plan index, i.e. N < M.

Action:

Try to specify the missing index columns. You should find their names in warning AW22 or AW34.

AW93 No catalog statistics available for the table

Reason:

No UPDATE STATISTICS command has been issued for the table.

Consequently, no information is available the DB2/VM Optimizer for determining the best data access path.

Action:

Issue the UPDATE STATISTICS command for the table or DBspace.

AW94 Catalog statistics available for index columns only

Reason:

An UPDATE STATISTICS command has been issued for the table without the ALL option.

Consequently, the DB2/VM Optimizer has statistics for the first column of the indexes only.

Action:

Issuing an UPDATE ALL STATISTICS for the table or DBspace may result in a better access path being adopted by the DB2/VM Optimizer.

Consider an UPDATE ALL STATISTICS for commands that perform poorly or that do not execute using the expected access path.

15 SQL/CA object notes

All the object warnings are followed by a list of objects, for which the warning applies.

ON01 DBspaces with freespace > 0:

The named DBspaces have the FREEPCT column > 0. Freespace is usually set when initially loading the DBspace and set then set to 0. This allows INSERT commands to use the DBspace freespace.

ON02 DBspaces with less than 10 active pages and with lockmode not "row":

Very small DBspaces should usually have the **row** lockmode. Other modes will lock too much data for each request and will be detrimental for concurrency.

ON03 DBspaces in storage pool 1:

In operational databases, DBspaces should be located in other storage pools. Storage pool 1 should be reserved for the DB2/VM catalog tables and the internal system DBspaces.

ON04 Tables with more than 10% overflow rows:

Tables with more than 10% overflow rows should be considered for reorganization.

ON05 Tables with less than 10 pages and non-unique indexes:

While indexes can provide faster access, they also impose a considerable overhead, especially during table update or insert activity. The designated indexes are not required to implement key-uniqueness or to implement referential constraints. For small tables, residing in a dedicated DBspace, a DBspace scan is probably the best access strategy.

ON06 Indexes with VARCHAR or VARGRAPHIC columns:

Indexes with VARCHAR or VARGRAPHIC columns prevent index-only access. Even when all data could be retrieved using the index, data pages must still be accessed for VARCHAR and VARGRAPHIC columns.

ON07 Indexes where the first column is not the most selective one:

In composite indexes, it is best to define the most selective column as the first index column. For the designated indexes, non-first columns actually have a better selectivity than the first one.

Note The above message will be issued only when an UPDATE **ALL** STATISTICS has been issued for the table, which ensures that the selectivity of all table columns is known. In the other case, the selectivity of the first index column only is known.

16 Predicate evaluation tables

16.1 Datatype evaluation table

The following table illustrates the rules governing datatype compatibility. Datatype combinations marked **Y** are compatible. Those marked **N** are not compatible.

For example: comparing an INTEGER column with a SMALLINT host variable adheres to the compatibility rules. Comparing a SMALLINT column with an INTEGER host variable violates these rules.[18]

Column Type	Hostvar SMALLINT	Hostvar INTEGER	Hostvar DECIMAL	Hostvar REAL	Hostvar FLOAT
SMALLINT	Y	N	N	N	N
INTEGER	Y	Y	N	N	N
DECIMAL	Y	Y	Y	N	N
REAL	Y	Y	Y	Y	N
FLOAT	Y	Y	Y	Y	Y

16.2 Decimal precision evaluation table

The following table defines the rules which the datatype, precision and scale of decimal columns and hostvars must adhere to, in order to be sargable.

Column Type	SMALLINT Hostvar	INTEGER Hostvar	DECIMAL(k,l) Hostvar
DECIMAL(m,n)	ensure: m>=4 and n=0	ensure: m>=10 and n=0	ensure: m>=k and n=l

[18]The rationale behind this: when comparing an INTEGER column and a SMALLINT variable; the variable contents can never exceed the magnitude of the column and a simple comparison is sufficient. When comparing a SMALLINT column with an INTEGER variable, the variable contents can exceed the magnitude of the column and a more sophisticated comparison is needed. The latter comparison cannot be performed by the DBSS (which basically performs byte-wise comparisons). It has to be carried out by the RDS. Therefore, an incompatible expression is never sargable.

16.3 Predicate operator evaluation table

OPERATOR	*KEYMATCH*	*SARGABLE*	*DEFAULT FILTER*
=	yes	yes	0.040
> < >= <= value	yes	yes	0.333
BETWEEN	yes	yes	0.100
IS NULL	yes	yes	0.040
IN(valuelist)	yes	yes	0.040*(size-of-list)
LIKE char	yes	yes	0.100
= **Q** (query)[19]	yes	yes	0.040
> < >= <= **Q** (query)	yes	yes	0.333
<> **Q** (query)	no	yes	0.960
<> value	no	yes	0.960
IS NOT NULL	no	yes	0.960
= (query)	no	no	0.040
<> (query)	no	no	0.960
> < >= <= (query)	no	no	0.333
= expression	no	no	0.040
<> expression	no	no	0.960
> < >= <= expression	no	no	0.333
[NOT] IN(query)	no	no	1.000
LIKE pattern	no	no	0.100
LIKE host variable	no	no	0.100
NOT BETWEEN	no	no	0.900
NOT IN(valuelist)	no	no	1-(0.040*(size-of-list))
NOT LIKE	no	no	0.900

[19] **Q** is one of the quantifiers **ANY**, **ALL** or **SOME**

16.4 Default filter factor table

Distinct table column values	*Range filter factor*	*LIKE/BETWEEN filter factor*
>= 100 000 000	0.0001	0.00003
>=10 000 000	0.0003	0.0001
>=1 000 000	0.001	0.0003
>= 100 000	0.003	0.001
>= 10 000	0.01	0.003
>= 1 000	0.033	0.01
>= 100	0.1	0.03
< 100	0.333	0.1
= -1 (no COLCOUNT available)	0.333	0.1

17 EXPLAIN tables usage by SQL/CA

SQL/CA must have **exclusive control** over the EXPLAIN tables used during analysis. Which EXPLAIN tables are used depends on the DB2/VM userid active during EXPLAIN. This userid becomes the creator of the explain tables: it can be specified in different ways on the SQL/CA menus.

Due to the method used by SQL/CA for accessing the EXPLAIN tables, it is **not possible** for the user to maintain its own explain output in those tables. These data will be cleared by SQL/CA at the next analysis request.

18 EXPLAIN performance

Generally, the EXPLAIN tables have very few rows and are best processed by a DBspace scan. For optimal performance, the user's 4 EXPLAIN tables should be created in a dedicated DBspace and no indexes should be defined on them.

19 Alias usernames

19.1 Functional Description

In some cases, it may be desirable that users perform program analysis using a DB2/VM authorization ID, other than their own. For example, several developers within a team may use the same DB2/VM userid when preprocessing their programs. This facilitates authorization management and allows developers to work on each other's programs and use each other's tables. In such cases, analysis must also be done under the common DB2/VM username. The alias facility allows the system administrator to define each of these users in the **SQLCA CONNECT** file and specify the DB2/VM username and password to be used by SQL/CA when the user forwards an analysis request. Since the CONNECT file contains sensitive data (the user password), it is kept in encrypted format by SQL/CA. The CONNECT file should be on a minidisk or in a directory that is accessible to the users during analysis. It may be placed on the disk containing the SQL/CA software material for instance.

19.2 Defining an alias

To define (or remove) an alias, access the minidisk or directory containing the SQLCA CONNECT file in filemode A. Issue **SQLCALIA** on the CMS prompt. The program saves the current SQLCA CONNECT as SQLCA CONNECTS. Then, the SQLCA CONNECT file is decrypted into a file named **CONNECT INPUT**. The CONNECT INPUT file is then XEDITed. Use XEDIT commands to insert (or remove) records that define the alias names. Issuing XEDIT **file** will encrypt and copy the CONNECT INPUT file to SQLCA CONNECT. The alias definitions take effect from this moment on.

Each alias description line has the following syntax:

CONNECT user_name TO database_name AS alias_name alias_password

Meaning: when **user_name** connects to **database_name**, connect should be done automatically by SQL/CA, using **alias_name** and **alias_password**. If **user_name** is specified as *****, all users connect to the database under the alias name.

SQL/CA scans the SQLCA CONNECT file sequentially and stops its search for alias replacements when a line is found where both **user_name** and **database_name** are matching.

For example: to allow a number of DB2/VM developers named SQLDEV1 thru SQLDEVn to connect under the common SQLDEV username, following lines should be coded in the CONNECT file:

- **CONNECT SQLDEV1 TO DBTEST AS SQLDEV SQLDEVPW**
- **other CONNECTs**
- **CONNECT SQLDEVn TO DBTEST AS SQLDEV SQLDEVPW**

20 Migrating to SQL/DS Version 3 Release 4

SQL/DS Version 3 Release 4 provides an enhanced EXPLAIN table structure. SQL/Command Analysis can process the EXPLAIN tables of SQL/DS versions before Version 3 Release 4 and those of Version 3 Release 4. During analysis, the DB2/VM release level is determined and the appropriate logic modules of SQL/CA are loaded accordingly. During installation, all support modules are installed. When migrating, nothing has to be done, except re-creating the user's explain tables, so that they conform to the new structure. To create the new tables, you can use the corresponding option of the SQL/CA main menu, which is release sensitive, that is, the explain tables will be created depending on the DB2/VM release actually installed.

21 Dynamic SQL/CA packages

After issuing the EXPLAIN command, SQL/CA has to process the EXPLAIN tables of the user. This cannot be achieved in static mode, since the creator of the EXPLAIN tables is unknown at preprocessing time. Instead, SQL/CA creates an extended dynamic package, the first time a user invokes Command Analysis. This package is named **userid.SQLCAXUX** or **userid.SQLCAXU2** (for SQL/DS Version 3.4 and later). These packages are managed in the same manner as static packages. If a package is dropped, it is automatically recreated by SQL/CA at the next analysis.

22 SQL/CA messages

Note

RC ... in the following message texts refers to the hexadecimal representation of the error code presented by the CMS file system during various stages of SQL/CA processing. A detailled description of these codes can be found in the IBM manual "CMS Macros and Function Reference" (paragraphs on the FSREAD and FSWRITE macros). The most likely errorcode after FSWRITE is 0D (A-disk full).

SQLCA001: RC ... AFTER FSREAD

Error when reading the application source file.

SQLCA002: RC ... AFTER FSWRITE

Error when writing the explain workfile.

SQLCA003: Application nnnn does not contain static SQL statements

The analyzed source does not contain static (prepped) analysable SQL statements at all.

SQLCA004: UPDATING STATISTICS FOR TABLE .....

Issued when performing the UPDATE STATISTICS SQL command for a table referenced in an application SQL command.

SQLCA005: RC ... AFTER FSWRITE

Error when writing the explain report file.

SQLCA006: RC ... AFTER HVWRITE

Error when writing the "host variable" workfile.

SQLCA007: SQLCODE ... WHEN CONNECTING DATABASE .....

Issued when CONNECT to the SQL/CA archive database fails.

SQLCA008: RETURNCODE x FROM SQLCPEX

If x = 8 or 12, CMS storage request fails when processing internal lists. Enlarge your virtual storage size.

SQLCA009: RC ... AFTER FSREAD

Error when reading the CSP application source.

SQLCA010: RC ... AFTER FSWRITE

Error when writing the explain workfile for CSP applications.

SQLCA011: ALL HOSTVAR REFERENCES REPLACED WITH CONSTANTS.

Issued when the SQL EXPLAIN command results in an SQLCODE indicating incompatible parameter marker usage. All hostvar references are then replaced by SQL/CA with a constant specification of the corresponding format and EXPLAIN is retried with the modified application command.

Hostvars and parameter markers are compatible, except for a number of cases, described in the IBM manual "DB2 Application Reference" (PREPARE command).

SQLCA012: TABLE (creator) tablename DOES NOT EXIST

The DB2 object name extracted from the SQL command is neither a table, a view or a synonym. Analysis for the application is aborted.

SQLCA013: BASE TABLE (creator) tablename DOES NOT EXIST

The application command references a view. When processing the view definition, SQL/CA is unable to locate the underlying table. Analysis for the application is aborted.

SQLCA014: UNABLE TO LOCATE INDEX indexname

The indexname occurring in the explain PLAN table is not found in the SQL/CA index list. This is probably an SQL/CA system error.

SQLCA015: RC ... AFTER FSREAD

Read error on SQLDBSU unload file during DB2 analysis.

SQLCA016: RC ... AFTER FSWRITE

Write error when creating the analysis workfile during DB2 analysis.

SQLCA017: SQLCODE ... ON EXPLAIN COMMAND

An SQLCODE has been returned when invoking SQL EXPLAIN for the application command displayed immediately before the above message. The application command is probably in error.

Press ENTER to continue with the analysis of the remaining application commands. The application command in error will not appear in the analysis report.

The above message may be signalled for SQL commands appearing in a CSP process that enables the "execution time statement build" option, if the command text of the process is such that execution time insertion of a CSP variable into the command is required to yield valid SQL syntax, for example when the WHERE clause is dynamically built by the application. If you wish to analyze such command, modify it in the CMS file <application CSPAPPL A> which is built by SQL/CA during analysis of CSP applications. Then execute SQLCA against this CMS file using the option "Analyze non_CSP application".

SQLCA099: Access denied.

The executing VM userid has not received an explicit authorization on SQL/CA.

Request your system administrator to add you to the SQL/CA authorization list.

23 SQL/CA sample analysis report

The following PL/1 sample program[20] contains several violations of DB2 coding rules.

```
EXEC SQL      SELECT * FROM SQLDBA.INVENTORY
              FOR UPDATE OF PARTNO,DESCRIPTION,QONHAND;
EXEC SQL      SELECT * FROM SQLDBA.INVENTORY
              WHERE PARTNO/1000 =:PARTGROUP;
EXEC SQL      SELECT SQLDBA.INVENTORY.PARTNO, DESCRIPTION, PRICE
              FROM SQLDBA.INVENTORY, SQLDBA.QUOTATIONS
              WHERE
              SQLDBA.INVENTORY.PARTNO=SQLDBA.QUOTATIONS.PARTNO
              AND SUPPNO=51;
EXEC SQL      SELECT * FROM SQLDBA.INVENTORY
              WHERE QONHAND >:QONHAND;
EXEC SQL      SELECT SUPPNO FROM SQLDBA.QUOTATIONS Q
              GROUP BY SUPPNO HAVING SUM(QONORDER) >=
              (SELECT AVG(QONORDER) FROM SQLDBA.QUOTATIONS
              WHERE SUPPNO=Q.SUPPNO);

EXEC SQL BEGIN DECLARE SECTION;
DECLARE PARTGROUP    BIN FIXED(15);
DECLARE QONHAND      BIN FIXED(15);
DECLARE PARTNO             BIN FIXED(31);
EXEC SQL END DECLARE SECTION;
```

When this program is submitted for analysis, the following analysis report will be produced.

[20]This program can be found as *SQLCADMO PLIOPT* on the SQL/CA product disk.

```
 ------------------------------------------------------------------------------
Cust : CATEST
File : SQLCADMO PLIOPT   A1    Line nr 2                          User=SQLAF
Date : 94/04/19  16:10:55      ArchQno = 501                     Dbase=CATEST
 ------------------------------------------------------------------------------

 COMMAND EXECUTION STRUCTURE BY BLOCK AND PARENT

 BLK Par  Command Text
 ---------------------

   1   0  SELECT * FROM SQLDBA.INVENTORY FOR UPDATE OF PARTNO,DESCRIPTION,
           QONHAND

           COMMAND COST SUMMARY

           Total Command Cost : 18 (ISQL like Cost : 1)

           COMMAND EXECUTION STRUCTURE

           Estimated number of rows SELECTed : 22 out of 22
           Estimated global command filter factor : 1
 >>> AW86  Estimated times predicate conditions satisfied : 22

           COMMAND EXECUTION DETAIL

           Plan : 1

 >>> AW80     Access : Scan of DBspace SAMPLE
                      DBspace pages scanned : 8
                      DBspace scan productivity : 13%
              Score  : 0
              Sort   : New table not sorted.


 >>> AW88  Indexing columns updated by command :

           PARTNO             : Column  1 in Index SQLDBA.INV1
                                          on Table SQLDBA.INVENTORY
           PREDICATE ANALYSIS WARNINGS

 >>>       AW23 No index columns referenced in the command predicate for table
                (SQLDBA) INVENTORY.
```

```
--------------------------------------------------------------------------------
Cust : CATEST
File : SQLCADMO PLIOPT   A1    Line nr 5                          User=SQLAF
Date : 94/04/19  16:10:56      ArchQno = 502                     Dbase=CATEST
--------------------------------------------------------------------------------

COMMAND EXECUTION STRUCTURE BY BLOCK AND PARENT

BLK Par  Command Text
---------------------

  1   0  SELECT * FROM SQLDBA.INVENTORY WHERE PARTNO/1000 =:PARTGROUP

          COMMAND COST SUMMARY

          Total Command Cost : 11 (ISQL like Cost : 1)

          COMMAND EXECUTION STRUCTURE

          Estimated number of rows SELECTed : 22 out of 22
          Estimated global command filter factor : 1
>>> AW86  Estimated times predicate conditions satisfied : 22

          COMMAND EXECUTION DETAIL

          Plan : 1

>>  AW84     Access : Non-selective index scan
                      using first, highly clustered, unique index
                      INV1 ON +PARTNO
              Score  : 3
              Sort   : New table not sorted.

          PREDICATE ANALYSIS WARNINGS

          PARTNO/1000=:PARTGROUP

>>>       AW01 Expression on index column PARTNO is suboptimal.
```

```
--------------------------------------------------------------------------------
Cust : CATEST
File : SQLCADMO PLIOPT   A1    Line nr 8                          User=SQLAF
Date : 94/04/19  16:10:58      ArchQno = 503                     Dbase=CATEST
--------------------------------------------------------------------------------

COMMAND EXECUTION STRUCTURE BY BLOCK AND PARENT

BLK Par  Command Text
---------------------

  1   0  SELECT SQLDBA.INVENTORY.PARTNO,DESCRIPTION,PRICE FROM SQLDBA.INVENTORY,
          SQLDBA.QUOTATIONS WHERE
          SQLDBA.INVENTORY.PARTNO=SQLDBA.QUOTATIONS.PARTNO AND SUPPNO=51

          COMMAND COST SUMMARY

          Total Command Cost : 22 (ISQL like Cost : 1)

          COMMAND PLAN SUMMARY

          Qry Plan

          1   1     Selective index scan of table QUOTATIONS
          1   2     Nested loop join using
                    selective index scan of table INVENTORY

          COMMAND EXECUTION STRUCTURE

          Estimated number of rows SELECTed : 3 out of 75
          Estimated global command filter factor : 0.040000
          Estimated times predicate conditions satisfied : 1

          COMMAND EXECUTION DETAIL

          Plan : 1

             Access : Selective index scan
                      using first, highly clustered, unique index
                      QUO1 ON +SUPPNO  +PARTNO
             Score  : 4
             Sort   : New table not sorted.

             COLUMN REFERENCE DETAILS

             Table : SQLDBA.QUOTATIONS
                Column : PARTNO
                    Filtering factor : 0.045
                    Estimated table rows filtered : 2.385 out of 53
                    Appears in WHERE clause such that index can be used.
                    Used as a JOIN column.
                Column : SUPPNO
                    Filtering factor : 0.074
                    Estimated table rows filtered : 3.922 out of 53
                    Appears in WHERE clause such that index can be used.
```

```
Plan : 2

   Method : Nested loop JOIN.
   Access : Selective index scan
            using first, highly clustered, unique index
            INV1 ON +PARTNO
   Score  : 4
   Sort   : New table not sorted.
            Composite not sorted.
            Number of rows in inner table : 22
            Number of rows in outer table : 2.385

   COLUMN REFERENCE DETAILS

   Table : SQLDBA.INVENTORY
      Column : PARTNO
           Filtering factor : 0.045
           Estimated table rows filtered : 0.990 out of 22
           Appears in WHERE clause such that index can be used.
           Used as a JOIN column.
```

```
--------------------------------------------------------------------------------
Cust : CATEST
File : SQLCADMO PLIOPT   A1    Line nr 13                        User=SQLAF
Date : 94/04/19  16:10:59      ArchQno = 504                    Dbase=CATEST
--------------------------------------------------------------------------------

COMMAND EXECUTION STRUCTURE BY BLOCK AND PARENT

BLK Par  Command Text
---------------------

  1   0  SELECT * FROM SQLDBA.INVENTORY WHERE QONHAND >:QONHAND

          COMMAND COST SUMMARY

          Total Command Cost : 6 (ISQL like Cost : 1)

          COMMAND EXECUTION STRUCTURE

          Estimated number of rows SELECTed : 7 out of 22
          Estimated global command filter factor : 0.318181
          Estimated times predicate conditions satisfied : 7.333

          COMMAND EXECUTION DETAIL

          Plan : 1

>>  AW84     Access : Non-selective index scan
                      using first, highly clustered, unique index
                      INV1 ON +PARTNO
              Score  : 3
              Sort   : New table not sorted.

              COLUMN REFERENCE DETAILS

              Table : SQLDBA.INVENTORY
                 Column : QONHAND
                     Filtering factor : 0.333
                     Estimated table rows filtered : 7.326 out of 22
                     Appears in WHERE clause such that index can be used.

          PREDICATE ANALYSIS WARNINGS

          QONHAND>:QONHAND

>         AW19 Range predicates used with hostvars may disqualify the index.
>         AW03 Datatype INTEGER of column QONHAND not compatible with datatype
               DECIMAL of hostvar QONHAND.
>              Expression is not sargable.
>         AW18 Default filter factor used for range predicate is 0.333.
>         AW36 BETWEEN is a more selective operator than >.
>>>       AW23 No index columns referenced in the command predicate for table
               (SQLDBA) INVENTORY.
```

```
--------------------------------------------------------------------------------
Cust : CATEST
File : SQLCADMO PLIOPT   A1    Line nr 16                          User=SQLAF
Date : 94/04/19  16:11:01      ArchQno = 505                      Dbase=CATEST
--------------------------------------------------------------------------------

COMMAND EXECUTION STRUCTURE BY BLOCK AND PARENT

BLK Par  Command Text
---------------------

  1   0  SELECT SUPPNO FROM SQLDBA.QUOTATIONS Q GROUP BY SUPPNO HAVING
          SUM(QONORDER) >= (
  2   1    SELECT AVG(QONORDER) FROM SQLDBA.QUOTATIONS WHERE SUPPNO=Q.SUPPNO)

          COMMAND COST SUMMARY

          Blk  Single_Cost  Exec_Times  Multi_Cost    SQL_Cost

            1       56.000       1.000      56.000    1169.000
            2       21.000      53.000    1113.000      21.000

          Total Command Cost : 1169 (ISQL like Cost : 2)
          Highest Subquery Cost : 1113  (Block 2)

          COMMAND PLAN SUMMARY

          Qry Plan
          1   1     Non-selective index scan of table QUOTATIONS
          2   1     Non-selective index scan of table QUOTATIONS

BLK   1 : SELECT SUPPNO FROM SQLDBA.QUOTATIONS Q GROUP BY SUPPNO HAVING
            SUM(QONORDER) >= (

          COMMAND EXECUTION STRUCTURE

          Estimated number of rows SELECTed : 53 out of 53
          Estimated global command filter factor : 1
>>> AW86  Estimated times predicate conditions satisfied : 53

          COMMAND EXECUTION DETAIL

          Plan : 1

>>  AW84     Access : Non-selective index scan
                      using first, highly clustered, unique index
                      QUO1 ON +SUPPNO  +PARTNO
              Score  : 3
              Sort   : New table not sorted.

              COLUMN REFERENCE DETAILS

              Table : SQLDBA.QUOTATIONS
                 Column : SUPPNO
                      Appears in GROUP BY clause on position 1.
```

```
           PREDICATE ANALYSIS WARNINGS

           SUM(QONORDER)>=(

>          AW16 Predicate operator >= is not a key-matching candidate.
>          AW17 Predicate operator >= is not sargable.
>>>        AW23 No index columns referenced in the command predicate for table
                (SQLDBA) QUOTATIONS.


BLK   2 : SELECT AVG(QONORDER) FROM SQLDBA.QUOTATIONS WHERE SUPPNO=Q.SUPPNO)

           COMMAND EXECUTION STRUCTURE

           Estimated number of rows SELECTed : 3 out of 53
           Estimated global command filter factor : 0.056603
>>> AW86   Estimated times predicate conditions satisfied : 53
           Estimated execution iteration by ancestor blocks : 53
>>  AW87   Block executed 53 times by parent block 1

           COMMAND EXECUTION DETAIL

           Plan : 1

>>  AW84      Access : Non-selective index scan
                       using first, highly clustered, unique index
                       QUO1 ON +SUPPNO  +PARTNO
              Score  : 3
              Sort   : New table not sorted.

              COLUMN REFERENCE DETAILS

              Table : SQLDBA.QUOTATIONS
                 Column : SUPPNO
                      Filtering factor : 0.074
                      Estimated table rows filtered : 3.922 out of 53

           PREDICATE ANALYSIS WARNINGS

>          AW39 JOIN is usually more efficient than a subquery.
>          AW34 No predicate specifications for trailing column(s) PARTNO of
                plan index QUO1.
```

TABLE LIST

Creator	Tablename	DBspace	RowL	Rows/Pg	NrPag	%Pag	Depend
SQLDBA	INVENTORY	SAMPLE	21	194	1	13	0
SQLDBA	QUOTATIONS	SAMPLE	24	170	1	13	0

TABLE COLUMN SELECTIVITY LIST

Table Name	Column Name	Distinct Values	Indexing Column
SQLDBA.INVENTORY	PARTNO	100%	Yes
SQLDBA.QUOTATIONS	PRICE	90%	Yes
SQLDBA.QUOTATIONS	PARTNO	41%	Yes
SQLDBA.QUOTATIONS	DELIVERY_TIME	37%	No
SQLDBA.QUOTATIONS	QONORDER	20%	No
SQLDBA.QUOTATIONS	SUPPNO	16%	Yes

TABLE INDEX LIST

Table Name	Index Name	ClustR	1st	Uni	Index Columns
INVENTORY	INV1	100	Yes	Yes	+PARTNO
QUOTATIONS	QUOPRICE	100	No	No	+PRICE
QUOTATIONS	QUO1	100	Yes	Yes	+SUPPNO +PARTNO

OBJECT RELATED NOTES

ON02 DBspaces with less than 10 active pages and lockmode not = "row" :
PUBLIC.SAMPLE

ON03 DBspaces in storage pool 1 :
PUBLIC.SAMPLE

ON05 Tables with less than 10 pages and non-unique indexes :
SQLDBA.QUOTATIONS

ANALYSIS SUMMARY

```
     Highest command cost : 1169
            DBspace scans : 1
 Non-selective index scans : 4
      Average access score : 2.857
       Severity 3 warnings : 10
       Severity 2 warnings : 5
       Severity 1 warnings : 8
```

Index