SQL COMMAND ANALYSIS

User's Guide

© Copyright Software Product Research 2000

All other product names, mentioned in this manual, are trademarks owned by International Business Machines Corporation, Armonk, NY.

1 Functional description

SQL Command Analysis **('SQL/CA')** is a software tool for DB2/VSE (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 DB2/VSE environment and is capable of analysing:

- Assembler, Cobol, Fortran or PL/1 source programs, residing in a VSE, an ICCF or a VOLLIE library
- ISQL routines and QMF procedures
- Files containing SQL statements in DB2 Database Services format
- Designated DB2 packages in the currently connected database

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

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

1.1 Statement extraction

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

1.2 SQL EXPLAIN

The SQL **EXPLAIN** statement is issued for all the extracted SQL statements 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 in the SQL/CA **Report Library** with a 2-part reportname (creator | packagename for packages filename | filetype for sources in a library). The analysis library is processed online, using the SQL/CA Librarian which runs as a CICS transaction.

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

1.2.1 Statement text

Prints the SQL statement in a manner that reflects the logical execution tree structure. If the statement contains subqueries, they are formatted by statement 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.

1.2.2 Statement Cost

Reports the cost of statement execution, as estimated by DB2. If the statement 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.

1.2.3 Statement Structure

Reports:

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

  If the statement 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.

1.2.4 Statement Plans

Reports:

- whether the statement 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 statement implies joining
- the number of rows in the inner and outer join table
- whether sort operations will be performed

If the statement 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.

1.2.5 Statement 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 statement

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.

1.2.6 Explain Warnings

In addition, an SQL statement 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 view materialization occurs
- 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 statement execution.

Note

If the statement contains **view references**, SQL EXPLAIN operates using the underlying real tables. Therefore, SQL/CA expands such statements by inserting the view definition(s) until the resulting statement contains real table references only. The original and the expanded statements are printed and the expanded statement is used for SQL/CA text analysis.[1]

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

1.3 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" manual and detects statement specifications that lead to suboptimal performance. Text analysis will specifically warn the following performance exposures:

- Indexing columns are being updated by the statement.

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

- Table indexes do exist, but the statement 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 85. 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 85.

- 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 statement 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 86.

- The statement 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 statement operators is not sargable. For a list of them, see the table on page 86.

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

- The entire statement 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 87.

- 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 statement omits necessary search conditions. If, in addition to the join predicate, the statement 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 statement predicate. The Glossary can be found on page 43 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.

1.4 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 statement 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[2]
- 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

[2] 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).

1.5 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

1.6 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.

1.7 Operation

1.7.1 User interface

SQL/CA analysis requests are usually issued from the CICS transaction **$CAM**. However, the analysis request can also be issued from a batch partition (e.g. a compile procedure), as described on page 25 Analysis itself is performed in a batch partition by the SQL/CA Analysis Server. The analysis reports must be examined online, using the$CAM transaction. After submitting an analysis request, the user has the option of waiting interactively on the analysis report.

1.7.2 SQL/CA help

Online help is provided at all levels of the SQL/CA user interface. The product 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 can be displayed by function key while the analysis report is being examined.

1.7.3 Explain tables

The SQL/CA explain DBspace and tables are created during product installation under the ownership of “SQLCA” which is the userid under which the SQL/CA server connects to DB2. Before starting an analysis, the server deletes all rows in the explain tables. No indexes are provided on the explain tables.

1.7.4 DB2 Connections

Analysis is always executed under the DB2 userid of the SQL/CA server “SQLCA”, which should have DBA authority. When submitting an analyis request, the user must specify the name of the DB2/VSE database where analysis should be performed. If the statements to be analyzed contain unqualified table names, a default creator name can be supplied in the user interface. SQL/CA will automatically use this default creator with unqualified table names.

1.8 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. This allows to compare the execution cost estimated by DB2, with the effective execution cost. To achieve this, the SQL/CA server should have access (as DBA) to the SQL/MF tables.

1.9 DB2 data modelling facility

It is desirable that 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/VSE 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 the utility program **SQLCADMF** that can be used to quickly copy the catalog statistics from one system to another.

2 SQL/CA Installation

2.1 Software Prerequisites

- VSE/ESA Version 2 Release 2 and later
- DB2/VSE Version 3.5 and later

2.2 Pre-installation tasks

2.2.1 Prepare a VSE library

If you did install another SPR product previously, SQL/CA should be stored in the same library as the other SPR product and you can skip the remainder of this paragraph.

Otherwise, define the VSE library where SQL/CA will be catalogued, by submitting the following jobstream:

```
// EXEC LIBR
 DEFINE SUBLIB=xxxxx.xxxx
/*
```

Notes

- The SQL/CA material requires about 5000 library blocks.
- If you have other SPR products installed, install SQL/CA in that library. All SPR products must reside in the same library and sublibrary.

2.2.2 Update LIBDEF and Standard Labels

- Add the SQL/CA library to the PHASE and OBJ SEARCH LIBDEF in your LIBDEF.PROC.
- Submit the updated LIBDEFs to the system before continuing installation. The installation procedures expect that the SQL/CA library is in the current chain.
- Ensure that the standard labels have a DLBL for the DB2 files SQLBIND, SQLGLOB and BINDWKF, as suggested by the DB2 installation guide. Submit the updated label definitions before continuing installation.

2.2.3 Prepare the SQL/CA Report Library

The SQL/CA analysis reports are stored in a VSAM cluster. These reports are managed by the SQL/CA users from the online SQL/CA application. Analysis reports are never deleted by SQL/CA. This is a user responsability. An installation may decide to keep the analysis reports in the library, as part of the application's documentation.

The space needed for the VSAM cluster depends on the number of active reports, the number of SQL statements in the applications and the number of SQL/CA warnings issued. SQL/CA compresses the text of the report by dropping leading, trailing and interspersed blanks. The SQL/CA library blocks are 16K long and in the average, a block will hold 3 to 4 analyzed SQL statements.

2.3 Installing SQL/CA

The SQL/CA software is delivered as a PC file in ZIP format.

Place the ZIP file in a dedicated directory (e.g. SQLCA) and unzip the file. Following files should now be present in the SQLCA directory:

INSTALL.BAT	installation procedure for Windows
SEND2RDR.BAT	installation procedure for Windows
SQLCA.PCF	SQL/CA software
SQLCASCF.PCF	software key
SQLCA.OPTIONS	upload the OPTIONS file
SQLCAI0.VSEJOB	setparm SPRLIB
SQLCAI1.VSEJOB	linkedit SQL/CA
SQLCAI2.VSEJOB	define the SQL/CA report library
SQLCAI3.VSEJOB	define the SQL/CA components to CICS
SQLCAI4.VSEJOB	install SQL/CA in a database
SQLCASRV.VSEJOB	job to start the SQL/CA server
SQLCA.BKS	bookshelf for IBM Library Reader
SQLCA.BOO	SQL/CA User's Guide in IBM Library Reader format

2.3.1 Preliminary Note

The INSTALL.BAT and SEND2RDR.BAT installation procedures use the "SEND.EXE" to upload the PC files to the POWER reader queue. Ensure that your 3270 emulator supports SEND (some emulators don't) and that the EXE is on your active path.

If you cannot use the SEND.EXE, use the upload facilities of your emulator to perform the equivalent of the SEND commands contained in the BAT files, that is:

for the INSTALL.BAT:

```
SEND SQLCASCF.PCF (FILE=RDR BINARY LRECL=80 NOUC
SEND SQLCA.PCF (FILE=RDR BINARY LRECL=80 NOUC
```

for the SEND2RDR.BAT:

```
SEND <filename> (FILE=RDR
```

2.3.2 Issue SETPARM for SPRLIB

Skip this step when running a VSE/ESA version lower than 2.4. The job submits a SETPARM SYSTEM statement that is not accepted in older VSE versions.

- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- Edit the file SQLCAI0.VSEJOB and insert the name of the target library on the SETPARM SPRLIB statement. The SETPARM statement remains active until the next VSE IPL. If you installed another SPR product previously into the SPRLIB, insert the name of that library on the SETPARM statement.
- Drag and drop the file SQLCAI0.VSEJOB on the SEND2RDR.BAT. This will send and start the job in class 0 and DISP D. The output listing is in DISP H.
- Check the output listing for errors.

2.3.3 Upload the SQL/CA software

- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- Execute the INSTALL.BAT by clicking. This will upload 2 jobs to the POWER/VSE reader queue with DISP D and class 0. The jobs catalog all SQL/CA components into the VSE library chosen as SQL/CA residence.
- Both jobs contain a // PAUSE statement. This allows to enter a // SETPARM SPRLIB='...'. If running VSE/ESA 2.4 or higher, ignore the PAUSE statement, as the SETPARM has been submitted by the SQLCAI0.VSEJOB, described above. If running a version older than 2.4, issue the SETPARM statement.
- After job completion, check the DISP=H listing for any errors.

2.3.4 Link SQL/CA

- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- In the file SQLCAI1.VSEJOB, assign the SETPARM symbol SPRLIB, specifying the library where SQL/CA has been uploaded. This is required for VSE versions lower than 2.4. If running 2.4 or higher, you can delete the SETPARM statement, as it has been submitted by the SQLCAI0.VSEJOB, described above.
- Drag and drop the file SQLCAI1.VSEJOB on the SEND2RDR.BAT. This will send and start the job in class 0 and DISP D. The output listing is in DISP H.
- Check the output listing for errors.

2.3.5 Upload the SQLCA OPTIONS file

- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- Edit the file SQLCA.OPTIONS. If needed, change the POWERCLASS statement, which indicates the class where analysis jobs should run.
- Drag and drop the file SQLCA.OPTIONS on the SEND2RDR.BAT. This will send and start the job in class 0 and DISP D. The output listing is in DISP H.
- Check the output listing for errors.

2.3.6 Define the SQL/CA Report Library

- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- Edit the file SQLCAI2.VSEJOB and assign the following SETPARM symbols to define the cluster SQLCA.REPORT.LIBRARY:
  - **CAT** the name of the VSAM catalog for the cluster
  - **VOLUME** the VOLUME parameter
  - **PALLOC** the primary space allocation value as a number of records (16K long)
  - **SALLOC** the secondary space allocation value as a number of records (16K long)
- Drag and drop the file SQLCAI2.VSEJOB on the SEND2RDR.BAT. This will send and start the job in class 0 and DISP D. The output listing is in DISP H.
- Check the output listing for errors.

2.3.7 Define the SQL/CA components to CICS

- The SQLCAI3.VSEJOB assumes that the CICS.CSD cluster is in the VSESPUC user catalog. If this not not the case, re-assign the CAT symbol in the SQLCAI3.VSEJOB file.
- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- Drag and drop the file SQLCAI3.VSEJOB on the SEND2RDR.BAT. This will send and start the job in class 0 and DISP D. The output listing is in DISP H.
- The job starts a DFHCSDUP run that defines the interactive SQL/CA components to CICS, using the group SQLCA.
- Check the output listing for errors.

2.3.8 Update the CICS FCT

- The CICS FCT must have an entry for the SQLCA.REPORT.LIBRARY.
- Copy the member SQLCAFCT.A in your DFHFCT and re-assemble it.

2.3.9 Install SQL/CA in a database

This installation step should be executed in all databases where SQL/CA will be used. Following functions will be performed in the target database:

- Define the userid SQLCA
- Load the SQL/CA packages
- Create the SQL/CA Explain tables

Edit the PC file SQLCAI4.VSEJOB and assign the following SETPARM symbols

CAT
the VSAM catalog for a SAM ESDS workfile (default is VSESPUC)
DB
The name of the target database.
DBAPASS
The password of user SQLDBA in that database.
CAPASS
The password to be assigned to user SQLCA in that database. The SQLCA userid is granted DBA authority by SQLCAI4. The CAPASS value is stored (in encrypted format) into a member of the SPRLIB, for retrieval by the SQL/CA components. The CAPASS value should be **the same in all databases** where SQL/CA is installed.
POOL
The storage pool number for the SQL/CA explain DBspace.
PAGES
The number of pages for the SQL/CA explain tables (minimal size of 128 pages is sufficient).

Submit the SQLCAI4.VSEJOB as follows:

- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- Drag and drop the file SQLCAI4.VSEJOB on the SEND2RDR.BAT. This will send and start the job in class 0 and DISP D. The output listing is in DISP H.
- Check the output listing for errors.

2.3.10 Update the VSE startup stream

If you installed the SPR product SQL/MF previously, skip this section.

If you are running VSE/ESA 2.4 or higher, insert a
// SETPARM SYSTEM,SPRLIB='library_name'
in the $0JCL or USERBG procedure, where 'library_name' is the library where SQL/CA has been catalogued. The SPRLIB variable is needed by several SQL/CA components during execution. VSE versions lower than 2.4 do not provide the SETPARM SYSTEM statement and the SPR library name will be obtained using a component provided by SQL/CA. The SETPARM SYSTEM statement will ensure better performance, as it allows to determine the library name without accessing the library.

2.3.11 Update the VSE standard labels

In your standard labels procedure, insert the following DLBL for the SQL/CA report library:

// DLBL SQLCLIB,'SQLCA.REPORT.LIBRARY',,VSAM,CAT=....,DISP=(OLD,KEEP)

2.3.12 Upload the SQLCASRV startup sample job

- The SQLCASRV VSEJOB starts the SQL/CA server partition.
- Examine the SQLCASRV.VSEJOB and modify the CLASS operand, if needed.
- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- Drag and drop the file SQLCASRV.VSEJOB on the SEND2RDR.BAT. This will store the job in the POWER reader queue, with DISP L.

3 Using SQL/CA

3.1 Using the SQL/CA interactive menu

In a CICS session, enter **$CAM**. This will display the main SQL/CA menu.

Enter one of the following function codes or select the function by positioning the cursor on the function description:

1 Invoke the SQL/CA Librarian (see page 20)
2 Analyze a source program residing in a VSE library (see page 17)
3 Analyze a source program residing in an ICCF library (see page 17)
4 Analyze a source program residing in a VOLLIE library (see page 18)
5 Analyze a DB2/VSE package (see page 18)
6 Analyze a QMF query (see page 18)
7 Analyze an ISQL query (see page 18)
8 Analyze a query entered on the terminal (see page 19)

3.2 Analyze a VSE library member

Supply following panel fields:

- the name of the VSE library
- the name of the VSE sublibrary
- the name of the VSE library member
- the type of the VSE library
- the name of the database where analysis must be performed
- optionally, the table creator to be used for unqualified table name references

Your input is saved in global variables and restored as default on the next invocation of this function.

3.3 Analyze an ICCF library member

Supply following panel fields:

- your ICCF userid
- the ICCF library number
- the name of the ICCF library member
- the name of the database where analysis must be performed
- optionally, the table creator to be used for unqualified table name references

Your input is saved in global variables and restored as default on the next invocation of this function.

3.4 Analyze a VOLLIE library member

Supply following panel fields:

- the name of the VOLLIE library member
- the type of the VOLLIE library member
- the name of the database where analysis must be performed
- optionally, the table creator to be used for unqualified table name references

Your input is saved in global variables and restored as default on the next invocation of this function.

3.5 Analyze a DB2 package

Supply following panel fields:

- the creator of the package
- the name of the package
- the name of the database where analysis must be performed
- optionally, the table creator to be used for unqualified table name references

Your input is saved in global variables and restored as default on the next invocation of this function.

3.6 Analyze a QMF query

Supply following panel fields:

- the creator of the query
- the name of the query
- the name of the database where analysis must be performed
- optionally, the table creator to be used for unqualified table name references

Your input is saved in global variables and restored as default on the next invocation of this function.

3.7 Analyze an ISQL query

Supply following panel fields:

- the creator of the ISQL ROUTINE table containing the query
- the name of the query
- the name of the database where analysis must be performed
- optionally, the table creator to be used for unqualified table name references

Your input is saved in global variables and restored as default on the next invocation of this function.

3.8 Analyze an interactive query

Supply following panel fields:

- the name of the database where analysis must be performed
- optionally, the table creator to be used for unqualified table name references
- the text of the SQL statement to be analyzed, with a maximum length of 10 terminal lines)

3.9 SQL/CA Librarian

3.9.1 Searching the SQL/CA Report Library

SQL/CA analysis reports are identified in the library by:

- the CICS userid that requested analysis
- a reportname
- a report timestamp (to allow multiple analysis reports for the same source)

The report name is the name of the analysis source, that is

- the VSE membername and the membertype
- the ICCF membername and library number
- the package creator and package name
- the query creator and query name
- the string “Interact Query” for interactive statement analysis

To search your reports in the library, enter following panel fields

- your userid (or leave the default CICS id)
- the 2-part reportname

Generic search arguments can be indicated by a trailing asterisk sign. The argument may consist of a single asterisk.

Examples

- to list all your reports, enter your userid and 2 asterisks for the reportname
- to list the analysis reports for all DB2 packagenames starting with ‘AB’, enter your userid, an asterisk for the creator and AB* as the packagename

3.9.2 Processing the Report selection list

Process the list of the selected analyis reports by using following PF key functions:

PF1
Display the SQL/CA help file.

PF2
Refresh the list from the report library.

PF3
Exit from this panel.

PF6
Display first page of the list.

PF7
Display previous page of the list.

PF8
Display next page of the list.

PF9
Display last page of the list.

PF10
Display the report under the cursor by invoking the SQL/CA report editor.

PF11
Print the report under the cursor. The function is not executed online but the request is transmitted to the batch SQL/CA analysis server. The report is printed to the POWER/VSE queue with an LDEST that specifies your CICS userid.

PF12
Delete the report under the cursor. The function is not executed online but the request is transmitted to the batch SQL/CA analysis server.

ENTER
Makes the report under the cursor “current” (and highlights the line).

3.9.3 Processing the analysis report

After invoking the report editor (PF10 on the previous screen), the report is read from the library into a storage list and the first page of the report is shown.

Process the analyis report by using following PF key functions:

PF1
Display the SQL/CA help file.

PF2
Searches a text string in the report.
On the search panel, specify the string to be located and the search direction as > (forward) or < (backward).
Searching is case insensitive and proceeds from the current line + 1. The first line located becomes the current line.

PF3
Terminate the report editor.

PF4
Display the SQL/CA glossary.
The glossary contains
- a detailed description of the warning messages issued during analyis
- suggestions for corrective action
- a description of the DB2 and SQL/CA terminology
- tables that describe some rules applied by the DB2 Optimizer

If the cursor is on an SQL/CA warning ("**AWxx**") when pressing PF4, the Glossary is positioned to the description of that warning.

PF5
Locates the next SQL/CA warning message.

PF6
Display first page of the list.

PF7
Display previous page of the list.

PF8
Display next page of the list.

PF9
Display last page of the list.

PF10
Displays the previous SQL statement in the report.

PF11
Displays the next SQL statement in the report.

PF12
Displays all warning issued for the current SQL statement on a single screen.
The SQL/CA Glossary can be called for each warning in this list, using PF4.

ENTER
Places the report line under the cursor on top of the screen and highlights the line.

3.9.4 Displaying the SQL/CA Glossary

Browse the Glossary by using following PF key functions:

PF2
Searches a text string in the glossary.
On the search panel, specify the string to be located and the search direction as > (forward) or < (backward).
Searching is case insensitive and proceeds from the current line + 1. The first line located becomes the current line.

PF3
Terminate the report editor.

PF6
Display first page of the list.

PF7
Display previous page of the list.

PF8
Display next page of the list.

PF9
Display last page of the list.

ENTER
Places the line under the cursor on top of the screen and highlights it.

3.10 Invoking SQL/CA from a batch job

SQL/CA analysis can be requested from batch, by submitting following JCL:

```
    // JOB
    // LIBDEF PROC,SEARCH=&SPRLIB
    // EXEC SQLCA,SIZE=SQLCA
(1) control_statement_1
(2) control_statement_2
(3) control_statement_3
(4) source text to be analyzed
    /*
    /&
```

Notes

(1) The first control statement specifies the name of the database where analysis should be performed. An optional default creator can be specified as the second word on this statement, if the SQL statements to be analyzed use unqualified tablenames.

(2) Specify an 8-character userid under which the analysis report should be stored in the SQL/CA report library. No checking is done on the validity of this userid and it is used solely as part of the report key. This userid should be used to search the report library after analysis.

(3) Identifies the source for analysis. The first word on the record is the sourcetype. The following words depend on the sourcetype used, as follows

Source type	**word 2**	**word 3**	**word 4**	**word 5**
VSE	library name	sublibrary name	member name	member type
ICCF	library number	member name		
PACKAGE	creator	packagename		
QMF	creator	queryname		
ISQL	creator	queryname		
*				

(4) If the source is in a library, use the * $$ SLI statement to insert it here. With sourcetype *, insert the text of the SQL statement. No data is expected when analyzing a package, a QMF query or an ISQL routine.

(5) By default, the report resulting from the batch analysis is stored in the SQL/CA Report Library. The report should be consulted using the $CAM CICS transaction.
However, if the **PARM='PRINT'** option is included on the // EXEC SQLCA statement, the analysis report will be printed to the POWER list queue only.
The **PARM='PRINTKEEP'** option will print the report to the POWER queue and store the report in the Report Library.

Example

```
// JOB SQLCA
// LIBDEF PROC,SEARCH=<SPRLIB>
// EXEC SQLCA,SIZE=SQLCA
SPRDB2
USER1
PACKAGE SQLDBA DRD10
/*
/&
```

4 The SQL/CA Server

4.1 Starting the SQL/CA server

The server is started with a PRELEASE RDR,SQLCASRV. This command should be in the system startup stream.

The online SQL/CA interface sends analysis requests to the server using XPCC messages to

- perform analysis
- print a named analysis report
- delete a named analysis report

On reception of an analysis request, SQLCASRV builds an SQL/CA jobstream and transmits it to POWER/VSE with the class specified as POWERCLASS in the SQLCA OPTIONS file. If no POWERCLASS statement is found, class Z is used. After the job has been sent, analysis is performed asynchronously and the server is free to accept a new request.

The analysis program itself (SQLCA) serializes the analysis facility, using a VSE lock request. This is necessary because SQLCA has only one set of explain tables. This method of operation will also reduce system load resulting from explain and predicate analysis.

4.2 Stopping the SQL/CA server

The stop the analysis server, issue a VSE **MSG** to the partition. No MSG data is expected.

5 Interpreting the analysis report

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

5.1 Statement Execution Structure by block and parent

SQL/CA formats the SQL statement in such a way, that the execution structure for a multiple query statement 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 statement 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 Statements

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

5.3 Statement Cost Summary

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

- *Total Statement Cost* The query cost estimate from the explain table. The value is zero for INSERT statements.
- *ISQL like Cost* (total_statement_cost / 1000) + 1. Interactive SQL systems such as ISQL present the Query Cost Estimate in this format.

For a multiple query statement, 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 statement. 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[3], 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 statement 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/VSE with the real execution cost.

[3] See *SQL/MF Integration* on page 9.

5.4 Statement Plan Summary

If the analyzed statement 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 statement
- the plan number within the query (1 to N)
- 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 **Statement execution detail** paragraph.

5.5 Statement Execution Structure

The paragraph is printed for each subquery in the SQL statement 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 statement. 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 statements, this total counts the rows for all tables participating in the join.

Estimated global statement filter factor

Represents the fraction of table rows estimated to satisfy the statement 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 statement.

Estimated times predicate conditions satisfied

The TIMES column from the EXPLAIN table. It estimates the probability that the predicate conditions of the statement 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 Statement Execution Detail

The following information is printed for each plan in the query. JOIN queries or statements 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 statement 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. 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[4] accessed in this statement plan needs [does not need] to be sorted.

New table sorted and duplicates removed

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

New table sorted due to ORDER BY

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

New table sorted due to GROUP BY

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

Composite [not] sorted

When a statement 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 statement plan needs to be sorted as part of a JOIN plan.

[4]See the Glossary on page 43 for a definition of "new table".

Composite sorted due to ORDER BY

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

Composite sorted due to GROUP BY

The composite table accessed in this statement plan needs to be sorted to satisfy the statement'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 statement 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 statement 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 statement 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 statement 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.

5.7 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 53 for a detailled description of the warning messages.

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

6 SQL/CA data modelling facility

The SQLCADMF utility programs copies catalog information for a named DBspace between databases.

The utility is invoked as follows:

```
// EXEC SQLCADMF
INSTALL source_database Dbspacename TO target_databasename
/*
```

Where

Source_database
Designates the database from which catalog modelling is performed.

Dbspace name
Is required and designates the PUBLIC 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.

Target_database
Designates the database into which catalog modelling is performed.

The utility copies following catalog columns from source to target:

- SYSDBSPACES
  - NACTIVE
  - NPAGES
- SYSCATALOG
  - ROWCOUNT
  - AVGROWLEN
- SYSCOLUMNS
  - COLCOUNT
  - AVGCOLLEN
- SYSCOLSTATS
  - FREQ1PCT
  - FREQ2PCT
- SYSINDEXES
  - FULLKEYCOUNT
  - FIRSTKEYCOUNT
  - NLEAF
  - NLEVELS
  - CLUSTER
  - CLUSTERRATIO

7 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 statement 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 statement 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 86), 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.

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.

Statement 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 statements 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 statement cost plus the cost of all its dependent blocks.

Highest_Subquery_Cost

The highest Multi_Cost found for a subquery in the statement.

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

Total Statement Cost

The cost estimate associated by DB2 with the statement as a whole.

ISQL-like Cost

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

COMPOSITE

When a statement 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 87.

ESTIMATED GLOBAL Statement FILTER FACTOR

Represents the fraction of table rows estimated to satisfy the statement 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 statement 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 statement. The filtering factors associated with the columns referenced in the statement 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 statements, 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 statement will be true. It also shows how many times eventual dependent blocks will be executed.

FILTER FACTOR

For each column reference in the statement 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 statement 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 statement 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 86.

FULLY-QUALIFIED INDEX SCAN

Refer to the keyword *ACCESS.*

INDEX DISQUALIFIED

An index is present but not used by the optimizer, because the application statement 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 statement 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 statement 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 statement'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/VSE 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/VSE 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. 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* .

8 SQL/CA analysis warning messages

The following messages are issued by the Text Analysis component of SQL/CA. The part of the statement 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/VSE (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 statement 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 statements.

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 85.

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 85.

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 statement 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 statement 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 statement execution.

Action

If the statement logic allows it, select a key-matching operator. See the table **Predicate operator evaluation** on page 86 and the Glossary on page 43 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 statement logic allows it, select a sargable operator. See the table **Predicate operator evaluation** on page 86 and the Glossary on page 43 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 87.

Action

You should try to keep the filter factor as small as possible, since it determines the estimated statement'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 statement 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 statement 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 statement 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 85.

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 85.

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 statement 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 statement 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 statement

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 statement predicate warned.

Action

Repeat the local predicate specified for the first join column as a local predicate for the other join column.

A statement 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 statement 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[5] only when the missing condition is sargable and an equijoin. This was the case for the statement predicate warned.

Action See the Notes under AW25 above.

The statement 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.

[5] in a process called *transitive closure*

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 85.

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 85.

Action:

The warning can be avoided be re-specifying the datatype or scale of the items involved.

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 85.

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 85.

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 85.

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 85.

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 87.

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 statement 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 statement 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 statement does not have the named column as the first index column. Therefore, fast index locate is not used.

Action:

Attempt to rewrite the statement, 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 statement 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/VSE 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 statement 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 statement 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 statement 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 43 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 statement 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 statement:

*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 statements 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 statement

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 statement 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 statement'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 statement has been issued for the table.

Consequently, no information is available the DB2/VSE Optimizer for determining the best data access path.

Action:

Issue the UPDATE STATISTICS statement for the table or DBspace.

AW94 Catalog statistics available for index columns only

Reason:

An UPDATE STATISTICS statement has been issued for the table without the ALL option.

Consequently, the DB2/VSE 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/VSE Optimizer.

Consider an UPDATE ALL STATISTICS for statements that perform poorly or that do not execute using the expected access path.

9 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 statements 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/VSE 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.

10 Predicate evaluation tables

10.1 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.[6]

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

10.2 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

[6]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.

10.3 Predicate operator evaluation table

OPERATOR	*KEYMATCH*	*SARGABLE*	*DEFAULT FILTER*
=	yes	yes	0,04
> < >= <= value	yes	yes	0,333
BETWEEN	yes	yes	0,1
IS NULL	yes	yes	0,04
IN(valuelist)	yes	yes	0.040*(size-of-list)
LIKE char	yes	yes	0,1
= **Q** (query)[7]	yes	yes	0,04
> < >= <= **Q** (query)	yes	yes	0,333
<> **Q** (query)	no	yes	0,96
<> value	no	yes	0,96
IS NOT NULL	no	yes	0,96
= (query)	no	no	0,04
<> (query)	no	no	0,96
> < >= <= (query)	no	no	0,333
= expression	no	no	0,04
<> expression	no	no	0,96
> < >= <= expression	no	no	0,333
[NOT] IN(query)	no	no	1
LIKE pattern	no	no	0,1
LIKE host variable	no	no	0,1
NOT BETWEEN	no	no	0,9
NOT IN(valuelist)	no	no	1-(0.040*(size-of-list))
NOT LIKE	no	no	0,9

[7] **Q** is one of the quantifiers **ANY**, **ALL** or **SOME**

10.4 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

11 SQL/CA messages

SQLCA003: Application nnnn does not contain static SQL statements

The analyzed source does not contain static (prepped) analysable SQL statements at all.

SQLCA005: RC xx VSAMRC yy AFTER {PDSOPEN | PDSWRITE}

An error has occurred when opening or writing to the SQL/CA Report Library using PDSAM.

XX represents the hexadecimal PDSAM returncode as follows:

08	ACB CONTAINS INVALID SPECIFICATIONS
0C	MEMBER NOT FOUND FOR DISP=OLD
10	VSAM ERROR (DETAILS IN ACBVSAMRC)
14	PDSOPEN HAS NOT BEEN ISSUED
18	INVALID RECORD LENGTH ON OUTPUT

YY represents the hexadecimal VSAM errorcode when XX=10

Action

If XX=10, use the value of YY to determine the VSAM errorcode. These errorcodes are documented in the VSE Messages and Codes manual.

In other cases, call for software support.

SQLCA007: SQLCODE ... WHEN CONNECTING DATABASE .....

Issued when a CONNECT to the database fails.

SQLCA011: ALL HOSTVAR REFERENCES REPLACED WITH CONSTANTS.

Issued when the SQL EXPLAIN statement results in an SQLCODE indicating incompatible parameter marker usage. Since an EXPLAIN statement cannot contain hostvariables, SQL/CA replaces them with parameter markers. Hostvariables and parameter markers are compatible, except for a number of cases, described in the IBM manual "DB2 Application Reference" (PREPARE statement). If such an exception is detected, SQL/CA replaces all hostvariable references with a constant specification of the corresponding format and retries EXPLAIN with the modified statement.

SQLCA012: TABLE (creator) tablename DOES NOT EXIST

The DB2 object name extracted from the SQL statement 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 statement 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.

SQLCA017: SQLCODE ... ON EXPLAIN Statement

An SQLCODE has been returned when invoking SQL EXPLAIN for the application statement displayed immediately before the above message. The application statement is probably in error and will not appear in the analysis report.

12 Index